ABSTRACT

Self-supervised learning (SSL) has shown promise in learning representations of audio that are useful for automatic speech recognition (ASR). But, training SSL models like wav2vec 2.0 requires a two-stage pipeline. In this paper we demonstrate a single-stage training of ASR models that can utilize both unlabeled and labeled data. During training, we alternately minimize two losses: an unsupervised masked Contrastive Predictive Coding (CPC) loss and the supervised audio-to-text alignment loss Connectionist Temporal Classification (CTC). We show that this joint training method directly optimizes performance for the downstream ASR task using unsupervised data while achieving similar word error rates to wav2vec 2.0 on the Librispeech 100-hours dataset. Finally, we postulate that solving the contrastive task is a regularization for the supervised CTC loss.

Index Terms— Self-supervision, Contrastive learning, Joint training, Semi-supervised, Speech recognition

1. INTRODUCTION

Deep learning has been impactful in building state-of-the-art end-to-end speech recognition systems [1, 2, 3]. But, they typically require large amounts of annotated speech data in the form of transcripts. Whereas, humans are able to learn language and speech with little supervision.

Recently, self-supervised learning (SSL) has been proposed as a method for training automatic speech recognition (ASR) models by pre-training on large amount of unlabeled data and then fine-tuning the speech recognition model on labeled data, for example contrastive predictive coding (CPC) [4]. While these methods [5, 6] have achieved impressive results on low-resource speech datasets, their goal is to learn speech representations that are useful for multiple speech-related tasks. Training an ASR model using SSL methods is a two-stage process as it requires running separate pre-training and fine-tuning experiments and jointly tuning hyperparameters for both stages. It is unclear how much pre-training is required to achieve reasonable performance on the downstream task of speech recognition.

In this paper, we propose a training method for ASR models that combines SSL and supervised learning in a single stage. The model is trained by jointly minimizing a loss on labeled data, and a loss on unlabeled data. The supervised loss is the Connectionist Temporal Classification (CTC) loss [7], while the unsupervised loss is based on a masked variant of CPC. As both losses are optimized jointly, our method allows early stopping by measuring the performance of the model for the downstream task on the validation dataset.

We show that a model trained using our method (with no quantization) achieves equivalent word error rate (WER) when trained on 960-hours of unlabeled data and 100-hours of labeled data to a model that is trained using the two-stage process of wav2vec 2.0 [8] (with quantization), which is a method based on masked CPC. Additionally, we verify that our method provides a regularization to the supervised loss when only using labeled data.

2. JOINT TRAINING

We propose to train our speech recognition model in a single stage, by jointly minimizing a supervised and an unsupervised loss. Our training procedure alternates between minimizing the unsupervised loss on unlabeled data and minimizing the supervised loss on labeled data.
2.1. Model

Our model is a neural network architecture which gets as input raw audio \((x)\) and outputs token \((y)\) probabilities \(p_\theta(y_t|x)\) at time \(t\) with the following functions:

\[
\begin{align*}
  z &= f(x) & (1) \\
  \hat{z} &= g(\text{mask}(z)) & (2) \\
  p_\theta(y_t|x) &= h(\hat{z}). & (3)
\end{align*}
\]

where a convolutional encoder \(f: X \rightarrow Z\) maps raw input audio into features at 20ms stride with a receptive field 30ms. These encoder features \(z\) (with optional masking of certain frames) are passed as input into a transformer-based \(g\) context network with full attention \(g: Z \rightarrow \hat{Z}\). Finally, the context features \(\hat{z}\) are used to generate output token probabilities \(p_\theta(y_t|x)\) at time frame \(t\) using a linear layer and softmax non-linearity \(h: \hat{Z} \rightarrow Y\).

2.2. Unsupervised and supervised losses

The supervised loss is CTC [7], denoted as \(L_s(\theta, x, y)\) in the paper. The unsupervised loss is the self-supervision loss used for pre-training in wav2vec [8]. This loss can be viewed as a contrastive predictive coding [4] loss, where the task is to predict the masked encoder features [10] rather than predicting future encoder features given past encoder features. In this loss, a certain percentage of the encoder features \(z\) (controlled by the masking probability) are masked at time frames \(t_{i_1}, t_{i_2}, ..., t_{i_T}\), where \(i_1, i_2, ..., i_T\) denote the masking indices. The features, for example \(z_{i_{1, t}}\), are masked by replacing it with a learnt mask embedding. The masked encoder features \(\hat{z} = \text{mask}(z)\) are passed as input to the context network, which is responsible for reproducing the features \(z\). The accuracy of reproduction is measured using a contrastive network, which is responsible for reproducing the features \(z\) given the input features \(\hat{z}\) of the context network at masked indices (anchor) and the input features \(z\) of the context network at masked indices (positive sample) against other encoder features at non-masked indices (negative samples):

\[
L_u(\theta, x) = \frac{1}{T} \sum_t -\log \frac{s(z_t, \hat{z}_t)}{s(z_t, \hat{z}_t) + \sum_{t'} s(z_{t'}, \hat{z}_t)} & (4)
\]

where \(s(z_t, \hat{z}_t) = \frac{1}{\tau} \exp\left(\frac{-\|z_t - \hat{z}_t\|}{\|z_t\| \|\hat{z}_t\|}\right)\). The time frame \(t\) denotes the index of the \(T\) masked features, \(z_{t'}\) are encoder features sampled from time frames \(t'\) other than time frame \(t\), \(\tau\) is a tunable hyperparameter called temperature.

2.3. Alternate minimization

The model is trained by alternately minimizing the two losses. Using a minibatch from the unlabeled data, the gradient of the unsupervised loss is used to update the model parameters for \(N\) steps, followed by the gradient of the supervised loss (using a minibatch from labeled data) for 1 step. This process is repeated until convergence of the word error rate on the validation dataset. A brief description is shown in Algorithm[1].

Separate adaptive momentum optimizers are used for each of the two losses with different learning rates: \(\eta_u\) for the unsupervised loss and \(\eta_s\) for the supervised loss. The two optimizers maintain their state independently, while sharing the parameters of the model. This ensures that the momentum averaging for one loss is not affected by the gradient updates from the other loss, leading to faster convergence. Experiments with a single optimizer show worse performance on the downstream task compared to the usage of two optimizers.

The ratio of unsupervised to supervised loss updates, \(N:1\), is chosen to be 1:1. This results in equal opportunity for the unsupervised and supervised tasks to affect the weights of the network as a function of the total number of updates. Choosing an update ratio that favors the unsupervised task results in a more computationally expensive training. While, an update ratio that is biased towards the supervised task produces an ASR model that does not improve over a supervised baseline.

The learning rate ratio is biased towards the unsupervised task as compared to the supervised task. Using a learning rate ratio of 1:1 or one that favors the supervised task results in an ASR model that does not improve over a supervised baseline.

3. EXPERIMENTAL SETUP

3.1. Datasets

The experiments use the Librispeech [11] 960-hours dataset as the unsupervised dataset. The supervised dataset is a subset of Librispeech: either 100-hours or 960-hours (full). During training, samples in the dataset that are smaller than 2 seconds or longer than 33 seconds are filtered out. The performance of the trained model is validated on the dev-clean/other datasets of Librispeech and tested on the test-clean/other datasets.

3.2. Architecture details

Similar to wav2vec 2.0 [8], the convolutional encoder network consists of a stack of 7 convolutions with kernel size \((10, 3, 3, 3, 3, 3, 2)\) and strides \((5, 2, 2, 2, 2, 2, 2)\) respectively. The number of input and output channels in the convolution is 512. Additionally, the input audio is normalized in the time dimension before it is passed into the convolutional encoder.

We use two versions of the model, \texttt{BASE} and \texttt{LARGE}. The transformer context network for the \texttt{BASE} model is composed of a convolutional relative positional embedding layer with kernel size 128 and group size 16, followed by a stack of 12 transformer layers with 8 heads. The hidden dimension is 768 and the feed-forward network dimension is 3072. Each transformer layer uses layer dropout [12] with probability 0.05 and dropout with probability 0.1. The transformer context network for the \texttt{LARGE} model uses a stack of 24 transformer
layers with 16 heads. The hidden dimension is 1024 and the feed-forward network dimension is 4096. Each transformer layer uses layer dropout with probability 0.2 and dropout with probability 0.1. The linear classifier is trained to output letter-based tokens, which consist of 26 English alphabet letters, augmented with the apostrophe and a word boundary token.

The total number of updates is 500K. The LR for the both losses is warmed up from 0 to their respective values in 20K updates. After the warmup period, the LR of the unsupervised loss $\eta_u$ is decayed to $0.1\eta_u$ at the end of training, whereas the LR of the supervised loss is kept constant. SpecAugment in the supervised loss update is activated after the warmup period.

Training is performed on 64 V100 GPUs with a batch size per GPU equal to 87.5s of audio for the BASE model and on 256 V100 GPUs with a batch size per GPU equal to 40s of audio for the LARGE model. The audio samples are batched together such that the total length of the samples does not exceed the batch size. The model is trained using the wav2letter++ toolkit [15] for approximately 4 days.

### 3.4. Beam-search decoding and rescoring

Besides reporting word error rate (WER) without a language model (LM), we also perform a one-pass beam-search decoder with a 4-gram word-level LM [16] and further the beam rescoring with a strong word-level Transformer LM [17]. We rely on the beam-search decoder from the wav2letter++ toolkit [15] and follow the procedure from [17].

### 4. RESULTS AND DISCUSSION

#### 4.1. Evaluation on standard SSL datasets

The single-stage training pipeline is evaluated in a setting where there is a large amount of unlabeled data compared to labeled data.

Table 1 shows word error rates (with and without an LM, see Section 3.4) for the BASE model trained on Librispeech 960-hours unlabeled data and 100-hours labeled data. The joint training procedure generates an ASR model that matches the WER of the wav2vec 2.0 BASE model on both the test-clean and test-other datasets. Unlike the wav2vec 2.0 model, this model does not include quantization, operates in the continuous space and does not use any unsupervised loss penalty terms during training. Using the two-stage pipeline of wav2vec 2.0 (reproduced in wav2letter++) to train the continuous BASE model results in slightly worse ASR performance compared to the quantized wav2vec 2.0 BASE model.

Table 2 shows word error rates (with and without an LM, see Section 3.4) for the LARGE model trained on Librispeech 960-hours unlabeled data and 100-hours labeled data. The joint training procedure generates an ASR model that matches the WER of the wav2vec 2.0 LARGE model on both the test-clean and test-other datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>LM</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noisy student</td>
<td>LSTM</td>
<td>3.9</td>
<td>8.8</td>
</tr>
<tr>
<td>wav2vec BASE</td>
<td>None</td>
<td>6.1</td>
<td>13.5</td>
</tr>
<tr>
<td>(quantized)</td>
<td>4-gram</td>
<td>2.7</td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>Transf.</td>
<td>2.6</td>
<td>7.0</td>
</tr>
<tr>
<td>wav2vec BASE</td>
<td>None</td>
<td>6.0</td>
<td>14.3</td>
</tr>
<tr>
<td>(continuous,</td>
<td>4-gram</td>
<td>3.2</td>
<td>8.9</td>
</tr>
<tr>
<td>reproduction)</td>
<td>Transf.</td>
<td>1.9</td>
<td>8.1</td>
</tr>
<tr>
<td>Joint BASE</td>
<td>None</td>
<td>6.1</td>
<td>13.7</td>
</tr>
<tr>
<td>(continuous)</td>
<td>4-gram</td>
<td>3.0</td>
<td>7.7</td>
</tr>
<tr>
<td></td>
<td>Transf.</td>
<td>2.1</td>
<td>6.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>LM</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noisy student</td>
<td>LSTM</td>
<td>3.9</td>
<td>8.8</td>
</tr>
<tr>
<td>wav2vec LARGE</td>
<td>None</td>
<td>4.6</td>
<td>9.3</td>
</tr>
<tr>
<td>(quantized)</td>
<td>4-gram</td>
<td>2.3</td>
<td>5.7</td>
</tr>
<tr>
<td></td>
<td>Transf.</td>
<td>2.1</td>
<td>4.8</td>
</tr>
<tr>
<td>Joint LARGE</td>
<td>None</td>
<td>4.2</td>
<td>8.9</td>
</tr>
<tr>
<td>(continuous)</td>
<td>4-gram</td>
<td>2.6</td>
<td>6.1</td>
</tr>
<tr>
<td></td>
<td>Transf.</td>
<td>2.0</td>
<td>5.1</td>
</tr>
</tbody>
</table>
4.2. Effect of hyperparameters on downstream task

Table 3 shows the effect of different hyperparameters on the ASR performance of the model trained using the single-stage training method. All models are trained for 500K updates using the Librispeech 960-hours dataset as the unsupervised dataset and the 100-hours dataset as the supervised dataset. The baseline model uses a \( \mathcal{L}_u \) to \( \mathcal{L}_s \) update ratio equal to 1:1, \( \mathcal{L}_u \) to \( \mathcal{L}_s \) learning rate ratio equal to 20:1 and separate optimizers for each of the two losses. Using a lower \( \mathcal{L}_u \) to \( \mathcal{L}_s \) learning rate ratio or using a single optimizer results in a higher WER on the dev-other dataset compared to the baseline. The training pipeline is not sensitive to the update ratio as can be seen by the negligible difference in WER between the models with a \( \mathcal{L}_u \) to \( \mathcal{L}_s \) loss update ratio 1:1 and 5:1.

Table 3. Word error rate (dev-other dataset, 4-gram LM) of models with different hyperparameters compared to baseline.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Updates</th>
<th>LR</th>
<th>dev-other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1:1</td>
<td>20:1</td>
<td>8.0</td>
</tr>
<tr>
<td>( \mathcal{L}_u ) to ( \mathcal{L}_s ) update ratio</td>
<td>5:1</td>
<td>20:1</td>
<td>7.9</td>
</tr>
<tr>
<td>( \mathcal{L}_u ) to ( \mathcal{L}_s ) learning rate ratio</td>
<td>1:1</td>
<td>4:1</td>
<td>9.0</td>
</tr>
<tr>
<td>Single optimizer</td>
<td>1:1</td>
<td>20:1</td>
<td>11.1</td>
</tr>
</tbody>
</table>

4.3. Regularization effect on supervised loss

Figure 1 shows a plot of the unsupervised loss \( \mathcal{L}_u \) and the supervised loss \( \mathcal{L}_s \) on the train (Librispeech 960-hours) and validation (Librispeech dev-other) datasets as a function of total number of updates for the BASE model trained using either joint training or supervised only training. Both models are trained for the same total number of updates, 500K. The supervised loss attains a lower value on the validation dataset and a higher value on the train dataset with joint training in comparison to supervised only training. Furthermore, Table 4 shows that a model trained using joint training achieves lower WER (with and without an LM) compared to a model trained using supervised loss only, even though it has a lower number of updates from this loss. This suggests that our method provides a regularizing effect to the supervised loss.

Table 4. Word error rates of models trained on Librispeech 960-hours labeled dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>LM</th>
<th>Dev clean</th>
<th>Test clean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>None</td>
<td>3.2</td>
<td>10.8</td>
</tr>
<tr>
<td></td>
<td>4-gram</td>
<td>2.1</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td>Transf.</td>
<td>1.5</td>
<td>5.4</td>
</tr>
<tr>
<td>Joint training</td>
<td>None</td>
<td>3.4</td>
<td>9.0</td>
</tr>
<tr>
<td></td>
<td>4-gram</td>
<td>2.1</td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td>Transf.</td>
<td>1.5</td>
<td>4.4</td>
</tr>
</tbody>
</table>

5. RELATED WORK

This paper draws upon recent advances in self-supervised contrastive learning \([8,18,19]\). It uses the principle of contrastive learning: similarity between an anchor and positive samples is compared against similarity with negative samples. But, the goal of self-supervised learning is to learn representations that are useful for multiple downstream tasks. Whereas, our method is designed to maximize performance on a single downstream task.

More broadly, our single-stage training method can be linked to semi-supervised learning or self-training methods \([20,21,3,22]\) for ASR. These methods bootstrap an acoustic model (AM) from transcriptions (labeled data), transcribe unlabeled audio with the trained AM (optionally with the help of an LM) and then retrain the AM on the generated pseudo-labels. Self-training methods are complementary to our method and there is potential to combine the two methods.

As our approach addresses both, a contrastive learning task and speech recognition task, this paper is related to the field of multi-task learning \([23,24]\). Recent approaches to multi-task learning \([25,26]\) solve the tasks by minimizing a loss, containing multiple terms, on the same supervised datasets. Whereas, in our method, the unsupervised and supervised losses are minimized on their respective datasets.

6. CONCLUSION

Our single-stage training method simplifies the process for learning speech recognition models jointly from labeled and unlabeled data and allows directly optimizing the model on the downstream task. Furthermore, the trained models match the performance of state-of-the-art self-supervised models for speech that use a two-stage pipeline. Finally, we demonstrate that solving the contrastive task provides a regularizing effect on the supervised loss when only using a labeled dataset.

Finally, we would like to thank Alexei Baevski and Michael Auli for helpful discussions regarding wav2vec 2.0.
7. REFERENCES


