ACOUSTIC NEIGHBOR EMBEDDINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper proposes a novel acoustic word embedding called Acoustic Neighbor Embeddings where speech or text of arbitrary length are mapped to a vector space of fixed, reduced dimensions by adapting stochastic neighbor embedding (SNE) to sequential inputs. The Euclidean distance between coordinates in the embedding space reflects the phonetic confusability between their corresponding sequences. Two encoder neural networks are trained: an acoustic encoder that accepts speech signals in the form of frame-wise subword posterior probabilities obtained from an acoustic model and a text encoder that accepts text in the form of subword transcriptions. Compared to a known method based on a triplet loss, the proposed method is shown to have more effective gradients for neural network training. Experimentally, it also gives more accurate results when the two encoder networks are used in tandem in a word (name) recognition task, and when the text encoder network is used standalone in an approximate phonetic match task. In particular, in an isolated name recognition task depending solely on Euclidean nearest-neighbor search between the proposed embedding vectors, the recognition accuracy is identical to that of conventional finite state transducer(FST)-based decoding using test data with up to 1 million names in the vocabulary and 40 dimensions in the embeddings.

1 INTRODUCTION

Acoustic word embeddings (Levin et al., 2013; Maas et al., 2012) are vector representations of words that capture information on how the words *sound*, as opposed to word embeddings that capture information on what the words *mean*. A number of acoustic word embedding methods have been proposed, applied to word discrimination (He et al., 2017; Jung et al., 2019), lattice rescoring in automatic speech recognition (ASR) (Bengio & Heigold, 2014), and query-by-example keyword search (Settle et al., 2017) or detection (Chen et al., 2015). Recently, triplet loss functions (He et al., 2017; Settle et al., 2019; Jung et al., 2019) have been used to train two neural networks simultaneously: an acoustic encoder $f(\cdot)$ network[1] that accepts speech, and a text encoder $g(\cdot)$ network that accepts text as the input. By training the two to transform their inputs into a common space where matching speech and text get mapped to the same coordinates, f and g can be used in tandem in applications where a speech utterance is compared against a database of text, or vice versa. They can also each be used as a standalone, general-purpose word embedding network that maps similar-sounding speech (in the case of f) or text (in the case of g) to similar locations in the embedding space.

An obvious application is highly-scalable isolated word (or name) recognition (e.g. in a music player app where the user can tap the search bar to say a song or album title), where a given speech input is mapped via f to an embedding vector $\mathbf{f}$, which is then compared against a database of embedding vectors $\{\mathbf{g}_1, \cdots, \mathbf{g}_N\}$, prepared via g, that represents a vocabulary of N words, to classify the speech. Using the Euclidean distance, the classification rule is:

$$i = \arg \min_{1 \leq j \leq N} ||\mathbf{f} - \mathbf{g}_j||^2, \tag{1}$$

where $|| \cdot ||$ is the L_2 norm. Vector distances have been used in the past for other matching problems (e.g. Schroff et al. (2015)). For speech recognition, a rule like (1) is interesting because it can be easily parallelized for fast evaluation on modern hardware (e.g. Garcia et al. (2008)), and the vocabulary

[1] It is interesting to note that the basic notion of "memorizing" audio signals in fixed dimensions can be traced back to as early as Longuet-Higgins (1968)

can also be trivially updated since each entry is a vector. If proven to work well in isolated word recognition, the embedding vectors could be used in continuous speech recognition where speech segments hypothesized to contain named entities can be recognized via nearest-neighbor search, particularly when we want an entity vocabulary that is both large and dynamically updatable. However, none of the aforementioned papers have reported results in isolated word recognition.

The main contribution in this paper is a new training method for f and g which adapts stochastic neighbor embedding (SNE) (Hinton & Roweis, 2003) to sequential data. It will be shown by analysis of the gradients of the proposed loss function that it is more effective than the triplet loss for mapping similar-sounding inputs to similar locations in a vector space. It will also be shown that the embeddings produced by the proposed method, called Acoustic Neighbor Embeddings (ANE), work better than embeddings produced by a triplet-loss-based method in isolated name recognition and approximate phonetic match experiments. This paper also claims to be the first work that successfully uses a simple L_2 distance between vectors to directly perform isolated word recognition that closely approaches the accuracy of conventional FST-based recognition over large vocabularies.

One design choice in this work for the acoustic encoder f is that instead of directly reading acoustic features, a separate acoustic model is used to preprocess them into (framewise) subword posterior probability estimates. "Posteriorgrams" have been used in past studies (e.g. Hazen et al. (2009)) for speech information retrieval. In the proposed work, they allow the f network to be much smaller, since the task of resolving channel and speaker variability can be delegated to a state-of-the-art ASR acoustic model. One can still use acoustic features with the proposed method, but in many practical scenarios an acoustic model is already available for ASR that computes subword scores for every speech input, so it is feasible to reuse those scores.

As inputs to the text encoder g, this study experiments with two types of text: phone sequences and grapheme sequences. In the former case, a grapheme-to-phoneme converter (G2P) is used to convert each text input to one or more phoneme sequences, and each phoneme sequence is treated as a separate "word." This approach reduces ambiguities caused by words that could be pronounced multiple ways, such as "A.R.P.A", which could be pronounced as (in ARPABET phones) "aa, r, p, aa", or "ey, aa, r, p, iy, ey", or "ey, d, aa, t, aa, r, d, aa, t, p, iy, d, aa, t, ey" (pronouncing every "." as "dot"). In the latter case using graphemes, more errors can occur in word recognition because a single embedding vector may not capture all the variations in how a word may sound. On the other hand, such a system can be more feasible because it does not require a separate G2P.

Also note that while we use the term "word embedding" following known terminology in the literature, a "word" in this work can actually be a sequence of multiple words, such as "John_W_Smith" or "The_Hilton_Hotel" as in the name recognition experiments in Section 5.

2 Review of stochastic neighbor embedding

In short, stochastic neighbor embedding (SNE) (Hinton & Roweis, 2003) is a method of reducing dimensions in vectors while preserving relative distances, and is a popular method for data visualization. Given a set of N coordinates $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, SNE provides a way to train a function $f(\cdot)$ that maps each coordinate $\mathbf{x}_i$ to another coordinate of lower dimensionality $\mathbf{f}_i$ where the relative distances among the $\mathbf{x}_i$'s are preserved among the corresponding $\mathbf{f}_i$'s.

The distance between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the input space is defined as the squared Euclidean distance with some scale factor σ_i:

$$d_{ij}^2 = \frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i^2}. \tag{2}$$

This distance is used to define the probability of $\mathbf{x}_i$ choosing $\mathbf{x}_j$ as its neighbor in the input space:

$$p_{ij} = \frac{\exp\left(-d_{ij}^2\right)}{\sum_{k \neq i} \exp\left(-d_{ik}^2\right)}. \tag{3}$$

The corresponding "induced" probability in the embedding space is

$$q_{ij} = \frac{\exp\left(-||\mathbf{f}_i - \mathbf{f}_j||^2\right)}{\sum_{k \neq i} \exp\left(-||\mathbf{f}_i - \mathbf{f}_k||^2\right)}. \tag{4}$$

The loss function for training f is the Kullback-Leibler divergence between the two distributions:

$$\mathcal{L}_f = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{5}$$

which can be differentiated to obtain this beautiful equation:

$$\frac{\partial \mathcal{L}_f}{\partial \mathbf{f}_i} = 2 \sum_j (\mathbf{f}_i - \mathbf{f}_j)(p_{ij} - q_{ij} + p_{ji} - q_{ji}). \tag{6}$$

Hinton & Roweis (2003)'s cogent interpretation of the above equation as "a sum of forces pulling $\mathbf{f}_i$ toward $\mathbf{f}_j$ or pushing it away depending on whether j is observed to be a neighbor more or less often than desired" is a seed for other arguments that will be made in the present paper.

3 Proposed embedding method

3.1 Method description

Consider a training set of N speech utterances. The n'th utterance is characterized by (S_n, X_n, Y_n) where S_n is the audio signal containing one or more words of speech, X_n is a sequence of subword posterior probability vectors for S_n, and Y_n is a sequence of subwords pertaining to the reference transcription of S_n.

$X_n = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T]$ is obtained from an acoustic model such as a DNN-HMM (Deep Neural Network-Hidden Markov Model) system, where the d'th element of $\mathbf{x}_t$ is an estimate of the posterior probability of subword w_d occurring at frame t given the speech signal S_n. There is a subword posterior vector for every speech frame, and T is the number of frames in the utterance. While monophone posteriors are used in this study, other subword posteriors could be used, such as HMM state or grapheme posteriors. Also note that raw acoustic features could be used for X_n, but posteriors were used here for reasons mentioned in Section 1.

The subword reference transcription of each utterance is also obtained, such as the phone sequence "k, r, ao, s, ih, ng," or the grapheme sequence "c, r, o, s, s, i, n, g" for the word "crossing," by force-aligning the utterance with its manual word transcription using an ASR with a pronunciation dictionary. This sequence is represented as a sequence of 1-hot vectors $Y_n = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M]$ where M is the number of subwords in the sequence. Each subword usually occupies at least one frame of speech, so $T \geq M$ for every utterance.

Note that the "subwords" used for the posteriors in X_n need not be the same as the "subwords" used for the transcriptions in Y_n (for ANE-g in Section 5, X_n is a monophone posterior vector sequence, whereas Y_n is a grapheme transcription).

The idea in this paper is to train an acoustic encoder neural network $f(\cdot)$ that will transform each posterior vector sequence to a single fixed-dimension embedding vector $\mathbf{f}_n = f(X_n)$ such that predefined relative distances between data samples in the space of X_n will be preserved in the space of $\mathbf{f}_n$ in a manner similar to SNE.

First, since each X_i is a sequence of vectors, the Euclidean distance in (2) does not make sense in our input space. Instead, we define the distance between X_i and X_j based on whether their transcriptions are an exact match in the space of Y:

$$d_{ij} = \begin{cases} 0 & \text{if } Y_i = Y_j \\ \infty & \text{else} \end{cases}. \tag{7}$$

It is worth noting that alternate forms of d_{ij} based on dynamic time warping between X_i and X_j (in a manner similar to Hazen et al. (2009)) with heuristic insertion, deletion, and substitution costs were also tried, but the binary distance above gave better accuracy.

The distance in (7) results in the following probability for (3):

$$p_{ij} = \begin{cases} 1/c_i & \text{if } Y_i = Y_j \\ 0 & \text{else} \end{cases}, \tag{8}$$

where c_i is the number of utterances (other than the i'th) that have the same subword sequence Y_i.

We use the same induced probability in (4) for the embedding space. Since this probability is explicitly based on the L_2 norm, all comparisons we do in the embedding space (e.g. word recognition using (1) or phonetic distance computations in Table 1) are done using the L_2 norm.

The loss function for training the acoustic encoder f is $\mathcal{L}_f$ as described in (5). Once we have fully trained f, we simply train the text encoder g such that its output for every subword sequence Y_n will match as closely as possible the output of f for the corresponding subword posterior sequence X_n, where we keep f fixed. A simple mean square error loss proves to be sufficient for this purpose:

$$\mathcal{L}_g = \sum_{n=1}^{N} ||g(Y_n) - f(X_n)||^2. \tag{9}$$

3.2 Training strategy for the f acoustic encoder

In practice, the training data for the proposed system is much larger (millions of utterances) than the original data (thousands of data points) for which SNE (Hinton & Roweis, 2003) was proposed. Hence, we sample the data into equally-sized "microbatches" for computing the loss in (5). Each "microbatch" consists of N data samples, each sample characterized by a subword posterior sequence and subword transcription (X, Y). A fixed number of microbatches then form a minibatch (as in "minibatch training"), and the minibatch loss is the average loss of the microbatches therein. Because the utterances are extremely diverse, forming each microbatch via purely random sampling often results in all the subword sequences being different from each other and $p_{ij} = 0$ everywhere in (8). In such a case, the loss in (5) would be 0, so all gradients would be 0, and the microbatch would have no effect on training. To avoid wasting training time on defunct microbatches, for every microbatch we designate (X_0, Y_0) the "pivot," and artificially search for at least one sample (X_n, Y_n) in the training data that satisfies $Y_n = Y_0$ and insert it in the microbatch. The other samples (with a different subword transcription from Y_0) in the microbatch are chosen purely randomly. For further simplicity, we compute the loss over only a subset of i, j pairings in (5), by fixing i to 0:

$$\mathcal{L}_f = \sum_j p_{0j} \log \frac{p_{0j}}{q_{0j}}. \tag{10}$$

3.3 Phonetic confusability reflected in the $\mathbf{g}$ vectors

In the proposed method, a clear intuition exists on how the Euclidean distance between two embedding vectors $\mathbf{g}_i$ and $\mathbf{g}_j$ can directly reflect the acoustic confusability of their corresponding (purely text) subword sequences Y_i and Y_j.

Since the text encoder g directly mirrors via (9) the embeddings generated by the acoustic encoder f, the $\mathbf{g}$ vectors – generated purely from text – will mirror the knowledge learned by f on how the text actually sounds.

Consider the two vowels "ae" (as in "bat") and "eh" as in ("bet"). Since they sound similar, it is likely that their posteriors will behave similarly; if one scores high, the other will also score high. On the other hand, the consonants "j" and "s" sound distinctly different, and therefore their scores will tend to be opposite of each other; if one scores high, the other will score low, and vice versa.

Now imagine acoustic instances of "Jackson" ("j, ae, k, s, ah, n"), "Jeckson" ("j, eh, k, s, ah, n"), and "Sackson" ("s, ae, k, s, ah, n") appearing as training samples for f. Both the second and third word are only one phoneme away from "Jackson", but the phone posteriorgram sequence for "Jeckson" will be close to that of "Jackson" due to the scores for "ae" and "eh" trending similarly, so their f embeddings will also be inevitably similar. On the other hand, the posteriorgram sequence for "Sackson" will be different from that for "Jackson" because the scores for "j" and "s" tend to be mutually exclusive, so their $\mathbf{f}$ embeddings will be different.

Now consider the $\mathbf{g}$ embeddings obtained from pure text inputs "Jackson", "Jeckson", and "Sackson". As we can see in Table. 1, "Jackson" is much closer to "Jeckson" than it is to "Sackson" in the $\mathbf{g}$ embedding space, which is consistent with what actually sounds more similar. Similarly, "game of thrones" is closer to "game of drones" than "fame of thrones."

Table 1: Euclidean distance between $\mathbf{g}$ vectors for different pairs of words, computed by the text encoder g from pure text. The differences in distances are consistent with the intuitive phonetic confusability between the words.

Words	Phone sequences fed to g network	Distance between $\mathbf{g}$'s
Jackson Jeckson	j, ae, k, s, ah, n j, eh, k, s, ah, n	0.0478
Jackson Sackson	j, ae, k, s, ah, n s, ae, k, s, ah, n	1.331
game of thrones game of drones	g, ey, m, ax, f, th, r, ow, n, z g, ey, m, ax, f, d, r, ow, n, z	0.122
game of thrones fame of thrones	g, ey, m, ax, f, th, r, ow, n, z f, ey, m, ax, f, th, r, ow, n, z	1.077

3.4 THE IMPORTANCE OF NORMALIZATION

It is interesting to see what would happen if no normalization were done in (3) and (4). The scores in (8) would be trivial (1 or 0) instead of scaled ($1/c_i$ or 0) binary values, and the computation would be significantly reduced with no denominator in (4). While this simplicity may be tempting, inspection of the resulting gradient shows that it is not a good idea:

$$\frac{\partial \mathcal{L}_f}{\partial \mathbf{f}_i} = 4 \sum_j (\mathbf{f}_i - \mathbf{f}_j) p_{ij}. \tag{11}$$

Comparing this with (6) (or later with (12)), we can see that q_{ij} has disappeared entirely from the gradient, meaning the similarity in the embedding space no longer plays a role in weighting the gradient. Only the similarity in the input space (p_{ij}) influences the gradient weight, so the fundamental notion of preserving relative distances is no longer being enforced.

4 COMPARISON WITH THE TRIPLET LOSS

Some insight into how the proposed method compares with existing triplet-loss-based methods (He et al., 2017) can be found by inspecting the gradients of the two methods' loss functions used in backpropagation.

For ANE's f network, we can apply (8) to (6) to obtain the gradient of the loss function. The summation can be split into the sum over the samples with the same subword sequence as i, i.e., $J_i^+ = \{j : Y_j = Y_i\}$, and the rest of the samples, i.e., $J_i^- = \{j : Y_j \neq Y_i\}$:

$$\frac{\partial \mathcal{L}_f}{\partial \mathbf{f}_i} = 2 \sum_{j \in J_i^+} (\mathbf{f}_i - \mathbf{f}_j) \left(\frac{1}{c_i} - q_{ij} + \frac{1}{c_j} - q_{ji} \right) - 2 \sum_{j \in J_i^-} (\mathbf{f}_i - \mathbf{f}_j)(q_{ij} + q_{ji}). \tag{12}$$

Now, let us consider the triplet loss, placed in the same context as ANE. For every given triplet of posterior sequences (X_0, X_m, X_n) with a corresponding triplet of subword sequences (Y_0, Y_m, Y_n), where $Y_0 = Y_m$ and $Y_0 \neq Y_n$, we have the following loss (He et al., 2017; Settle et al., 2019; Jung et al., 2019):

$$\mathcal{L}_{\text{trip}} = \max \left\{ 0, \alpha - \text{Sim}(\mathbf{f}_0, \mathbf{g}_m) + \text{Sim}(\mathbf{f}_0, \mathbf{g}_n) \right\}, \tag{13}$$

where α is some constant and $\text{Sim}(\mathbf{f}_i, \mathbf{g}_j)$ is some pre-defined similarity function between $\mathbf{f}_i$ and $\mathbf{g}_i$. The idea is to make the embeddings attract each other if their utterances have identical subword sequences, and repel each other if the subword sequences are different.

For the purpose of this analysis, we can ignore the $\max$ operator and set $\alpha = 0$ in (13), since they are merely for data selection. Let us consider a "batch" triplet loss that is summed over all the samples in our set of N speech utterances:

$$\mathcal{L}_{\text{trip}} = \sum_{i,j} s_{ij} \text{Sim}(\mathbf{f}_i, \mathbf{g}_j), \tag{14}$$

where we have defined

$$s_{ij} = \begin{cases} -1 & \text{if } Y_i = Y_j \\ +1 & \text{else} \end{cases}. \tag{15}$$

Also, since the training tries to make $\mathbf{f}_j$ as close as possible to $\mathbf{g}_j$ for every j, we make the approximation $\mathbf{g}_j \approx \mathbf{f}_j$, resulting in

$$\mathcal{L}_{\text{trip}} = \sum_{i,j} s_{ij} q'_{ij}, \tag{16}$$

where $q'_{ij} \triangleq \text{Sim}(\mathbf{f}_i, \mathbf{f}_j)$.

A variety of possibilities exist for $\text{Sim}(\mathbf{f}_i, \mathbf{g}_j)$ in (13). If we assume a Euclidean-distance-based form similar to (4),

$$\text{Sim}(\mathbf{f}_i, \mathbf{g}_j) \triangleq \exp\left\{-||\mathbf{f}_i - \mathbf{g}_j||^2\right\}, \tag{17}$$

it can be determined that the gradient of the loss in (16) is

$$\frac{\partial \mathcal{L}_{\text{trip}}}{\partial \mathbf{f}_i} = 4 \sum_{j \in J_i^+} (\mathbf{f}_i - \mathbf{f}_j) q'_{ij} - 4 \sum_{j \in J_i^-} (\mathbf{f}_i - \mathbf{f}_j) q'_{ij}. \tag{18}$$

Comparing (18) with (12), we notice a key difference in the contribution of the samples in J_i^+ to the gradient. In the case of the triplet loss, the contribution to the gradient is amplified as the similarity q'_{ij} increases, causing more perturbation to the model parameters during backpropagation. This behavior is counterintuitive because a high q'_{ij} means the embedding for X_i and X_j are already almost the same, which is exactly what we're trying to achieve for $j \in J_i^+$, so there is no need to change them further. In contrast, in the ANE loss in (12) one can see in the summation over J^+ that as q_{ij} gets higher, we amplify the gradient less and therefore perturb the model parameters less, which is consistent with intuition.

Alternate similarity functions for (17) can also be tried. If $\text{Sim}(\mathbf{f}_i, \mathbf{g}_j) \triangleq -||\mathbf{f}_i - \mathbf{g}_j||^2$, the gradient in (18) becomes $-4\sum_j s_{ij}(\mathbf{f}_i - \mathbf{f}_j)$, and if $\text{Sim}(\mathbf{f}_i, \mathbf{g}_j) \triangleq \mathbf{f}_i^T \mathbf{g}_j / (||\mathbf{f}_i|| \cdot ||\mathbf{g}_j||)$ (cosine similarity), the gradient is $-2\sum_j s_{ij}\left\{\mathbf{f}_i q'_{ij}/||\mathbf{f}_i||^2 - \mathbf{f}_j/(||\mathbf{f}_i|| \cdot ||\mathbf{f}_j||)\right\}$, and we can make similar arguments as above that they are less effective than the proposed method.

In summary, both the triplet-loss-based method (He et al., 2017) and the proposed method ANE seem to share the same basic notion of pulling vectors closer if they represent the same word and pushing them apart if not. However, ANE has the added subtlety of pushing or pulling with more "measured strength" based on how good the embeddings currently are.

5 NAME RECOGNITION EXPERIMENTS

Three f-g encoder pairs were trained, then tested using the Euclidean nearest-neighbor matching rule in (1). ANE-p is the proposed system using monophone posteriorgram sequences for every X_n and monophone sequences for every Y_n in Section 3.1, and ANE-g is the same but using graphemes for Y_n. Trip-p uses the same inputs and outputs as ANE-p but is trained by a triplet distance per Section 4 and (17), under the intent of comparing the proposed method with existing methods (He et al., 2017) in the same setting. For all systems, both f and g were Bi-LSTM networks with 51 input dimensions (1 dimension for each monophone, including a "silence" monophone). All f networks had 2 layers and 100 nodes in each direction, and all g networks had 1 layer and 200 nodes in each direction (with the exception of ANE-p-40, which had 300 nodes), with an additional regression layer applied to the last output of every sequence to produce the embedding vector.

For training data, speech utterances of varying lengths were extracted from a large set of proprietary data, and their monophone posterior vector sequences and phonetic transcriptions as described in Section 3.1 were obtained using a Finite State Transducer(FST)-based ASR with a DNN-HMM acoustic model. The lengths of the utterances were randomly chosen so that the overall distribution of the number of phones per utterance roughly matched that of a database of music titles and person names. The process resulted in 4.2M utterances, from which 678K microbatches (160 utterances per microbatch, and 32 microbatches per minibatch) were created for training the f for ANE-p and Trip-p, and 783K microbatches for ANE-g. Prepared in a similar manner, the cross-validation data

Table 2: Results of name recognition based on nearest-neighbor search in (1) using the acoustic encoder f and the text encoder g in tandem, compared with the result from a whole-word FST recognizer. ANE-p used f and g trained by the proposed method using monophone transcriptions for each Y_n (see Section 3.1), while ANE-g used grapheme transcriptions for Y_n. Trip-p used a triplet distance with the same data as ANE-p. The numeric suffixes indicate the number of dimensions in the embedding vectors.

Method	Accuracy (%)
FST	97.5
Trip-p-20	65.5
Trip-p-30	65.1
ANE-g-20 (proposed method)	96.0
ANE-g-30 (proposed method)	96.4
ANE-p-20 (proposed method)	96.9
ANE-p-30 (proposed method)	97.3
ANE-p-40 (proposed method)	97.4

Table 3: Name recognition results for increasing phonebook size using f and g in tandem. See caption for Table 2 for description of methods. When the number of dimensions reaches 40, the Euclidean nearest-neighbor search between Acoustic Neighbor Embedding vectors (ANE-p-40) is as accurate as FST decoding.

Method	Accuracy (%) for phonebook size			
	20K	100K	500K	1M
FST	88.1	84.4	76.8	72.1
ANE-g-20	79.6	75.6	67.4	62.1
ANE-g-30	81.8	78.6	71.3	66.3
ANE-p-20	86.5	81.7	73.1	68.2
ANE-p-30	88.1	84.1	76.4	71.9
ANE-p-40	88.2	84.4	77.0	72.7

(used to stop training) consisted of 1.4M utterances, organized into 130K microbatches for ANE-p and Trip-p, and 141K microbatches for ANE-g.

For the isolated name recognition experiment, 19,646 audio utterances of spoken names were prepared as evaluation data. Each utterance had a corresponding list of possible names (i.e., the speaker's "phonebook") of varying size, with an average 1,055 names per phonebook. The phonebook was used to build a whole word FST recognizer for every utterance, using pronunciations obtained from G2P. The same list of pronunciations was used to generate the $\mathbf{g}$ embeddings. The FST recognizer was a subword-to-word transducer designed to directly consume subword posterior sequences instead of audio signals, to ensure that the same inputs were used for both the FST and the f networks. A single acoustic model was used to generate all training and testing inputs for the f networks, as well as the inputs to the FST wherever needed.

Table 2 shows the accuracy of each method in the name recognition task, for varying dimensions in the embedding vectors. For ANE-p and Trip-p, the result of the rule (1) was a best-matching phone sequence, and the match was deemed correct when it was an exact match with any pronunciation in the phonebook for the reference (manually-transcribed) name. For ANE-g, a crude normalization was first done on all names (removing special characters and converting to lowercase) for both training and testing data. The result of the rule in (1) was a best-matching normalized name, and the match was deemed correct when it was an exact match with the normalized reference name. ANE-p was more accurate than ANE-g because ambiguities in word pronunciation were better resolved. With the triplet distance, competitive accuracy could not be achieved in word recognition.

Table 4: Approximate phonetic match accuracy for isolated names using g. The FST performs poorly because it uses only a general vocabulary where many of the phonebook names are missing. For ANE-p and Trip-p, the g network was used to transform the FST's 1-best phone sequence into an embedding vector, then a nearest neighbor search was done on the actual phonebook names. See caption for Table 2 for further details on the methods shown.

Method	Accuracy (%)
FST using a default vocabulary	67.8
Trip-p-20	86.3
Trip-p-30	85.4
ANE-p-20 (proposed method)	93.5
ANE-p-30 (proposed method)	93.9

In a second experiment, the accuracy was measured for increasing phonebook size. For simplicity, a unified phonebook was used for all test samples, where the list contains all the reference names and is padded with other random names to attain the sizes in Table 3. There is increasing confusion between the names as the list gets larger, especially because most of the names are short (e.g. "Lian" and "Layan"), and the accuracy gap between FST and ANE also widens.

As seen in both Tables 2 and 3, the accuracy of ANE-p increases as the number of dimensions increases because less discriminative information is discarded. For every utterance, the average number of dimensions in the original input used by the FST is 3,662 (51 $\times$ the length of the utterance, where the mean length is 71.8 frames). This is reduced to 20, 30, or 40 in ANE-p in Table 3, so the dimension reduction is significant. However, the name recognition accuracy of ANE-p is essentially identical to that of the FST when the number of embeddings is 40.

The third experiment uses the phonebooks in the first experiment to perform recognition via approximate phonetic match using only the g networks. First, a general large-vocabulary (900K) continuous ASR was run. Since many of the spoken names did not exist in the vocabulary, accuracy is low for the FST in Table 4. Next, from the FST's 1-best phone sequence, a $\mathbf{g}$ vector was computed, followed by a nearest neighbor search over the $\mathbf{g}$ vectors of the corresponding phonebook. Accuracy was computed based on how often the correct name was chosen, and ANE-p gave the best result.

6 Conclusion and Discussion

This paper has proposed a method of adapting stochastic neighbor embedding (SNE) to sequential data so that words of arbitrary length can be represented as fixed-dimension vectors, called Acoustic Neighbor Embeddings, where the acoustic confusability between words is represented by the Euclidean distance between the vectors. The gradients of the loss function was inspected to show that this neural network training method can be more effective than a known method based on the triplet distance, and the efficacy of ANE in isolated name recognition and phonetic matching experiments was demonstrated.

It is possible that creating only one embedding for each phone sequence does not sufficiently capture all the acoustic variabilities in its pronunciation. SNE (Hinton & Roweis, 2003) already has extensions that allow multiple embeddings per input in the dimension-reduced space, which we may be able to leverage in future work to make more robust matches.

References

Samy Bengio and Georg Heigold. Word embeddings for speech recognition. In *Proceedings of the 15th Conference of the International Speech Communication Association, Interspeech*, 2014.

G. Chen, C. Parada, and T. N. Sainath. Query-by-example keyword spotting using long short-term memory networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5236–5240, 2015.

V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–6, 2008.

T. J. Hazen, W. Shen, and C. White. Query-by-example spoken term detection using phonetic posteriorgram templates. In *2009 IEEE Workshop on Automatic Speech Recognition Understanding*, pp. 421–426, 2009.

Wanjia He, Weiran Wang, and Karen Livescu. Multi-view recurrent neural acoustic word embeddings. In *International Conference on Learning Representations (ICLR)*, 2017.

Geoffrey E Hinton and Sam T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer (eds.), *Advances in Neural Information Processing Systems 15*, pp. 857–864. MIT Press, 2003.

M. Jung, H. Lim, J. Goo, Y. Jung, and H. Kim. Additional shared decoder on Siamese multi-view encoders for learning acoustic word embeddings. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 629–636, 2019.

K. Levin, K. Henry, A. Jansen, and K. Livescu. Fixed-dimensional acoustic embeddings of variable-length segments in low-resource settings. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 410–415, 2013.

Hugh Christopher Longuet-Higgins. The non-local storage of temporal information. In *Proceedings of the Royal Society of London B*, 1968.

Andrew L. Maas, Stephen D. Miller, Tyler M. O'Neil, Andrew Y. Ng, and Patrick Nguyen. Word-level acoustic modeling with convolutional vector regression. In *ICML 2012 Representation Learning Workshop*, 2012.

F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, June 2015. doi: 10.1109/CVPR.2015.7298682.

S. Settle, K. Levin, H. Kamper, and K. Livescu. Query-by-example search with discriminative neural acoustic word embeddings. In *INTERSPEECH 2017*, 2017.

S. Settle, K. Audhkhasi, K. Livescu, and M. Picheny. Acoustically grounded word embeddings for improved acoustics-to-word speech recognition. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5641–5645, 2019.