Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

A transfer learning approach for pronunciation scoring

Tesis de Licenciatura en Ciencias de la Computación

Marcelo Agustín Sancinetti

Directora: Dra. Luciana Ferrer

Codirectora: Lic. Jazmín Vidal Domínguez

Buenos Aires, 2022

A TRANSFER LEARNING APPROACH FOR PRONUNCIATION SCORING

Evaluating a student's pronunciation is a key aspect of language education. This is commonly done by human teachers, but can also be done by computers, in which case it is called computer aided pronunciation training (CAPT). CAPT tools have become increasingly popular in recent years, allowing students to improve their pronunciation through constant feedback. Automatic pronunciation assessment can be done at different levels, such as phrase, word or phone-level; the last one being the most granular and useful. At that level, automatic pronunciation is a challenging task, with performance far from that of human annotators. Standard systems generate a score for each phone in a phrase using models trained for automatic speech recognition (ASR), a different but related task, using recordings of native speakers only. However, better performance has been shown when using systems that are trained specifically for the task of interest using non-native data labelled at phone level as correctly or incorrectly pronounced. Yet, such systems face the challenge that datasets labelled for this task are scarce and usually small.

In this work, we present a transfer learning-based approach that leverages a neural network model trained for ASR on native data, adapting it for the task of English phone-level pronunciation scoring by replacing the model's output layer and re-training it with non-native data. We train the model on EpaDB, a database of English phrases read by Argentinians designed specifically for pronunciation scoring research. We analyze the effect of several design choices and compare the performance with a state-of-the-art goodness of pronunciation (GOP) system. Our final system is 20% better than the GOP system on EpaDB, for a cost function that prioritizes low rates of unnecessary corrections.

Keywords: Computer Assisted Language Learning, Phone level Pronunciation Scoring, Goodness of Pronunciation, Transfer Learning.

AGRADECIMIENTOS

CONTENTS

1. INTRODUCTION

1.1 Problem description

In recent years, globalization has made language learning more important than ever. Because of this, as well as the advancements in information technology, computer assisted language learning (CALL) has started to become more widespread. CALL systems allow students to practice and progress in their efforts to acquire a second language without requiring a human teacher guiding them and providing them with continuous personalized feedback [1]. Computer aided pronunciation training (CAPT) systems are a family of CALL systems that are used to improve pronunciation, which is otherwise particularly difficult to practice without a personal tutor. These systems assess pronunciation quality for a whole phrase, each word or each phone in an utterance. Phone-level pronunciation scoring is the most granular and challenging task, since the amount of data used to judge each segment is smaller. However, these systems may be the best for beginners, as they provide students with more detailed information and can point to phone-specific mistakes[2], whereas phrase-level pronunciation assessment only sees the "big picture".

CAPT systems can be classified into two groups. Systems in the first group are trained with non-native speech labelled as correctly or incorrectly pronounced [3][4]. Systems in the second group, on the other hand, rely on models trained only with correctly pronounced speech (usually from native speakers) and measure pronunciation quality based on the acoustic similarity between the student's speech and the speech in the training data [5][6][7]. Systems that use annotated non-native data have been found to be more accurate [2]. However, non-native speech datasets annotated at phone level are usually scarce or even non-existent for many L1-languages (target students' native tongue), making the task difficult. It should also be noted that these systems are generally L1-dependent, meaning they only work

for the intended L1s. For example, a system designed to score English pronunciation trained only on Argentinian speakers should not be expected to work well for Chinese speakers, whereas systems in the second group can be used regardless of the L1.

The aim of this work is to propose and analyze a neural network-based phone-level English pronunciation scoring system designed to be used by students in Argentinian schools. It will be trained using EpaDB [8], a database of Argentinian speakers annotated at a detailed phonetic level. The system proposed in this work will belong to the first of the two groups mentioned in the previous paragraph.

1.2 Previous work

In this section, we will review several existing approaches for pronunciation scoring. First, we will introduce a traditional, non-DNN-based system. Then, we will move on to more modern DNN-based methods.

1.2.1 Original Goodness of Pronunciation method

One of the earliest phone-level pronunciation scoring systems is called Goodness of Pronunciation or GOP and relies on Automatic Speech Recognition (ASR) models [7], which solve the task of automatically transcribing speech. When the GOP method was first proposed, ASR models relied on *Hidden Markov Models* [9] (HMM) trained for each phone that used *Gaussian Mixture Models* [10] (GMM) as probability density functions for the emission of acoustic events. We explain GMM-HMM acoustic modeling in sections 2.2.1 and 2.2.2.

The GOP measure estimates the pronunciation quality of a given phone instance p as the logarithm of the posterior probability $P(p|O^{(p)})$, that is, the probability that the speaker uttered the phone p given the observation of the acoustic segment $O^{(p)}$, normalized by the duration:

$$GOP(p) \equiv \frac{\left|\log\left(P(p|O^{(p)})\right)\right|}{NF(p)} \tag{1.1}$$

By applying Bayes' theorem, the score can be estimated using the likelihoods $p(O^{(p)}|p)$, which can be obtained from the previously mentioned GMM-HMM model. The GOP formula then becomes:

$$GOP(p) \equiv \left| \log \left(\frac{p(O^{(p)}|p)P(p)}{\sum_{q \in Q} p(O^{(p)}|q)P(q)} \right) \right| \Big/ NF(p) \tag{1.2}$$

where Q is the set of all phones and NF(p) is the number of frames spanned by p. The likelihoods $p(O^{(p)}|p)$ in the numerator are determined by aligning a known orthographic transcription of the utterance to the recording using the Viterbi algorithm using the GMM-HMM model. The likelihoods $p(O^{(p)}|q)$ in the denominator are calculated with the GMM-HMM model as well, but unconstrained to any transcription (see section 2.2.4).

We can think of GOP scores as a measure of how confident an ASR model is in its prediction of a phone. The closer a speaker's pronunciation is to that of a native speaker, the higher the likelihood of the target phones should be, so the GOP score should be higher (The GOP score is a log-likelihood, so it will be zero if the likelihood is one, and negative if the likelihood is lower than one).

Formula 1.2 can be simplified in a few ways, such as approximating the sum in the denominator by its maximum, which can be more efficient to calculate. It can also be assumed that all phones in the phone set have the same prior probability so that they cancel out. With these modifications, we arrive at the following, simpler formula:

$$GOP(p) \equiv \left| \log \left(\frac{p(O^{(p)}|p)}{\max_{q \in Q} p(O^{(p)}|q)} \right) \right| \Big/ NF(p) \tag{1.3}$$

It should be noted that this method does not require annotated non-native speech to be trained, as it relies only an ASR model which is trained on native data.

1.2.2 Deep Neural Network-based methods

In the past few years, deep neural networks have become one of the most widely used artificial intelligence models, showing great advancements in many different fields, including Automatic Speech Recognition [11]. Since pronunciation scoring

systems often rely on ASR technology, these advancements triggered new work on applying deep neural networks for this task, obtaining improvements over traditional methods, both for systems trained with non-native data as well as those that only use native data. Such is the case of the GOP method, which was reformulated by replacing the GMM-HMM based acoustic models and aligners with DNN-HMM alternatives [12].

The original GOP method came down to calculating the posteriors $P(p|O^{(p)})$. In a GMM-HMM setup, these posterior probabilities were estimated using the likelihoods $p(O^{(p)}|p)$, as mentioned previously in section 1.2.1. In the DNN variant of the GOP method this is no longer needed as the posterior probabilities can be obtained from the DNN acoustic model directly. In DNN acoustic modeling, DNNs take a few frames of acoustic input (a sequence of feature vectors extracted from the audio) and output the probability of the given acoustic features being emitted by each HMM state. In other words, they are trained to predict HMM states given an acoustic input (we explain this in more detail in section 2.2). Since each HMM state corresponds to a single phone, a DNN acoustic model's output can be used to obtain $P(p|O^{(p)})$, where $O^{(p)}$ is the input to the DNN. This way, it is no longer necessary to apply Bayes' theorem as in equation 1.2. This method was proposed in [12] and will be used as baseline for this work. It will be described in detail in section 2.3. There have since been other DNN-GOP variants, such as [13][14].

As mentioned, systems trained with L1-specific non-native data often perform better, but face the challenge of data scarcity. For this reason, some works on pronunciation scoring have opted for transfer learning approaches. In transfer learning, models trained for a certain source task are re-purposed to solve the target task. If models that are already good at the source task have learnt something that is also relevant to the target task, it is possible to achieve better performance with the same amount of data than by training a completely new model for the target task. [15]. This is very popular in data scarcity scenarios across many different domains, and there are successful examples of this in pronunciation scoring. [16] proposed an approach where the output layer of the ASR DNN was replaced with a new

layer trained to detect incorrectly pronounced phones. Similarly, [17] started from a CNN trained with a large dataset of images and trained it to classify mispronounced phones from spectrograms. As will be discussed later, the method proposed in this work is strongly based on [16], relying on a similar transfer-learning strategy.

2. METHODS

In this chapter, we will describe the DNN-GOP system used as baseline, as well as the newer transfer-learning approach explored in this work. The features used to represent speech and the metrics used to evaluate the system's performance will also be explained.

2.1 Features

In machine learning, features are attributes, most often numeric, that describe the data used in a system. For any machine learning task, it is always important to choose good features, that is, features that describe the data in a way that is relevant to the problem. In this case, features will be extracted from the waveforms used to train and test the system before being passed as input for the neural networks. The features used in this work are *Mel Frequency Cepstral Coefficients* and *iVectors*.

2.1.1 Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCCs) are features that are used for many speech processing tasks. They are designed to represent audio in a way that relates to human perception. They are frame-level features, meaning they are extracted from every frame, which are small chunks of the audio signal, usually around 25 ms long. By computing these features for every frame, we can describe the perceptually-relevant acoustic properties of the signal at every point in time.

The computation of MFCCs consists of the following steps:

1. Pre-emphasis: In human speech recordings, low frequencies usually have a higher amplitude than high frequencies, meaning that high frequencies become attenuated, even though they are an important part of speech. To remedy this, a high-pass filter is applied to the signal, which amplifies high frequencies.

2. Windowing: As mentioned, MFCCs are frame-level features. Hence, the signal must be divided into small frames. Frames should be small enough so that the signal does not change too much within that time window, but long enough to provide enough samples to accurately describe the frequencies present in that window. The standard frame size is 25 ms, and the shift between the start of one frame and the next is 10 ms. This produces an overlap of 15 ms between frames, so that the information between two adjacent frames is captured.

3. Spectral analysis: Using the Fast Fourier Transform (FFT), the signal is converted from the time domain to the spectral domain. This means that instead of measuring the amplitude of the signal for each moment in time, spectral analysis describes the power of the frequencies present in the signal during the whole frame.

4. Mel scale: Humans do not perceive all frequencies equally. Small changes in pitch are distinguished more easily in low frequencies than in high frequencies. The mel frequency scale is a transformation that can be applied to a signal in order to represent this phenomenon. It is linear up to 1000 Hz and logarithmic for higher frequencies, which roughly approximates human perception of pitch. A signal can be converted to the mel scale by applying a series of 40 triangular filters to the spectral representation computed in the previous step where the filters are located with centers defined by the mel scale . These filters produce values called filterbank energies that represent the loudness of frequencies along different ranges of the linear scale. Finally, the natural logarithm of each filterbank energy is computed to account for the fact that the human hearing system perceives small fluctuations in loudness more easily in quieter than in louder signals (just like pitch differences). As mentioned, 40 filters are used in this work, but different amounts may be used for different tasks.

5. Cepstral Coefficients: The Discrete Cosine Transform (DCT) is applied to the filterbank energies, resulting in the final cepstral coefficients. These are the final MFCCs.

2.1.2 iVectors

Identity vectors or iVectors, for short, are acoustic features first introduced in [18] for the task of speaker verification, i.e, verifying a speakers identity. However, they are also useful for speech recognition and pronunciation scoring, since they provide a big picture of the signal's speech content. Unlike MFCCs, they are not calculated for each frame in an utterance, but for whole audio signals or long portions of them, such as ten seconds. iVectors are calculated by adapting the means of a *Universal Background Model* (UBM), which is a global GMM trained with a large number of speakers. The UBM is trained to model a set of features extracted from the speakers' utterances, in this case, MFCCs. By adapting the means of the UBM based on a new utterance's MFCCs, it is possible to find such a GMM representation for any new speaker based on the features of the speakers used to train the UBM. However, the full GMM representation of a speaker has a vector of means that is very large, usually in the order of thousands. Thus, iVectors are projections of these mean vectors onto a lower-dimensional space, in this case, a subspace of 100 dimensions. The in-depth mathematics of how iVectors are calculated are too complex and out of scope to fully describe in this work. For more detail, see [18].

2.2 Automatic Speech Recognition fundamentals

The GOP method, both in its original GMM-HMM version as well as the more modern DNN-based variant used as baseline for this work, relies on ASR models to compute the necessary probabilities and forced alignments. In this section, we will introduce some elements of traditional ASR systems that are important to fully understand the GOP method in both of its variants.

2.2.1 Hidden Markov Models

In order to understand how GMM-HMM acoustic models work, we need a brief introduction to HMMs. Hidden Markov Models are statistical models consisting of

"hidden" states, transitions between these states, and emissions, which are observable features. In each state, there is a probability distribution for the emissions, as well as a certain probability of transitioning to any other state or staying in the current state. The emissions can be observed directly, but the internal state sequence that produced a certain sequence of emissions cannot, so it must be inferred based on the observations, hence the name "hidden" state. Transition probabilities depend only on the current state, so there is no memory of the previous state sequence. There is also a starting state. In general, when the observations are continuous in nature, Gaussian Mixture Models are used as probability density functions.

HMMs are useful at representing certain phenomena where there is an unknown sequence that must be inferred by observing a sequence of events that relates to the unknown sequence in some way. For more detail on the topic of Hidden Markov Models, such as how they are trained, see [9].

In the context of ASR, HMMs are used to model phone (or word) sequences, which are inferred based on a sequence of acoustic observations (a set of features extracted from a speech signal). In modern ASR systems, HMM states correspond to subphonetic units, called senones [19]. The motivation for this subphonetic modeling stems from the fact that the acoustic features of phones behave differently depending on the context, that is, which phones come before and after them. Each possible phone conditioned to a certain context is called a triphone. Instead of having a single state per phone, HMMs can have three states per triphone: one for the start, middle, and end. With this level of detail, acoustic emissions for each HMM state can be modelled more accurately than using only a single state per phone. However, this also causes the HMM to have a huge amount of states, which makes them impractical. This is addressed by clustering states that have similar emission probability distributions into one, which is possible because some triphones are similar. Decision trees such as the one shown in figure 2.1 are used to decide how triphones can be clustered based on phonetic properties. This cluster of hidden states is what is finally called a senone, and the final HMM has a single state for each senone. Figure 2.2 helps visualize the process of creating senone HMMs.

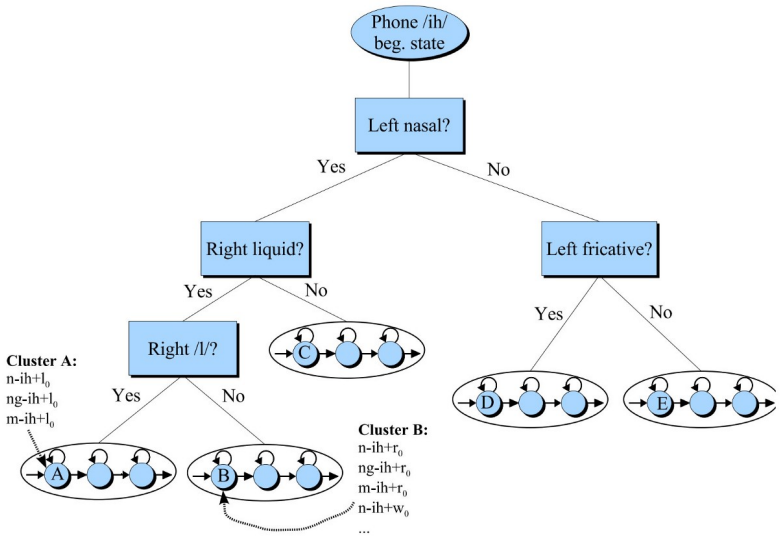

Fig. 2.1: A decision tree to find possible clusters of triphones for phone /ih/ based on the phonetic properties of the phones that appear to the left and right of a given instance of phone /ih/ in a word. The flow of these trees start at the top, in the beginning state. From there, a series of questions about the adjacent phones are made, leading to the leaves of the tree, where a certain cluster is assigned. In this case, only five senones are built, greatly reducing the amount of states in the HMM. For example, using this phonetic tree, the phone /ih/ in the word "miss" would be assigned cluster E, since the phone /m/, to the left of phone /ih/ is nasal and not fricative. Taken from Jonathan Hui's blog on ASR.

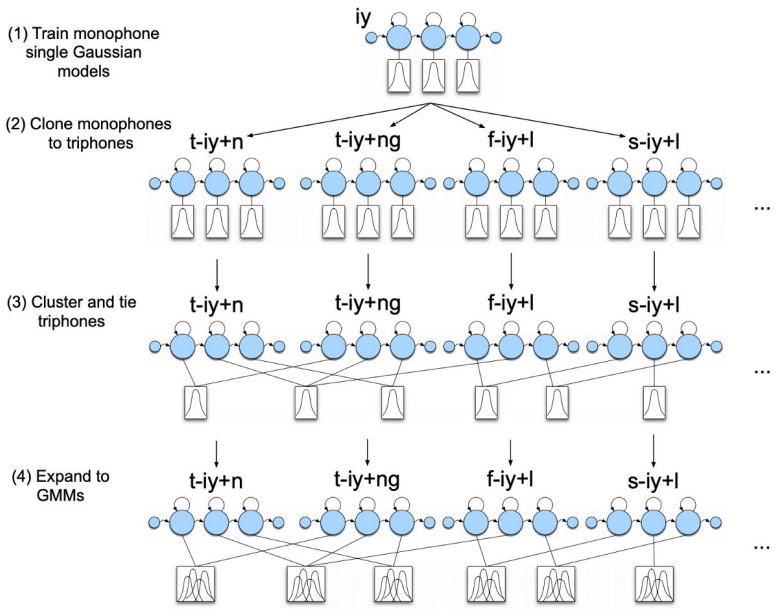

Fig. 2.2: An example of the steps involved in expanding phones to triphones and clustering triphones to senones for phone /ih/. First, a single 3-state HMM is trained, using a single gaussian for the emissions in each state. This monophone model is cloned for each possible triphone, which are then clustered based on a phonetic decision tree such as the one shown in figure 2.1. Finally, the emissions for each senone are modeled with a new GMM trained using the single gaussians from each triphone belonging to that senone. Taken from Jonathan Hui's blog on ASR.

2.2.2 GMM-HMM-based acoustic model

Traditionally, GMMs were used as the probability density functions for the emissions in the HMMs. In this case, GMMs are trained to model MFCCs of each triphone (which are then clustered into senones as explained in the previous section). Formally, the GMMs for the HMM state corresponding to senone s allow us to calculate the likelihoods $P(O|s)$, where O is an acoustic input. These are the likelihoods that are used in the GOP method, as we will explain in section 2.3.

2.2.3 DNN-HMM-based acoustic model

Nowadays, it is common to use DNNs instead of GMMs to model the emissions of HMMs [11]. We saw in section 2.2.2 that the emission probabilities of an HMM-based acoustic model are $P(O|s)$, where O is an acoustic input and s is a senone. Hence, in order to build a DNN-HMM-based acoustic model, we need to calculate these likelihoods using DNNs. For this purpose, DNN-based acoustic models are trained to classify senones given an acoustic input. This is equivalent to determining which HMM state (corresponding to a single senone of a previously trained acoustic model) could most probably have emitted a set of acoustic features. Formally, the output of the DNN at frame t can be expressed as $P_t(s|O)$, where s is a senone and O is the full sequence of acoustic features. These are posterior probabilities, not the likelihoods that we need for the emissions of our HMM. However, since the probabilities $P(s)$ are known from the data used to train the DNN, a scaled version of the likelihood $p(O|s)$ can be calculated by dividing $P(s|O)$ by $P(s)$. This is because of Bayes' theorem:

$$p(O|s) = \frac{P(s|O)p(O)}{P(s)} \tag{2.1}$$

$$p(O|s) \propto \frac{P(s|O)}{P(s)} \tag{2.2}$$

The resulting likelihood is scaled by the unknown factor $p(O)$, but this has shown not to seriously affect the performance of ASR systems [11]. These scaled likelihoods can be used for the emission probabilities of the HMM used in ASR decoding and to compute forced alignments, which will be explained in the next section.

2.2.4 ASR decoding and forced alignments

Now that we have some insight into how both types of acoustic models work, we can understand how ASR systems decode sequences of acoustic features into words. The task of an ASR system is to find the sequence of words that was most probably uttered in an audio recording. So, given a sequence of acoustic featuers X, the task is to find the word sequence W that maximizes $P(W|X)$. Acoustic models can

be used to find out which phones were uttered in a given speech segment. This, however, is not enough for ASR decoding, as we want a word-level sequence and not a phone-level sequence. Additional steps need to be taken to determine which words were uttered based on the phone sequence. This is not trivial, since there are cases where two different words or word sequences can be pronounced the same way. For example, *flour* and *flower*. For this reason, there are also two additional components to ASR systems: a language model and a lexicon. The language model is trained on text data and describes the probability of finding a given word in a sentence based on a certain amount of words that came before it (for example, a well trained language model would output a high probability for the phrase *Bread is made with flour* compared to *Bread is made with flower*. The lexicon contains all the possible phonetic transcriptions of each word, so as to know which words a given phone sequence could represent. When decoding an acoustic sequence, the language model and lexicon are used together with the acoustic model to find the most likely word sequence. This is done using the Viterbi algorithm [9]. Figure 2.3 shows the composition between a language model for words one, two, and zero and the respective HMMs for these words. HMMs for each word are built by concatenating the HMMs for each phone according to the possible pronunciations of each word, which are given by the lexicon.

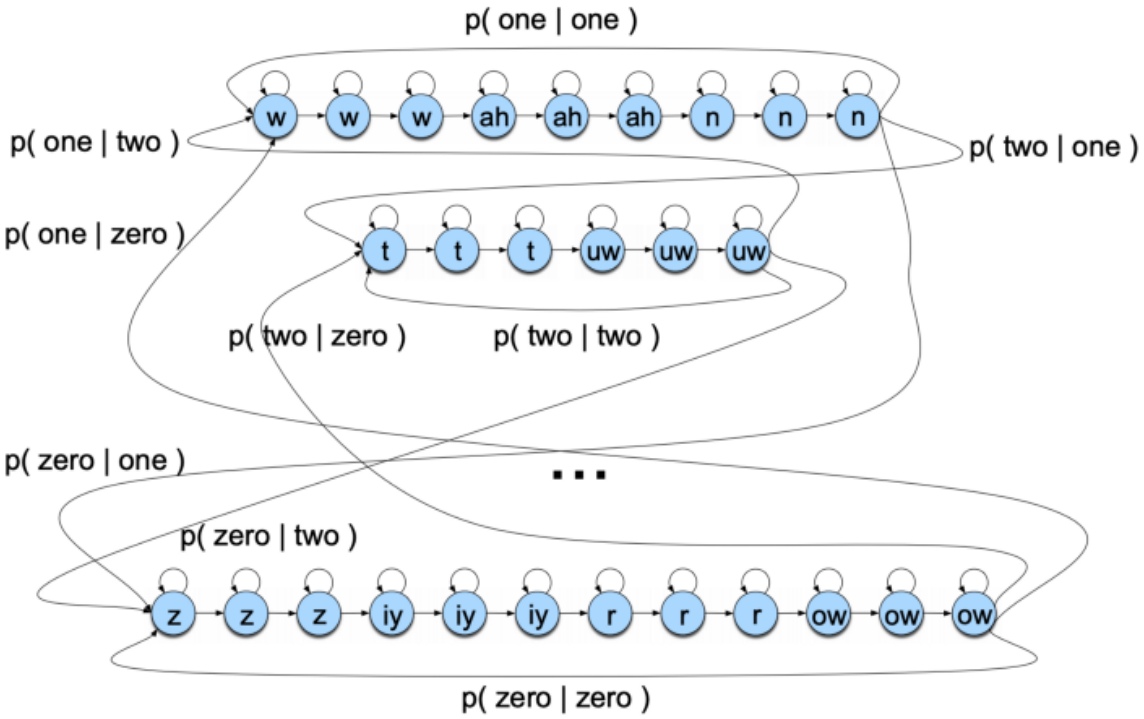

Fig. 2.3: An Example of a language model for words one, two, and zero as a bi-gram (only the previous word is considered to determine the probability of the next word) composed with the HMM for each word. Emission probabilities for each state are omitted. Taken from Jonathan Hui's blog on ASR

If the orthographic transcription of an utterance is known, this same method can be applied to align that transcription to a recording by constraining the language model to that word sequence. This is done in the original GOP method both to calculate the necessary likelihoods and phone boundaries.

2.3 Baseline: DNN-based GOP system

The baseline for this work is the DNN-GOP method proposed in [12]. GOP uses estimates of posterior probabilities of the target phone as scores. Recalling section 1.2.2, the DNN version of this method obtains these posterior probabilities from an acoustic model such as the one detailed in section 2.2.3, which is trained only

on native data and predicts senones given an acoustic input. The formula for the DNN-GOP score of a phone p starting at frame T, with a duration of D frames is:

$$GOP(p) = -\frac{1}{D} \sum_{t=T}^{T+D-1} \log P_t(p|O) \tag{2.3}$$

where O is the full sequence of features for the waveform and $P_t(p_t|O)$ is the DNN's estimate of the posterior probability for phone p at frame t. The start and end frames for each target phone are obtained using a forced-aligner given the word transcription. Note that the DNN acoustic model provides likelihoods for each senone, $P(s|O)$, and not for each phone. To obtain $P(p|O)$, we average all the likelihoods for senones corresponding to phone p.

2.4 Proposed method

As mentioned previously, CAPT systems based solely on measuring the speaker's similarity to native speech can fall short. This is because these systems never see examples of what incorrectly pronounced speech looks like as part of their training. Again, the inherent assumption is that correctly pronounced speech will be more similar to the samples seen during training than incorrectly pronounced speech. This is a fairly simplistic assumption that ignores the boundary between correct and incorrect pronunciations. Furthermore, using the confidence of an ASR model as a pronunciation assessment is specially problematic. Even though ASR is a task that is related to pronunciation scoring, it is not the same task. The goal of an ASR system is to recognize speech as well as possible, so a good ASR system should in fact be able to accept a wide variety of pronunciations, which may give it a high confidence even with somewhat incorrectly pronounced speech, defying the aforementioned assumption. On the other hand, systems trained with labelled samples of both correctly and incorrectly pronounced phones have a clear advantage in terms of the available data, because it stands to reason that a model trained for a specific task with specially designed data should perform better than applying a model trained for a similar but different task. The method proposed in this

section uses annotated non-native data, which is why it is expected to outperform the baseline system.

The DNN-GOP system described in the previous section uses an ASR model that classifies senones. The posterior probabilities obtained from this model are then used to compute the GOP score as described in equation 2.3. However, it is possible to fine-tune said model to output pronunciation scores directly. To achieve this, the DNN ASR acoustic model's output layer, which has one output node per senone, will be replaced by a new layer with one output node per phone in the English language. This layer is composed of an affine transformation followed by sigmoid activation functions. Sigmoid activations are used because we want to make a binary decision: correctly or incorrectly pronounced phone. Hence, we do not want a probability distribution over all of the phones, but over the correctly/incorrectly pronounced class for each phone individually. The original acoustic model used softmax activations because its goal was to generate a probability distribution for all the senones. For each frame in the utterance, the new model will output a score for each phone. Figure 2.4 shows a schematic of the new method alongside the baseline method.

Forced alignments are still necessary to determine which phone the speaker should have pronounced in each frame. This is called the target phone. Recall that in the GOP method, the DNN outputs posteriors for all 6024 senones, but only those corresponding to each frame's target phone are taken into account (i.e. averaged). In the same way, the new model will output 39 scores (one for each phone), but only the score from the node corresponding to each frame's target phone is relevant (see the *phone selection* step in figure 2.4). Thus, during training, only one node's score will be used to calculate the loss function for each frame. During inference, only the target phone's scored will be considered to evaluate pronunciation. The alignments will be calculated in the same way as in the baseline system (see section 2.2.4). To obtain a single, final score for a phone in the utterance, the relevant scores for each frame during which the phone was uttered are averaged.

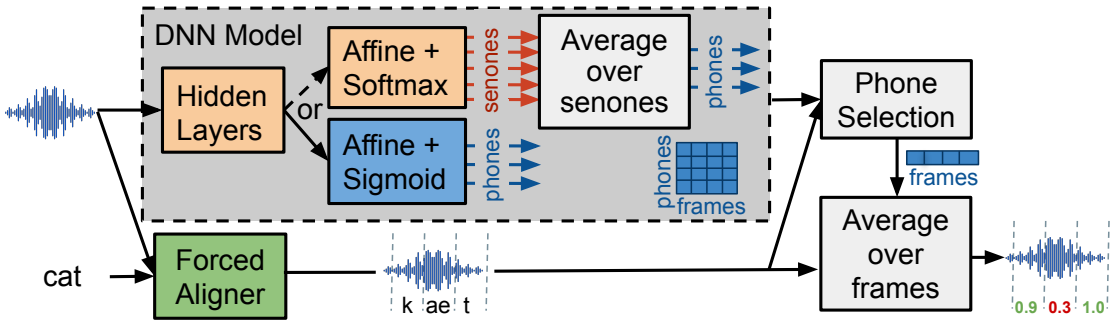

Fig. 2.4: Schematic for the GOP and proposed models. A waveform and its transcription are fed to a forced aligner to obtain the start and end times of each target phone. A DNN model is then used to generate frame-level scores for each phone in the language. For the baseline GOP system (top branch in grey block), this block includes the step where senone posteriors are summed over all senones corresponding to each phone to get scores per phone for each frame. In the case, of the proposed model (bottom branch in grey block), the DNN directly produces scores (probabilities of correct pronunciation) per phone. In the next step, the score corresponding to the phone found by the forced-aligner at each frame is selected, resulting in one score for each frame in the signal. Finally, pronunciation scores are computed by averaging the frame-level scores over all the frames for each phone in the alignments.

The loss function used for fine-tuning the model's parameters, including the replaced output layer, is given by:

$$L = -\sum_{p \in P} \sum_{y \in Y} w_{py} \sum_{t \in T_{py}} y_t \log \hat{y}_t + (1 - y_t) \log(1 - \hat{y}_t) \tag{2.4}$$

where the first sum goes over P, the set of all phones in the model; the second sum goes over the two classes, $Y = \{0, 1\}$, incorrectly and correctly pronounced, respectively; the third sum goes over $T_{yp} = \{t | p_t = p \wedge y_t = y\}$, all the frames in the waveform for which the target phone found by the forced aligner for time t, p_t, is p and the pronunciation label for that frame (inherited from the label for p_t), y_t, is y; and $\hat{y}_t = P_t(p_t|O)$ is the posterior generated by the DNN for frame t and phone

p given the sequence of observations O.

The weights w_{py} are used to adjust the influence of the samples from each phone and class. Two approaches were evaluated in this regard: **flat weights** and **balanced weights**. With **flat weights**, all w_{py} are set to 1, resulting in all frames having the same influence on the loss. In this case, this is just the standard binary cross-entropy loss function. With **balanced weights**, on the other hand, $w_{py} = 1/N_{py}$, where N_{py} is the number of frames for phone p and class y (when $N_{py} = 0$, w_{py} is set to 0). The second approach is meant to compensate for the imbalance in pronunciation scoring datasets, where some phones are more frequent than others and, for most phones, the positive class is more frequent than the negative class (because mispronunciations are not as common as correct pronunciations). The problem with this imbalance is that when the loss function is dominated by some phone-classes more than others, parameter adjustments will be motivated by the model's errors on those samples, thus leading to an imbalanced performance across the different phone-classes.

The model is trained using Adam optimization [20]. The training loss is given by Equation 2.4 averaged over all samples in a mini-batch. In the case of balanced weights, the N_{py} are computed over the complete mini-batch rather than independently for each sample, to give more stability to the loss since each individual sample contains only a subset of all phones making $N_{py} = 0$ for most phones and classes on most individual samples.

2.5 Metrics

In this section, we discuss how the performance of both the baseline and the new method will be evaluated. When testing these systems, our ground truth are the labels present in the manual annotations. What we want to see is a system that agrees with these labels as much as possible, so the metrics used in this evaluation will measure that in different ways. We will call samples classified as correctly pronounced *positives* and samples classified as incorrectly pronounced *negatives*.

Both the baseline and the proposed method output numeric scores for each phone. Thus, a threshold needs to be set to make a hard decision on which samples are classified as correct and which samples are classified as incorrect. The threshold can be different for each phone. Some of the metrics that will be discussed in this section measure performance for a specific set of thresholds, while others do not depend on them.

False Positive Rate and False Negative Rate

A false positive occurs when the system classifies a given phone as correctly pronounced but the ground truth labels for that phone are negative. As its name suggests, the *False Positive Rate* (FPR) is the percentage of ground-truth negatives that are wrongly classified as positives. The *False Negative Rate* is defined conversely. We would like to have both a FPR and a FNR that are as low as possible. The formula for the FPR is:

$$FPR = \frac{FP}{FP + TN} \tag{2.5}$$

where FP is the amount of false positives and TN is the amount of true negatives. Similarly, the FNR is defined as:

$$FNR = \frac{FN}{FN + TP} \tag{2.6}$$

Equal Error Rate

We use a threshold to decide whether a score obtained from the model corresponds to a correct or incorrect pronunciation. The threshold that is chosen affects the FPR and FNR. With a low enough threshold, all samples would be classified as positive. Thus, the FNR would be zero, since there would be zero samples detected as negative. Conversely, a high threshold, i.e. a having a very strict model, would result in a low FPR, because there would be few positives. These two metrics can be balanced: there is threshold value for which the FPR and FNR are equal or as

close to each other as possible. The FPR/FNR value for such a threshold is called *Equal Error Rate* (EER). We want the Equal Error Rate to be as low as possible.

ROC curve and AUC

Receiver Operating Characteristic (ROC) curves allow us to see the TPR (1-FNR) and FPR values as the threshold is varied across its full range of possible values.

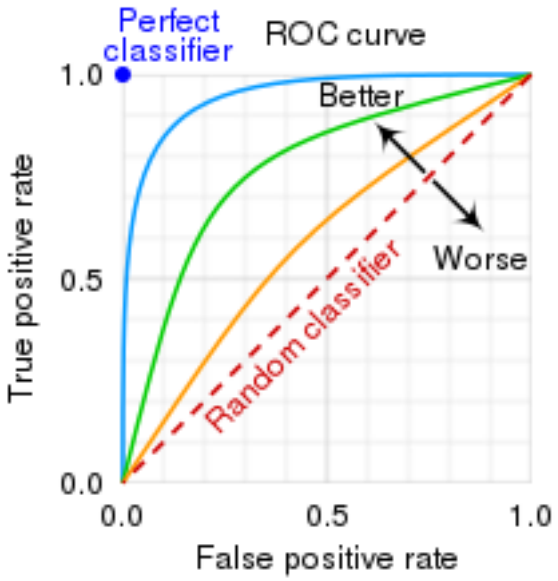

Fig. 2.5: An example of ROC curves. Taken from Wikimedia Commons.

As can be seen in figure 2.5, the better a classifier is, the further its ROC curve goes above the center line which represents a random classifier. If a system A has a ROC that is always above that of another system B, it means that, for the same FPR, the system A has a higher TPR and, hence, it is a more discriminative system than B. Thus, the *Area Under Curve* (AUC) is a good metric to evaluate a classifier. The area under the ROC curve will be higher if the ROC curve is higher. AUC is a very widely used metric. A perfect classifier would have an AUC of 1.0.

Cost

In the context of education, it is important not to frustrate students with inaccurate or unnecessary corrections. On the other hand, failing to detect a mistake is not as serious. For this reason, we have also defined a cost that prioritizes a low FNR over a low FPR:

$$Cost = 0.5FPR + FNR \tag{2.7}$$

False negatives contribute to this cost metric twice as much as false positives. Therefore, minimizing this metric means minimizing false negatives more than false positives. This type of cost function is widely used in speaker verification and language detection tasks [21], where the weights are determined depending on the application scenario. When setting the threshold to classify scores as positive or negative, one possible approach is to choose the threshold that minimizes the cost for each phone on the test data itself, resulting on the best possible cost on that data (MinCost). Selecting the optimal threshold on the test data, though, leads to optimistic estimates of the cost. Hence, for the evaluation data we also compute the cost obtained when the threshold is selected as the one that optimizes the cost on the development data for each phone. We call this the Actual Cost (ActCost). We can see an example of this in figures 2.6 and 2.7. Figure 2.6 shows the score distributions for correctly and incorrectly pronounced samples of phone /AY/ for one of the systems tested on the development set, with a vertical line showing the threshold that minimizes the cost function. Figure 2.7 shows the score distributions for the same phone and the same model configuration, but tested on the evaluation data. In this figure we can see two thresholds, the bold line being the threshold that was calculated on the development set, whereas the dotted line represents the threshold that was calculated on the evaluation set.

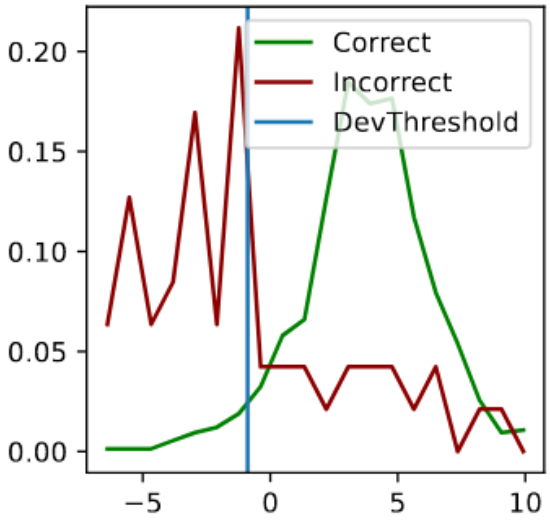

Fig. 2.6: Score distributions for correctly and incorrectly pronounced samples of phone /AY/ when testing a model on the development data. Vertical line shows the threshold that minimizes the cost function for the development data.

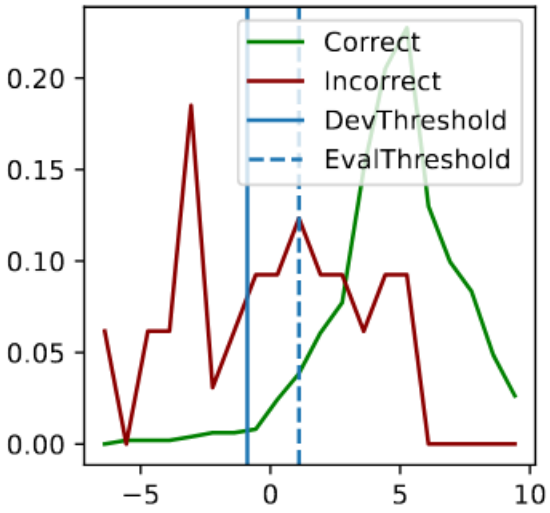

Fig. 2.7: Score distributions for correctly and incorrectly pronounced samples of phone /AY/ when testing a model on the evaluation data. Vertical lines shows the thresholds that minimize the cost function for the development and the evaluation data.

3. EXPERIMENTAL SETUP

In this chapter, we will describe the original ASR model used to compute the forced alignments and the baseline GOP scores, as well as the modified version of the model used in the new method. We will describe the setup used to train this model, including the non-native speech dataset. Much of this work was focused on the process of fine-tuning the model itself, which meant testing many different hyper parameter configurations, so these will also be outlined.

3.1 Non-native dataset description

The non-native speech data corpus used in this work is EpaDB (English Pronounced by Argentinians Database)[8]. It consists of 3200 short English utterances from 50 Spanish speakers from Argentina in different age and proficiency groups and balanced by gender. The recordings were manually annotated with phonetic transcriptions. These annotations, along with a lexicon containing a subset of all the correct pronunciations of each word, are used to generate correct/incorrect labels for training. The utterance list was designed to contain enough samples of each phone, with more emphasis on phones that are more likely to be mispronounced by speakers in the region. This dataset was randomly split in two groups speaker-wise. The utterances of 30 speakers were used as a development set and the 20 remaining speakers were held-out for final evaluation.

3.2 Acoustic Model

The acoustic model used for both the forced alignments and computing the posterior probabilities necessary for calculating GOP scores is a *Factorized Time Delay Neural Network* (FTDNN) [22] trained to classify 6024 senones. This ASR model is used in Kaldi's [23] official GOP recipe. We recreated its architecture using PyTorch and

extracted its parameters from the original Kaldi model to initialize the PyTorch version.

This architecture is a modification of the TDNN architecture, which was first proposed in [24]. The TDNN architecture was designed specifically for speech recognition, and it exhibits the property of shift invariance, simply meaning that the output of the model does not depend on the point in time where the input happened. This has been found to be useful in ASR systems because the sound of a phone does not depend very much on the point in time where it occurs within an utterance. However, this architecture does allow the model to detect temporal relations between different parts of an input sequence within a certain window (given by the amount of context frames), which is key in speech recognition because the pronunciation of a phone highly depends on the phones that surround it. This is accomplished by first feeding each frame to the network together with some number of adjacent frames for context. Similarly, when propagating features through the hidden layers, the feature sequence is also extended with the features from the sequence displaced by a certain amount of frames (the time delay). This process is shown in figure 3.1. This allows the layer to establish connections between the features at different nearby points in time.

The FTDNN architecture is similar to the TDNN architecture, with a slight difference. Weight matrices are factorized as a means for dimensionality reduction. Normally, trained weight matrices of linear layers can be factorized through *Singular Value Decomposition* [25]. The size of a weight matrix can be reduced by using only some of its singular values in the factorization, on the basis that not all of them are needed to capture most of the important information in the model. After this factorization, the model would be further fine-tuned. This led to the idea of using such a structure (layers with bottlenecks) in training from a random start, instead of first training a model normally and then factorizing its weight matrices and fine-tuning. In [22], the authors propose a method for training a factorized-TDNN (FTDNN) model[1] in such a way, with an additional "semi-orthogonality" constraint

[1] The architecture of the Factorized-TDNN proposed in [22] is similar, but not exactly the same

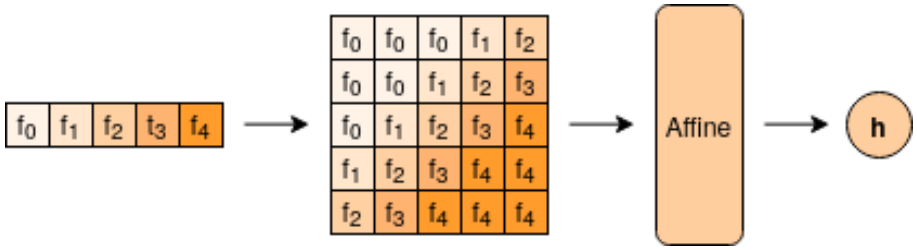

Fig. 3.1: Example of a TDNN hidden layer. In this example, a five frame long sequence is fed to the layer and a time-delay of two frames is used for context by concatenating the sequence displaced one and two frames backward and forward to the original sequence. f_t represents the features at frame t. The first and last frames are used for padding. The resulting sequence has the same amount of frames as the original sequence, but each frame contains the features of five frames of the original sequence. The time-delayed sequence is then fed to the affine operation. An activation function h is then applied to the output of the affine operation. The features f can be the input features to the network itself or the output of a previous layer.

for one of the factors for improved training stability.

The FTDNN acoustic model used in this work takes 220-dimensional features as input. The first 120 values correspond to the 40 MFCCs of the frame that is being processed, plus the 40 MFCCs of the previous and next frame as context. The last 100 values are the components of the iVector. There are 17 hidden layers in this FTDNN. Layers 2 through 17 are factorized into a linear plus an affine operation. Time-delayed concatenated features are created before feeding them to the linear and affine operations. These layers also have ReLU activations, batch normalization and dropout (these will be explained in section 3.4.2). The operations happening within these layers is shown in figure 3.2. Layers 2 through 4 use an offset of 1 frame for time-delays. Layers 6 through 17, on the other hand, use an offset of 3 frames. Layer 5 is the only hidden layer in the system that does not apply time-delays.

as the one that is implemented in the Kaldi ASR toolkit, and recreated in PyTorch for this work.

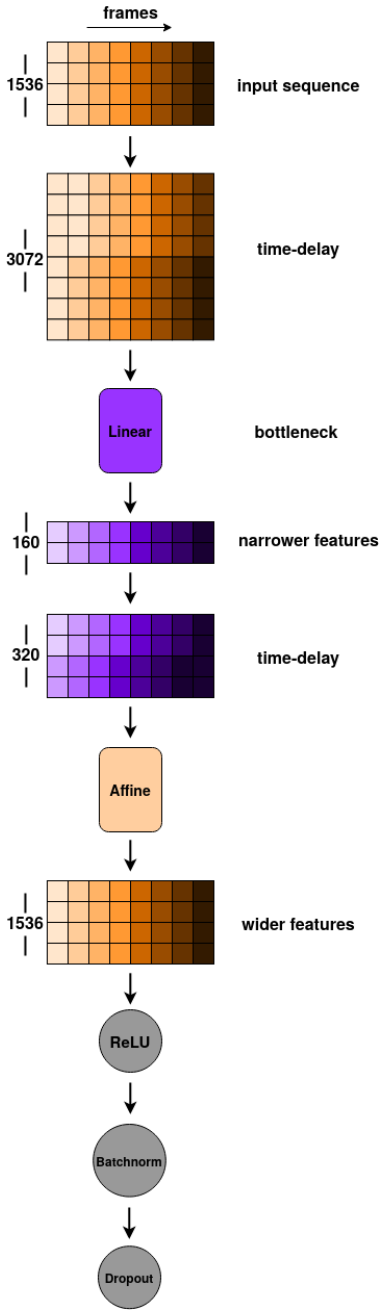

Fig. 3.2: Operations in an FTDNN layer used in the system. The first time-delay is applied only with negative offsets in the time dimension (feature sequence is shifted back). The time-delayed sequence is fed to the linear bottleneck, resulting in a narrower sequence. The second time-delay is applied only with positive offsets in the time dimension (feature sequence is shifted forward). The resulting sequence is fed to the affine transformation, which outputs a sequence as wide as the input sequence to be propagated to the next layer after ReLU activations, batch normalization, and dropout are applied. Numbers to the left of the sequences represent the width of the features. Note that the use of a linear bottleneck results in a much smaller amount of parameters than if the layer simply used a 1536x1536 affine transformation, which is the motivation for this factorization.

Layer 18 is a simple linear bottleneck layer with a width of 256 nodes. The original output layer has a dimension of 6024 (the number of senones in the Kaldi ASR model) and uses softmax activations. When fine-tuning this model to output pronunciation scores directly, we replace this last layer with one with 39 nodes, one for each target phone, each with a sigmoid activation.

Another interesting aspect of this architecture is how features are propagated through the model. Instead of only propagating each layer's outputs to the next layer, skip-connections are used: each layer takes a weighted sum of the previous two layers as input. Figure 3.3 shows the full architecture of the FTDNN with both possible output layers (the original used for senone classification, and the new one used for mispronunciation detection).

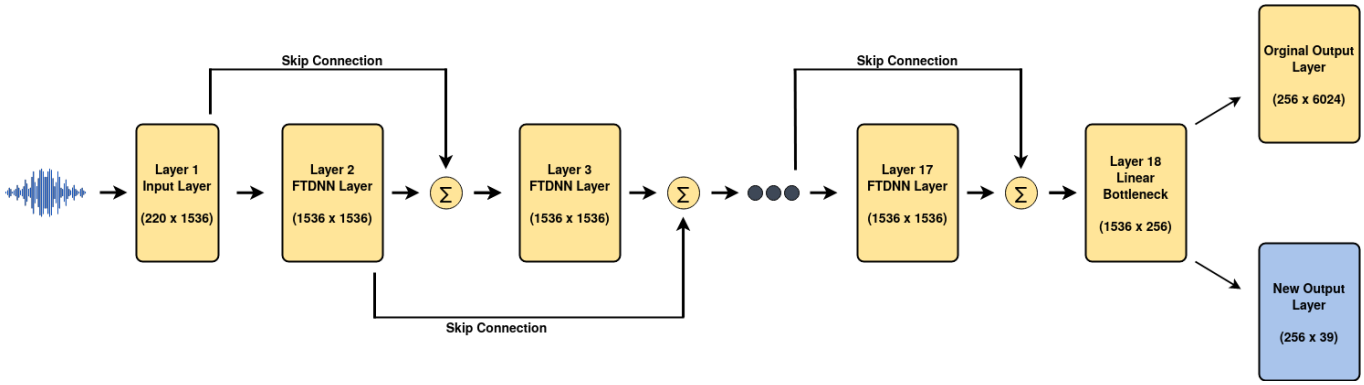

Fig. 3.3: Full architecture of the FTDNN used in this system with both possible output layers, the original one used for senone classification and the new one used for mispronunciation detection.

3.3 Baseline GOP

As a baseline, GOP scores were calculated using formula 2.3 by replicating the official Kaldi GOP recipe using PyKaldi [26]. The posterior probabilities, as well as the necessary forced alignments were computed using the above-mentioned acoustic model.

3.4 Fine-tuning experiments

As mentioned, once the Kaldi acoustic model was recreated in PyTorch, its 6024-node output layer was replaced with one with only 39 nodes (one per phone). During fine-tuning experiments, this layer was trained to output posterior probabilities for classes correctly/incorrectly pronounced for each phone directly. When training the new layer, we retain the parameters of the pre-trained model for all layers except the last as initial values. The replaced output layer is initialized with random values. We explore two approaches for fine-tuning the new model: **LayO**, where only the new output layer is trained, keeping all other parameters frozen at their pre-trained values, and **LayO+1**, where the last hidden layer is also trained. In the second case, we train the model in two stages: first, only the output layer is trained over several epochs and then the second to last layer is unfrozen and both layers are further fine-tuned. In our preliminary experiments this procedure gave better results than training both layers together from the first epoch, in agreement with many other works that do fine-tuning on different tasks [27]. However, further experiments showed comparable performance when training two layers in one or two stages, as we will see in chapter 4. We will also show that training three layers instead of two did not lead to further improvements, probably due to the third-to-last layer having too many parameters for the amount of training data available.

As mentioned, we use Adam optimization with loss function shown in equation 2.4 for fine-tuning. We used a batch size of 32 samples during all of the experiments.

3.4.1 Cross-Validation

During development, we used K-Fold Cross Validation to make the most out of the available training data. In K-Fold Cross Validation, the development set is divided in disjoint train and test sets in K different ways, each leaving out a different group of samples to test on. In this case, we used 6 folds. The 30 speakers were divided into 6 groups with 5 speakers each. During every training experiment, 6 different models were trained, each one using a different combination of 5 of the 6 groups of

speakers, leaving out the remaining group for testing. This ensures that no model is tested on the same data that it was trained on, which is always important to avoid reporting overly optimistic results. However, this technique also allows for both training and testing on the whole development set (even though no model is trained on the whole set), taking more advantage of the dataset than if the data was simply split in training and testing sets. Initially, the latter, simpler approach was used, but the data was found to be insufficient to effectively train and then validate the model.

Once the hyperparameters for a model configuration were tuned using cross-validation, a final evaluation of that configuration was done by training a single, new model on the 30 development speakers and then testing on the 20 held-out evaluation speakers.

3.4.2 Hyperparameters

Several combinations of hyperparameters were explored, with varying results that will be reported in chapter 4.

Batch normalization is the process of normalizing a batch of inputs by subtracting the mean and dividing by the standard deviation for each feature, where the statistics are computed over the batch's samples. This has shown to contribute to faster and more numerically stable training. All layers in the original model used batch normalization, but we tested the performance of the trained models with and without using batch normalization on the new output layer, and found that the models that used it performed better.

Dropout refers to randomly setting some percentage of the outputs of a linear layer to zero during each forward pass in training. This has shown to reduce overfitting to training data, the reason being that the model's layers see a slightly different input every time the same sample is fed into the model, since varying sets of weights get set to zero each time. We evaluated multiple possible values for dropout

probability, ranging from 0 (no dropout) to 0.5 with a granularity of 0.1. As we will see, the best results were found using a dropout probability of 0.4.

Learning rate is a parameter of the optimizer and refers to the size of the optimization steps that are taken, i.e. by how much the weights are adjusted in the direction opposite to the loss function's gradients in each step. A bigger learning rate means the model learns more quickly, but can lead to "overshooting" the optimal parameter values. On the other hand, a small learning rate causes slower convergence, but can also allow for a smoother approximation of the optimal parameters. This phenomenon is illustrated in figure 3.4. We tuned the learning rate by evaluating the values 0.001, 0.002, 0.005 and 0.01, and found 0.005 to be the best one when using a fixed learning rate.

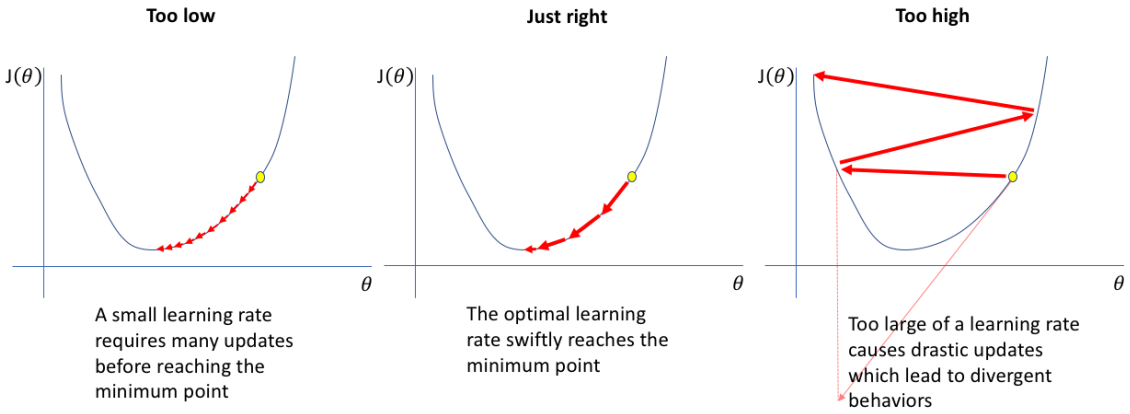

Fig. 3.4: Example of too low, too high, and optimal learning rates.
Taken from Jeremy Jordan's blog on learning rate tuning.

LR scheduling is a technique that consists of automatically tuning the learning rate between training iterations. It is often better than using a constant value because the optimal learning rate may be bigger when parameters are too far off in a certain direction, but smaller when they are close to a point of minimum loss. Typically, at the start of training, parameters can be tuned more aggressively, since

the loss is still very high. We experimented with multiple different schedulers and ultimately settled for exponential decay scheduling where the learning rate decays by a factor of 0.9 every 10 epochs, starting from a relatively large learning rate of 0.01, which proved to be better than using a fixed learning rate.

Balanced loss weights were used to prevent overly represented classes in the dataset from contributing too much to the loss while classes with less occurrences contribute less, as explained in section 2.4. This showed better results than using flat weights. Some phones in the dataset have very few instances in the minority class (usually the incorrectly pronounced class). We considered 50 to be the minimum amount of instances in a given class necessary to effectively train the model. Hence, phones with less than 50 occurrences in the minority class had their weights set to zero so that they do not add noise to the loss (both when using balanced as well as flat weights).

4. RESULTS

In this chapter, we show the results of the experiments. In sections 4.1 and 4.2 are the experiments for finding the best dropout probability and learning rate, respectively. Section 4.3 compares results when training different amounts of layers, showing that training two layers provided better performance than training one or three. In that section, we also provide an analysis on whether or not it is better to use gradual unfreezing when training two layers or training them both from the start. In section 4.4, we compare models with different hyper parameter configurations. Finally, section 4.5 compares the GOP system against the best configuration for the fine-tuning method on the evaluation data.

4.1 Dropout tuning

Figure 4.1 shows average MinCost and 1-AUC (so that lower values are better for both metrics) across epochs for models trained with batch normalization and different dropout probabilities. Using a dropout probability of 0.3 gives a gain of 1.1% in 1-AUC and 2.1% in MinCost over using a probability of 0.2. Increasing the dropout probability to 0.4 results in a gain of 0.5% in 1-AUC and no visible gain in MinCost. A further increase to 0.5 degrades performance, resulting in an almost equal 1-AUC and MinCost to that of the system trained with a dropout probability of 0.2. This suggests that a probability of 0.5 is too high, making it difficult for the model to fit to the training data.

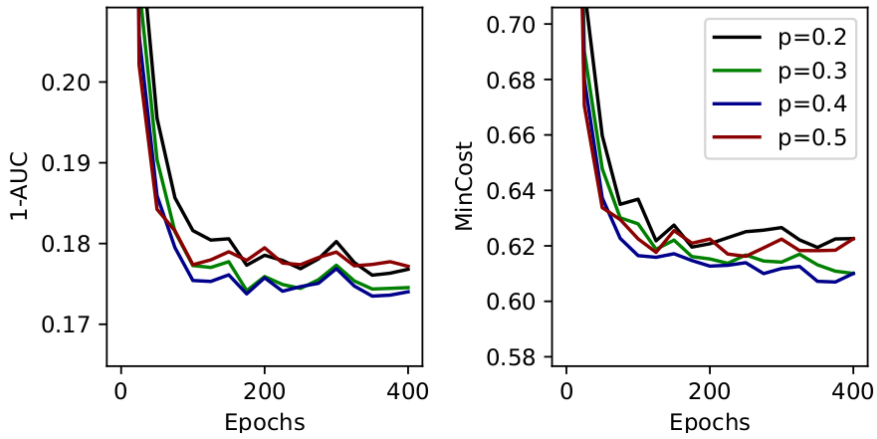

Fig. 4.1: Average 1-AUC and MinCost over phones with more than 50 samples of each class for the development data across epochs for models trained with different dropout probabilities. These models were trained with a fixed learning rate of 0.005, batch normalization and the balanced loss function.

.

4.2 Learning rate

As mentioned, two approaches were explored in regards to learning rate. The first option was using a fixed learning rate. After the best fixed learning rate was found, we experimented with a learning rate schedule starting at a bigger learning rate than the best fixed value that would then decrease by a factor of 0.9 every 10 epochs.

Figure 4.2 shows average MinCost and 1-AUC across epochs for models trained with the different fixed learning rate values. The model trained with the highest learning rate, 0.01, converges more quickly than the rest, but provides a 1-AUC and MinCost that are 0.6% and 1% worse than the model trained with a learning rate of 0.005 after 400 epochs. This indicates that a learning rate of 0.01 is too high, causing training to overshoot the optimal parameters.

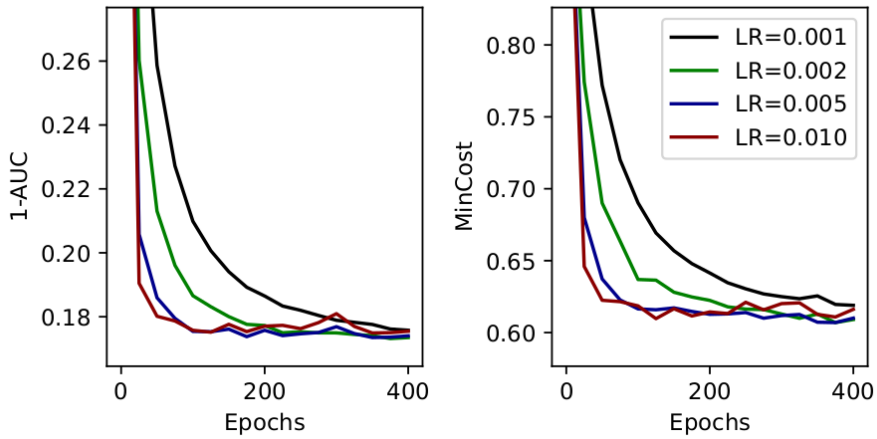

Fig. 4.2: Average 1-AUC and MinCost over phones with more than 50 samples of each class for the development data across epochs for models trained with different fixed learning rates, with batch normalization, a dropout probability of 0.4 and the balanced loss function.

We can also see that learning rates lower than 0.005 provide a more stable descent in both metrics, but take longer to converge. Considering this trade-off between smooth descent and how long it takes to converge, we consider 0.005 and 0.002 to be the best fixed learning rates, as 0.002 is only slightly slower than 0.005 and still provides similar results after 400 epochs with a smoother descent. We consider the learning rate of 0.001 to be too slow.

On the other hand, we see that the higher value of 0.01 works well at the start of training. Up to epoch 100, the model trained with the higher learning rate has lower 1-AUC and MinCost than all the other models. This is because aggressive updates to the parameters make sense at the beginning, when they are randomly initialized and very far off the optimal values. As mentioned, this motivates starting training at such a high learning rate value, and then progressively lowering it. Figure 4.3 shows average MinCost and 1-AUC across epochs for three models, one trained with the fixed learning rate of 0.005, another with a fixed learning rate of 0.002, and the last one using the mentioned learning rate scheduler. We can see that the model trained with the scheduler converges more quickly and smoothly than both models trained with fixed learning rates, starting with a bigger drop in both metrics and a

smoother and more consistent descent thereafter. This scheduler combines the best of both worlds, providing both a faster convergence than the higher fixed learning rate and a descent as smooth as that of the lower fixed learning rate.

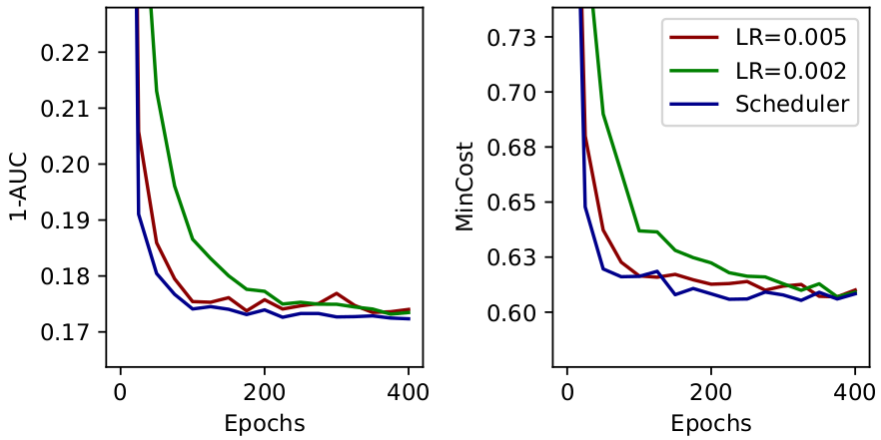

Fig. 4.3: Average 1-AUC and MinCost over phones with more than 50 samples of each class for the development data across epochs for models trained with fixed learning rates of 0.005 and 0.002, and one trained with a with a learning rare scheduler starting at 0.01 and decreasing by a factor of 0.9 every 10 epoch. The three models were also trained with batch normalization, a dropout probability of 0.4, and the balanced loss function.

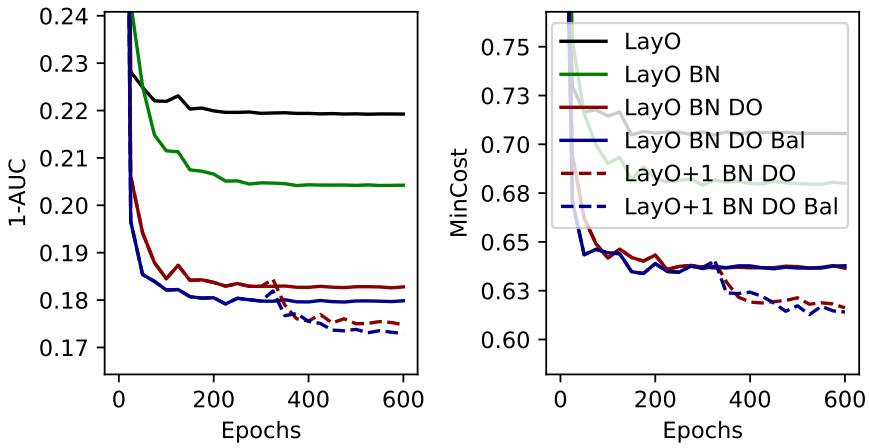

Fig. 4.4: Average 1-AUC and MinCost over phones with more than 50 samples of each class for the development data across epochs for various fine-tuning approaches. LayO and LayO+1: the last layer or the last two layers are fine-tuned. BN: batch-normalization is used as the first component in the output layer. DO: dropout is used in all layers (the dropout probability is 0.4, which is the optimal value that was found in the dropout tuning experiments). Bal: the loss with balanced weights is used in training. For reference, the GOP system has 1-AUC of 0.286 and MinCost of 0.801. All the models shown were trained with the learning rate schedule covered in section 4.2.

.

4.3 Training multiple layers

During fine-tuning, we tried training three different amounts of layers: just the new output layer, the last two layers, and the last three layers. When training more than one layer, the output layer can be trained on its own for a certain number of epochs, and then the previous layers can be unfrozen and further trained for an additional amount of epochs. In fine-tuning tasks, this is often better than training both layers in a single stage [27]. This is due to the fact that the new output layer is initialized randomly, whereas the layers before are pre-trained. Hence, training all layers at the same time can cause the pre-trained layer to forget what it had learnt before.

We found this to be the case in some of our preliminary experiments. However, with other setups, we found it did not make a difference whether or not we used gradual unfreezing to train more than one layer. Such is the case with the results shown in figure 4.5, where we compare 1-AUC and MinCost across epochs for four models: one where only one layer is trained (1-Layer), one where two layers are trained with gradual unfreezing (2-Layers-Double-Stage), one without gradual unfreezing (2-Layers-Single-Stage), and one where three layers are trained (3-Layers-Double-Stage), also with gradual unfreezing. 2-Layers-Double-Stage converges to the same performance as 2-Layers-Single-Stage, but it takes slightly longer, which means it may even be better to train both at the same time to have a lower training time. However, we still found gradual unfreezing preferable for the rest of the experiments since it implies training for only 300 epochs if the last layer has already been trained on its own, whereas training both layers in a single stage takes at least 400 epochs to converge.

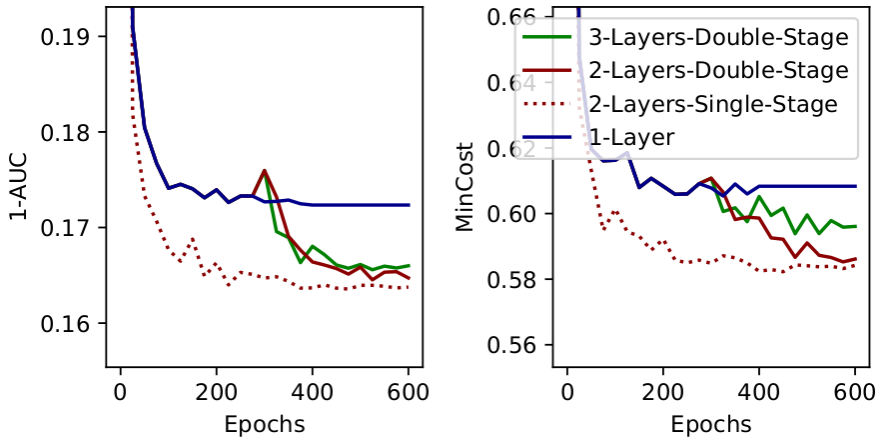

Fig. 4.5: Average 1-AUC and MinCost over phones with more than 50 samples of each class for the development data across epochs for four models. In one model, only one layer was trained. In another, three layers were trained. There are two setups where two layers were trained. Single-Stage means both layers were trained from the start, whereas in Double-Stage, the last layer was trained on its own first for 300 epochs, and then the second-to-last layer was unfrozen and both layers were trained for an additional 300 epochs. In the setup where three layers were trained, the last layer was trained on its own for 300 epochs, then both additional layers were unfrozen at the same time and all three layers were trained for an additional 300 epochs. All models were also trained with batch normalization, a dropout probability of 0.4, the learning rate schedule and the balanced loss function.

4.4 Comparison of different configurations

Figure 4.6 shows average MinCost and 1-AUC across epochs for different fine-tuning approaches on the cross-validation data. The first system (LayO) corresponds to fine-tuning only the output layer, without batch normalization or dropout and using the flat loss function. Adding batch normalization as the first block in the output layer gives a relative gain of 6% in 1-AUC and 4% in MinCost, while doing dropout during training with a probability of 0.4 gives gains of 10% and 6%. Finally, LayO+1, the system where the last two layers are fine-tuned, gives an additional

gain of 5% and 3%. Using the balanced loss function gives a small but consistent gain over using the flat loss. Interestingly, the trends on 1-AUC and MinCost are similar. The best system, "LayO+1 BN DO Bal", gives a gain of 40% in 1-AUC and 23% in MinCost over the baseline GOP system on this data.

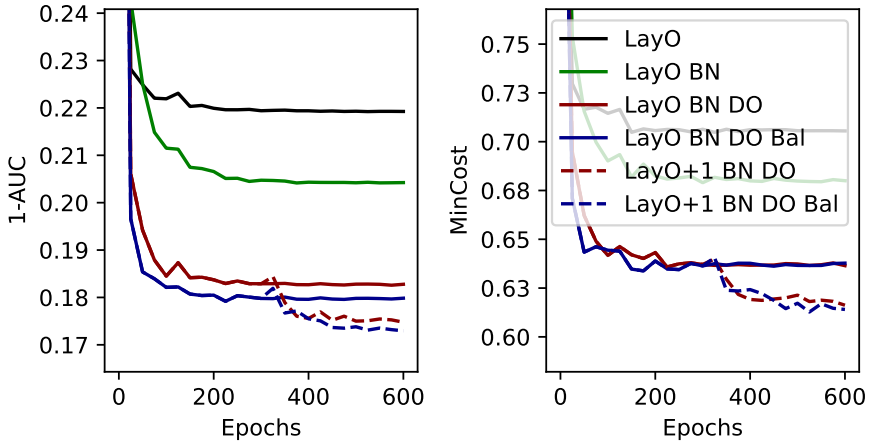

Fig. 4.6: Average 1-AUC and MinCost over phones with more than 50 samples of each class for the development data across epochs for various fine-tuning approaches. LayO and LayO+1: the last layer or the last two layers are fine-tuned. BN: batch-normalization is used as the first component in the output layer. DO: dropout is used in all layers (the dropout probability is 0.4, which is the optimal value that was found in the dropout tuning experiments). Bal: the loss with balanced weights is used in training. For reference, the GOP system has 1-AUC of 0.286 and MinCost of 0.801. All the models shown were trained with the learning rate schedule covered in section 4.2.

.

4.5 Final results against GOP

Figure 4.7 shows the results on the 20 evaluation speakers for the GOP baseline and the best fine-tuned model (referred to as **LayO+1 BN DO Bal** in 4.6), which we call GOP-FT for short. The bars with a solid black line show the MinCost, where the threshold for each phone is given by the one that optimizes the cost on the evaluation

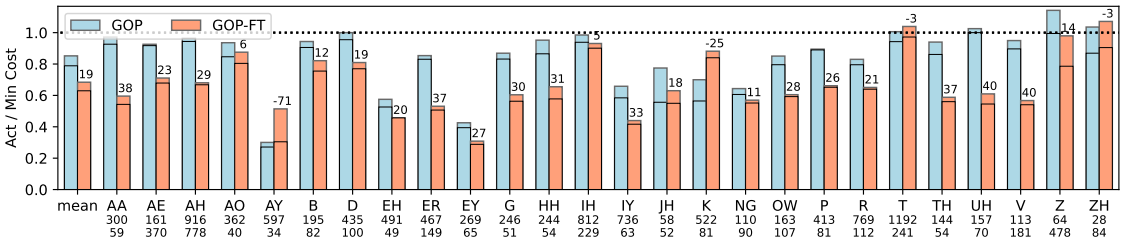

Fig. 4.7: ActCost (bar height) and MinCost (black lines) per phone on the evaluation data. Values under the x-axis are the number of correctly and incorrectly pronounced instances of each phone. Numbers on top of the GOP-FT bars are the relative gains. The dashed horizontal line indicates the value of ActCost for a naïve system that always decides the phone was correctly pronounced.

data itself. These are optimistic estimates of the cost, since, in practice, one never has the full evaluation data to estimate the thresholds on. The top bars, on the other hand, are the ActCost, where the thresholds are estimated using the cross-validation scores, as would be done in practice. We can see that the ActCost is within 10% of the MinCost for most phones, indicating that the thresholds chosen on development speakers generalize well to the unseen speakers. The "AY" phone is a particularly striking exception to this, as there is a big difference between MinCost and ActCost for this phone. The average FNR rate corresponding to these thresholds is 10% and 13%, for the GOP and GOP-FT systems, respectively, an acceptable level for real use scenarios [28, 29]. The average FPR is 64% for GOP and 41% for GOP-FT, showing a large relative improvement from the fine-tuning approach where 23% more of the incorrect pronunciations are detected as such.

Figure 4.7 shows a wide range of cost values across phones. In most cases, the proposed fine-tuning approach leads to gains over the GOP baseline. We hypothesize these are cases where the original ASR DNN was too permissive, allowing wrong pronunciations to get large senone posteriors for the target phone. This is corrected by fine-tuning since the system learns to distinguish what annotators considered good and bad pronunciations. For a few phones, the cost degrades with fine-tuning, though the degradation is relatively small in most cases. These might be cases where

the model has overfitted to the training data. Again, the biggest exception to this is phone "AY". Finally, note that for those phones where the cost is close to or above 1.0, the value for a naive system that always decides correct pronunciation, the system should probably not be used in practice since it would not provide useful information. Broadly, these results are similar to those obtained by Huang and others in [16] and in other works on this task. Clearly, despite the gains obtained with the proposed approach, there is still work to do to improve performance on this task, as the performance of this system and others is still far from that of a human teacher.

4.5.1 Phone "AY"

As mentioned, the proposed method is worse than the baseline method for the phone "AY", more so than for any other phone. In this section, we will analyze why this may have happened.

In order to assess this issue, we will analyze the cases where the proposed system made mistakes but the baseline system was right. Figure 4.8 shows the scores assigned by both systems for these cases in the evaluation data. There are 11 false positives and 5 false negatives.

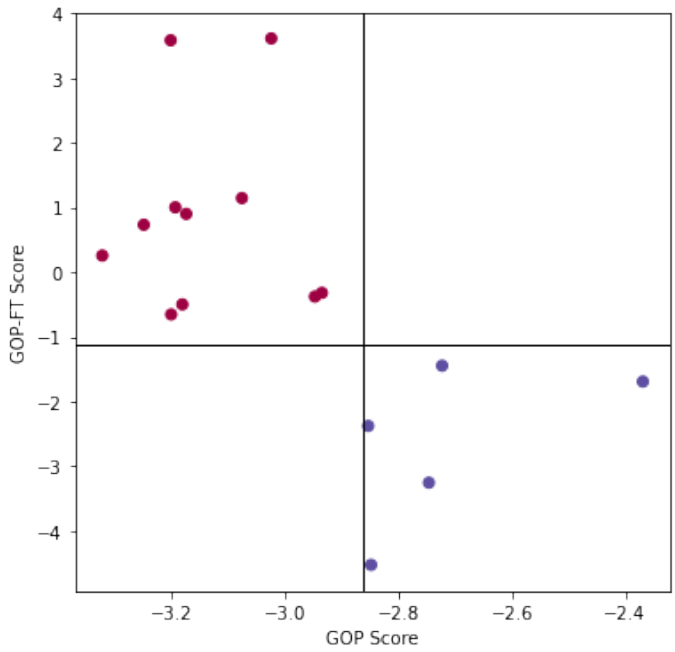

Fig. 4.8: Scatter plot of the scores from both systems for samples of phone "AY" in the evaluation data where the fine-tuned system was wrong and the baseline system was right. The horizontal line represents the ActCost threshold for the fine-tuned system and the vertical line represents the threshold for the baseline system. Dots marked in red represent samples of incorrectly pronounced phones, while the blue dots represent samples of correctly pronounced phones according to the ground-truth labels.

We analyzed each of these cases to see if there was a common explanation for the mistakes. A priori, we considered a few possibilities. Firstly, it could be that the manual annotations themselves were wrong, and the system was right. Secondly, it could be a problem with the forced alignments. Since the phone boundaries used during inference are given by the forced alignments, inaccuracies of the forced alignments will result in mistakes in the downstream network. Lastly, it could simply be that the fine-tuned model was wrong due to over fitting to the training data or that there were low quality samples in the test data.

After listening to the recordings and reviewing the manual annotations, we concluded that the annotations were in fact correct in all cases, so the first explanation

was ruled out. To analyze the quality of the phone boundaries given by the forced alignments for these samples, we trimmed the recordings and listened to the audio between these boundaries. Two of the false negatives shown in figure 4.8 did in fact have inaccurate phone boundaries. After listening to the audio, we found that one did not include the full phone and the other included a big part of the preceding phone, which evidently confused the model. However, there are still 14 mistakes to account for where the forced alignments are reasonable. The best explanation for this is that the model is indeed not fit for these samples. This may be due to the fact that there are only 52 samples of the incorrectly pronounced class for this phone in the training data.

4.5.2 Phone "V"

Contrary to phone "AY", phone "V" is a phone for which the proposed method is much better than the baseline method in terms of ActCost and MinCost on the evaluation data. In this section we will analyze why this improvement might have occured.

Figure 4.9 shows the scores assigned by both systems for samples in the evaluation data where the fine-tuned system was right and the baseline system was wrong. The mistakes made by the GOP system are mostly false positives (76 out of the 85 mistakes), meaning that the phone was incorrectly pronounced (according to the manual annotations), but the GOP system classified them as correctly pronounced. Hence, the focus of this analysis will be on the false positives. This can be explained by considering that the GOP approach relies solely on the posterior probabilities from the DNN acoustic model, which was trained for the ASR task. As such, the acoustic model benefits from outputting large posterior probabilities for phones that do not sound perfect, since this allows the ASR system to recognize phones correctly even if the audio is noisy or low-quality. An ASR system can benefit from a "wide" acoustic model. However, this means that also slightly incorrectly pronounced phones can get large posterior probabilities, which

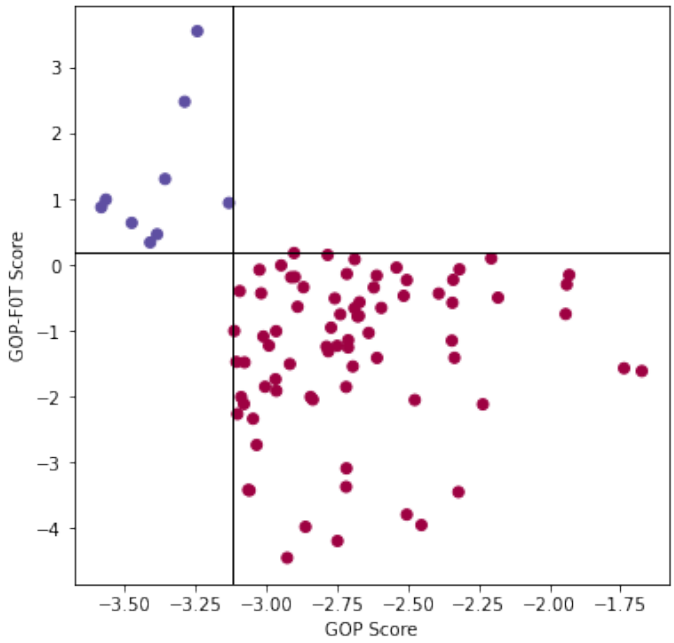

Fig. 4.9: Scatter plot of the scores from both systems for samples of phone "V" in the evaluation data where the fine-tuned system was right and the baseline system was wrong. The horizontal line represents the ActCost threshold for the fine-tuned system and the vertical line represents the threshold for the baseline system. Dots marked in red represent samples of incorrectly pronounced phones, while the blue dots represent samples of correctly pronounced phones.

defeats the purpose of using them to decide if a phone is correctly pronounced or not. This possible downside of the GOP method was already mentioned in section 2.4. We hypothesize that the fine-tuned model was able to better distinguish between correctly and incorrectly instances of phone "V" thanks to having seen what mispronounced instances of this phone look like in Argentine speakers during training, thereby learning to reject them.

5. CONCLUSIONS

In this work we present a simple transfer-learning approach for pronunciation scoring, where the DNN acoustic model from an ASR system is fine-tuned for the task of detecting correctly versus incorrectly pronounced phones by changing only the output layer's architecture and training it on non-native data labelled with pronunciation quality. The motivation for the transfer-learning approach (as opposed to training an entirely new model from a random start) is that non-native datasets designed for pronunciation scoring are very scarce. To our knowledge, EpaDB is the only one designed for Argentinian speakers (or spanish speakers in general). By fine-tuning a pre-trained acoustic model, we can take advantage of what the model has already learnt on a large amount of native data, getting a head start. This is possible because the ASR task is tightly related to the pronunciation scoring task. On the other hand, training a neural network as big as the one we fine-tuned from a random start would have been impossible with the amount of data available.

We explored several approaches for fine-tuning. The largest gains were shown using batch normalization and dropout. Unfreezing the last hidden layer in the DNN also provided an additional substantial improvement in performance compared to training only the new output layer. We also found small but consistent gains by using a loss function normalized by the amount of samples of each class in a batch to balance the contribution of each class to the loss during training. Finally, we proposed the use of a cost function designed to encourage low false correction rates to evaluate the system's performance, something the community agrees to be essential for the practical use of systems intended for education. We show that our best fine-tuned model is, on average, 20% better in terms of cost compared to a state-of-the-art GOP system that uses the same acoustic model.

We believe there are several improvements that could be made to the method proposed in this work. For instance, the loss is currently only computed for each

frame individually. However, each phone in an utterance spans several frames, so we could calculate the loss for the whole phone instance by averaging the model's outputs for all the frames spanned by it and then do back-propagation based on the loss for that average of phone-level output values. This could potentially lead to a more robust loss function, as some of the loss values for individual frames may not be very representative of the quality of the model's prediction across the whole phone instance. This may specially be true for frames that are close to phone boundaries, due to inaccuracies in the aligner. In the end, it is the phone-level prediction that matters for this task, and not the prediction for each frame. Thus, intuitively, it would be better to optimize the model as an end-to-end system, giving good phone-level scores directly instead of frame-level scores. We could also try using attention to give some frames more weight than others when calculating the phone-level scores.

Another aspect that could be further explored is the model's architecture. It would be interesting to see, for instance, if we can obtain similar or better results using a smaller ASR model, such as a small TDNN. The architecture of the model we fine-tuned in this work is considerably large. This may be justified for the ASR task, but we believe it may be unnecessary for pronunciation scoring, which is explored in Cyntia Bonomi's thesis [30]. Furthermore, if the layers in the model we fine-tune had fewer nodes, it may be possible to fine-tune three or more layers with the amount of training data available, which we found to be worse than training just two layer in this case. We could also try adding more than one new layer to the end of the acoustic model.

As mentioned in chapter 4, although we saw significant improvements compared to the baseline method, this system's performance is still not good enough to be used in a real scenario, as is the case with state-of-the-art systems for this task in general. There is still much work to be done in phone-level pronunciation scoring, but we think these systems will eventually work well enough to be an useful tool for language learning.

BIBLIOGRAPHY

[1] C. Tejedor-García, D. Escudero-Mancebo, E. Cámara-Arenas, C. González-Ferreras, and V. Cardeñoso-Payo, "Assessing pronunciation improvement in students of english using a controlled computer-assisted pronunciation tool," *IEEE Transactions on Learning Technologies*, vol. 13, no. 2, pp. 269–282, 2020.

[2] N.F. Chen and H. Li, "Computer-assisted pronunciation training: From pronunciation scoring towards spoken language learning," in *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE, 2016, pp. 1–7.

[3] S. Wei, G. Hu, Y. Hu, and R. Wang, "A new method for mispronunciation detection using support vector machine based on pronunciation space models," *Speech Communication*, vol. 51, no. 10, pp. 896–905, 2009.

[4] H. Franco, L. Ferrer, and H. Bratt, "Adaptive and discriminative modeling for improved mispronunciation detection," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 7709–7713.

[5] H. Franco, L. Neumeyer, Y. Kim, and O. Ronen, "Automatic pronunciation scoring for language instruction," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1997, vol. 2, pp. 1471–1474.

[6] Yoon Kim, Horacio Franco, and Leonardo Neumeyer, "Automatic pronunciation scoring of specific phone segments for language instruction," in *Proc. 5th European Conference on Speech Communication and Technology (Eurospeech 1997)*, 1997, pp. 645–648.

[7] S.M. Witt and S.J. Young, "Phone-level pronunciation scoring and assessment

for interactive language learning," *Speech communication*, vol. 30, no. 2-3, pp. 95–108, 2000.

[8] J. Vidal, L. Ferrer, and L. Brambilla, "Epadb: a database for development of pronunciation assessment systems," *INTERSPEECH*, pp. 589–593, 2019.

[9] Lawrence R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[10] Douglas A. Reynolds, "Gaussian mixture models," in *Encyclopedia of Biometrics*, 2009.

[11] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[12] W. Hu, Y. Qian, and F.K. Soong, "An improved dnn-based approach to mispronunciation detection and diagnosis of l2 learners' speech.," in *SLaTE*, 2015, pp. 71–76.

[13] Jiatong Shi, Nan Huo, and Qin Jin, "Context-Aware Goodness of Pronunciation for Computer-Assisted Pronunciation Training," in *INTERSPEECH*, 2020, pp. 3057–3061.

[14] S. Sudhakara, M.K. Ramanathi, C. Yarra, and P. K. Ghosh, "An improved goodness of pronunciation (gop) measure for pronunciation evaluation with dnn-hmm system considering hmm transition probabilities.," in *INTERSPEECH*, 2019, pp. 954–958.

[15] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He, "A comprehensive survey on transfer learning," 2019.

[16] H. Huang, H. Xu, Y. Hu, and G. Zhou, "A transfer learning approach to goodness of pronunciation based automatic mispronunciation detection," *The Journal of the Acoustical Society of America*, vol. 142, no. 5, pp. 3165–3177, 2017.

[17] F. Nazir, M. N. Majeed, M. A. Ghazanfar, and M. Maqsood, "Mispronunciation detection using deep convolutional neural network features and transfer learning-based model for arabic phonemes," *IEEE Access*, vol. 7, pp. 52589–52608, 2019.

[18] N. Dehak, P.J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2010.

[19] M.Y. Hwang and X. Huang, "Subphonetic modeling with markov states-senone," in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1992, vol. 1, pp. 33–36 vol.1.

[20] D. P Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[21] D.A. Van Leeuwen and N. Brümmer, "An introduction to application-independent evaluation of speaker recognition systems," in *Speaker classification I*, pp. 330–353. Springer, 2007.

[22] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, and S. Khudanpur, "Semi-orthogonal low-rank matrix factorization for deep neural networks.," in *INSTERSPEECH*, 2018, pp. 3743–3747.

[23] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al., "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number CONF.

[24] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989.

[25] Jian Xue, Jinyu Li, and Yifan Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *INTERSPEECH*, 2013.

[26] C. Dogan, Martinez V., Papadopoulos P., and Narayanan S.S, "Pykaldi: A python wrapper for kaldi," in *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on*. IEEE, 2018.

[27] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv:1801.06146*, 2018.

[28] Claudia Baur, Cathy Chua, Johanna Gerlach, Manny Rayner, Martin Russell, Helmer Strik, and Xizi Wei, "Overview of the 2017 Spoken CALL Shared Task," in *Proc. 7th ISCA Workshop on Speech and Language Technology in Education (SLaTE)*, 2017, pp. 71–78.

[29] Sandra Kanters, Catia Cucchiarini, and Helmer Strik, "The goodness of pronunciation algorithm: a detailed performance study," in *Proc. Speech and Language Technology in Education (SLaTE)*, 2009, pp. 49–52.

[30] Cyntia Bonomi, "Desarrollo y evaluación de sistemas de calificación de pronunciación basado en redes neuronales," 2022.