```
import
torch
          import torch.nn as nn
          import torch.nn.functional as F
          from torch.distributions import Categorical # 다항분포


          from tqdm import tqdm # for문 돌아갈때 확인해주는 패키지임 (print로 확인하고 그런거)
          import numpy as np
          from preprocess import mulaw_decode  # https://en.wikipedia.org/wiki/%CE%9C-law_algorithm 8비트 16
          비트 그런거 맞춰주는과정이라는데
```

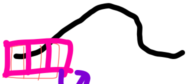
GRU 를 만듬

```
          def get_gru_cell(gru):
            gru_cell = nn.GRUCell(gru.input_size, gru.hidden_size)
            gru_cell.weight_hh.data = gru.weight_hh_l0.data          # hidden hidden  # hidden과 hidden사이
            gru_cell.weight_ih.data = gru.weight_ih_l0.data          # input hidden  # input과 hidden사이
            gru_cell.bias_hh.data = gru.bias_hh_l0.data              # hidden hidden
            gru_cell.bias_ih.data = gru.bias_ih_l0.data
            return gru_cell
```

예상

300Hz
200Hz
100Hz

Stride=1



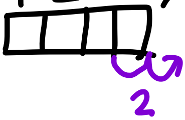
3x1

1차원 convolution

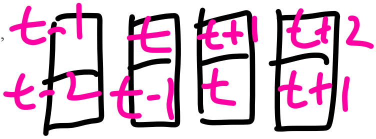
encoder 부분

```
          class Encoder(nn.Module):
            def __init__(self, in_channels, channels, n_embeddings, embedding_dim, jitter=0):
              super(Encoder, self).__init__()
              self.encoder = nn.Sequential(
                nn.Conv1d(in_channels, channels, 3, 1, 0, bias=False),# 1차원 커널  인풋크기, 아웃풋크기, 커널크
          기=3, stride=1,padding=0
                nn.BatchNorm1d(channels),                     #stride 1칸씩 이동 , 패딩 0으로 채워놓겠따
                nn.ReLU(True),          #batch normalization 기존 Deep Network에서는 learning rate를 너무 높게 잡
          을 경우 gradient가 explode/vanish 하거나, 나쁜 local minima에 빠지는 문제가 있었다. 이는 parameter들의
          scale 때문인데, Batch Normalization을 사용할 경우 propagation 할 때 parameter의 scale에 영향을 받지 않게
          된다. 따라서, learning rate를 크게 잡을 수 있게 되고 이는 빠른 학습을 가능케 한다.



                nn.Conv1d(channels, channels, 3, 1, 1, bias=False),
                nn.BatchNorm1d(channels),
                nn.ReLU(True),
                nn.Conv1d(channels, channels, 4, 2, 1, bias=False),
                nn.BatchNorm1d(channels),
                nn.ReLU(True),
                nn.Conv1d(channels, channels, 3, 1, 1, bias=False),
                nn.BatchNorm1d(channels),
                nn.ReLU(True),
                nn.Conv1d(channels, channels, 3, 1, 1, bias=False),
                nn.BatchNorm1d(channels),
                nn.ReLU(True),
                nn.Conv1d(channels, embedding_dim, 1)
              )
              self.codebook = VQEmbeddingEMA(n_embeddings, embedding_dim) # 코드북이 벡터양자화과정에서 나오는 결과물담은거같음
              self.jitter = Jitter(jitter)   # jitter는 과적합방지 근데 여기선 0넣서 안한듯 decoder에서 0.5로 잡음
```

4칸 , stride=2



Continuous =

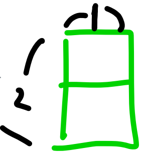


시간순

보라색 jitter

```
    def forward(self, mels):
        z = self.encoder(mels)   # 멜스펙트로그램 걍 넣는거임
        z, loss, perplexity = self.codebook(z.transpose(1, 2))     # 밑에부분
        z = self.jitter(z)
        return z, loss, perplexity
```

encoder를 통해



모양벡터로 바뀜 continuous 하게

```
    def encode(self, mel):         # 멜스펙트로그램 넣기
        z = self.encoder(mel)
        z, indices = self.codebook.encode(z.transpose(1, 2))
        return z, indices
```

시계열에서 쓰는 과적합 방지 방법 인듯?

Jitter =

```
class Jitter(nn.Module):         # 노이즈넣어서 과적합막아주는거  # p가머지 확률같긴한데  # nn.moduler?
    def __init__(self, p):
        super().__init__()
        self.p = p
        prob = torch.Tensor([p / 2, 1 - p, p / 2])   # 확률을 텐서형태로?  p=0.1 이면  0.05   0.9   0.05
        self.register_buffer("prob", prob)


    def forward(self, x):                      # x가 머지
        if not self.training or self.p == 0:
            return x
        else:
            batch_size, sample_size, channels = x.size()
```

이부분.

```
        dist = Categorical(self.prob)       #0.05, 0.9, 0.95 인 다항분포
        index = dist.sample(torch.Size([batch_size, sample_size])) - 1        # ???
        index[:, 0].clamp_(0, 1)
        index[:, -1].clamp_(-1, 0)
        index += torch.arange(sample_size, device=x.device)


        x = torch.gather(x, 1, index.unsqueeze(-1).expand(-1, -1, channels))
    return x
```

범위 안 값 변환

x가 이제 encoder를 통해 나온 정보임.

VQ bottleneck

```
# 이게 벡터 양자화 과정
class VQEmbeddingEMA(nn.Module):
  def __init__(self, n_embeddings, embedding_dim, commitment_cost=0.25, decay=0.999, epsilon=1e-5):
                  # n_embedding 임베딩벡터 개수? , embedding_dim= 임베딩 백터 크기
    super(VQEmbeddingEMA, self).__init__()
    self.commitment_cost = commitment_cost
    self.decay = decay
    self.epsilon = epsilon


    init_bound = 1 / 512
    embedding = torch.Tensor(n_embeddings, embedding_dim)
    embedding.uniform_(-init_bound, init_bound)     # -init~ init 균등분포 난수생성인듯
    self.register_buffer("embedding", embedding)
    self.register_buffer("ema_count", torch.zeros(n_embeddings))
    self.register_buffer("ema_weight", self.embedding.clone())
```

연속적으로 들어온 데이터를 $\frac{1}{512}$로 쪼갠다.

VQ bottleneck

```
  def encode(self, x):
    M, D = self.embedding.size()
    x_flat = x.detach().reshape(-1, D)
     # addmm 은  n*m   m*p
    distances = torch.addmm(torch.sum(self.embedding ** 2, dim=1) +
                            torch.sum(x_flat ** 2, dim=1, keepdim=True),
                            x_flat, self.embedding.t(),
                            alpha=-2.0, beta=1.0)


    indices = torch.argmin(distances.float(), dim=-1)
    quantized = F.embedding(indices, self.embedding)
    quantized = quantized.view_as(x)
```

addmm = 좀 특이한 연산 검색~

```
        return quantized, indices


    def forward(self, x):
        M, D = self.embedding.size()
        x_flat = x.detach().reshape(-1, D)


        distances = torch.addmm(torch.sum(self.embedding ** 2, dim=1) +
                                torch.sum(x_flat ** 2, dim=1, keepdim=True),
                                x_flat, self.embedding.t(),
                                alpha=-2.0, beta=1.0)


        indices = torch.argmin(distances.float(), dim=-1)
        encodings = F.one_hot(indices, M).float()
        quantized = F.embedding(indices, self.embedding)
        quantized = quantized.view_as(x)


        if self.training:
            self.ema_count = self.decay * self.ema_count + (1 - self.decay) * torch.sum(encodings, dim=0)


            n = torch.sum(self.ema_count)
            self.ema_count = (self.ema_count + self.epsilon) / (n + M * self.epsilon) * n


            dw = torch.matmul(encodings.t(), x_flat)
            self.ema_weight = self.decay * self.ema_weight + (1 - self.decay) * dw


            self.embedding = self.ema_weight / self.ema_count.unsqueeze(-1)


        e_latent_loss = F.mse_loss(x, quantized.detach())
        loss = self.commitment_cost * e_latent_loss


        quantized = x + (quantized - x).detach()


        avg_probs = torch.mean(encodings, dim=0)
        perplexity = torch.exp(-torch.sum(avg_probs * torch.log(avg_probs + 1e-10)))
```

Code book은
train 하기위해
지수이동 평균 하고
ARIMA 모델
사용

벡터
양자화
과정
VQ

ema
과정
지수이동
평균.

```
        return quantized, loss, perplexity



class Decoder(nn.Module):
    def __init__(self, in_channels, n_speakers, speaker_embedding_dim,
                 conditioning_channels, mu_embedding_dim, rnn_channels,
                 fc_channels, bits, hop_length):
        super().__init__()
        self.rnn_channels = rnn_channels
        self.quantization_channels = 2**bits
        self.hop_length = hop_length


        self.speaker_embedding = nn.Embedding(n_speakers, speaker_embedding_dim)
        self.rnn1 = nn.GRU(in_channels + speaker_embedding_dim, conditioning_channels,
                           num_layers=2, batch_first=True, bidirectional=True)
        self.mu_embedding = nn.Embedding(self.quantization_channels, mu_embedding_dim)
        self.rnn2 = nn.GRU(mu_embedding_dim + 2*conditioning_channels, rnn_channels, batch_first=True)
        self.fc1 = nn.Linear(rnn_channels, fc_channels)
        self.fc2 = nn.Linear(fc_channels, self.quantization_channels)
```

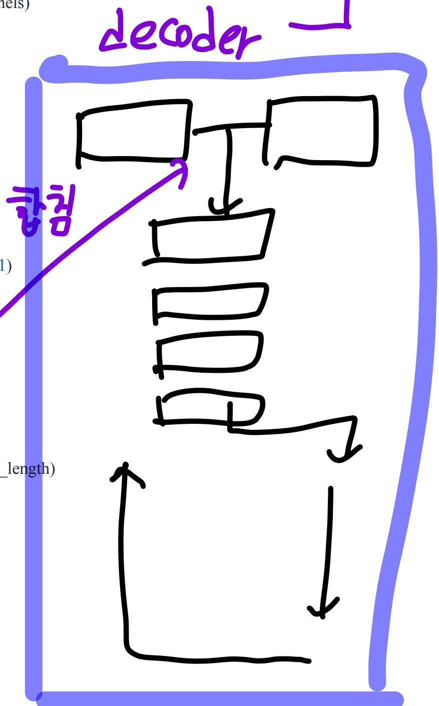
decoder

복원

embedding

GRU

```
    def forward(self, x, z, speakers):
        z = F.interpolate(z.transpose(1, 2), scale_factor=2)
        z = z.transpose(1, 2)


        speakers = self.speaker_embedding(speakers)
        speakers = speakers.unsqueeze(1).expand(-1, z.size(1), -1)


        z = torch.cat((z, speakers), dim=-1)
        z, _ = self.rnn1(z)


        z = F.interpolate(z.transpose(1, 2), scale_factor=self.hop_length)
        z = z.transpose(1, 2)


        x = self.mu_embedding(x)
        x, _ = self.rnn2(torch.cat((x, z), dim=2))
```



```
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x

def generate(self, z, speaker):
    output = []
    cell = get_gru_cell(self.rnn2)

    z = F.interpolate(z.transpose(1, 2), scale_factor=2)
    z = z.transpose(1, 2)

    speaker = self.speaker_embedding(speaker)
    speaker = speaker.unsqueeze(1).expand(-1, z.size(1), -1)

    z = torch.cat((z, speaker), dim=-1)
    z, _ = self.rnn1(z)

    z = F.interpolate(z.transpose(1, 2), scale_factor=self.hop_length)
    z = z.transpose(1, 2)

    batch_size, sample_size, _ = z.size()

    h = torch.zeros(batch_size, self.rnn_channels, device=z.device)
    x = torch.zeros(batch_size, device=z.device).fill_(self.quantization_channels // 2).long()

    for m in tqdm(torch.unbind(z, dim=1), leave=False):
        x = self.mu_embedding(x)
        h = cell(torch.cat((x, m), dim=1), h)
        x = F.relu(self.fc1(h))
        logits = self.fc2(x)
        dist = Categorical(logits=logits)
        x = dist.sample()
        output.append(2 * x.float().item() / (self.quantization_channels - 1.) - 1.)

    output = np.asarray(output, dtype=np.float64)
    output = mulaw_decode(output, self.quantization_channels)
```

앞부분이랑
비슷한데
잘 모르
겠다...

```
        return output
```

- Terms
- Privacy
- Security
- Status
- Help
- Contact GitHub
- Pricing
- API
- Training
- Blog
- About

Vector-quantized neural networks for acoustic unit discovery in the ZeroSpeech 2020 challenge

Benjamin van Niekerk *Leanne Nortje* *Herman Kamper*

E&E Engineering, Stellenbosch University, South Africa

benjamin.l.van.niekerk@gmail.com, nortjeleanne@gmail.com, kamperh@sun.ac.za

Abstract

In this paper, we explore vector quantization for acoustic unit discovery. Leveraging unlabelled data, we aim to learn discrete representations of speech that separate phonetic content from speaker-specific details. We propose two neural models to tackle this challenge. Both models use vector quantization to map continuous features to a finite set of codes. The first model is a type of vector-quantized variational autoencoder (VQ-VAE). The VQ-VAE encodes speech into a discrete representation from which the audio waveform is reconstructed. Our second model combines vector quantization with contrastive predictive coding (VQ-CPC). The idea is to learn a representation of speech by predicting future acoustic units. We evaluate the models on English and Indonesian data for the *ZeroSpeech 2020* challenge. In ABX phone discrimination tests, both models outperform all submissions to the 2019 and 2020 challenges, with a relative improvement of more than 30%. The discovered units also perform competitively on a downstream voice conversion task. Of the two models, VQ-CPC performs slightly better in general and is simpler and faster to train. Probing experiments show that vector quantization is an effective bottleneck, forcing the models to discard speaker information.

Index Terms: unsupervised speech processing, acoustic unit discovery, voice conversion, representation learning

1. Introduction

Modern speech and language technologies are developed with massive amounts of annotated data. However, large datasets of transcribed speech are not available for low-resource languages and building new corpora can be prohibitively expensive. As a result, tools like automatic speech recognition and text-to-speech are not available for many of the world's languages.

To address this problem, *zero-resource speech processing* aims to develop methods that can learn directly from speech without explicit supervision. The goal is to leverage unlabelled data to discover representations that capture meaningful phonetic contrasts while being invariant to background noise and speaker-specific details. These representations can then be used to bootstrap training in downstream speech systems and reduce requirements on labelled data. Additionally, since infants acquire language without explicit supervision, the discovered representations can be used in cognitive models of language learning [1–3].

Over the last few years, progress in this area has been driven by the *ZeroSpeech Challenges* [4–6]. *ZeroSpeech 2020* consolidates previous challenges, allowing submissions to both the 2017 and 2019 tracks. We focus on *ZeroSpeech 2019: Text-to-Speech Without Text*, which requires participants to discover *discrete* acoustic units from unlabelled data. From the discovered units, the task is then to synthesize speech in a target speaker's voice. Synthesized utterances are evaluated in terms of intelligibility, speaker-similarity, and naturalness. While similar to voice conversion [7, 8], an explicit goal of *ZeroSpeech 2019* is to learn *low-bitrate* representations that perform well on phone discrimination tests. In contrast to work on continuous representation learning [9–13], this encourages participants to find discrete units that correspond to distinct phones[1].

Early approaches to acoustic unit discovery typically combined clustering methods with hidden Markov models [15–19]. More recent studies have explored neural networks with intermediate discretization [20–23]. In this paper, we investigate vector quantized (VQ) neural networks for acoustic unit discovery, and propose two models for the *ZeroSpeech 2020* challenge.

The first model is a type of vector-quantized variational autoencoder (VQ-VAE) [24]. The VQ-VAE maps speech into a discrete latent space before reconstructing the original waveform. Instead of using WaveNet [25], we opt for a lightweight recurrent network as the decoder. The result is a smaller, faster model that can be trained on a single GPU.

The second model is a combination of vector-quantization and contrastive predictive coding (VQ-CPC). The model learns a discrete representation of speech that can distinguish future acoustic units from negative examples drawn from other utterances. We compare across-speaker and within-speaker sampling for negative examples and show that the latter is important for speaker invariance.

In ABX phone discrimination tests on English and Indonesian data, the models outperform all other submissions to the *ZeroSpeech 2019* and *2020* challenges. On the voice conversion task, both models are competitive, with VQ-CPC achieving the best naturalness and speaker-similarity scores on the English dataset. Finally, in probing experiments, we analyze the effect of VQ. We show that VQ imposes an information bottleneck that separates phonetic and speaker content.

2. Vector-quantized neural networks

In this section we first explain vector quantization and then discuss the two models in detail.

2.1. Vector quantization

The VQ layer consists of a trainable codebook $\{e_1, e_2, \dots, e_K\}$ with K distinct codes. In the forward pass, a sequence of continuous feature vectors $z := \langle z_1, z_2, \dots, z_T \rangle$ is discretized by mapping each z_i to it's nearest neighbor in the codebook. Concretely, we find $k := \arg\min_j ||z_i - e_j||^2$ and replace z_i with the code e_k, resulting in the quantized sequence $\hat{z} := \langle \hat{z}_1, \hat{z}_2, \dots, \hat{z}_T \rangle$. Since the arg min operator is not differentiable, in the backward pass, gradients are approximated using the straight-through estimator [26]. To train the codebook, we use an exponential moving average of the continuous features. Finally, a *commitment cost* is

gra
done

[1] As a point of reference, phonetic transcriptions encode speech at a rate of about 50 bits per second [14].

1 gra done
2 Codebook train 위해
지수이용

added to the loss to encourage each z_i to commit to the selected code. For a more detailed explanation see [24].

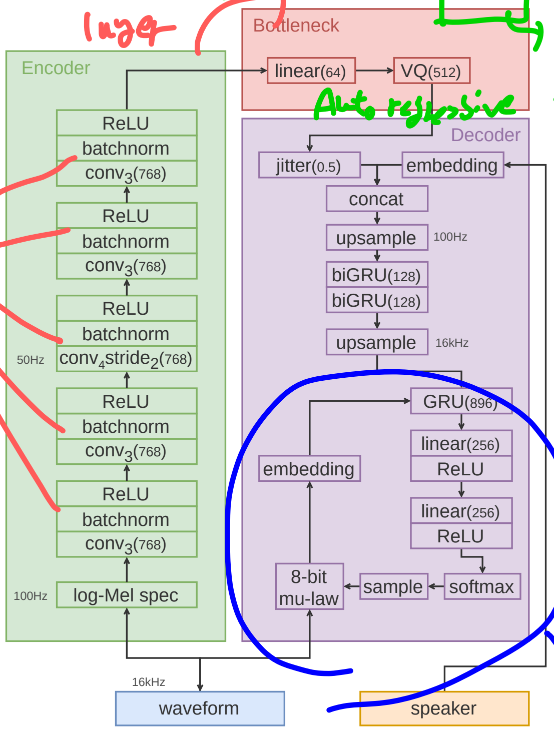
2.2. Vector-quantized variational autoencoder

Inspired by the WaveNet autoencoder proposed in [22], our first model is a type of VQ-VAE. We replace the WaveNet decoder [25] with a lightweight RNN based vocoder [27]. Together with automatic mixed precision [28], this allows us to train on a single GPU. Additionally, to learn a low-bitrate representation, we use a much smaller codebook. Finally, we release code and pretrained weights.[2]

Model description. The VQ-VAE can be divided into the three components shown in Figure 1. The *encoder* takes a speech waveform sampled at 16 kHz as input and computes a log-Mel spectrogram. The spectrogram is processed by a stack of 5 convolutional layers, which downsamples the input by a factor of 2. In the *bottleneck*, the output of the encoder is projected into a sequence of continuous features. The representation is then discretized using a VQ layer with 512 codes. Finally, the *decoder* tries to reconstruct the original waveform. To predict the next sample, we condition an autoregressive model on the output of the bottleneck, the speaker identity, and past waveform samples.

2배로
축?,

For acoustic unit discovery, the VQ-VAE balances two opposing pressures. On the one hand, the encoder must preserve information from the input to accurately reconstruct the waveform. On the other hand, vector quantization imposes an information bottleneck, forcing a compressed representation that discards non-essential details. To encourage the bottleneck to specifically discard speaker information, we condition the decoder on speaker identity during training.



5개 Convolution layer

통과

Continuous 로 출력됨)

512개로 출력

Auto regressive

VQ Coder.

Figure 1: *VQ-VAE: A convolutional encoder (green) takes a speech waveform as input and outputs downsampled continuous features. These are discretized (red) using vector quantization. The decoder (purple) then tries to reconstruct the input waveform from the discrete representation using an RNN-based vocoder conditioned on a speaker embedding.*

Training details. We train the model to maximize the log-likelihood of the waveform given the bottleneck, i.e. we minimize the sum of the reconstruction error and the commitment cost:

train

$$\mathcal{L} := -\frac{1}{N}\sum_{i=1}^{N}\log p(x_i|\hat{z}) + \beta\frac{1}{T}\sum_{i=1}^{T}||z_i - \text{sg}(\hat{z}_i)||^2,$$

where $\langle x_1, x_2, \ldots, x_N\rangle$ is a sequence of waveform samples, β is the commitment cost weight, and sg$(\cdot)$ denotes the stop-gradient operator. The model is trained on minibatches of 52 segments, each 320 ms long. We use the Adam optimizer [29] with an initial learning rate of $4 \cdot 10^{-4}$, which is halved after 300k and 400k steps. The network is trained for a total of 500k steps.

Voice conversion. At test time, we can generate speech in a target voice by conditioning the decoder on a specific speaker. First, we encode a source utterance into a sequence of acoustic units. Since the bottleneck separates speaker details form phonetic information, we can replace the speaker while retaining the content of the utterance. Specifically, the output of the bottleneck is concatenated with the target speaker embedding and piped to the decoder.

Practical considerations. Our goal is to discover phone-like acoustic units. Ideally, adjacent frames within the same phone would be mapped to the same unit. In practice, to encourage consistency across frames, we use time-jitter regularization [22]. During training, the code assigned to each frame may be replaced by one of its neighbors. Jitter forces the discovered codes to be useful across multiple time steps. We apply jitter directly after the bottleneck, with a replacement probability of 0.5. Another common issue with vector quantization is codebook collapse, where only a few codes are ever selected [30, 31]. We found that batch normalization, coupled with large batch sizes, improved codebook utilization.

2.3. Vector-quantized contrastive predictive coding

Contrastive predictive coding (CPC) is a recently proposed framework for unsupervised learning [32]. The idea is to learn representations by predicting future observations in latent space. Models are trained, with a contrastive loss, to distinguish future frames from negative examples. The motivation behind CPC is that the model must infer global structure in speech (e.g. phone identity) to make accurate predictions. At the same time, low-level details which do not improve prediction can be discarded.

Recent studies have shown that CPC learns representations that capture phonetic contrasts and transfer well across languages [33, 34]. In this paper, we adapt CPC to the task of acoustic unit discovery.[3] We incorporate vector quantization to learn discrete units, and investigate different negative sampling strategies to encourage speaker-invariant representations.

Model description. The VQ-CPC model is illustrated in Figure 2. First, the *encoder* maps input speech (parametrized as a log-Mel spectrogram) into a sequence of continuous features. The encoder consists of a strided convolutional layer (downsampling the input by a factor of 2), followed by a stack of 4 linear layers with ReLU activations. Layer normalization is applied after each layer. The *bottleneck* is identical to the one described in §2.2. The output of the encoder is projected into a sequence of

[2] https://github.com/bshall/ZeroSpeech

[3] https://github.com/bshall/VectorQuantizedCPC

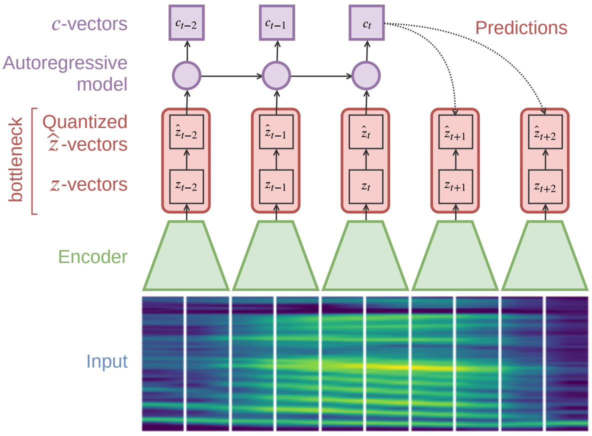


Figure 2: *VQ-CPC: An encoder (green) encodes speech (parametrized as a log-Mel spectrogram) to a sequence of continuous vectors z. Using a VQ bottleneck (red) the z-vectors are quantized. The quantized $\hat{z}$-vectors are summarised by an autoregressive RNN (purple) into context vectors c. Using this context, the model is trained to predict future codes.*

output of encoder.

continuous latent vectors which are discretized using a VQ layer with 512 codes. Finally, the *autoregressive model* summarizes the discrete representations (up to time t) into a context vector c_t. Using this context, the model is trained to discriminate future codes from negative examples drawn from other utterances.

C_t

Training details. Given a prediction horizon of M steps, a trainable predictor matrix W_m, and a set $\mathcal{N}_{t,m}$ containing negative examples and the positive code $\hat{z}_{t+m}$, we minimize the InfoNCE loss [32]:

$$\mathcal{L}_t := -\frac{1}{M}\sum_{m=1}^{M}\log\left[\frac{\exp(\hat{z}_{t+m}^{\intercal}W_m c_t)}{\sum_{\tilde{z}\in\mathcal{N}_{t,m}}\exp(\tilde{z}^{\intercal}W_m c_t)}\right].$$

The loss is averaged over segments of 1.28 seconds and a VQ commitment cost is added. We set the prediction horizon to $M = 6$ steps and sample 17 negative examples per step. We use the Adam optimizer, with a batches size of 64, and a learning rate of $4 \cdot 10^{-4}$. Each minibatch is divided into groups of 8 segments from which negative examples are sampled. To address codebook collapse, we use a warm-up phase where we linearly increase the learning rate from $1 \cdot 10^{-5}$ over the first 150 epochs.

Sampling negative examples. We investigate *across-speaker* and *within-speaker* sampling for negative examples. In across-speaker sampling, negatives are drawn from a mix of speakers, while within-speaker sampling uses the same speaker. We hypothesize that within-speaker sampling will encourage speaker-invariant representations since speaker information cannot be used to identify the positive example.

Voice conversion. VQ-CPC is not a generative model, so we train a separate vocoder on top of the discovered acoustic units for voice conversion. The vocoder is similar to the decoder in Figure 1, except the jitter layer is replaced with an embedding which reads in the code indices from the VQ-CPC bottleneck. Again, the target voice can be controlled by conditioning the vocoder on a specific speaker.

3. Experimental setup

Datasets. We evaluate our models on the English and Indonesian datasets from the *ZeroSpeech 2019 Challenge*. Indonesian is a low-resource Austronesian language widely used as a lingua franca [35, 36]. Following the challenge guidelines, we use English as the development language. After finalizing the models, we apply the same procedure to the Indonesian data. For both languages, training data consists of about 15 hours of speech from 100 speakers. An additional hour is provided per target speaker for voice conversion. Finally, the test set contains approximately 30 minutes of speech from unseen speakers.

ABX evaluation. ABX phone discrimination tests are used to evaluate the discovered acoustic units [37]. The tests ask whether triphone X is more similar to triphones A or B. Here, A and X are instances of the same triphone (e.g. "beg"), while B differs in the middle phone (e.g. "bag"). To measure speaker-invariance, A and B come from the same speaker, but X is taken from a different speaker. As a similarity metric, we use the average cosine distance along the dynamic time warping alignment path. ABX is reported as an aggregated error rate over all pairs of triphones in the test set.

Voice conversion. To assess voice conversion quality, human evaluators judge intelligibility, speaker-similarity, and naturalness. For intelligibility, the evaluators orthographically transcribe the synthesized speech. By comparing the transcriptions to the ground truth, a character error rate (CER) is calculated. The evaluators score speaker-similarity and naturalness on a scale from 1 to 5 (higher is better), with the latter reported as a mean opinion score (MOS).

Baselines. The challenge baseline system combines a Dirichlet process Gaussian mixture model (DPGMM) for acoustic unit discovery [19] with a parametric speech synthesizer based on Merlin [38]. The topline system feeds the output of a supervised speech recognition model to a text-to-speech system, both trained on ground-truth transcriptions. See [6] for details.

We also include results for two other approaches. The first is the VQ-VAE-based system we submitted to the previous challenge [21], referred to here as VQ-VAE(spec). Instead of generating audio waveforms directly, VQ-VAE(spec) uses a two-stage approach. The model reconstructs log-Mel spectrograms, which are then fed to a separately trained FFTNet vocoder [39] for synthesis. Secondly, we include results for the system of Chen and Hain [40], one of the other top-performing submissions to *ZeroSpeech 2020*. Their system is similar to the WaveNet autoencoder of [22], but uses instance-norm layers in the encoder and adaptive instance normalization for speaker conditioning. In contrast to our models, Chen and Hain also downsample by a factor of 4 and use a much larger codebook with 2^{16} codes.

4. Experimental results

Table 1 shows the evaluation results for the *ZeroSpeech 2020 Challenge*.[4] On ABX tests, our models achieve the best scores, outperforming all submissions to the 2019 and 2020 challenges. Over our closest competitor [40], we improve ABX scores on the English and Indonesian datasets by more than 30% and 50%, respectively. On the English voice conversion task, VQ-CPC also achieves top naturalness and speaker-similarity results, marginally beating the VQ-VAE. However, on Indonesian some of the other submissions perform better. This discrepancy may be explained by a mismatch in the volume of our synthesized speech and the source utterances. On the English dataset, the

[4] The leader-board can be viewed at https://zerospeech.com/2020/results.html. Voice conversion samples for our models can be found at https://bshall.github.io/ZeroSpeech/ and https://bshall.github.io/VectorQuantizedCPC/, respectively.

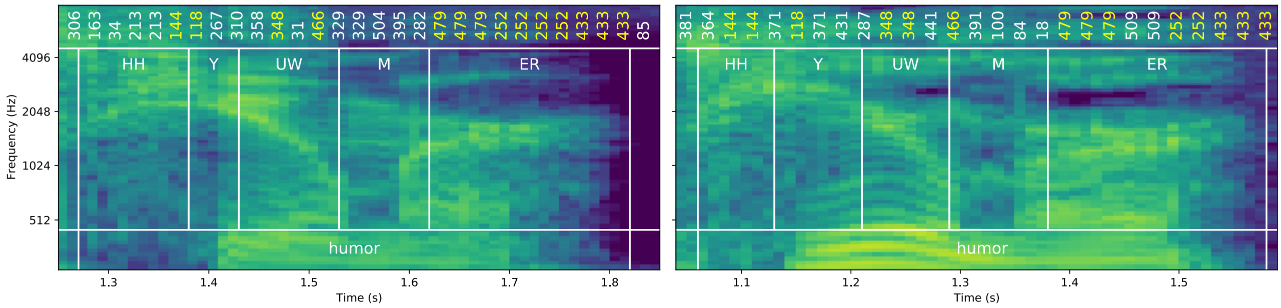


Figure 3: *The log-Mel spectrograms of speech segments taken from two different speakers. Overlaid are the aligned transcriptions and acoustic units from VQ-CPC. Common units in the two code sequences are highlighted in yellow.*

Table 1: *Human and machine evaluations on the English and Indonesian test sets. For MOS and similarity scores, higher is better. For CER, ABX, and bitrate, lower is better. ABX scores for the discrete codes and auxiliary representations are shown under the "code" and "aux" columns respectively.*

	CER	MOS	Similarity	ABX (%)		
Model	(%)	[1, 5]	[1, 5]	code	aux	Bitrate
English:						
DPGMM-Merlin	77	2.14	2.98	35.6	-	72
VQ-VAE(spec) [21]	67	2.18	2.51	27.6	23.0	173
Chen and Hain [40]	**18**	3.61	2.57	20.2	-	386
VQ-VAE	39	3.62	3.49	14.0	13.2	412
VQ-CPC	38	**3.64**	**3.80**	**13.4**	**12.5**	421
Supervised	43	2.52	3.10	29.9	-	**38**
Indonesian:						
DPGMM-Merlin	67	2.23	3.26	27.5	-	75
VQ-VAE(spec) [21]	60	1.96	1.76	19.8	14.5	140
Chen and Hain [40]	**15**	**4.06**	2.67	12.5	-	388
VQ-VAE	21	3.71	2.59	6.2	8.3	424
VQ-CPC	27	3.49	2.68	**5.1**	**4.9**	420
Supervised	33	3.49	**3.77**	16.1	-	**35**

volume difference is moderate, at around 6.1 LUFS[5]. But, a larger disparity of 9.4 LUFS on Indonesian may have negatively impacted our scores.

Chen and Hain [40] perform the best on intelligibility (CER) across both languages. These results seem to indicate a trade-off between intelligibility and voice conversion quality. By using a larger codebook, Chen and Hain are able to improve CER at the cost of speaker-similarity. A different trade-off is bitrate against CER and ABX score. While our models outperform the supervised topline, they operate at a much higher bitrate. In contrast, the topline has a similar bitrate to phonetic transcriptions.

Comparing our two models, it is clear that the VQ-VAE and VQ-CPC perform similarly across all metrics. However, VQ-CPC is an order of magnitude faster to train and was more robust to codebook collapse in our experiments. A comparison to the VQ-VAE(spec) (from our previous submission [21]), suggests that training an autoregressive decoder jointly with the encoder is beneficial. Finally, it is interesting to note that our models (trained exclusively on unlabelled speech) achieve comparable ABX scores to the visually grounded VQ model of [41], which is trained on paired images and unlabelled spoken captions.

Table 2: *Speaker classification results at probe points before and after quantization (shown under the "pre-quant" and "code" columns respectively).*

	Spkr. class. accuracy (%)		ABX (%)	
Model	code	pre-quant	code	aux
log-Mel spectrogram	98.9	-	27.0	-
VQ-VAE	65.8	98.8	14.0	13.2
VQ-CPC (within)	47.4	94.9	13.4	12.5
VQ-CPC (across)	80.3	98.5	36.2	31.7
CPC (within)	99.7	-	16.4	13.8

To show that the VQ bottleneck discards speaker information, we analyze representations before and after quantization. At each probe point, we train a multilayer perceptron with 2048 hidden units to predict the speaker identity. We use mean-pooling after the non-linearity to aggregate features. Table 2 shows the results of the probing experiments on English data. Based on the drop in speaker classification accuracy across the probe points, the VQ layer clearly acts as an information bottleneck, forcing the models to discard speaker details. Interestingly, CPC without vector quantization performs well on ABX tests but does not explicitly discard speaker information. As a result, CPC alone was not capable of voice conversion in our experiments. Table 2 also compares within-speaker and across-speaker negative sampling for VQ-CPC (see §2.3). Within-speaker sampling results in better speaker invariance (lower speaker classification accuracy) and significantly lower ABX scores (13.4% vs. 36.2%).

To examine a few of the acoustic units discovered by VQ-CPC, Figure 3 plots two utterances along with the extracted codes. We can see that the utterances are encoded as a similar sequence of units despite coming from different speakers. Additionally, adjacent frames within a phone are often mapped to the same code.

5. Conclusions and future work

We presented two neural models for acoustic unit discovery from unlabelled speech. Using vector quantization, both models learn discrete representations of speech that capture phonetic content but discard speaker information. They performed competitively on phone discrimination tests and a voice conversion task for the *ZeroSpeech 2020* challenge. Despite these merits, the models operate at high bitrates compared to phonetic transcriptions and a supervised topline. In future work, we aim to lower bitrates and discover acoustic units that are consistent across phones.

[5] Loudness Units relative to Full Scale (LUFS), see the `ITU-R BS.1770-4` standard.

6. References

[1] O. J. Räsänen, "Computational modeling of phonetic and lexical learning in early language acquisition: Existing models and future directions," *Speech Commun.*, vol. 54, pp. 975–997, 2012.

[2] T. Schatz and N. H. Feldman, "Neural network vs. HMM speech recognition systems as models of human cross-linguistic phonetic perception," in *Proc. CCN*, 2018.

[3] C. Shain and M. Elsner, "Measuring the perceptual availability of phonological features during language acquisition using unsupervised binary stochastic autoencoders," in *Proc. HLT-NAACL*, 2019.

[4] M. Versteegh, X. Anguera, A. Jansen, and E. Dupoux, "The Zero Resource Speech Challenge 2015: Proposed approaches and results," in *Proc. SLTU*, 2016.

[5] E. Dunbar, X. N. Cao, J. Benjumea, J. Karadayi, M. Bernard, L. Besacier, X. Anguera, and E. Dupoux, "The Zero Resource Speech Challenge 2017," in *Proc. ASRU*, 2017.

[6] E. Dunbar, R. Algayres, J. Karadayi, M. Bernard, J. Benjumea, X.-N. Cao, L. Miskic, C. Dugrain, L. Ondel, A. W. Black *et al.*, "The Zero Resource Speech Challenge 2019: TTS without T," in *Proc. Interspeech*, 2019.

[7] A. Kain and M. W. Macon, "Spectral voice conversion for text-to-speech synthesis," in *Proc. ICASSP*, 1998.

[8] J.-c. Chou, C.-c. Yeh, H.-y. Lee, and L.-s. Lee, "Multi-target voice conversion without parallel data by adversarially learning disentangled audio representations," in *Proc. Interspeech*, 2018.

[9] N. Zeghidour, G. Synnaeve, N. Usunier, and E. Dupoux, "Joint learning of speaker and phonetic similarities with Siamese networks," in *Proc. Interspeech*, 2016.

[10] M. Heck, S. Sakti, and S. Nakamura, "Learning supervised feature transformations on zero resources for improved acoustic unit discovery," *IEICE T. Inf. Syst.*, vol. 101, no. 1, pp. 205–214, 2018.

[11] Y.-A. Chung, W.-N. Hsu, H. Tang, and J. Glass, "An unsupervised autoregressive model for speech representation learning," in *Proc. Interspeech*, 2019.

[12] W. Wang, Q. Tang, and K. Livescu, "Unsupervised pre-training of bidirectional speech encoders via masked reconstruction," in *Proc. ICASSP*, 2020.

[13] P.-J. Last, H. A. Engelbrecht, and H. Kamper, "Unsupervised feature learning for speech using correspondence and siamese networks," *IEEE Signal Proc. Let.*, vol. 27, pp. 421–425, 2020.

[14] J. L. Flanagan, *Speech analysis synthesis and perception*. Springer Science & Business Media, 2013, vol. 3.

[15] B. Varadarajan, S. Khudanpur, and E. Dupoux, "Unsupervised learning of acoustic sub-word units," in *Proc. ACL*, 2008.

[16] C.-y. Lee and J. R. Glass, "A nonparametric Bayesian approach to acoustic model discovery," in *Proc. ACL*, 2012.

[17] M.-H. Siu, H. Gish, A. Chan, W. Belfield, and S. Lowe, "Unsupervised training of an HMM-based self-organizing unit recognizer with applications to topic classification and keyword discovery," *Comput. Speech Lang.*, vol. 28, no. 1, pp. 210–223, 2014.

[18] C.-y. Lee, T. O'Donnell, and J. R. Glass, "Unsupervised lexicon discovery from acoustic input," *Trans. ACL*, vol. 3, pp. 389–403, 2015.

[19] L. Ondel, L. Burget, and J. Černockỳ, "Variational inference for acoustic unit discovery," *Procedia Comput. Sci.*, vol. 81, pp. 80–86, 2016.

[20] L. Badino, A. Mereta, and L. Rosasco, "Discovering discrete subword units with binarized autoencoders and hidden-Markov-model encoders," in *Proc. Interspeech*, 2015.

[21] R. Eloff, A. Nortje, B. L. Van Niekerk, A. Govender, L. Nortje, A. Pretorius, E. Van Biljon, E. Van der Westhuizen, L. Van Staden, and H. Kamper, "Unsupervised acoustic unit discovery for speech synthesis using discrete latent-variable neural networks," in *Proc. Interspeech*, 2019.

[22] J. Chorowski, R. J. Weiss, S. Bengio, and A. van den Oord, "Unsupervised speech representation learning using WaveNet autoencoders," *IEEE Trans. Audio, Speech, Language Process.*, vol. 27, no. 12, pp. 2041–2053, 2019.

[23] A. Tjandra, B. Sisman, M. Zhang, S. Sakti, H. Li, and S. Nakamura, "VQVAE unsupervised unit discovery and multi-scale code2spec inverter for Zerospeech Challenge 2019," in *Proc. Interspeech*, 2019.

[24] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proc. NeurIPS*, 2017.

[25] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: a generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[26] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[27] J. Lorenzo-Trueba, T. Drugman, J. Latorre, T. Merritt, B. Putrycz, R. Barra-Chicote, A. Moinet, and V. Aggarwal, "Towards achieving robust universal neural vocoding," in *Proc. Interspeech*, 2019.

[28] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *Proc. ICLR*, 2018.

[29] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.

[30] L. Kaiser, S. Bengio, A. Roy, A. Vaswani, N. Parmar, J. Uszkoreit, and N. Shazeer, "Fast decoding in sequence models using discrete latent variables," in *Proc. ICML*, 2018.

[31] A. Baevski, S. Schneider, and M. Auli, "vq-wav2vec: Self-supervised learning of discrete speech representations," in *Proc. ICLR*, 2020.

[32] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[33] M. Rivière, A. Joulin, P.-E. Mazaré, and E. Dupoux, "Unsupervised pretraining transfers well across languages," in *Proc. ICASSP*, 2020.

[34] J. Kahn, M. Rivière, W. Zheng, E. Kharitonov, Q. Xu, P.-E. Mazaré, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen *et al.*, "Libri-light: A benchmark for asr with limited or no supervision," in *Proc. ICASSP*, 2020.

[35] S. Sakti, R. Maia, S. Sakai, T. Shimizu, and S. Nakamura, "Development of HMM-based Indonesian speech synthesis," in *Proc. O-COCOSDA*, 2008.

[36] S. Sakti, E. Kelana, H. Riza, S. Sakai, K. Markov, and S. Nakamura, "Development of Indonesian large vocabulary continuous speech recognition system within A-STAR project," in *Proc. TCAST*, 2008.

[37] T. Schatz, V. Peddinti, F. Bach, A. Jansen, H. Hermansky, and E. Dupoux, "Evaluating speech features with the minimal-pair ABX task: Analysis of the classical MFC/PLP pipeline," in *Proc. Interspeech*, 2013.

[38] Z. Wu, O. Watts, and S. King, "Merlin: An open source neural network speech synthesis system." in *Proc. SSW*, 2016.

[39] Z. Jin, A. Finkelstein, G. J. Mysore, and J. Lu, "FFTNet: A real-time speaker-dependent neural vocoder," in *Proc. ICASSP*, 2018.

[40] M. Chen and T. Hain, "Unsupervised acoustic unit representation learning for voice conversion using WaveNet auto-encoders," in *submitted to Interspeech*, 2020.

[41] D. Harwath, W.-N. Hsu, and J. Glass, "Learning hierarchical discrete linguistic units from visually-grounded speech," in *Proc. ICLR*, 2020.