Forced Alignment with Deep Convolutional Neural Networks

Sean Farhat

Abstract

In this report, we introduce a novel method for Forced Alignment of audio transcripts using Convolutional Neural Networks. By examining neural network based methods for automatic speech recognition (ASR), we realized that the reliance of such models on the standard sequence-to-sequence loss function, Connectionist Temporal Classification (CTC), prevents such networks from achieving this task; instead, CTC's main benefit, being alignment-free, becomes the main hindrance. By defining a 1-to-1 alignment for input/output sequences, we were able to use the Negative Log Likelihood instead, whose easily interpretable nature allows for a Forced Alignment to be created. By utilizing a fully convolutional model for transcription generation and a word-boundary generator based solely off the Levenshtein-distance and a pronunciation dictionary, we achieve a highly efficient method relative to comparable methods (DTW, HMM-GMM/DNN, RNN) in addition to a low average alignment error of 0.067 seconds on the TIMIT dataset.

1. Introduction

The problem of forced alignment has been approached from multiple angles, sometimes directly via methods such as Dynamic Time Warping, yet mostly from the perspective of a side effect of Automatic Speech Recognition (ASR): Hidden Markov Models (and their hybird Gaussian Mixture Models or Deep Neural Network variants), as well as recurrent models such as Recurrent Neural Networks, Gated Recurrent Units, Long Short-Term Memory, and recently, Transformers. These models explicity model each timestep conditioned on the past and/or future, so a Forced Alignment could be easily generated as an interpretability step. Some work has been done on convolutional models, yet these have mostly been in tandem with recurrent models, as they attempt to leverage the advantages of both (Gulati et al., 2020).

While newer flavors of models for ASR follow a sequence-to-sequence scheme as popularized in the Listen-Attend-Spell architecture (Chan et al., 2015), classic models utilized a loss and inference method known as Connectionist Temporal Classificatiton (CTC) (Graves et al., 2006; Hannun, 2017). Since its inception, it has been ubiquitous for a variety of sequence classification tasks, from ASR to handwriting recognition. In the realm of ASR, it has been especially useful given its versatility to work in the realm of high level outputs such as characters instead of phonemes, allowing it to be used with most datasets.

However, CTC's alignment-free nature causes it to lack interpretability, therefore preventing many models from obtaining a Forced Alignment or even working with standard interpretability methods such as activation maps (Zhou et al., 2016; Selvaraju et al., 2019) or saliency maps (Simonyan et al., 2014).

We investigate CTC's tradeoff on being alignment-free vs. its saved cost in preprocessing, and how utilizing the TIMIT dataset to create an optimal alignment allows us to leave CTC behind, thus enabling faster traininig/inference in addition to being able to generate Forced Alignments and Class Action Maps.

2. Motivation

For other Forced Alignment pipelines, the process usually starts with an existing ASR model and leveraging its properties to extract the alignment. So, we will do the same, but consider a CNN as the main underlying model instead. We begin by considering the task of speech recognition, which can formally defined as an instance of the sequence classification problem: for a given input sequence $X = x_1 \dots x_n$ we want to know its corresponding output sequence $Y = y_1 \dots y_m$, which contains labels from an alphabet $\mathcal{X}$.

2.1. Maximum Likelihood Expectation (MLE)

One approach to this problem is a conditionally independent discriminative model, a notable assumption that differentiates it from other popular methods such as HMM's or RNN's. We generate a probability distribution $\hat{P}(Y|X;\theta) \in [0,1]^{|\mathcal{X}|\times m}$. Training is equivalent to maximizing the probability of the correct output given the model and inputs; this quantity is also known as the **likelihood**, hence the name

Maximum Likelihood Estimation (MLE):

$$\max_{\theta} \hat{P}(Y|X;\theta) = \prod_{t=1}^{m} \hat{P}_t(y_t|X;\theta), Y = y_1 \dots y_m \quad (1)$$

Inference comes from:

$$\hat{y}_t = \arg\max_{y_i} \hat{P}_t(y_i|X;\theta), \forall t \in [1,m] \quad (2)$$

To utilize popular descent methods as well as avoid numerical underflow, instead of maximizing Equation 1, we minimize its negative logarithm instead, hence the name Negative Log Likelihood (NLL) Loss. From here onwards, we will use the terms "training" and "aligning" interchangeably with regards to this problem, as we are intuitively teaching the model to align the input sequence to its corresponding output sequence during training. In addition, "alignment" will refer to the output of the model, $\hat{Y}$.

2.2. Connectionist Temporal Classification

A common case to consider is when the length of the input and output differ, $n \neq m$, and no direct alignment between $X \to Y$ exists, i.e. $x_i \to y_j, \forall i, j$. For these instances, the most common approach is a sequence-to-sequence model that will generate $\hat{P} \in [0,1]^{|\mathcal{X}| \times n}$, that is a distribution with respect to the length of the input, *not* the ouput, and use Connectionist Temporal Classification (CTC) for training and decoding. Below we will briefly cover the intuition behind CTC and why it is *not* helpful for our purposes.

2.2.1. The Loss Function

As CTC functions with probabilistic models, it follows the logical approach to maximize the likelihood of the desired sequence. However, what makes this algorithm stand out is its loose definition of "desired"; that is, CTC computes the MLE over a set of length n sequences it refers to as "valid alignments".

To do so, CTC adds a blank label $(-)$ to $\mathcal{X}$, which the model can learn to infer, but which is not present in any of the training/test data. This character intuitively represents a transition between labels in the sequence. It then defines a collapsing function $\beta : (\mathcal{X} \cup -)^n \to \mathcal{X}^m$ which takes in a sequence, collapses all repeats, then removes all blanks. So the set of valid CTC alignments is formally defined as:

$$\mathcal{A} = \beta^{-1}(Y) \quad (3)$$

and an illustrative description of β can be found in Figure 1.

CTC, as a loss function, is equivalent to NLL, except that it sums over all valid alignments and treats them as equally desirable outputs. Mathematically,

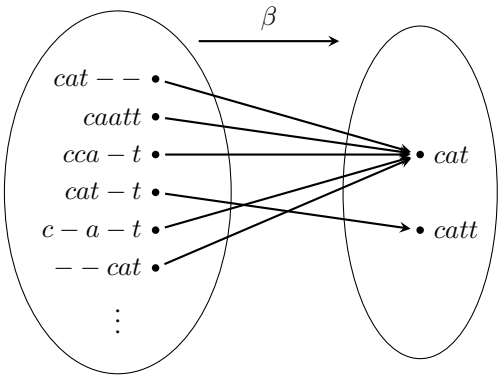

Figure 1. An illustration of CTC's collapsing procedure for $Y = cat, |X| = 5$. Every element that maps to cat is a valid alignment. Its surjective nature hampers interpretability and adds computational complexity.

$$\max_{\theta} \hat{P}(Y|X;\theta) = \sum_{A \in \mathcal{A}} \prod_{t=1}^{m} \hat{P}_t(a_t|X;\theta), A = a_1 \dots a_n \quad (4)$$

When implemented, the summation is intractable, so often a Dynamic Programming approach is taken to reduce the runtime.

2.2.2. Inference

For inference, multiple heuristics can be chosen between. One approach is to greedily take the $\arg\max \hat{P}_t, \forall t$, similar to Equation 2. However, this is not always optimal given CTC's conditional independence assumption. More likely, a beam search is utilized in combination with a language model for decoding, which in practice has led to high accuracy results.

2.3. Shortcomings of CTC

While CTC's lack of reliance on a predefined alignment is a huge benefit in terms of versatility with datasets, this alignment-free nature is a double-edged sword.

Having mutliple valid alignments is realized mathematically as multiple valid paths through $\hat{P}$ that must all be considered at training and test time, necessitating a computationally expensive DP approach. In addition, there is no clear output class to take the gradient with respect to, preventing most kinds of classification interpretability methods such as saliency maps or class activation maps. Lastly, a Forced Alignment cannot be generated, as, at each timestep, more than 1 label could be valid, e.g. the blank and another character.

2.4. The Optimal Alignment

Consider this: if we are able to make a strong guarantee that $|X| = |Y|$ via a pre-defined alignment, then we can compute a straightforward NLL and have no need for CTC's training or inference complexity, while now being able to generate a Forced Alignment as well as use interpretability methods. This is our **main contribution**.

To differentiate from CTC's multiple alignments, we will refer to our choice for the pre-defined alignment as the Optimal Alignment. Its properties should be intuitive:

Definition 1 (Optimal Alignment): The optimal alignment Z for an input sequence X is a strict one-to-one mapping ($|Z| = |X|$) where each label is present for the entire duration of its existence in X.

For the use case of speech recognition, we can make this guarantee by utilizing the phonetic transcriptions of the TIMIT dataset.

2.4.1. Why Phonemes?

In CTC-based ASR, it is more common to use a character-level model rather than a phoneme-level one, so it is worth justifying our choice. To begin, it has been shown that when using CTC, a character-level model will successfully learn the correct acoustics, but not spelling. Intuitively, this makes sense: given solely an audio input, how would the model be able to learn a high-level feature such as spelling rules in a language? So often, an external Language Model is utilized, but when done, does lead to a competitive Word Error Rate (WER). To illustrate this shortcoming, below is an example of a transcript generated by an acoustic model using CTC over characters:

> Guessed transcript: *the dus were sufeor to ecs ail*
> True transcript: *the dews were suffered to exhale*

Therefore, we will use phonemes. While this does introduce the extra step of moving from phonemes to words, we will show in Section 3.4 that this won't be necessary to do Forced Alignment (though it is worth mentioning that if we wished to do ASR, it would).

2.5. TIMIT

Generating the Optimal Alignment is only possible if we have the duration information of each phoneme in an audio sample. There exists a dataset that has precisely this information: TIMIT (Garofolo et al., 1988).

The DARPA TIMIT (Texas Instruments + MIT) Acoustic-Phonetic Continuous Speech Corpus is the de-facto speech dataset for tasks involving speech-phoneme relationships. The dataset contains 5.4 hours (6300 sentences) of spoken sentences. 630 speakers across 8 different dialects each recite 10 sentences. These sentences are split into 2 dialect sentences (SA), 450 phonetically compact sentences (SX), and 1890 phonetically diverse sentences (SI). By convention, we omit the SA sentences. The dataset provides a set-aside partition of the dataset as the test set. For each sample, 4 pieces of data are provided: a .WAV audio file, a .TXT sentence transcription, a .WRD time-annotated word transcription, and a .PHN tim-annotated phoneme transcription. There are 61 phonemes from its self-defined TIMITBET (similar to ARPABET), which are mapped to 39 phones at test time (Lee & Hon, 1989). A great summary of work on the TIMIT dataset can be found in (Lopes & Perdigao, 2011).

We can leverage the .PHN information to obtain the optimal alignment. The implementation details can be found in Section 3.1.2.

3. Method

3.1. Preprocessing

For each TIMIT sample, we generate input-output pairs for training and testing.

3.1.1. Input

While utilizing the raw waveform has had some success, we chose to preprocess the waveform by generating log-Mel Spectrograms, i.e. taking its Short Term Fourier Transform (STFT), projecting the result onto the Mel Scale, and taking the logarithm of its magnitude. The number of mel filterbanks is left as a hyperparamter. In addition, first and second derivatives along the frequency dimension are appended as well. Lastly, these features were normalized to have 0 mean and unit variance.

We chose to use 40 mel filterbanks, leaving the final shape of the input to the model as $(120, *)$.

3.1.2. Output

Diverging from the usual choice to use the phonetic transcript as-is, we modify it by encoding duration information, thus creating the optimal alignment. Naively, we could accomplish this by listing each phoneme for the durations provided in these .PHN files.

However, TIMIT time-annotates the phonemes w.r.t. the time dimension of the waveform (t), whereas our input spectrograms have a modified time dimension (τ). While the exact function for this mapping differs depending on the implementation parameters of the underlying Short-Term Fourier Transform (STFT), we will show our implementation and leave appropriate adjustments to the reader.

We assume that that the STFT is computed via padding the input to always ensure a centered window. Two hyperparameters are important: the hop length (HL) between each successive Fourier Transform, and the window length (WL) over which each Fourier Transform is computed over. We define a time mapping function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ as:

$$f(t) = \left\lfloor \max\left(0, \left\lfloor \frac{t - \frac{WL}{2}}{\text{HL}} \right\rfloor\right) \right\rfloor \tag{5}$$

The derivation of this function and additional commentary regarding its properties can be found in Appendix A. For the rest of this paper, we will refer to the pre-processed, duration encoding phonetic transcript that will be used as the output in the model as $Z = z_1 \dots z_n$.

Figure 2 illustrates the entire process for an example from the TIMIT training dataset.

3.2. Model

Given our desire for effiency and interpretability, in combination with the 2-dimensional nature of our audio features, we chose to utilize a Deep Convolutional Neural Network to model $P(Z|X)$. There has been several instances of success in using these models for ASR accuracy tasks (Collobert et al., 2016; Li et al., 2019), though we chose to use one heavily inspired by (Zhang et al., 2017).

The input is passed into 10 convolutional layers, the first 4 generating 128 activaton maps, while the last 6 generate 256. A kernel of size $(3, 5)$, longer along the time dimension, and PReLU activation with $\alpha = 0.1$ are used. To keep the size of the time dimension unchanged, we only pad along that dimension and allow the feature dimension to shrink with each layer. The first layer has pooling across the feature dimension of shape $(3, 1)$.

In order to preserve the time dimension and allow for a discriminative model that models the probaility at each timestep of the input, we *do not* completely flatten the final convolutional activation maps into a singular vector to feed into the fully-connected layers. Instead, we only flatten across the *channel* dimension, leaving us with a 2-dimensional $(channels * features, time)$ input to the FC layers. Here, we simply *duplicate* the DNN and apply it to each column separately, each generating a conditionally independent softmax distribution. Figure 3 illustrates this process.

The 3 fully connected layers all have 1024 neurons, with a final layer mapping the neurons to the number of labels, which in our case is 61 given the non-collapsed TIMIT phonemes. Finally, a log-softmax layer is added to generate a probability distribution for each time-step. Every layer in the network aside from the first and last make use of a 0.3 Dropout (Srivastava et al., 2014).

3.3. Training

The model was trained on a single NVIDIA RTX 2060 for 15 epochs, with a batch size of 3, the ADAM (Kingma & Ba, 2017) optimizer with a learning rate of 10^{-5}, and model weights initialized uniformly between $[-0.05, 0.05]$.

3.3.1. Loss Function

At each time step, a probability distribution exists over all the labels. Therefore, our chosen loss function was a slightly modified Negative Log Likelihood that averaged the NLLs across all timesteps.

$$\mathcal{L}(\hat{P}, Z) = \frac{-\sum_{t=1}^{n} \ln\left(\hat{P}_t(z_t|X;\theta)\right)}{n}, Z = z_1 \dots z_n \tag{6}$$

3.4. Inference

As is customary when using the TIMIT dataset, we first map all output phonemes to the 39 test phonemes before any inference is done.

With the output of the model $\hat{Y} \in \mathcal{X}^n$, which encodes phonetic as well as duration information, we can generate a transcript $\tilde{Y} \in \mathcal{X}^m$, reminiscent of the original output Y, by simply collapsing all repeats. If we were interested in the efficacy of our method on creating an ASR system, we could compute the Levenshtein (edit) distance between Y and $\tilde{Y}$ to get the Phoneme Error Rate.

3.4.1. Forced Alignment

The provided output of our model $\hat{Y}$ naturally provides a Forced Alignment of the phonemes. In practice though, an alignment of words to timestamps is much more useful. Since most datasets for supervised ASR include a word transcript, we already know where the spaces (which are equivalent to the times where words start/end) should exist. The challenge is figuring out how to infer the position of the spaces in our phonetic output $\tilde{Y}$.

One possibility is to map $\tilde{Y}$ to words, but we decided not to pursue this route due to the complexity it introduces. Phoneme-to-word models warrrant an entirely separate research question.

Instead, we decided to stay in the realm of phonemes. Our algorithm only requires a pronunciation dictionary; we used the CMU Pronunciation Dictionary. The main idea is to look on the edit path between Y and $\tilde{Y}$ and find which substring of $\tilde{Y}$ corresponds to when the $\langle$sp$\rangle$'s occur in Y. The details can be found in Algorithm 1. This is our **secondary contribution**.

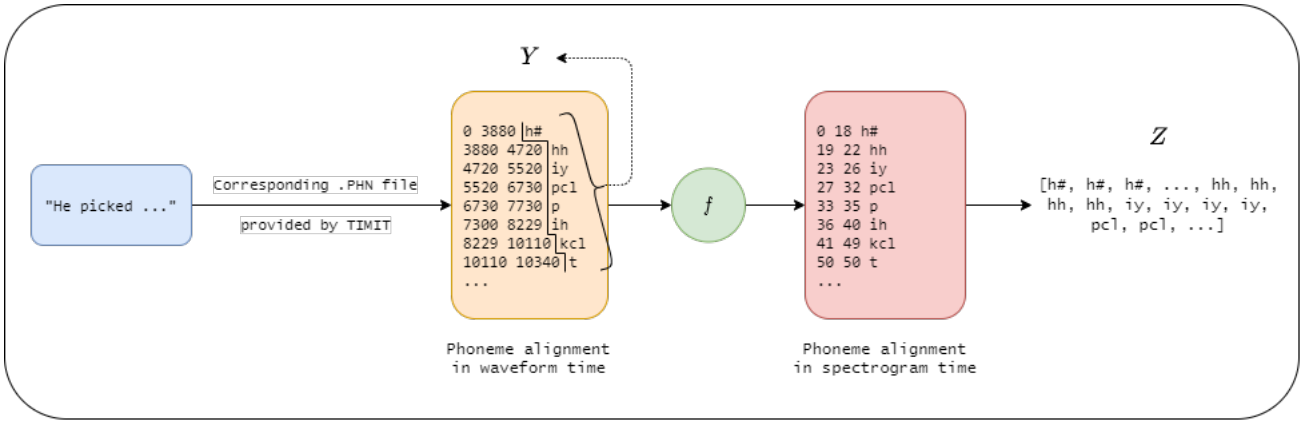

Figure 2. Preprocessing to generate duration-encoding output transcript. The example shown above is from TRAIN/DR5/FKKH0/SX390.

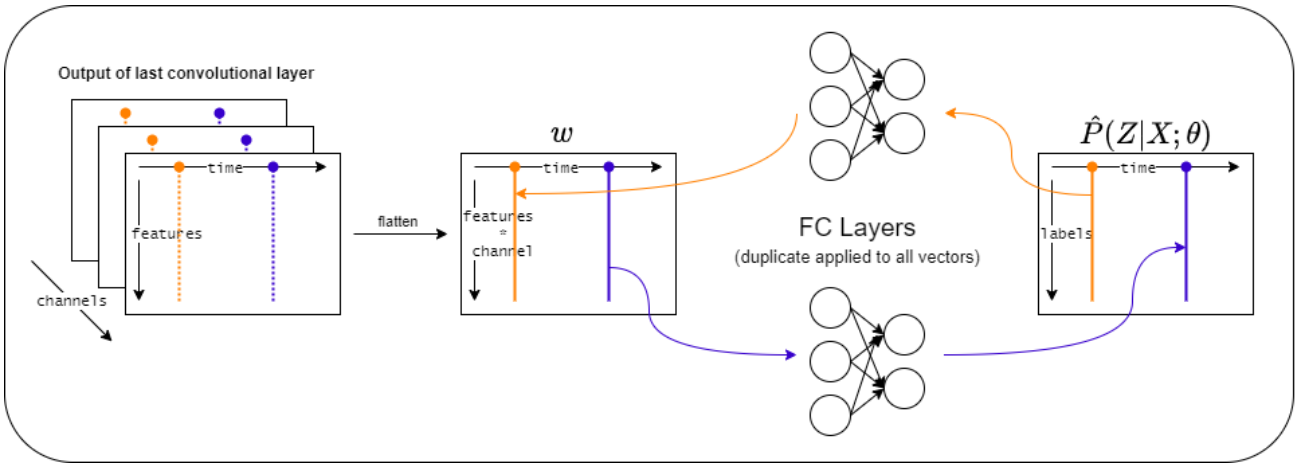

Figure 3. The latter half of the neural network pipeline. The blue path follows the network modelling $\hat{P}_t(z_t|X;\theta)$ by feeding the channel-flattened t^{th} timestep of the final convolutional activation map, the column vector $w_{[*,t]}$, into the fully-connected layers. This is repeated $\forall t \in [1, n]$.The orange path shows the gradient flow backwards which is helpful for GRAD-CAM.

Algorithm 1 Forced Alignment

Input: Ground truth Y, guess $\tilde{Y}$, pronunciation dictionary d
Output: Guess transcript s with $\langle$sp$\rangle$'s in estimated locations
$p := []$
$s := []$
for w = word in Y **do**
  Add $d(w)$ to p
  Add $\langle$sp$\rangle$ to p
end for
Note: DP graph has words starting at index 1
$e := \text{EditPath}(p, \tilde{Y})$
prev := None
for $(i, j) \in e$ **do**
  curr := $\tilde{Y}_{i-1}$
  if curr != prev **then**
    Add curr to s
    if $p_{j-1} == \langle$sp$\rangle$ **then**
      Add $\langle$sp$\rangle$ to s
    end if
  end if
  prev = curr
end for

With the location of where the word boundares exist wrt the phonemes, i.e. "after the third p", we can return to $\hat{Y}$ to get the spectrogram-time indices where the boundaries occur. Thus the process is complete without ever moving to word-space. An visualization of this algorithm can be found in Figure

3.5. Interpretability

Since our decoding scheme is a straightforward $\arg\max$, and this is the optimal method, we can now leverage standard interpretability methods as each timestep has a single, well-defined target class that we can take gradients with respect to.

Recall the details of our convolutional layers $\rightarrow$ FC layers: we do not flatten across the time dimenson, so each slice of the output $\hat{P}_t$ is computed independently of all other timesteps via the FC layers. Therefore, the gradient flows backwards until the tth slice of the output of the final convolutional layer. When using a method like GRAD-CAM, we only receive a 1-dimensional heatmap.

As an added bonus, we can utilize the word boundaries from our Forced Alignment to get activation maps for entire words. So we can normalize and concatenate the generated heatmaps to get the activation map for entire words, or even phrases, as well.

Ground Truth w/ <sp> inserted

$\tilde{Y}$		-	s	p	r	ih	ng	<sp>	s
	-	0	1	2	3	4	5	6	7
	sil	1	1	2	3	4	5	6	7
	s	2	1	2	3	4	5	6	6
	sil	3	2	2	3	4	5	6	7
	p	4	3	2	3	4	5	6	7
	r	5	4	3	2	3	4	5	6
	ih	6	5	4	3	2	3	4	5
	ng	7	6	5	4	3	2	3	4
<sp>	n	8	7	6	5	4	3	3	4
	s	9	8	7	6	5	4	4	3

Figure 4. The space-inserting algorithm based on the Levenshtein Distance matrix. The edit path is highlighted in green while the suggested index of the $\langle$sp$\rangle$ is in red.

4. Results

4.1. Alignment Error (AE)

To test the efficacy of our model to create a Forced Alignment, we define the Alignment Error (AE) to be the average difference between the computed word boundaries and the ground truth.

$$AE = \frac{\sum_i |\text{guess}_{i,\text{end}} - \text{truth}_{i,\text{end}}|}{\#\text{ of words}} \tag{7}$$

Our model achieved an average AE of 67 ms. This is the first result for pure CNN models to our knowledge, though a discussion of the great performance of HMM-based models can be found in (Hosom, 2009).

4.2. Accuracy

To be thorough, we tested the accuracy of the model. It achieved a Phoneme Error Rate of 22.8% on TIMIT test set. This could probably be improved upon using maxout networks or architecture changes, but we leave that to future work.

5. Conclusion

In this paper, we investigated whether purely Convolutional Neural Networks could generate Forced Alignents of speech to text. By leveraging the phonetic information provided in the TIMIT dataset, we were able to create an Optimal

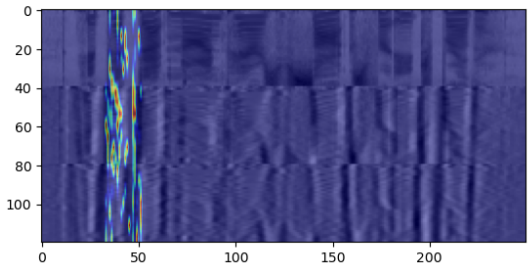

Figure 5. The Class Activation Map using GRAD-CAM for the word 'picked' for the example TRAIN/DR5/FKKH0/SX390 in TIMIT.

Alignment that removed the need to use CTC, enabling not only Forced Alignment, but GRAD-CAM as well. On top of a phoneme alignment, we also introduced a Levenshtein-distance based algorithm for learning word boundaries without ever moving into word space. On the entire TIMIT dataset, our model achieved an average error of 67 milliseconds compared to the true word boundaries, while also achieving a 22.8% error on phoneme accuracy given our severly lacking compute for such a deep model.

6. Future Work

While the focus of this work was on possibility, the major improvement from here would be on optimizing for better error rates. It is somewhat shortsighted to restrict ourselves to purely convolutional models, as recurrent models intuitively make more sense for sequential information at a higher cost. For accuracy tasks, CNN's in combination with recurrent modules such as RNNs/GRUs/LSTMs (Sainath et al., 2015; Amodei et al., 2015) and residual connections (Zhang et al., 2016; Wang et al., 2017) show improved performance. Even just a pure BLSTM model shows fantastic accuracy (Fernández et al., 2008).

In addition, given the depth of our model, it would be interesting to see whether having access to more compute would lead to a better result as we could search over a greater range of hyperparameters.

References

Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J., and Zhu, Z. Deep speech 2: End-to-end speech recognition in english and mandarin, 2015.

Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. Listen, attend and spell, 2015.

Collobert, R., Puhrsch, C., and Synnaeve, G. Wav2letter: an end-to-end convnet-based speech recognition system, 2016.

Fernández, S., Graves, A., and Schmidhuber, J. Phoneme recognition in timit with blstm-ctc, 2008.

Garofolo, J. et al. Darpa timit acoustic-phonetic continuous speech database. *National Institute of Standards and Technology (NIST)*, 1988.

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

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. Conformer: Convolution-augmented transformer for speech recognition, 2020.

Hannun, A. Sequence modeling with ctc. *Distill*, 2017. doi: 10.23915/distill.00008. https://distill.pub/2017/ctc.

Hosom, J.-P. Speaker-independent phoneme alignment using transition-dependent states. *Speech Communication*, 51(4):352–368, 2009. ISSN 0167-6393. doi: https://doi.org/10.1016/j.specom.2008.11.003. URL https://www.sciencedirect.com/science/article/pii/S0167639308001775.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017.

Lee, K.-F. and Hon, H.-W. Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37 (11):1641–1648, 1989.

Li, J., Lavrukhin, V., Ginsburg, B., Leary, R., Kuchaiev, O., Cohen, J. M., Nguyen, H., and Gadde, R. T. Jasper: An end-to-end convolutional neural acoustic model, 2019.

Lopes, C. and Perdigao, F. Phoneme recognition on the timit database. In Ipsic, I. (ed.), *Speech Technologies*, chapter 14. IntechOpen, Rijeka, 2011. doi: 10.5772/17600. URL https://doi.org/10.5772/17600.

Sainath, T., Vinyals, O., Senior, A., and Sak, H. Convolutional, long short-term memory, fully connected deep neural networks. In *ICASSP*, 2015.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2): 336–359, Oct 2019. ISSN 1573-1405. doi: 10.1007/s11263-019-01228-7. URL http://dx.doi.org/10.1007/s11263-019-01228-7.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. In *The Journal of Machine Learning Research, vol. 15, no. 1*, pp. 1929–1958, 2014.

Wang, Y., Deng, X., Pu, S., and Huang, Z. Residual convolutional ctc networks for automatic speech recognition, 2017.

Zhang, Y., Chan, W., and Jaitly, N. Very deep convolutional networks for end-to-end speech recognition, 2016.

Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Bengio, C. L. Y., and Courville, A. Towards end-to-end speech recognition with deep convolutional neural networks, 2017.

Zhou, B., Khosla, A., A., L., Oliva, A., and Torralba, A. Learning Deep Features for Discriminative Localization. *CVPR*, 2016.

A. Appendix

Intuitively, we are looking for which "hops" of the Fourier Transform cover a sample in the waveform. This is equivalent to its corresponding waveform times. As is illustrated in Figure 6, from the perspective of the spectrogram time τ, the range of covered t for a τ can be defined as:

$$t \in \left[\tau * \text{HL} - \frac{WL}{2}, \tau * \text{HL} + \frac{WL}{2}\right]$$

However, there is no guarantee that the mapping of time-to-spectrogram is one-to-one. With our parameters, a 400 sample Window Length and a 200 sample Hop Length, each sample of the waveform will be covered by exactly 2 slices of the spectrogram. We heuristically choose to utilize the

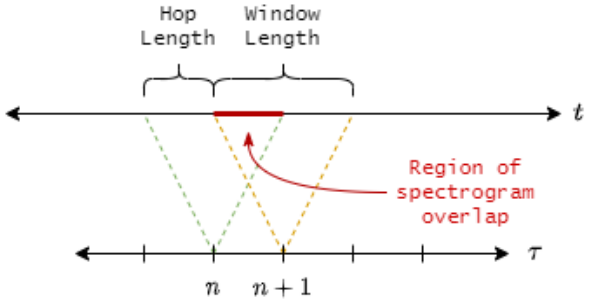

Figure 6. An illustration of how f works. Although overlapping regions exist, we choose the first satisfying τ.

first such τ. This corresponds to the first integer satisfying:

$$t = \tau * HL + \frac{WL}{2}$$
$$\tau = \frac{t - \frac{WL}{2}}{HL}$$

Accounting for the implicit initial padding of the Fourier Transform and non-negative spectrogram times, we arrive at our final expression:

$$f(t) = \max\left(0, \left\lfloor \frac{t - \frac{WL}{2}}{\text{HL}} \right\rfloor\right)$$

A natural question regarding this function is whether the choice to use the first satisfying τ will cause overlap problems. Specifically, is it possible for us to omit a unique label using our method? To answer this, we can look at the underlying parameters of the STFT.

We utilized a 400-point DFT and window length, and a hop length equal to half the window length, all defaults in Pytorch, in addition to a standard 16.5 kHz sampling rate. Therefore, each time-slice of the resultant spectrogram covers $\frac{400}{16500} \approx 0.024$ seconds, or 24 ms. For our method to "miss" a phoneme, it would need to be spoken for less than 24 ms before the next one, which we believe does not happen often in practice. Our results support this belief.