ESPRESSO: A FAST END-TO-END NEURAL SPEECH RECOGNITION TOOLKIT

Yiming Wang[1], *Tongfei Chen*[1], *Hainan Xu*[1], *Shuoyang Ding*[1], *Hang Lv*[1,4], *Yiwen Shao*[1],
Nanyun Peng[3], *Lei Xie*[4], *Shinji Watanabe*[1], *Sanjeev Khudanpur*[1,2]

[1] Center of Language and Speech Processing, [2] Human Language Technology Center of Excellence,
Johns Hopkins University, Baltimore, MD, USA
[3] Information Sciences Institute, University of Southern California, Los Angeles, CA, USA
[4] ASLP@NPU, School of Computer Science, Northwestern Polytechnical University, Xi'an, China
{yiming.wang,tongfei,hxu31,dings,yshao18,shinjiw,khudanpur}@jhu.edu, npeng@isi.edu, {hanglv,lxie}@nwpu-aslp.org

ABSTRACT

We present ESPRESSO, an open-source, modular, extensible end-to-end neural automatic speech recognition (ASR) toolkit based on the deep learning library PyTorch and the popular neural machine translation toolkit FAIRSEQ. ESPRESSO supports distributed training across GPUs and computing nodes, and features various decoding approaches commonly employed in ASR, including look-ahead word-based language model fusion, for which a fast, parallelized decoder is implemented. ESPRESSO achieves state-of-the-art ASR performance on the WSJ, LibriSpeech, and Switchboard data sets among other end-to-end systems without data augmentation, and is 4–11× faster for decoding than similar systems (e.g. ESPNET).

***Index Terms*—** automatic speech recognition, end-to-end, parallel decoding, language model fusion

1. INTRODUCTION

Various open-source toolkits for building automatic speech recognition (ASR) systems have been created, with a notable example being Kaldi [1], a weighted finite state transducer based framework with extensive linear algebra support, that enables traditional hybrid ASR systems [2].

With advances in deep learning, recent work in ASR begun paying attention to so-called *neural end-to-end* systems [3, 4, 5, *inter alia*], which are characterized by generally smaller code size, and greater portability and maintainability across hardware platforms and software environments. This shift is analogous to the one in the machine translation (MT) community: from feature- and syntax-based statistical machine translation (SMT) systems (e.g. Moses [6], Joshua [7]) to end-to-end neural machine translation (NMT) systems (e.g. OPENNMT [8], OPENSEQ2SEQ [9], FAIRSEQ [10]). As a result, there have been a few efforts in the ASR research community to create open source neural end-to-end frameworks, most notably ESPNET [11] (See also, Table 1). However, ESPNET has some important shortcomings: (i) the code is not very easily extensible and has portability issues due to its mixed dependency on two deep learning frameworks PyTorch [12] and Chainer [13]; (ii) the decoder, which uses a simple but relatively slow beam search algorithm, is not fast enough for quick turnaround of experiments.

To address these problems, we present ESPRESSO, a novel neural end-to-end system for ASR.[1] ESPRESSO builds upon the popular NMT framework FAIRSEQ, and the flexible deep learning framework PyTorch. By extending FAIRSEQ, ESPRESSO inherits its excellent extensibility: new modules can easily be plugged into the system by extending standard PyTorch interfaces. Additionally, we gain ability to perform distributed training over large data sets for ASR.

We also present the first fully parallelized decoder for end-to-end ASR, with look-ahead word-based language model fusion [19], tightly integrated with the existing sets of optimized inference algorithms (e.g. beam search) inherited from FAIRSEQ and tailored to the scenario of speech recognition. Furthermore, an improved *coverage* mechanism is proposed to further reduce deletion and insertion errors, and is compared with related techniques such as EOS threshold [20]. ESPRESSO provides recipes for a variety of benchmark ASR data sets, including WSJ [21], LibriSpeech [22], and Switchboard [23], and achieves *state-of-the-art results* on these data sets.

ESPRESSO, by building upon FAIRSEQ, also has the potential to integrate seamlessly with sequence generation systems from natural language processing (NLP), such as neural machine translation and dialog systems. We envision that ESPRESSO could become the foundation for unified speech + text processing systems, and pave the way for future end-to-end speech translation (ST) and text-to-speech synthesis (TTS) systems, ultimately facilitating greater synergy between the ASR and NLP research communities.

2. SOFTWARE ARCHITECTURE AND DESIGN CHOICES

We implement ESPRESSO with the following design goals in mind:

- Pure Python / PyTorch that enables modularity and extensibility;
- Parallelization and distributed training and decoding for quick turnaround of experiments;
- Compatibility with Kaldi / ESPNET data format to enable reuse of previous / proven data preparation pipelines;
- Easy interoperability with the existing FAIRSEQ codebase to facilitate future joint research areas between speech and NLP.

We elaborate our technical rationale in the following sections.

2.1. Input format and dataset classes

Our speech data follows the format in Kaldi, where utterances are stored in the Kaldi-defined SCP format, consisting of space-

This work was partially supported by the IARPA MATERIAL program and by an unrestricted gift from Mobvoi. The authors also thank Ziyan Jiang for running speed tests on ESPNET.

[1] `https://github.com/freewym/espresso`.

Table 1. Popular end-to-end neural ASR systems and our system.

Name	Language	Deep Learning Framework	Recipes	Other Applications
EESEN [14]	C++	—	WSJ, LibriSpeech, SWBD, TED-LIUM, HKUST	—
ESPNET [11]	Python	Chainer & PyTorch	various	TTS, ST
E2E LF-MMI [15]	C++	Kaldi	various	—
LINGVO [16]	Python	TensorFlow	LibriSpeech	NMT
OPENSEQ2SEQ [9]	Python	TensorFlow	LibriSpeech	NMT, TTS
RETURNN [17]	Python	Theano, TensorFlow	WSJ, LibriSpeech, SWBD, CHiME	NMT
WAV2LETTER++ [18]	C++	ArrayFire	WSJ, LibriSpeech, TIMIT	—
ESPRESSO	Python	PyTorch	WSJ, LibriSpeech, SWBD	NMT

delimited lines that follows this template:

```
<UttID> <FeatureFile>:<Offset>
```

where <UttID> is the utterance ID, a key that points to any utterance in the dataset, and <FeatureFile> is a string interpreted as an extended filename for reading from a binary file (ARK format) storing the actual acoustic feature data, following the practice[2] in Kaldi.

In theory, any kind of acoustic feature vectors (e.g. MFCC) can be stored in the feature file. In ESPRESSO, we follow ESPNET and employ the commonly used 80-dimensional log Mel feature with the additional pitch features (in total, 83 dimensions for each frame).

In FAIRSEQ, there is a concept called "datasets", which contains a set of training samples and abstracts away details such as shuffling and bucketing. We follow this and create the following dataset classes in ESPRESSO:

- `data.ScpCachedDataset`: this contains the real-valued acoustic features extracted from the speech utterance. Each training batch drawn from this dataset is a real-valued tensor of shape [BatchSize × TimeFrameLength × FeatureDims] that will be fed to the neural speech encoder (Section 2.3). Since the acoustic features are large and cannot be loaded into memory all at once, we also implement sharded loading, where given the order of the incoming examples in an epoch, a bulk of features is pre-loaded once the previous bulk is consumed for training / decoding. This helps balance the file system's I/O load and the memory usage.
- `data.TokenTextDataset`: this contains the gold speech transcripts as text. Each training batch is a integer-valued tensor of shape [BatchSize × SequenceLength], where each integer in this tensor is the index of the character / subword unit in the token dictionary (see below).
- `data.SpeechDataset`: this is a container for the two datasets above: each sample drawn from this dataset contains two fields, `source` and `target`, that points to the speech utterance and the gold transcripts respectively.

Note that in speech recognition, the token dictionary (set of all vocabulary) is different from the common practice in FAIRSEQ due to the additional special token <space>. For this reason, we do not directly use the `data.Dictionary` class from FAIRSEQ, instead, we inherit that class and create our `data.TokenDictionary` class for this purpose, with the extra functionality of handling <space>.

For speech decoding purposes rather than NMT (default in FAIRSEQ), normally the output unit for each decoding step is a *subword unit* instead of a word, since it is shown that for ASR using whole words as modeling units is only possible when large amounts of training data (at least tens of thousands of hours) is available [24, 25]. A subword unit can either be a *character* or *character sequence* like BPE [26] or a SentencePiece[3] [27]. Both are supported in ESPRESSO and experimental results will follow.

2.2. Output format

ESPRESSO supports two output format: a *raw* format and a more detailed *aligned results* version that helps debugging.

The raw format just consists of space-delimited lines that follows this template:

```
<UttID> <DecodedSequence>
```

where <UttID> is the original utterance ID from the SCP dataset, and the <DecodedSequence> is the raw output of the speech recognition system.

The *aligned results* provide an aligned sequence between the gold reference transcript and the predicted hypothesis. An example is shown below:

```
4k9c030b
REF: "QUOTE AN EYE FOR AN EYE "UNQUOTE
HYP: "QUOTE AN EYE FOR    ANY "END-QUOTE
STP:                   D  S   S
WER: 42.86%
```

Each such record consists of 5 rows: the first line is the utterance ID; `REF` and `HYP` is the reference transcript and the system output hypothsis respectively – these two are aligned using minimal edit distance. The fourth line, `STP` (step), contains the error the system makes at each decoding step: it could be one of `S` (substitution), `I` (insertion) and `D` (deletion), corresponding to the three types of errors when evaluating the word error rate (WER) commonly used to evaluate speech recognition systems. The last line is WER calculated on this utterance. Such output format facilitates easy human inspection to the different error types made by the system, rendering debugging easier for researchers.

2.3. Encoder-Decoder

ESPRESSO supports common sequence generation models and techniques arisen from the research in the ASR and NLP community. The *de facto* standard model, the encoder-decoder with attention [28, 29] (also successfully pioneered by [4] in the speech community), is implemented as our `models.speech_lstm.SpeechLSTMModel` class. Owing to the modularity and extensibility of ESPRESSO, other models, e.g., Transformer [30], can be easily integrated from the underlying FAIRSEQ.

[2] `https://kaldi-asr.org/doc/io.html`.

[3] `https://github.com/google/sentencepiece`.

CNN-LSTM Encoder The default encoder we used is a 4-layer stacked 2-dimensional convolution (with batch normalization between layers), with kernel size $(3, 3)$ on both the time frame axis and the feature axis [31, 11]. 2×-downsampling is employed at layer 1 and 3, resulting in $1/4$ time frames after convolution. The final output channel dimensionality is 128, with the 21 downsampled frequency features, hence a total of $128 \times 21 = 2688$ dimensional features for each time frame.

Then 3 layers of bidirectional LSTMs [32] are stacked upon the output channels yielded by the stacked convolution layers.

This architecture, with the various dimensionality, number of layers, and other hyperparameters customizable, is implemented in our `models.speech_lstm.SpeechLSTMEncoder` class.

LSTM Decoder with Attention We use a 3-layer LSTM decoder by default, with Bahdanau attention [28] on the encoded hidden states (Luong attention [29] is also implemented). We follow the architecture in the Google Neural Machine Translation (GNMT) system [33], where the context vectors generated by the attention mechanism is fed to all 3 layers of the decoding LSTM. Residual connections [34] are added between the decoder layers. These are implemented in the `models.speech_lstm.SpeechLSTMDecoder` class.

2.4. Training Strategies

Scheduled Sampling Scheduled sampling [35] is supported by our system, with promising results in end-to-end speech recognition [36]. With scheduled sampling, at each decoder step, the gold label is fed to the next step with p probability, whereas the predicted token[4] is fed with $(1 - p)$ probability. In our implementation such mechanism will start at an intermediate epoch N in the training process: the first few epochs are always trained with gold labels. The probability p can be scheduled in the training process: later epochs might use a smaller probability to encourage the model not to rely on the gold labels.

Label Smoothing Label smoothing [37] has been proposed to improve accuracy by computing the loss (i.e., cross entropy here) not with the "hard" (one-hot) targets from the dataset, but with a weighted mixture of these targets with some distribution [38]. We support three kinds of these distributions in ESPRESSO:

- *Uniform smoothing* [37]: The target is a mixture of $(1 - p)$ probability of the one-hot target and the rest of the p probability mass uniformly distributed across the vocabulary set;
- *Unigram smoothing* [39]: Mixed with a unigram language model trained on the gold transcripts;
- *Temporal smoothing* [40]: Mixed with a distribution assigning probability mass to neighboring tokens in the transcript. Intuitively, this smoothing scheme helps the model recover from beam search errors: the network is more likely to make mistakes that simply skip a subword unit of the transcript.

Model Selection via Validation At the end of each training epoch, we compute the WER on the validation set using greedy-search decoding without language model fusion (see Section 3). This is different from the approach in previous frameworks such as ESPNET and FAIRSEQ, where they compute the loss function on the validation set (the gold labels are fed in) to perform model selection. We argue that our approach may be more suitable since free decoding on the validation set is a closer scenario to the final metric on test sets. Owing to efficiency concerns, we do not use full-blown language model fused beam search decoding for validation (arguably this is even better).

We employ learning rate scheduling following FAIRSEQ: at the end of an epoch, if the metric on the validation set is not better than the previous epoch, the learning rate is reduced by a factor (e.g. $1/2$). Empirically we found that the reduction of the learning rate will be less frequent if using WER as the validation metric as compared to the loss value on the validation set. According to [41], a less frequent learning rate reduction generally leads to better performance.

3. LANGUAGE MODEL-FUSED DECODING

It is shown in recent research that a pure sequence-to-sequence transduction model for ASR without an external language model component (which is used in traditional hybrid ASR systems) is far from satisfactory [42]. This is in contrast with neural machine translation (NMT) models, where normally no external language model is needed. This performance gap is hypothesized to be caused by the fact that the ASR model is only trained on speech-transcript pairs. The gold transcript set is not large enough to produce state-of-the-art neural language models, which are typically trained on a corpus on the scale of 1 billion words.

In ESPRESSO, we employ *shallow fusion* [43] as a language model integration technique, which is proven to be effective in speech recognition [42, 44]. The LSTM decoder with shallow fusion computes a weighted sum of two posterior distributions over subword units: one from the end-to-end speech recognition model, the other from the external neural language model.

We support 3 types of external neural language models:

- *Subword-unit language model*: A vanilla LSTM-based language model trained on subword units. Here subword units can either just be characters (with `<space>` as an additional special token) or trained subword units (e.g. BPE [26] or SentencePiece [27]);
- *Multi-level language model* [45]: This is a combination of character-based and word-based language models. Hypotheses in the beam are first scored with the character-based language model until a word boundary (`<space>`) is encountered. Known words are then re-scored using the word-based language model, while the character-based language model provides for out-of-vocabulary scores;
- *Look-ahead word-based language model* [19]: This model enables outputting characters for each decoding step with a pre-trained word-based language model, by providing look-ahead word probabilities based on the word prefix (sequence of characters) decoded. This is shown in [19] to be superior to the multi-level language model.

3.1. Parallelization with Look-ahead Word-based LMs

The original implementation of the look-ahead word-based language model in ESPNET [11] is not operating on batches, making the decoding speed slow. In ESPRESSO, we devise a fully-parallelized version of the decoding algorithm on GPUs.

In [19], a word-based language model is converted to a character-based one via a technique using *prefix trees*. The *prefix tree automaton* $T = (\Sigma, Q, \varepsilon, \tau, V)$ is a finite-state automaton (see Fig. 1):

- Σ is the character set (including `<space>`);
- $V \subseteq \Sigma^*$ is the *word vocabulary set* and also the final state set;

[4] This is the token with the maximum posterior probability resulting from the previous LSTM decoder step. It may not necessarily be gold.

- $Q = \{p \sqsubseteq w \mid w \in V\} \subseteq \Sigma^*$ is the set of all *prefixes*[5] of the words in V and also the state set;
- ε is the empty string, which also serves as the initial state;
- $\tau : Q \times \Sigma \to Q$ is the state transition function, where given a state and an input character, $\tau(p, \mathtt{c}) = p\mathtt{c}$, i.e. a simple concatenation.

A look-ahead word-based LM computes the probability of the next character $\mathtt{c} \in \Sigma$ based on a given word history $\mathbf{h}$ and a word prefix $p \in Q$ (i.e., a state in the prefix tree automaton):

$$P(\mathtt{c} \mid p, \mathbf{h}) = \frac{\sum_{s:\, p\mathtt{c}s \in V} P_{\mathrm{W}}(p\mathtt{c}s \mid \mathbf{h})}{\sum_{s:\, ps \in V} P_{\mathrm{W}}(ps \mid \mathbf{h})}. \tag{1}$$

where $P_{\mathrm{W}}(w \mid \mathbf{h})$ is the probability of the word w predicted by the word-based LSTM language model. In Eq. (1), the numerator is the sum of the probability of all words prefixed by $p\mathtt{c}$, i.e. all possible words that could be generated from $p\mathtt{c}$ if the state is moved from p to $p\mathtt{c}$; the denominator is the sum of the probability of all possible words at the current state p (see Fig. 1).

[19] proposed an efficient way to compute the sum in Eq. (1). We denote p precedes q *lexicographically* as $p \prec q$, and define the *upper bound* $\overline{p}$ (the lexicographically greatest element prefixed by p) and *lower bound* $\underline{p}$ (the greatest element lexicographically less than any word prefixed by p) as:

$$\overline{p} = \max_{w \in V, p \sqsubseteq w} w; \quad \underline{p} = \max_{w \in V, p \not\sqsubseteq w, w \prec p} w \tag{2}$$

Given that the vocabulary set is sorted lexicographically, we can efficiently compute the sum of the probability of all words *preceding or equal to* a given word, using efficient routines like `cumsum`:

$$g(w \mid \mathbf{h}) = \sum_{w' \preceq w} P_{\mathrm{W}}(w' \mid \mathbf{h}) \tag{3}$$

Using these definitions, Eq. (1) can be rewritten as

$$P(\mathtt{c} \mid p, \mathbf{h}) = \frac{g(\overline{p\mathtt{c}} \mid \mathbf{h}) - g(\underline{p\mathtt{c}} \mid \mathbf{h})}{g(\overline{p} \mid \mathbf{h}) - g(\underline{p} \mid \mathbf{h})} \tag{4}$$

This method is illustrated in Fig. 1, showing that the probability of a character given a prefix can be efficiently computed via a `cumsum` operation and simple arithmetics.

In our parallelized implementation, each prefix p (corresponding to a state in the automaton) is indexed as a unique integer. Hence the batch of decoding states is compactly stored as a vector of integers, each corresponding to a state on the prefix tree automaton. The automaton itself is stored as a matrix $\mathbf{T}$ with shape [NumberOfStates × MaxOutDegree], where the row T_p contains the index of all descendants of p, logically forming an adjacency list. The index of the $\overline{p}$ and $\underline{p}$ for each state p is also precomputed and cached. In sum, the entire prefix tree automaton is vectorized.

To compute $P(\mathtt{c} \mid p, \mathbf{h})$ for all $\mathtt{c} \in \Sigma$ over batches and beam hypotheses, the following steps are executed:

(i) Update $P_{\mathrm{W}}(w \mid \mathbf{h})$ for all $w \in V$ from a specific decoding step of the word-based language model for those hypotheses that encounter the end-of-word (`<space>`) symbol in the batch;

(ii) Update the $g(w \mid \mathbf{h})$ function using $P_{\mathrm{W}}(w \mid \mathbf{h})$ for all $w \in V$;

(iii) Get all possible successive states;

(iv) Get all upper and lower bounds for all successive states;

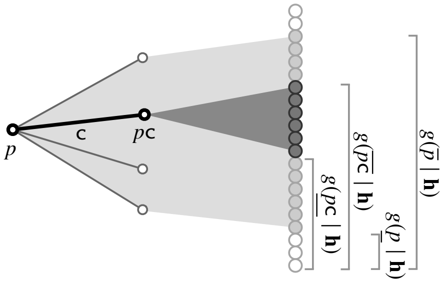
p
c
pc
$g(\underline{pc} \mid \mathbf{h})$
$g(\overline{pc} \mid \mathbf{h})$
$g(\underline{p} \mid \mathbf{h})$
$g(\overline{p} \mid \mathbf{h})$

Fig. 1. The efficient look-ahead LM algorithm. The dark gray and light gray subtrees correspond to the probability mass spanned by prefix $p\mathtt{c}$ and p (numerator and denominator in Eq. (1)) respectively. These can be efficiently computed via Eq. (4) using `cumsum`.

(v) Compute the probability for each $\mathtt{c} \in \Sigma$ according to Eq. (4).

Note that the first step follows a batched forward computation of a neural language model; the third and the fourth steps can be computed via tensorized advanced indexing; and the second and the last steps can be executed using vectorized arithmetics. Hence we obtain a fully parallelized algorithm that runs on GPUs.

As mentioned in the first step, at a specific decoding step, some elements in the batch may encounter the end-of-word symbol, whereas others may not: running this conditional operation requires special treatment. We devise an algorithm that shares the spirit with [46], an parallelized algorithm for stack LSTM parsing: first we record the elements in the batch that has not reached the end of a word with a mask, then progress the state for one step for all of these elements, finally, for those masked states, their original states are restored, and only $P_{\mathrm{W}}(w \mid \mathbf{h})$ of those not masked are updated.

3.2. Improved Coverage

With language model fusion, the decoder tends to make more deletion errors when the language model weight becomes larger [47]. A coverage term (a scalar value assigned to each hypothesis in the beam) is first proposed in [40] to promote longer transcripts and also to prevent attentions from looping over utterance (repeating certain n-grams when decoding) when using shallow fusion:

$$\text{coverage} = \textstyle\sum_j \mathbb{1}\left[\sum_i a_{ij} > \tau\right] \tag{5}$$

where a_{ij} is the attention weight for the decoder step i and encoder frame j, $\tau > 0$ is a tunable hyper-parameter, and $\mathbb{1}[\cdot]$ is the indicator function. This is the total number of encoder frames that has been sufficiently "attended" to. However, based on our experiments, applying the coverage term Eq. (5) is not sufficient to prevent words from being repeated. Instead we propose a modified version of the coverage term which penalizes looping attentions more aggressively:

$$\begin{aligned} \text{coverage} \quad = \quad & \textstyle\sum_j \Big(\mathbb{1}\left[\sum_i a_{ij} > \tau_1\right] \\ & -\mathbb{1}\left[\sum_i a_{ij} > \tau_2\right] \cdot \Big(c + \sum_i a_{ij} - \tau_2\Big)\Big) \end{aligned} \tag{6}$$

where $\tau_2 > \tau_1 > 0, c > 0$ are tunable hyper-parameters. While the first term in Eq. (6) is exactly the same as Eq. (5), the second term penalizes the hypothesis score when the accumulated attention on encoder frame j exceeds τ_2. Specifically, if the accumulated attention weight on frame j exceeds τ_1 but not τ_2, only the first term is activated, increasing the coverage score to encourage more attention on

[5] We denote "p is a prefix of q" as $p \sqsubseteq q$: e.g. "stand" $\sqsubseteq$ "standard".

frame j; if the accumulated attention weight further exceeds τ_2, the second term (with the minus sign) is also activated and its magnitude is the amount of the exceeding value plus a constant c, discouraging further attention accumulated on the same frame. Therefore, the new coverage mechanism enforces a soft constraint on the accumulated attention weight on each frame to be between τ_1 and τ_2, leading to both less deletion errors and less repeating n-grams (shown in the WSJ part of Section 4). During beam search decoding this new coverage term as a whole is added to the hypothesis score with a weight (e.g. 0.01). In our experiments, $\tau_1 = 0.5, \tau_2 = 1.0, c = 0.7$.

3.3. EOS Threshold

We implement the end-of-sentence threshold technique proposed in [20] to bias the decoder away from short transcriptions when decoding with a fused language model. End-of-sentence (`<eos>`) tokens can only be emitted if its probability is greater than a specific factor of the top output token candidate during beam search:

$$\log P(\texttt{<eos>} \mid \mathbf{h}) > \gamma \cdot \max_{t \in V} \log P(t \mid \mathbf{h}) \tag{7}$$

where V is the vocabulary set. In our experiments, γ is set to 1.5.

4. RECIPES AND RESULTS

ESPRESSO provides running recipes for a variety of well-known data sets. We elaborate the details of our recipes on Wall Street Journal [21] (WSJ), an 80-hour English newspaper speech corpus, LibriSpeech [22], a corpus of approximately 1,000 hours of read English speech, and Switchboard [23] (SWBD), a 300-hour English telephone speech corpus.

Besides the transcripts, all of these data sets have their own extra text corpus for training language models. Input and output word embeddings are tied [48] to reduce model size. All the models are optimized using Adam [49] with an initial learning rate 10^{-3}, and then halved if the metric on the validation set at the end of an epoch does not improve over the previous epoch. The training process stops if the learning rate is less than 10^{-5}. Curriculum learning [50], which is quite helpful to stabilize training with long sequences (e.g. LibriSpeech) and improve performance (esp. SWBD), is employed for the first 1 (LibriSpeech) or 2 (WSJ / SWBD) epochs. All the models are trained / evaluated using NVIDIA GeForce GTX 1080 Ti GPUs. If not otherwise specified, all the models throughout this paper are trained with 2 GPUs using FAIRSEQ built-in distributed data parallelism. Note that no data augmentation techniques such as speed-perturbation [51] or the more recent SpecAugment [41] is applied.

The hyper-parameters for the recipes are listed in Table 2.

Table 2. Hyper-parameters for the three recipes.

Hyper-parameter	WSJ		LibriSpeech		SWBD	
	LM	ASR	LM	ASR	LM	ASR
Vocab. size	65k	52	5k	5k	1k	1k
Encoder # layers	-	3	-	4	-	4
Decoder # layers	3	3	4	3	3	3
Emb. dim.	1,200	48	800	1,024	1,800	640
Hidden dim.	1,200	320	800	1,024	1,800	640
# Params.	113M	18M	25M	174M	80M	70M
Dropout rate	0.35	0.4	0.0	0.3	0.3	0.5
Avg. batch size	435	48	1,733	34	1,783	69
Beam size	50		60		35	
LM fusion weight	0.9		0.47		0.25	

Table 3. WERs (%) on the WSJ dev93 and eval92 set.

	dev93	eval92
Hadian et al.[8] [15]	-	4.1
Baskar et al. (ESPNET) [36]	-	3.8
Likhomanenko et al. [55]	6.4	3.6
Zeghidour et al. [53]	6.8	3.5
Amodei et al.[9] (Deep Speech 2) [56]	*4.4*	*3.1*
ESPRESSO LSTM	14.8	12.1
+Look-ahead Word LM	7.4	5.1
+Improved Coverage	**5.9**	3.5
+EOS Threshold	**5.9**	**3.4**

WSJ We adopt the look-ahead word-based language model [19] as the external language model. We report the perplexities on the validation / evaluation set: the external word-based language model achieves 72.0 perplexity on dev93 and 59.0 on eval92.

For the encoder-decoder model, the vocabulary size of subword units (characters) for WSJ is 52, the same as in ESPNET.[6] Temporal label smoothing with $p = 0.05$ and scheduled sampling with $p = 0.5$ starting at epoch 6 are adopted.

Baseline end-to-end systems are compared: Hadian et al. [15], an end-to-end[7] model with the lattice-free MMI objective [52]; Baskar et al. [36], an encoder-decoder model with discriminative training in ESPNET; Zeghidour et al. [53], a pure convolutional network with ASG loss [54]; Likhomanenko et al. [55], a lexicon-free decoding method with the acoustic model proposed in [53]; and the last one, Deep Speech 2 [56].

We show the beam search decoding results of various configurations of ESPRESSO in Table 3 with beam size 50. The breakdown of the three kinds of errors is shown in Table 4. The first row gives WERs where no language model fusion is applied. The second row is after integrating the look-ahead word-based language model, with its optimal LM fusion weight 0.5. Although it has already significantly reduced the overall WER, deletion errors increase. Further applying the improved coverage yields better performance by suppressing deletion errors. If we only use the first term in Eq. (6) which is equivalent to the coverage term in [40], the insertions errors will increase from 0.8 to 1.3 on dev93, and from 0.6 to 0.9 on eval92. A manual inspection of the decoded results reveals that these additional insertions are mostly repeated words. This observation validates the effectiveness of our improved coverage mechanism. Alternatively, if we apply the EOS threshold, we achieve state-of-the-art performance on WSJ among end-to-end models.

LibriSpeech SentencePiece is used as the subword units in our LibriSpeech setup for both external language modeling and encoder-decoder model. We combine dev-clean and dev-other sets together as a single validation set for both language model and encoder-decoder model training.

[6] It includes 45 characters constituting the training transcripts, plus 3 atomic symbols: `<*IN*>`, `<*MR.*>` and `<NOISE>`, plus 4 special symbols: `<pad>`, `<eos>`, `<unk>` and `<space>`.

[7] This is a hybrid system. "End-to-end" here is in the sense that it does not need HMM-GMM training or tree-building steps.

[8] The result is with "full bichar" (using all possible 2-gram characters as context-dependent modeling units), data speed-perturbation, and a 3-gram word language model.

[9] It uses 12,000 hours transcribed data for acoustic model training and large crawled text for language model training, making it not comparable.

Table 4. Breakdown of the WERs (%) on WSJ.

	dev93			eval92		
	Sub	Ins	Del	Sub	Ins	Del
Espresso LSTM	12.0	1.4	1.4	9.7	1.5	1.0
+Look-ahead Word LM	4.6	**0.8**	2.0	3.1	0.7	1.3
+Improved Coverage	4.3	**0.8**	**0.8**	2.7	0.6	**0.3**
+EOS Threshold	**4.1**	0.9	**0.8**	**2.6**	**0.5**	**0.3**

Table 5. WERs (%) on the LibriSpeech dev and test sets.

	dev		test	
	clean	other	clean	other
Zeghidour et al. [53]	3.1	10.0	3.3	10.5
Hannun at al. [20]	3.0	8.9	3.3	9.8
Park et al. [41] (w/o SpecAugment)	-	-	3.2	9.8
Lüscher et al. [57]	**2.6**	**8.4**	**2.8**	9.3
Espresso LSTM	3.8	11.5	4.0	12.0
+Subword LM	3.3	8.9	3.4	9.5
+Improved Coverage	2.9	8.8	3.2	9.0
+EOS Threshold	2.8	**8.4**	**2.8**	**8.7**

Table 6. WERs (%) on the SWBD Hub5'00 evaluation set.

	Switchboard	CallHome
Cui et al. [58] (w/ speed-pertubation)	12.0	23.1
Zeyer et al. [59]	11.0	23.1
Park et al. [41] (w/o SpecAugment)	10.9	19.4
Espresso LSTM	10.7	20.7
+Subword LM	**9.2**	**19.1**

Table 7. Training (per epoch) and decoding wall time on WSJ.

	Training		ASR Decoding (eval92)	
	LM	ASR	w/o LM	w/ look-ahead LM
ESPnet	56min	36min	5min 21s	29min 16s
Espresso	**46min**	**31min**	**1min 27s**	**2min 44s**
Speedup	17.9%	16.0%	3.7×	10.7×

Again, we report the perplexities on the validation / evaluation sets. Espresso obtains 35.4 and 37.3 on the dev-clean and dev-other sets, and 37.2 and 37.2 on test-clean and test-other.

Uniform smoothing with $p = 0.1$ is applied throughout the entire training. No scheduled sampling is used. The vanilla shallow fusion is used without the "look-ahead" technique. The results, along with several baseline systems, are demonstrated and compared in Table 5,

We can see that both the improved coverage or EOS threshold help in this setup as well, where actually deletion error reductions contribute mostly. In addition, we achieve state-of-the-art results on end-to-end models for LibriSpeech without any data augmentation.

Switchboard The vocabulary consists of 1,000 subword units segmented by SentencePiece[10] trained on both Switchboard and Fisher transcripts. As there is no official validation set, we hold out the same 4,000-example subset of the training data as in Kaldi for validation. We apply scheduled sampling starting at epoch 6 with probability from 0.9 to 0.6. Uniform smoothing is used with $p = 0.1$.

The language model achieves a validation perplexity of 17.3. No coverage is used during decoding. The results of our current system and 3 other competitive end-to-end baselines are shown in Table 6. Again, we obtain state-of-the-art results without SpecAugment. The coverage term or EOS threshold does not help on this dataset, and we suspect it is because its optimal LM fusion weight is not as large as those for the other two datasets, resulting in less deletion errors.

5. TRAINING AND DECODING SPEED

In this section we compare Espresso and ESPnet on the training and decoding time with single GPU, using the WSJ dataset.

For fair comparison, we create architectures in Espresso (different from those in Section 4) that mimics the WSJ recipe in ESPnet as closely as possible. Data preparation and vocabulary building are identical. The neural architecture is mostly the same (e.g. number and dimension of LSTM layers), with a few minor exceptions: e.g. ESPnet's use of location-based attention (which Espresso does not employ), pooling CNN layers (Espresso uses strided CNN without pooling), and joint cross-entropy+CTC loss (Espresso uses only cross-entropy loss).

Table 7 gives training wall time comparisons on both the external word-based language model and the encoder-decoder model, which are averaged over 20 epochs and 15 epochs respectively. Espresso is 17.9% faster than ESPnet on the language model training, and 16.0% faster on the encoder-decoder model training. We conjecture that the main reason of such speed gain for language model training is that in fairseq (and hence in Espresso) training examples are sorted based on input sequence lengths before batching (i.e., bucketing; ESPnet does not use it for language modeling), so that the average sequence length in batches is smaller.

A notable advantage of Espresso compared to ESPnet is the decoding speed. In order to have a more fair comparison, we enable GPU batch decoding in ESPnet [60], and make batch and beam sizes of the two systems the same. We measure the decoding time on the WSJ eval92 data set, which consists of 333 utterances. Table 7 shows that, without language model fusion, Espresso is 3.7× faster than ESPnet. With the look-ahead language model fusion, the speedup is even more prominent—more than 10× faster—mostly due to our parallelized implementation of the look-ahead language model fusion (cf. Section 3.1), as opposed to ESPnet, where LM scores are computed iteratively over examples within a minibatch.

6. CONCLUSION

In this paper we present Espresso, an open-source end-to-end ASR toolkit built on top of fairseq, an extensible neural machine translation toolkit. In addition to advantages inherited from fairseq, Espresso supports various other features for ASR that are seamlessly integrated with fairseq, including reading data in Kaldi format, and efficient parallelized language model-fused decoding. We also provide ASR recipes for WSJ, LibriSpeech, and Switchboard data sets, and achieve state-of-the-art performance among end-to-end systems. By sharing the underlying infrastructure with fairseq, we hope Espresso will facilitate future joint research in speech and natural language processing, especially in sequence transduction tasks such as speech translation and speech synthesis.

[10] It includes additional special tokens `[laughter]`, `[noise]`, and `[vocalized-noise]`.

7. REFERENCES

[1] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *Proc. ASRU*, 2011.

[2] Dong Yu and Li Deng, *Automatic Speech Recognition: A Deep Learning Approach*, Springer Publishing Company, Incorporated, 2014.

[3] Alex Graves and Navdeep Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. ICML*, 2014, pp. 1764–1772.

[4] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "End-to-end continuous speech recognition using attention-based recurrent NN: first results," *Deep Learning and Representation Learning Workshop, NeurIPS*, vol. abs/1412.1602, 2014.

[5] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. ICASSP*, 2016, pp. 4960–4964.

[6] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al., "Moses: Open source toolkit for statistical machine translation," in *Proc. ACL: Demo and Poster Sessions*, 2007, pp. 177–180.

[7] Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren NG Thornton, Jonathan Weese, and Omar F Zaidan, "Joshua: An open source toolkit for parsing-based machine translation," in *Proc. the Fourth Workshop on Statistical Machine Translation*, 2009, pp. 135–139.

[8] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush, "OpenNMT: Open-source toolkit for neural machine translation," in *Proc. ACL: System Demonstrations*, 2017, pp. 67–72.

[9] Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman, Vitaly Lavrukhin, Carl Case, and Paulius Micikevicius, "OpenSeq2Seq: Extensible toolkit for distributed and mixed precision training of sequence-to-sequence models," in *Proc. Workshop for NLP Open Source Software (NLP-OSS)*, 2018, pp. 41–46.

[10] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proc. NAACL-HLT: Demonstrations*, 2019, pp. 48–53.

[11] Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai, "ESPnet: End-to-end speech processing toolkit," in *Proc. INTERSPEECH*, 2018, pp. 2207–2211.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in PyTorch," in *NeurIPS Autodiff Workshop*, 2017.

[13] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton, "Chainer: a next-generation open source framework for deep learning," in *Proc. Workshop on Machine Learning Systems (LearningSys) in NeurIPS*, 2015.

[14] Yajie Miao, Mohammad Gowayyed, and Florian Metze, "EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding," in *Proc. ASRU*, 2015, pp. 167–174.

[15] Hossein Hadian, Hossein Sameti, Daniel Povey, and Sanjeev Khudanpur, "End-to-end speech recognition using lattice-free MMI," in *Proc. INTERSPEECH*, 2018, pp. 12–16.

[16] Jonathan Shen, Patrick Nguyen, Yonghui Wu, Zhifeng Chen, et al., "Lingvo: a modular and scalable framework for sequence-to-sequence modeling," *CoRR*, vol. abs/1902.08295, 2019.

[17] Albert Zeyer, Tamer Alkhouli, and Hermann Ney, "RETURNN as a generic flexible neural toolkit with application to translation and speech recognition," in *Proc. ACL, System Demonstrations*, 2018, pp. 128–133.

[18] Vineel Pratap, Awni Hannun, Qiantong Xu, Jeff Cai, Jacob Kahn, Gabriel Synnaeve, Vitaliy Liptchinsky, and Ronan Collobert, "Wav2letter++: A fast open-source speech recognition system," in *Proc. ICASSP*, 2019, pp. 6460–6464.

[19] Takaaki Hori, Jaejin Cho, and Shinji Watanabe, "End-to-end speech recognition with word-based RNN language models," in *Proc. SLT*, 2018, pp. 389–396.

[20] Awni Hannun, Ann Lee, Qiantong Xu, and Ronan Collobert, "Sequence-to-sequence speech recognition with time-depth separable convolutions," in *Proc. INTERSPEECH*, 2019, pp. 3785–3789.

[21] Douglas B. Paul and Janet M. Baker, "The design for the wall street journal-based CSR corpus," in *Proc. ICSLP*, 1992, pp. 357–362.

[22] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. ICASSP*, 2015, pp. 5206–5210.

[23] John Godfrey, Edward Holliman, and Jane McDaniel, "Switchboard: Telephone speech corpus for research and development," in *Proc. ICASSP*, 1992, vol. 1, pp. 517–520.

[24] Hagen Soltau, Hank Liao, and Hasim Sak, "Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition," in *Proc. of INTERSPEECH*, 2017, pp. 3707–3711.

[25] Kartik Audhkhasi, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, and Michael Picheny, "Building competitive direct acoustics-to-word models for english conversational speech recognition," in *Proc. ICASSP*, 2018, pp. 4759–4763.

[26] Rico Sennrich, Barry Haddow, and Alexandra Birch, "Neural machine translation of rare words with subword units," in *Proc. ACL*, 2016, pp. 1715–1725.

[27] Taku Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," in *Proc. ACL*, 2018, pp. 66–75.

[28] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR*, 2015.

[29] Thang Luong, Hieu Pham, and Christopher D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. EMNLP*, 2015, pp. 1412–1421.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Proc. NeurIPS*, 2017, pp. 6000–6010.

[31] Yu Zhang, William Chan, and Navdeep Jaitly, "Very deep convolutional networks for end-to-end speech recognition," in *Proc. ICASSP*, 2017, pp. 4845–4849.

[32] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber, "Bidirectional LSTM networks for improved phoneme classification and recognition," in *Proc. ICANN*, 2005, pp. 799–804.

[33] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.

[35] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. NeurIPS*, 2015, pp. 1171–1179.

[36] Murali Karthick Baskar, Lukáš Burget, Shinji Watanabe, Martin Karafit, Takaaki Hori, and Jan Honza Černocký, "Promising accurate prefix boosting for sequence-to-sequence ASR," in *Proc. ICASSP*, 2019, pp. 5646–5650.

[37] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna, "Rethinking the inception architecture for computer vision," in *Proc. CVPR*, 2016, pp. 2818–2826.

[38] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton, "When does label smoothing help?," *CoRR*, vol. abs/1906.02629, 2019.

[39] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton, "Regularizing neural networks by penalizing confident output distributions," in *Proc. ICLR, Workshop Track*, 2017.

[40] Jan Chorowski and Navdeep Jaitly, "Towards better decoding and language model integration in sequence to sequence models," in *Proc. INTERSPEECH*, 2017, pp. 523–527.

[41] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," in *Proc. INTERSPEECH*, 2019, pp. 2613–2617.

[42] Anjuli Kannan, Yonghui Wu, Patrick Nguyen, Tara N. Sainath, Zhijeng Chen, and Rohit Prabhavalkar, "An analysis of incorporating an external language model into a sequence-to-sequence model," in *Proc. ICASSP*, 2018, pp. 5824–5828.

[43] Çaglar Gülçehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loïc Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "On using monolingual corpora in neural machine translation," *CoRR*, vol. abs/1503.03535, 2015.

[44] Shubham Toshniwal, Anjuli Kannan, Chung-Cheng Chiu, Yonghui Wu, Tara N. Sainath, and Karen Livescu, "A comparison of techniques for language model integration in encoder-decoder speech recognition," in *Proc. SLT*, 2018, pp. 369–375.

[45] Takaaki Hori, Shinji Watanabe, and John R. Hershey, "Multi-level language modeling and decoding for open vocabulary end-to-end speech recognition," in *Proc. ASRU*, 2017, pp. 287–293.

[46] Shuoyang Ding and Philipp Koehn, "Parallelizable stack long short-term memory," in *Proc. the Third Workshop on Structured Prediction for NLP*, 2019, pp. 1–6.

[47] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio, "End-to-end attention-based large vocabulary speech recognition," in *Proc. ICASSP*, 2016, pp. 4945–4949.

[48] Ofir Press and Lior Wolf, "Using the output embedding to improve language models," in *Proc. EACL*, 2017, pp. 157–163.

[49] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.

[50] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston, "Curriculum learning," in *Proc. ICML*. ACM, 2009, pp. 41–48.

[51] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur, "Audio augmentation for speech recognition," in *Proc. INTERSPEECH*, 2015, pp. 3586–3589.

[52] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur, "Purely sequence-trained neural networks for ASR based on lattice-free MMI," in *Proc. INTERSPEECH*, 2016, pp. 2751–2755.

[53] Neil Zeghidour, Qiantong Xu, Vitaliy Liptchinsky, Nicolas Usunier, Gabriel Synnaeve, and Ronan Collobert, "Fully convolutional speech recognition," *CoRR*, vol. abs/1812.06864, 2018.

[54] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *CoRR*, vol. abs/1609.03193, 2016.

[55] Tatiana Likhomanenko, Gabriel Synnaeve, and Ronan Collobert, "Who needs words? lexicon-free speech recognition," in *Proc. INTERSPEECH*, 2019, pp. 3915–3919.

[56] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al., "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *Proc. ICML*, 2016, pp. 173–182.

[57] Christoph Lüscher, Eugen Beck, Kazuki Irie, Markus Kitza, Wilfried Michel, Albert Zeyer, Ralf Schlüter, and Hermann Ney, "RWTH ASR systems for LibriSpeech: Hybrid vs attention," in *Proc. INTERSPEECH*, 2019, pp. 231–235.

[58] Jia Cui, Chao Weng, Guangsen Wang, Jun Wang, Peidong Wang, Chengzhu Yu, Dan Su, and Dong Yu, "Improving attention-based end-to-end ASR systems with sequence-based loss functions," in *Proc. SLT*, 2018, pp. 353–360.

[59] Albert Zeyer, André Merboldt, Ralf Schlüter, and Hermann Ney, "A comprehensive analysis on attention models," in *Proc. IRASL Workshop, NeurIPS*, 2018.

[60] Hiroshi Seki, Takaaki Hori, Shinji Watanabe, Niko Moritz, and Jonathan Le Roux, "Vectorized beam search for CTC-attention-based speech recognition," in *Proc. INTERSPEECH*, 2019, pp. 3825–3829.