Learning Speech Representations with Uncertainties

Siqi Sun and **Nicholas Sanders** and **Wenyu Huang** and **Oli (Danyi) Liu**

Abstract

The Autoregressive Predictive Coding model (Chung et al., 2019)learns speech representations by predicting the acoustic feature at a future frame given past frames. However, the model can only make predictions in the form of a unimodal distribution in the acoustic feature space, which could not fit the target distribution, which is multimodal by the nature of human speech. We proposed three adaptations to address this shortcoming, among which the mixture density model proved to outperform the baseline APC in the downstream phoneme probing task, achieving a reduction of 1.2% in phoneme error rate.

1 Introduction

Representations containing higher level information are desirable for most machine learning tasks, due to real world data often being noisy and difficult to process for machine learning. In the context of speech processing, speech representation learning aims to learn the transformation of surface level features, like speech waveforms or log-mel spectrograms as used in this study and seen in Figure 1, into higher level representations of prosodic, phonetic, and speaker characteristics. Automatically learning feature representations, as opposed to manual feature engineering, is very convenient.

One strategy is supervised learning, where the objective is to learn representations from scratch for a specific task (e.g., automatic speech recognition, speaker identification, etc.), using a set of labeled training data. However, a weakness of representation learning in supervised learning (e.g., automatic speech recognition, speaker classification, etc.) is the requirement of large amounts labeled data to learn the feature representations and achieve good performance on diverse test data. Additionally, representations learned through supervised learning are typically very specialized and can only used for the same or similar tasks. Conversely, unsupervised methods like clustering algorithms learn feature representations from unlabeled data; however, often cannot be used for supervised learning tasks.

A viable solution that is similar to unsupervised learning, is self-supervised learning. In self-supervised learning, representations can be learned by first training on unlabeled data for a more general task, such as language modeling in NLP (i.e., predicting words or subwords) and then later used as initialization or preprocessing for more specific supervised learning tasks, a process called transfer learning. This strategy of using self-supervised learning as a pretraining for use in transfer learning has seen great success, as more general representations are more useful for a larger amount of downstream supervised learning tasks.

Two approaches to self-supervised learning are autoencoders like BERT (Devlin et al., 2019) and autoregressive models like GPT (Radford et al., 2018). The autoencoder approach to representation learning typically is to train a model to encode data into generalized representations and then decode said representations into reconstructions of the original input data, as seen in variational autoencoders (Kingma and Welling, 2014). BERT is a masked language modeling technique, similar to a denoising autoencoder (Vincent et al., 2010), which during training encodes corrupted inputs and decodes learned representations into non-corrupted input data. Masking is an extension of corrupting inputs, where certain parts of the input are obscured or masked and the unmasked input is used as the target for prediction. On the other hand, the autoregressive approach to representation learning is to treat inputs sequentially, where current time step inputs are encoded into representations that are then used to predict future time step inputs. Therefore, autoregressive approaches can be viewed as directly modeling the conditional probabilities of an input sequence, whereas autoencoders can be viewed as directly modeling the joint probability of an input sequence.

One effective self-supervised, speech representation learning method is to autoregressively predict future frames of surface-level acoustic features (i.e., 25 millisecond segments of 40 dimensional log-mel spectrograms), by encoding previous frames, known as autoregressive predictive coding (APC)

(Chung et al., 2019). APC achieves better performance than surface level features and other self-supervised methods on a diverse array of downstream, supervised learning tasks and is a self-supervised learning objective that is able to learn general high-level representations.

Still, APC is limited by its ultimate objective, which is to minimize the mean squared error, known as L2 loss or distance, between a predicted frame from a linear predictor and the ground truth. This objective only enables the model to learn the mean of the training data (Bishop, 1994), which is limiting because speech varies vastly even for the same speakers and phonemes. Therefore, we hypothesize that the original APC learns a unimodal distribution, but the true output space is multimodal and that the uncertainty can be modelled with a mixture of means and variances.

Our method expands on the original APC model by examining the effect of different objective functions used during pretraining on the quality of the information in learned hidden representations; which, are demonstrable through model performance on downstream tasks. To examine this effect, we test two methods, piecewise regression (i.e., a mixture of many means) and mixture density networks (i.e., a mixture of Gaussians). We also propose a method called target-quantized APC, where we cluster the surface level features using k-means and change the objective to classification; which, in the context of APC, the future frame clusters are autoregressively predicted instead of the frames themselves. We analyze how these methods are affected by increasing the amount of components in the mixtures and observe that mixture density network achieves the best performance on downstream tasks, improving as the number of components increases.

2 Background

2.1 Model Architecture

Inspired by language models for text, the Autoregressive Predictive Coding (APC) model aims to predict the acoustic features at a future frame x_{t+k} given features in the history $(x_1, x_2, ..., x_t)$. The APC model also resembles most autoregressive language models in using an RNN architecture to perform sequence modeling. The key difference between language models and APC, as well as other models for speech, is that the units being modeled are discrete tokens and continuous acoustic features respectively. Hence APC takes the acoustic features as direct input without applying an embedding layer that maps discrete tokens to vectors. It also replaces the softmax layer used in language models that yields a discrete probability distribution of the target token in with a regression layer for reconstructing the future frame.

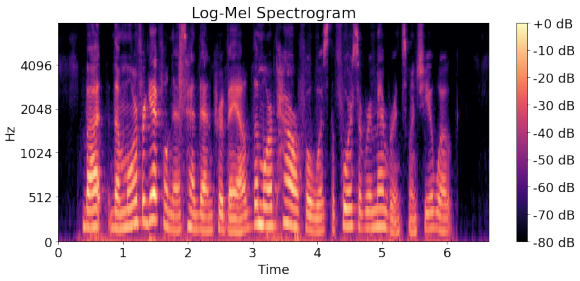


Figure 1: An example log-mel spectrogram (y-axis is in seconds) from the Librispeech dataset. All spectrograms were generated with a sampling rate of 16,000 samples per second, fast Fourier transform frame sizes of 400 samples or 25 milliseconds, with 160 frames or 10 milliseconds between successive frames, and with 40 Mel bands.

Figure 2 shows the overall architecture of APC. The architecture of APC can be divided to two parts. The **encoder** consists of a multi-layered RNN that takes in acoustic features of past frames, in which the hidden representations of the final layer are taken to be the representations extracted by APC. The encoder is followed by the **predictor**, which is a linear mapping from the representations to be extracted back to the acoustic feature space in the original APC. The encoder and the predictor are simultaneously optimised as a whole during training, but only the encoder is used to generate representations for speech features in deployment.

Formally, we denote the surface acoustic feature at time step t as x_t. The representations corresponding to time step t is then:

$$h_t = \text{RNN}(x_1, x_2, ..., x_t; \theta_{rnn}) \quad (1)$$

The prediction for the target frame at k time steps away is a linear transformation of the representation, parameterised by a matrix W.

$$\hat{y}_t = W h_t \quad (2)$$

θ_{rnn} and W are optimised to minimise the L2 distance between the predicted and observed features of the target frames. When trained on an input

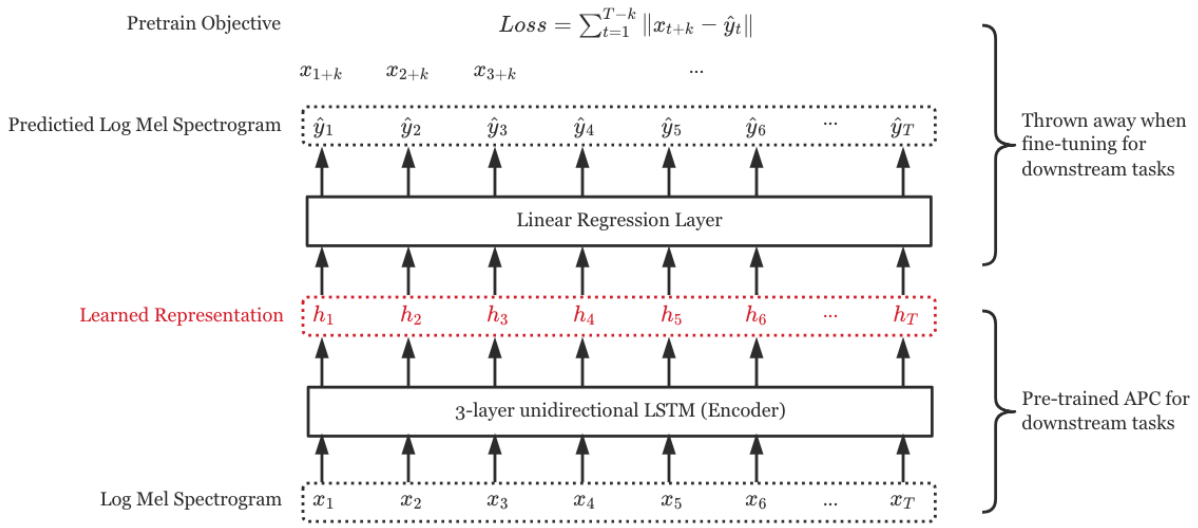


Figure 2: Overall Architecture of APC.

sequence $(x_1, x_2, ..., x_T)$, the loss is given by:

$$\mathcal{L} = \sum_{t=1}^{T-k} ||x_{t+k} - \hat{y}_t||_2 \quad (3)$$

2.2 Limitation

The distribution of the acoustic features at a future time step can be multimodal since a phone could be followed by many different phones and also people pronounce the same phone in many different ways. This reveals a critical limitation of the predictor module in APC since neural networks are known to approximate conditional mean of the output (Bishop, 1994). When the output conforms to a multimodal distribution, the conditional mean may be close to none of the existing modes. There are two additional inflexibilities with using a linear mapping as the predictor and optimising it with L2 loss. While optimising for L2 loss is equivalent to finding a isotropic Gaussian distribution that best approximates the output distribution, it only searches for the optimal mean, leaving out the variances and co-variances. Moreover, it models the mean in an implicit manner, via a linear mapping from the representation space to the acoustic feature space.

3 Proposed Approaches

To address the predictor module's failure in working with multimodal output distribution, we propose a piecewise regression predictor where multiple linear mappings are used in the hope that each will specialise in predicting one of the modes. However, there remains the problem of the variance parameters being neglected and the means being learnt implicitly. We thus applied mixture density networks to explicitly fit the output space with multimodal Gaussian distributions parameterised with both means and variances. Finally, we also proposed a target-quantized APC model that uses explicit clustering to induce clusters before training an APC model, where the predictor would predict cluster ID of the future frame.

3.1 Mixture Density Network

A Mixture Density Network (MDN) aims to model a Probabilistic Density Function (PDF) of a mixture distribution through a neural network. Instead of directly outputting the feature vectors as in the original APC, the neural network is used to predict the distributional parameters of the PDF (Zen and Senior, 2014).

Depending on whether the random variables we model are discrete or continuous, we should choose an appropriate mixture distribution. Since we are modelling the continuous speech feature vectors, we choose a continuous mixture distribution here. One of the most widely used continuous mixture distributions is the Gaussian Mixture Model (GMM).

For each output channel $y^c, c \in [1, C]$, where C is the total number of channels, a GMM-based MDN $\mathcal{M}$ first maps an input feature vector $\boldsymbol{x}$ into a set of GMM parameters, including the weights $\boldsymbol{w^c} = [w_1^c, \ldots, w_M^c]$, means $\boldsymbol{\mu^c} = [\mu_1^c, \ldots, \mu_M^c]$ and variances $\boldsymbol{v^c} = [v_1^c, \ldots, v_M^c]$, where M is the total number of components. Then, a GMM PDF can be written using these parameters, as shown in

Eq. 4.

$$p(y^c|\boldsymbol{x}, \mathcal{M}) = \sum_{m=1}^{M} w_m^c(\boldsymbol{x}) \cdot \mathcal{N}(y^c; \mu_m^c(\boldsymbol{x}), v_m^c(\boldsymbol{x})) \tag{4}$$

, where $w_m^c(\boldsymbol{x})$, $\mu_m^c(\boldsymbol{x})$ and $v_m^c(\boldsymbol{x})$ correspond to the weight, the mean and the variance of the mth GMM component modelling channel c, conditioned on $\boldsymbol{x}$.

The neural network doesn't predict these distributional parameters directly, because there are some mathematical constraints on these parameters. Instead, the neural network predicts the unconstrained version of these parameters using Eq. 5:

$$\hat{\boldsymbol{w}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{v}} = \boldsymbol{W}_p \text{RNN}(\boldsymbol{x}) + \boldsymbol{b}_p \tag{5}$$

, where $\boldsymbol{W}_p \in \mathbb{R}^{n \times 3CM}$ and $\boldsymbol{b}_p \in \mathbb{R}^{3CM}$ are the linear transformation parameters of the linear layer. $\hat{\boldsymbol{w}}$ is the concatenation of $[\hat{\boldsymbol{w}}^1, \cdots, \hat{\boldsymbol{w}}^C]$, $\hat{\boldsymbol{\mu}}$ is the concatenation of $[\hat{\boldsymbol{\mu}}^1, \cdots, \hat{\boldsymbol{\mu}}^C]$ and $\hat{\boldsymbol{v}}$ is the concatenation of $[\hat{\boldsymbol{v}}^1, \cdots, \hat{\boldsymbol{v}}^C]$. Then these unconstrained parameters are transferred to the constrained ones using the following transformations:

$$w_m^c = \frac{\exp(\hat{w}_m^c)}{\sum_{l=1}^{M} \exp(\hat{w}_l^c)} \tag{6}$$

$$\mu_m^c = \hat{\mu}_m^c \tag{7}$$

$$v_m^c = \text{Softplus}(\hat{v}_m^c) \tag{8}$$

, in which Softmax is used to normalize the weights and Softplus is used to ensure that the variance is always positive. However, other transformations are also possible, which we haven't explored in our experiments.

In order to train the MDN $\mathcal{M}$, we use maximum likelihood estimation, given the training data, as shown in Eq. 9.

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M}} \sum_{n=1}^{N} \sum_{t=1}^{T} \sum_{c=1}^{C} \log p(y_t^{c,(n)} | \boldsymbol{x}_{\boldsymbol{t}}^{(\boldsymbol{n})}, \mathcal{M}) \tag{9}$$

, where n is the index of the training utterance, t is the timestep of the frame within the training utterance and c is the output channel. Equivalently, we can minimize the Negative Log Likelihood (NLL) loss to train $\mathcal{M}$.

Shared-weight MDN Besides the general form of the MDN mentioned above, an alternative form of the MDN is to share the weight of a GMM component (i.e., w_m) among all the output channels. In this way, the total number of weights predicted by the neural network is M, instead of $C \times M$.

We implement both the non-shared-weight MDN and the shared-weight MDN in our experiments, and compare their performance in Section 4.2.2.

3.2 Piecewise Regression

Also known as segmented regression, piecewise regression is a regression method that partitions the input into intervals and fits one linear segment to each interval. With an increasing number of partitions, the piecewise regressor provides a closer approximation to the underlying nonlinear function.

In the context of the APC model, incorporating piecewise regression into the predictor module can be viewed as having multiple independent components of predictors all of which are linear transformations. During training, only the component that gives the best prediction, i.e. the smallest L2 loss, is activated and its gradient back-propagated.

When there are M components, the candidate predictions are

$$\hat{\mathbf{y}}_{\mathbf{t}} = [W_1 h_t, W_2 h_t, ..., W_M h_t] \tag{10}$$

And the loss would be computed as

$$\mathcal{L} = \sum_{t=1}^{T-k} \min_i ||y_t - (\hat{\mathbf{y}}_{\mathbf{t}})_i||_2 \tag{11}$$

3.3 Target-quantized APC

Instead of predicting speech surface feature vector, HuBERT (Hsu et al., 2021) tried to predict the speech frame cluster for masked frame as the pretrain objective. Inspired by their positive result, we also tried to build our HuBERT-like, target-quantized APC model. Given the previous speech surface feature $\boldsymbol{x}$, we predict the future frame cluster $l \in [1, L]$ by adding an extra classification head on top of the APC features using Eq.12

$$\hat{\boldsymbol{l}} = \text{Softmax}(\boldsymbol{W}_l \text{RNN}(\boldsymbol{x}) + \boldsymbol{b}_l) \tag{12}$$

, where $\hat{\boldsymbol{l}}$ is the predicted probability of each clusters, $\boldsymbol{W}_l \in \mathbb{R}^{d \times L}$ and $\boldsymbol{b}_l \in \mathbb{R}^L$ are the linear transformation parameters of the cluster classification head. For training the target-quantized APC, we

calculate the cross entropy of the predicted probability and the true label pre-clustered with k-means. For comparing the target-quantized APC with the vanilla APC, we also record the L2 loss of the target-quantized APC. When pre-clustering frame surface features with k-means, we got the centroid embeddings matrix $\boldsymbol{C} \in \mathbb{R}^{L \times C}$ for each cluster, where C is the dimension of surface feature vectors (equals to the total channel numbers in MDN). Then we are able to produce predicted feature vectors by Eq.13.

$$\boldsymbol{x'} = \hat{\boldsymbol{l}}\boldsymbol{C} \tag{13}$$

By applying the same L2 loss in vanilla APC, we could get the comparable L2 loss of target-quantized APC. Besides, $\boldsymbol{x'}$ can actually be seen as the predicted mel-spectrograms by a simplified two-step MDN. The predicted logits $\hat{\boldsymbol{l}}$ are actually equal to the weights in the shared-weight MDN described in Section 3.1. Here each cluster could refer to a specific Gaussian component with a mean for the centroid embeddings of the cluster and variance $\mathbf{0}$. The main difference between target-quantized APC and shared-weight MDN is that the means and variances of different Gaussian components are now fixed. In short, target-quantized APC can be seen as an ablation study of shared-weight MDN, which remove the Gaussian uncertainty assumption to have a fixed centroid for each Gaussian components in MDN. However, this ablation study is not very rigorous since target-quantized APC is not trained by the L2 loss. So we just take it for reference.

4 Experiments

In this section, we first discuss the experimental setups, and then we demonstrate the effectiveness of the learned representations from various proposed models. First, we reproduce the result of the original APC, which is used as our baseline model, following the configuration in (Chung et al., 2019). Then, we show the results of various models proposed in Section 3, including MDN, Piecewise Regression and Hubert-like APC. By using the phoneme classification as the probing task just as in (Chung et al., 2019), we can examine how much phonetic information has been successfully captured by the learned representation.

4.1 Experimental Setups

4.1.1 Self-supervised training

For the baseline model and the other proposed models, we use the 360-hour clean subset of LibriSpeech (Panayotov et al., 2015) corpus as our training set for self-supervised training. 40-dimensional log Mel-spectrograms are extracted every 10ms using a 25ms window as the feature vectors, served as the input features $\boldsymbol{x} \in \mathbb{R}^{40}$ and the training target $\boldsymbol{y} \in \mathbb{R}^{40}$, which are shifted by 3 timesteps into the future. All the models are trained for 10 epochs using the Adam optimizer (Kingma and Ba, 2014) with a batch size of 1 and an initial learning rate of 10^{-4}. Note that the feature dimension, the batch size and the initial learning rate are slightly different from those in (Chung et al., 2019), but the differences aren't significant according to one of the authors of (Chung et al., 2019).

4.1.2 Probing task

Phoneme classification is used as a probing task in our experiments to verify that the learned representation has successfully captured the phonetic information present in the speech. Other probing tasks such as speaker verification (Chung et al., 2019) and speaker classification (Chung et al., 2020) are also possible. However, due to time constraints, we only restrict our analysis to the phoneme classification.

The phoneme classification is conducted on the Wall Street Journal (WSJ) (Paul and Baker, 1992) corpus. In this task, 40-dimensional log Mel-spectrogram are used as the input features, whereas the phoneme identity of each frame (marked every 10ms) is used as the training target. Taking the extracted features as input, a logistic regression layer is trained with cross entropy loss to correctly classify each frame into one of the 42 phonemes. All self-supervised models are kept frozen during the training of the classification layer.

We follow the standard split of WSJ as in (Chung et al., 2020), in which 90% of `si284` are used for training, while the remaining 10% are used for validation. For each self-supervised model, the classification layer is trained for 10 epochs using the Adam optimizer with a batch size of 1 and an initial learning rate of 10^{-4}. Phoneme Error Rate (PER) is used as the evaluation metric in the phoneme classification task.

4.2 Results

4.2.1 Baseline APC

In spite of the experimental setup specified in Section 4.1, we largely follow the configuration specified in (Chung et al., 2019). A 3-layer unidirectional LSTM model is used for feature extrac-

Proposed Model	Num. Component	PER (%)
Surface features	-	53.80
Baseline APC	-	25.20
MDN (shared)	128	24.80
	256	24.50
	512	24.30
MDN (non-shared)	128	24.40
	256	24.30
	512	**24.00**
Piecewise Regression	2	25.24
	64	31.13
	128	31.56
Target-Quantized APC	100	42.45

Table 1: The Phoneme Error Rate (PER) of the 10th epoch on the validation set. The best result is shown in bold. The Num. Component for Target-Quantized APC represents cluster numbers.

tion (with each layer of hidden size $= 512$), and the hidden vectors of the last layer are used as the hidden representations for the probing task. Note that instead of using L1 loss as the training objective as in (Chung et al., 2019), we use L2 loss in our baseline, since minimizing an L2 loss is equivalent to minimizing a NLL loss, which may explain why our baseline PER is better than the PER reported in (Chung et al., 2019).

The validation PER of the baseline APC is shown in the second line in Table 1. The baseline APC achieves a much better PER than a classification layer directly trained on the surface features (here log Mel-spec) does, as shown in the first line. We can see that there is a huge reduction in PER from 0.538 to 0.252, which meets our expectation.

A confusion matrix visualizing the correspondence between the actual phoneme and the phoneme predicted by the baseline APC is shown in Appendix A. Note that the confusion matrix shows an almost diagonal pattern, which is desirable. Nevertheless, APC doesn't perform well on `spn` (spoken noise) and `nsn` (non-spoken noise), despite that they are very rare.

Appendix A also shows the confusion matrix associated with the surface features (log Mel spectrograms). Compared to the surface features, APC shows a much clearer diagonal pattern, showing that the baseline APC improves by a lot compared to the surface features, which also agrees with the gap between the PERs shown in Table 1.

4.2.2 Mixture Density Network

As discussed in Section 3.1, there are two alternative forms of MDN, which are the non-shared-weight MDN and the shared-weight MDN. We implement both forms and compare their performances in this section.

Like the baseline APC, a 3-layer uni-directional LSTM model is used for feature extraction in both forms. For a fair comparison, we keep all the model hyper-parameters identical to the baseline APC. Nevertheless, compared to the baseline APC, the postnet in MDN has much more training parameters, which is of size $\mathbb{R}^{512\times 3CM} = \mathbb{R}^{512\times 120M}$ (non-shared-weight) or $\mathbb{R}^{512\times(2C+1)M} = \mathbb{R}^{512\times 81M}$ (shared-weight), where M is the number of GMM components. Although the MDN postnet has much more training parameters than the baseline APC does, remember that the postnet is only used during self-supervised training. The postnet is never used in extracting the features, meaning that the time overhead for feature extraction in MDN is the same as in the baseline APC.

The validation PERs of both MDN forms are listed in Table 1. For each form, we vary M to compare PER against the number of GMM components. As we can see, both MDN forms are better than the baseline APC, showing the effectiveness of using MDN as our self-supervised model. Moreover, the non-shared-weight MDN is better than the shared-weight version, which meets our expectation, since more training parameters provide larger modelling capacity. Lastly, it is clear that with the increase of GMM components, the PER becomes lower, showing the effectiveness of multi-modal modelling.

Appendix A also shows the confusion matrix of the non-shared-weight MDN and the matrix of the shared-weight MDN. The matrices look very similar. In addition, there isn't a significant difference between the MDN matrix and the baseline APC matrix, which agrees with the gap between the PERs shown in Table 1.

4.2.3 Piecewise Regression

In the initial implementation of piecewise regression, every frame in the same utterance share the same component of the predictor, which was chosen to be the one that gave the lowest average of L2 loss for the utterance. Since each utterance was produced by a single speaker, we expect each component to specialise in modeling a type of cer-

tain speaker. However, we noted that there was always one component that dominated the selection process and then the model would revert to the baseline APC. Note that this does not imply that the components could not be trained to specialise in speakers. Since each batch in training time was a single utterance with thousands of frames, we observed that the component selected for the first few training utterance was trained to outperform other components after seeing a few utterances.

Thus we use component selection by frame as the standard setting for piecewise regression, and trained models with the number of components of 2, 64, and 128. As shown in Table 1, APC adapted with piecewise regression underperformed the baseline APC and the PER increased with the number of components used.

4.2.4 Target-quantized APC

Our target-quantized APC is built on top of feature extraction LSTM layers of the baseline APC model described in section 4.2.1. Instead of having a linear regression layer to produce 40-dimensional log Mel-spectrograms predictions, we predict the cluster of future frame shifted by 3 timesteps into the future. We follow the model construction described in section 3.3, and set the cluster number L to 100. The model is trained with cross entropy loss, and we report both training loss and mean squared loss for comparing with vanilla APC. We train the target-quantized APC for 10 epochs following the training settings described in section 4.1. We get the converted L2 loss 0.344 in the final epoch, which is far more larger than 0.153 in the baseline APC. It it not surprised since we are now predicting quantized clusters. The validation PER of target-quantized APC is shown in Table 1. In the result, we can find that even the baseline APC outperforms target-quantized APC, which is also consistent to the large margin increasing in the loss. These observations, in a sense, indicate that the Gaussian uncertainty assumption in MDN is necessary.

4.3 Discussion

An interesting behavior to observe is how the training process proceeds for different models, which reveals some important properties behind the scenes. In Figure 3, the learning curves for the baseline APC and the shared-weight MDN ($M = 512$) are shown for 10 epochs. Note that the MSE loss is used as the training objective for the baseline APC.

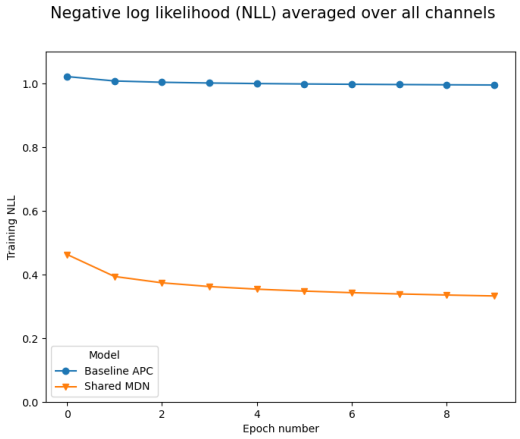


Figure 3: The training curve of baseline APC and MDN.

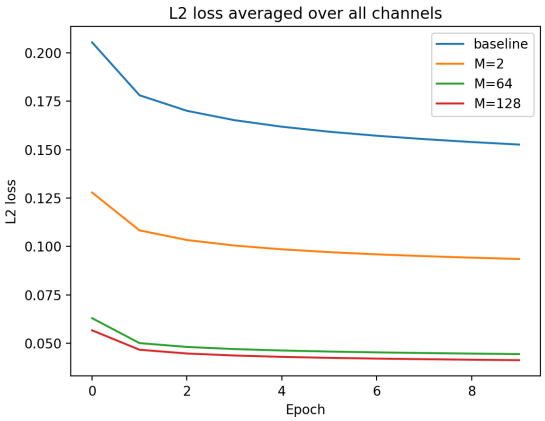


Figure 4: The training curve of APC adapted with piecewise regression using $M = 2, 64, 128$ components

For a direct comparison, the MSE loss is first converted to the NLL loss, which is the training objective for the MDN models.

As shown in Figure 3, both learning curves drop rapidly at the first few epochs and then converge after around the 4th epoch. This confirms that choosing the 10th epoch for feature extraction after the self-supervised training is a reasonable choice. Also, note that the training NLL loss for the shared-weight MDN is much lower than that for the baseline APC, showing the effectiveness of multi-modal modelling in MDN.

APC adapted with piecewise regression also achieved significantly lower L2 training losses, that is less than a third of the baseline. However, the PER obtained in the downstream probing task appear to correlate negatively with the training loss. This raises the question that whether a more expressive predictor module can indeed lead to better representations in the encoder. One possible ex-

planation for the observed negative correlation is that, the predictor is doing more heavy-lifting since it has become more powerful than before. This hypothesis can be evaluated by predicting further away in the future. As the task gets harder, the advantage of having a stronger predictor would be expected to diminish, resulting in a smaller difference in training loss. Further analysis on the correlation between predictor component and phoneme identity is in progress since we observed that each component was activated almost uniformly across $M = 2, 64, 128$.

5 Related Work

MDN applications in speech Previous works have shown Gaussian mixture models, or mixture density networks, perform well in the speech domain, mainly in speech synthesis. Zen and Senior (2014) introduced a Deep Mixture Density Network (DMDN) approach for speech synthesis and provides a speech synthesis framework. The inputs of the DMDN are the linguistic features converted from text. The linguistic features are then propagated through Deep Neural Networks (DNNs) for generating parameters of each Gaussian component. Then, a likelihood-based speech parameter generation algorithm is applied on top of the predicted GMMs to generate speech. Followed DMDN, Wang et al. (2017) later introduced an autoregressive recurrent mixture density network (AR-RMDN) for speech synthesis, which is based on Recurrent Mixture Density Networks (RMDN) (Schuster, 1999). Instead of using DNNs, RMDN applies Recurrent Neural Networks (RNNs) to generate Gaussian parameters. Compared to RMDN, Wang et al. (2017) further hypothesize that the mean of the future target feature distribution is dependent on the observed features of previous steps, which is also the main hypothesis in our works. The benefit of this hypothesis is that RMDN is be able to have an autoregressive training objective, and accomplish better performance in generation tasks. The main difference of our MDN-based approach to the related works is that we are modeling speech features to hidden representations, while speech synthesis works tries to model linguistic features from text to speech features.

Self-supervised learning with clusters As the inspiration of our target-quantized APC, HuBERT (Hsu et al., 2021) is a novel self-supervised speech representation learning model. Like what its name implies, HuBERT is a BERT-like model which pretrains the model with masked language model. By masking out some of the frames in the speech sequence, HuBERT is required to reconstruct such masks. Since the speech feature is hardly to be masked or reconstructed, HuBERT quantizes the features into predetermined clusters produced by k-means. So given the masked continuous-value speech features, HuBERT is required to predict the cluster of the masked frames. Our approach is somewhat similar to HuBERT, but applying the regime to an autoregressive setting. Since autoregressive learning does not require such discrete labels in training, we are more likely to quantize the target features in the baseline APC.

6 Conclusion

The self-supervised APC model extracts speech representations that achieve good performance on probing tasks for phoneme and speaker identification. However, the design of the model did not account for the fact that the distribution of speech features is multimodal. We propose to adapt the original APC model by incorporating piecewise regression and mixture density network. Apart from improving the expressiveness of the predictor module in APC, we propose an additional approach where we change the predicting objective of the predictor module in APC from acoustic features of the future frame to the cluster it belongs to in the multimodal distribution. Among the three propsed adaptations, MDN yielded the best performance in the downstream task for probing phoneme identity, outperforming the baseline APC across three choice of component numbers by as much as 1.2% in PER. APC adapted with piecewise regression displayed an opposite trend in PER value as the number of component increased. While we have confirmed that all of the components have been activated during training and there are no significantly dominating branches for both MDN and piecewise regression, further investigation is under way to explore potential correlations between components and phonemes.

References

Christopher M. Bishop. 1994. Mixture density networks. Technical report, Aston University.

Yu-An Chung, Wei-Ning Hsu, Hao Tang, and James Glass. 2019. An Unsupervised Autoregressive Model

for Speech Representation Learning. In *Proc. Interspeech 2019*, pages 146–150.

Yu-An Chung, Hao Tang, and James Glass. 2020. Vector-Quantized Autoregressive Predictive Coding. In *Proc. Interspeech 2020*, pages 3760–3764.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE ACM Trans. Audio Speech Lang. Process.*, 29:3451–3460.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An ASR Corpus Based on Public Domain Audio Books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210.

Douglas B. Paul and Janet M. Baker. 1992. The design for the wall street journal-based csr corpus. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, page 357–362, USA. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Technical report, OpenAI.

Mike Schuster. 1999. Better generative models for sequential data problems: Bidirectional recurrent mixture density networks. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 589–595. The MIT Press.

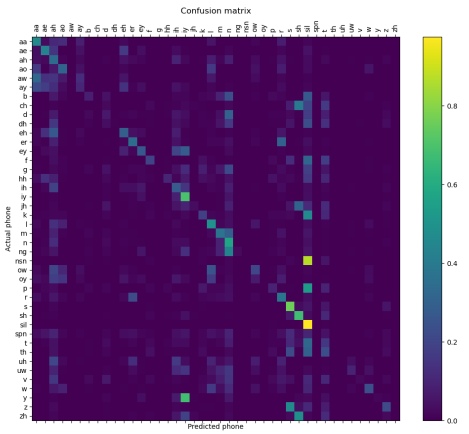
Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408.

Xin Wang, Shinji Takaki, and Junichi Yamagishi. 2017. An autoregressive recurrent mixture density network for parametric speech synthesis. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pages 4895–4899. IEEE.

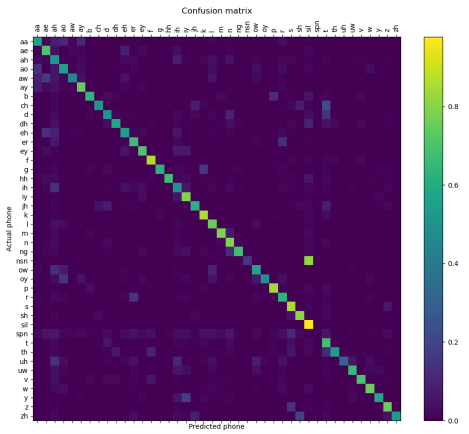
Heiga Zen and Andrew Senior. 2014. Deep Mixture Density Networks for Acoustic Modeling in Statistical Parametric Speech Synthesis. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3844–3848.

A Appendix

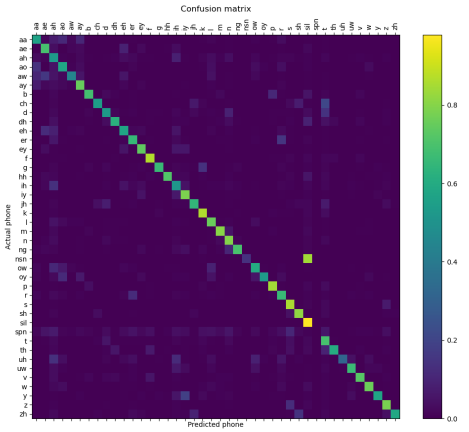
The confusion matrices visualizing the correspondence between the actual phoneme and the phoneme predicted by the various models are shown in Figure 5. The vertical axis corresponds to the actual phonemes, whereas the horizontal axis corresponds to the predicted phonemes. Note that all rows are normalized so that the elements in each row sum to 1.0, i.e., the element of position (i, j) in the confusion matrix gives $Pr(i, j) = Pr(\hat{p}_j|p_i)$, which is the probability of predicting phoneme $\hat{p}_j$ given the actual phoneme is p_i.



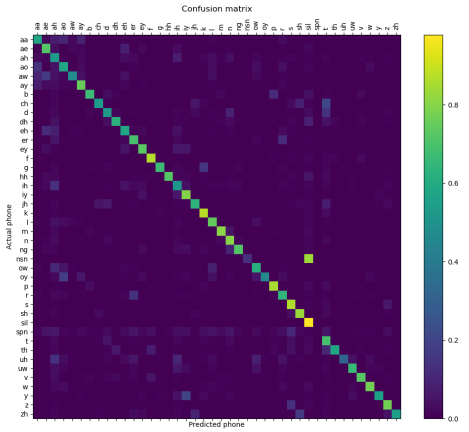
(a) The surface features



(b) The baseline APC



(c) The shared-weight MDN



(d) The non-shared-weight MDN

Figure 5: The confusion matrix between the actual phonemes and the predicted phonemes. The vertical axis corresponds to the actual phonemes, whereas the horizontal axis corresponds to the predicted phonemes. The element of position (i, j) gives $Pr(i, j) = Pr(\hat{p}_j|p_i)$, which is the probability of predicting phoneme $\hat{p}_j$ given the actual phoneme is p_i. These matrices are the results of the 10th epoch, respectively. (Best viewed in color.)