

wav2vec working principles

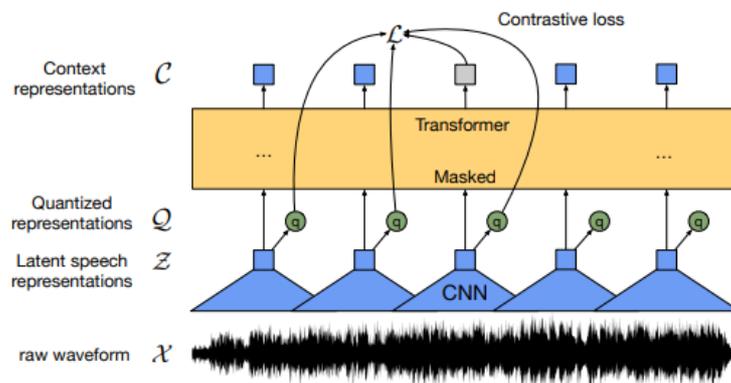
Publication - <https://arxiv.org/pdf/2006.11477.pdf>

CATEGORICAL REPARAMETERIZATION WITH GUMBEL-SOFTMAX

<https://arxiv.org/pdf/1611.01144.pdf>

Our approach encodes speech audio via a multi-layer convolutional neural network and then masks spans of the resulting latent speech representations. The latent representations are fed to a Transformer network to build contextualized representations and the model is trained via a contrastive task where the true latent is to be distinguished from distractors.

Robust wav2vec 2.0 - pre-trained on multiple domains (read audiobooks, phone calls, speeches...)



Training objective - identifying the correct quantized latent audio representation in a set of distractors for each masked time step.

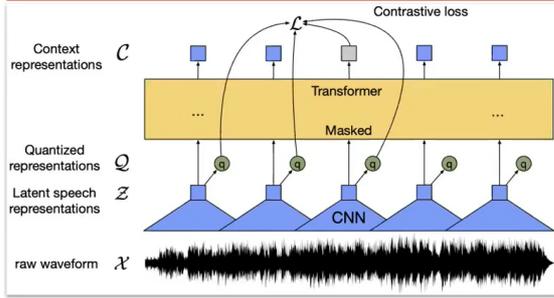
With cross-lingual training, wav2vec 2.0 learns speech units that are used in multiple languages. We find that some units are used for only a particular language, whereas others are used in similar languages and sometimes even in languages that aren't very similar.

Video that describes this architecture:

https://www.youtube.com/watch?v=8Kpowre6yyk&ab_channel=MaziarRaissi



wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations



$f: \mathcal{X} \mapsto \mathcal{Z}$
 multi-layer convolutional feature encoder
 $\mathcal{X} \rightarrow$ input raw audio
 $\mathcal{Z} = (z_1, z_2, \dots, z_T) \rightarrow$ latent speech representations
 $g: \mathcal{Z} \mapsto \mathcal{C}$
 transformer
 $\mathcal{C} = (c_1, c_2, \dots, c_T) \rightarrow$ representations capturing information from the entire sequence

Instead of fixed positional embeddings which encode absolute positional information, use a convolutional layer which acts as relative positional embedding.

$\mathcal{Z} \mapsto \mathcal{Q}$
 quantization module
 $\mathcal{Q} = (q_1, q_2, \dots, q_T)$

diversity loss: encourage the model to use the codebook entries equally often

Quantization Module

$G \rightarrow$ number of codebooks/groups
 $V \rightarrow$ number of entries
 $e \in \mathbb{R}^{V \times d/G}$

Choose one entry/row from each codebook e and concatenate the resulting vectors e_1, \dots, e_G and apply a linear transformation $\mathbb{R}^d \rightarrow \mathbb{R}^f$ to obtain $g \in \mathbb{R}^f$. The Gumbel softmax enables choosing discrete codebook entries in a fully differentiable way!

$z \mapsto l$
 $z \rightarrow$ feature encoder output
 $l \in \mathbb{R}^{G \times V} \rightarrow$ logits

$$p_{g,v} = \frac{\exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^V \exp(l_{g,k} + n_k)/\tau}$$

$\tau \rightarrow$ non-negative temperature
 $n = -\log(-\log(u)) \rightarrow$ Gumbel noise $u \sim U(0, 1)$

Forward pass: $i = \arg \max_j p_{g,j} \rightarrow$ codeword i

Backward pass: true gradient of the Gumbel softmax outputs

Pre-training
 $\mathcal{L}_m = -\log \frac{\exp(\text{sim}(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathcal{Q}_t} \exp(\text{sim}(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)}$ $\text{sim}(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b} / \|\mathbf{a}\| \|\mathbf{b}\|$

contrastive loss: identify the true quantized latent speech representation for a masked time step within a set of distractors.

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^G -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^G \sum_{v=1}^V \bar{p}_{g,v} \log \bar{p}_{g,v} \quad \mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d$$

	avg. WER	std.
Continuous inputs, quantized targets (Baseline)	7.97	0.02
Quantized inputs, quantized targets	12.18	0.41
Quantized inputs, continuous targets	11.18	0.16
Continuous inputs, continuous targets	8.58	0.08

	TIMT phoneme recognition accuracy in terms of phoneme error rate (PER)	
	dev PER	test PER
CNN + TD-filterbanks [59]	15.6	18.0
PASE+ [47]	-	17.2
Li-GRU + MLLR [46]	-	14.9
wav2vec [49]	12.9	14.7
vq-wav2vec [5]	9.6	11.6
This work (no LM)	7.4	8.3
LARGE (LS-960)	7.4	8.3

Quantizer implementation:

```

class Wav2Vec2GumbelVectorQuantizer(nn.Module):
    """
    Vector quantization using gumbel softmax. See [CATEGORICAL REPARAMETERIZATION WITH
    GUMBEL-SOFTMAX](https://arxiv.org/pdf/1611.01144.pdf) for more information.
    """

    def __init__(self, config):
        super().__init__()
        self.num_groups = config.num_codevector_groups
        self.num_vars = config.num_codevectors_per_group

        if config.codevector_dim % self.num_groups != 0:
            raise ValueError(
                f"config.codevector_dim {config.codevector_dim} must be divisible "
                f"by config.num_codevector_groups {self.num_groups} for concatenation"
            )

        # storage for codebook variables (codewords)
        self.codevectors = nn.Parameter(
            torch.FloatTensor(1, self.num_groups * self.num_vars, config.codevector_dim // self.num_groups)
        )
        self.weight_proj = nn.Linear(config.conv_dim[-1], self.num_groups * self.num_vars)

        # can be decayed for training
        self.temperature = 2

    @staticmethod
    def _compute_perplexity(probs, mask=None):
        if mask is not None:
            mask_extended = mask.flatten()[:, None, None].expand(probs.shape)
            probs = torch.where(mask_extended, probs, torch.zeros_like(probs))
            marginal_probs = probs.sum(dim=0) / mask.sum()
        else:
            marginal_probs = probs.mean(dim=0)

        perplexity = torch.exp(-torch.sum(marginal_probs * torch.log(marginal_probs + 1e-7), dim=-1)).sum()
        return perplexity

    def forward(self, hidden_states, mask_time_indices=None):
        batch_size, sequence_length, hidden_size = hidden_states.shape

        # project to codevector dim
        hidden_states = self.weight_proj(hidden_states)
        hidden_states = hidden_states.view(batch_size * sequence_length * self.num_groups, -1)
  
```

```

if self.training:
    # sample code vector probs via gumbel in differentiable way
    codevector_probs = nn.functional.gumbel_softmax(
        hidden_states.float(), tau=self.temperature, hard=True
    ).type_as(hidden_states)

    # compute perplexity
    codevector_soft_dist = torch.softmax(
        hidden_states.view(batch_size * sequence_length, self.num_groups, -1).float(), dim=-1
    )
    perplexity = self._compute_perplexity(codevector_soft_dist, mask_time_indices)
else:
    # take argmax in non-differentiable way
    # compute hard codevector distribution (one hot)
    codevector_idx = hidden_states.argmax(dim=-1)
    codevector_probs = hidden_states.new_zeros(*hidden_states.shape).scatter_(
        -1, codevector_idx.view(-1, 1), 1.0
    )
    codevector_probs = codevector_probs.view(batch_size * sequence_length, self.num_groups, -1)

    perplexity = self._compute_perplexity(codevector_probs, mask_time_indices)

codevector_probs = codevector_probs.view(batch_size * sequence_length, -1)
# use probs to retrieve codevectors
codevectors_per_group = codevector_probs.unsqueeze(-1) * self.codevectors
codevectors = codevectors_per_group.view(batch_size * sequence_length, self.num_groups, self.num_vars, -1)
codevectors = codevectors.sum(-2).view(batch_size, sequence_length, -1)

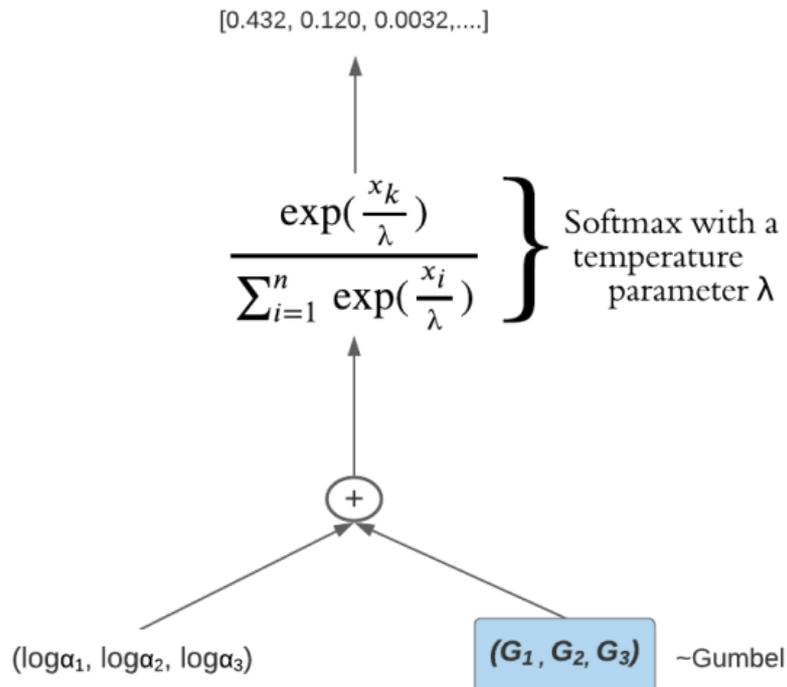
return codevectors, perplexity

```

Gumbel softmax - interpolation between a discrete one-hot encoded categorical distribution and continuous categorical densities.

Low temperatures - approaches argmax

High temperatures - approaches uniform distribution



Pytorch implementation:

<https://github.com/pytorch/pytorch/blob/15f9fe1d92a5d1e86278ae25f92dd9677b4956dc/torch/nn/functional.py#L1237>

Great explanations:

<https://datascience.stackexchange.com/questions/58376/gumbel-softmax-trick-vs-softmax-with-temperature>