MAAD: Week 3

Self Supervised Models

July 25th, 2026

Agenda

1. **Generative**
   i. WaveNet
2. **Self Supervised**
   i. Wav2Vec
   ii. VQ-Wav2Vec
   iii. Wav2Vec 2.0
   iv. Hubert
   v. WavLM
3. **A[illegible] Language [illegible]dels**
   i. [illegible]
   ii [illegible]
   iii [illegible]Qwen-Audio-[illegible]

https://www.everypixel.com/image-419543868743882478

What are we going to talk about?

We are here to learn about how these models work.

No time is spent on the specific datasets or results.

All of these models build on each other.

Understanding one helps you understand the rest.

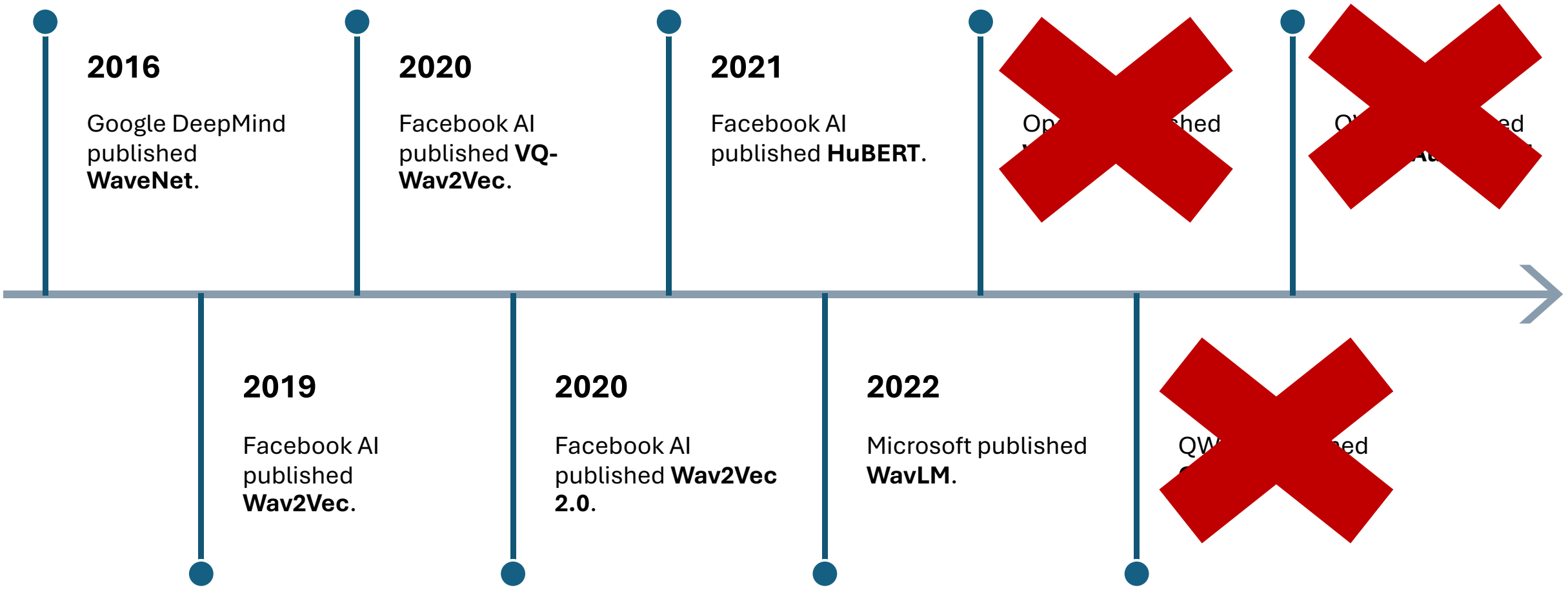
After WaveNet, the tasks remain the same: **learn a representation that helps on downstream tasks**.

After Wav2Vec, the results improve for each iteration.

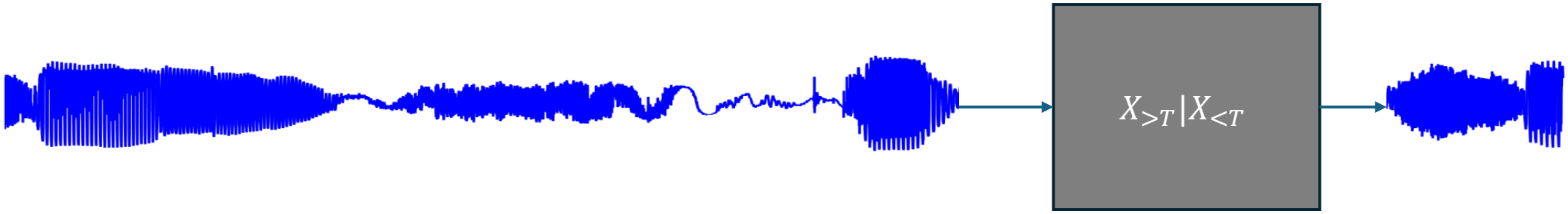
What are we going to talk about?

This is just one of those things that are good for you.

Timeline


2016
Google DeepMind published **WaveNet**.
2020
Facebook AI published **VQ-Wav2Vec**.
2021
Facebook AI published **HuBERT**.
2019
Facebook AI published **Wav2Vec**.
2020
Facebook AI published **Wav2Vec 2.0**.
2022
Microsoft published **WavLM**.

WaveNet



Our goal is to learn the joint probability of a waveform $\boldsymbol{x} = \{x_1, \dots, x_T\}$.

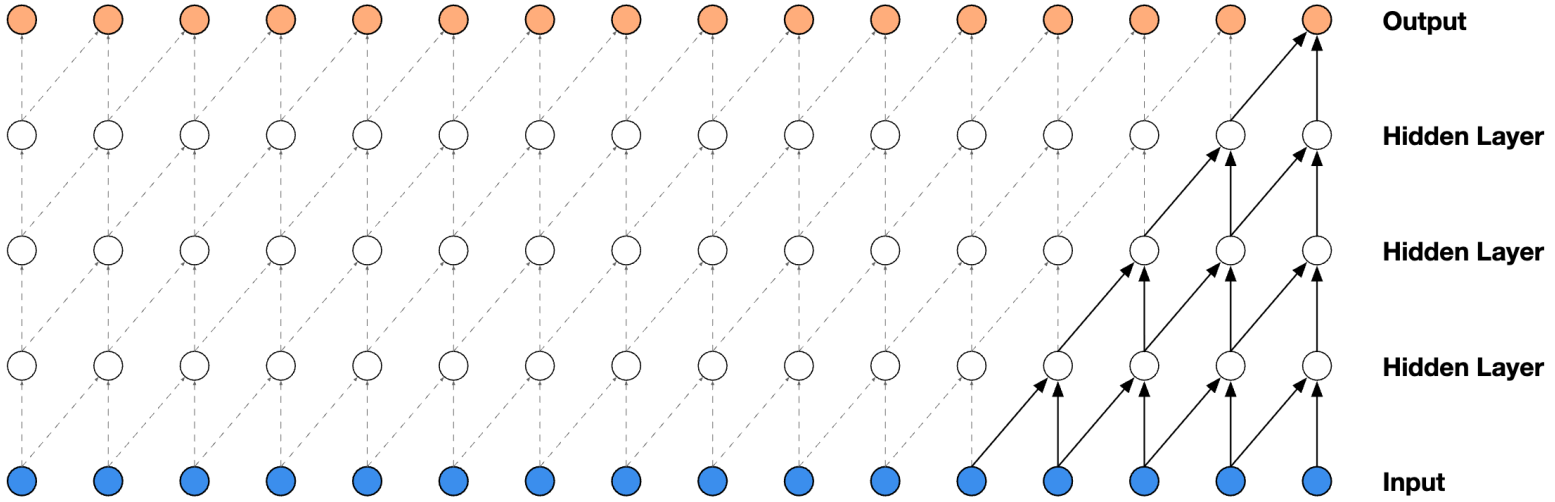
We can obtain this distribution by considering the following:

$$p(x_1) = p(x_1), \qquad p(x_2, x_1) = p(x_2|x_1) \cdot p(x_1), \qquad p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t|x_1, \dots, x_{t-1})$$

We can also condition on some additional information $\boldsymbol{h}$ as: $p(\boldsymbol{x}|\boldsymbol{h}) = \prod_{t=1}^{T} p(x_t|x_1, \dots, x_{t-1}, \boldsymbol{h})$

WaveNet

The first ingredient is the **causal convolution**.



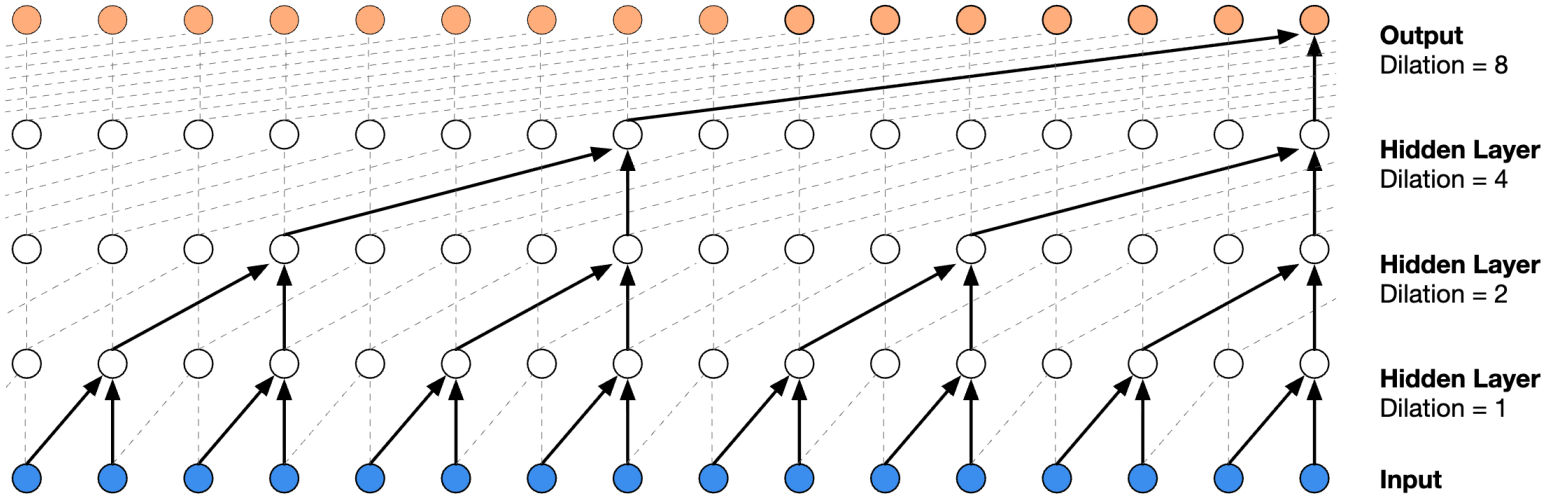
With any convolution block, we will have a fixed receptive field.
Therefore, we will approximate the joint distribution as such:

$$p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t|x_1, \dots, x_{t-1}) \approx \prod_{t=1}^{T} p_\theta(x_t|x_{t-r}, \dots, x_{t-1}),$$

where r is the size of the receptive field and θ is learnable parameters.

WaveNet

The next ingredient is the **dilated causal convolution**.

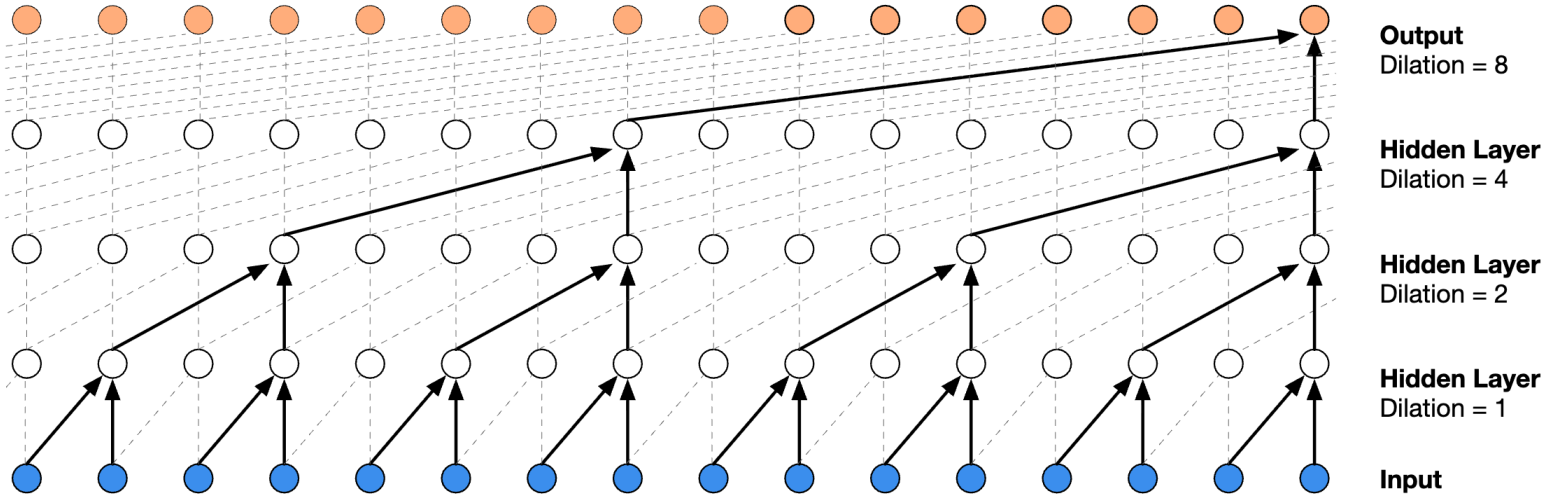


Dilation is conceptually adding zeros in between kernel components.

No Dilation $\mathrm{v} = \begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}, k_{s=3,d=1} = \begin{bmatrix}5\\6\\7\end{bmatrix} \rightarrow \begin{bmatrix}5\cdot 1 + 6\cdot 2 + 3\cdot 7\\5\cdot 2 + 6\cdot 3 + 7\cdot 4\\5\cdot 3 + 6\cdot 4 + 7\cdot 5\end{bmatrix}$

WaveNet

The next ingredient is the **dilated causal convolution**.



Dilation is conceptually adding zeros in between kernel components.

Dilation

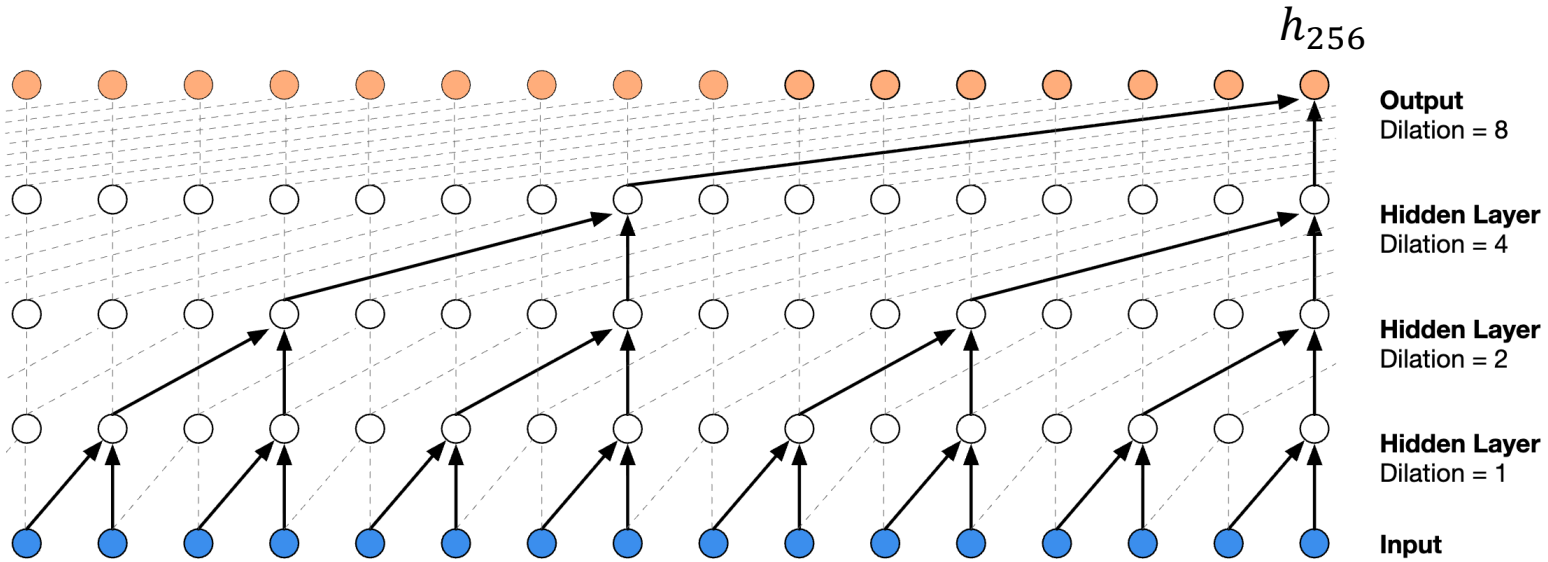
$$\mathrm{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}, \; k_{s=3,d=2} = \begin{bmatrix} 5 \\ 0 \\ 6 \\ 0 \\ 7 \end{bmatrix} \rightarrow [5 \cdot 1 \; + \; 6 \cdot 3 + 7 \cdot 5], \qquad y = \sum_{m=0}^{\mathrm{K}-1} w[m]v[t - dm]$$

WaveNet

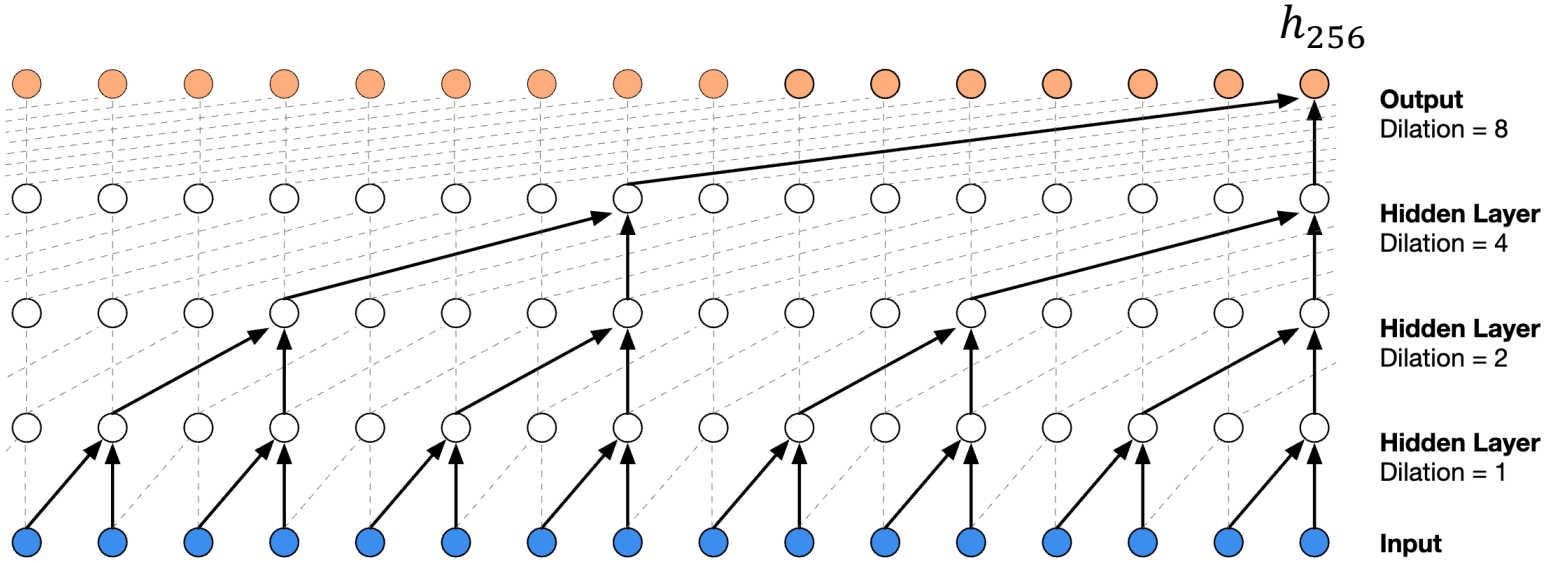
Before data enters the network, it is compressed with a μ-law transformation and then quantized to to 256 bins:

$$f(x_t) = \text{sign}(x_t)\frac{\ln(1+\mu|x_t|)}{\ln(1+\mu)},$$

where $-1 < x_t < 1$ and $\mu = 255$.



WaveNet



We can now define our conditional distribution as:

$$p_\theta(\mathrm{f}(x_t) = \mathrm{i}|\mathrm{f}(x_{t-R}), \ldots, \mathrm{f}(x_{t-1})) = \frac{e^{h_i}}{\sum_{j=1}^{256} e^{h_j}}, \exists i \in \{1, 2, \ldots 256\}$$

WaveNet

We can now define our conditional distribution as:

$$\pi_{t,\theta} = p_\theta(\mathrm{f}(x_t = \mathrm{i})|\mathrm{f}(x_{t-R}), \dots, \mathrm{f}(x_{t-1})) = \frac{e^{h_i}}{\sum_{j=1}^{256} e^{h_j}}, \exists i \in \{1,2, \dots 256\}$$

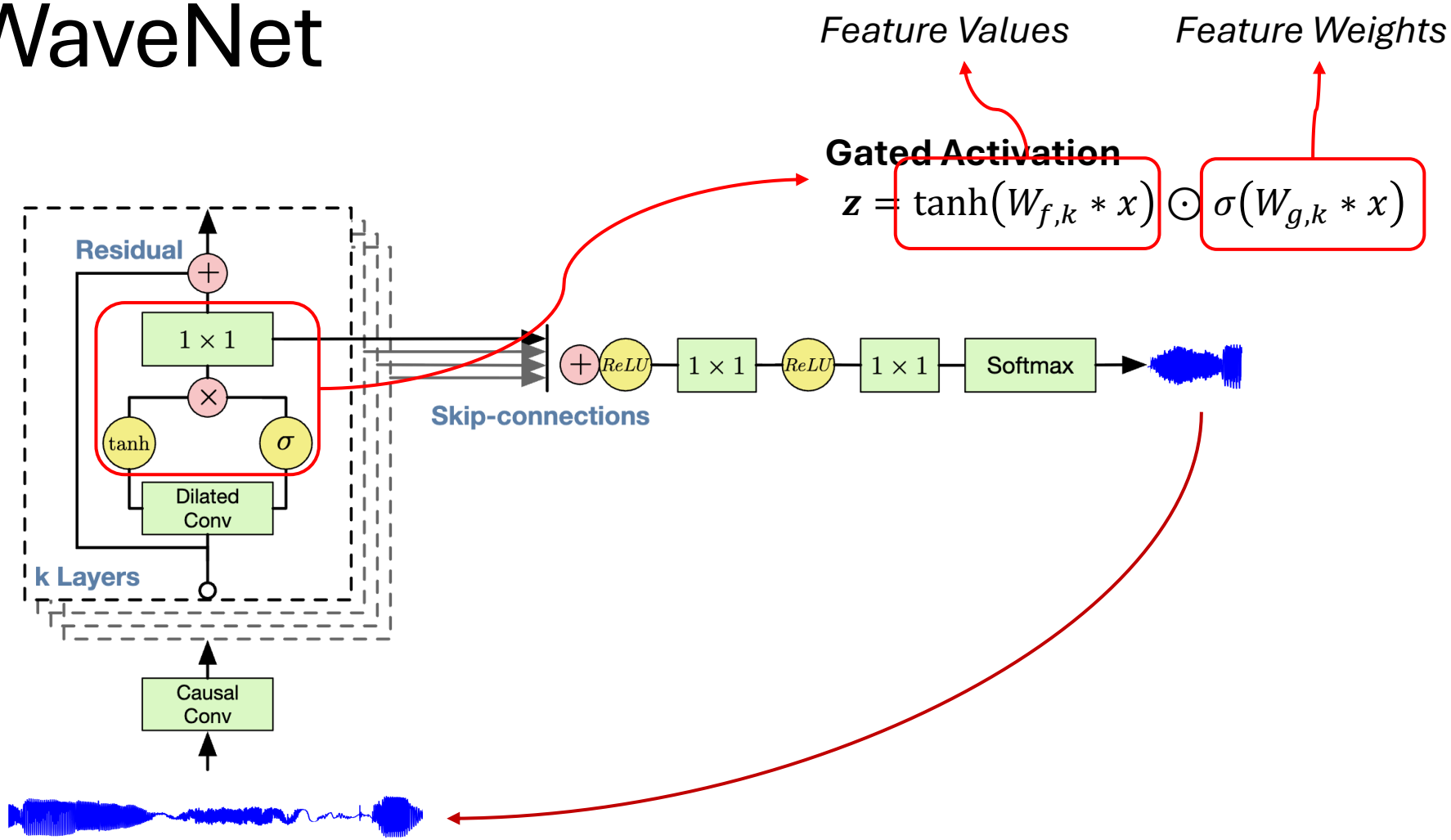
$$\mathrm{q}_t \in \{1, \dots, 256\}$$

The objective function is now simply:

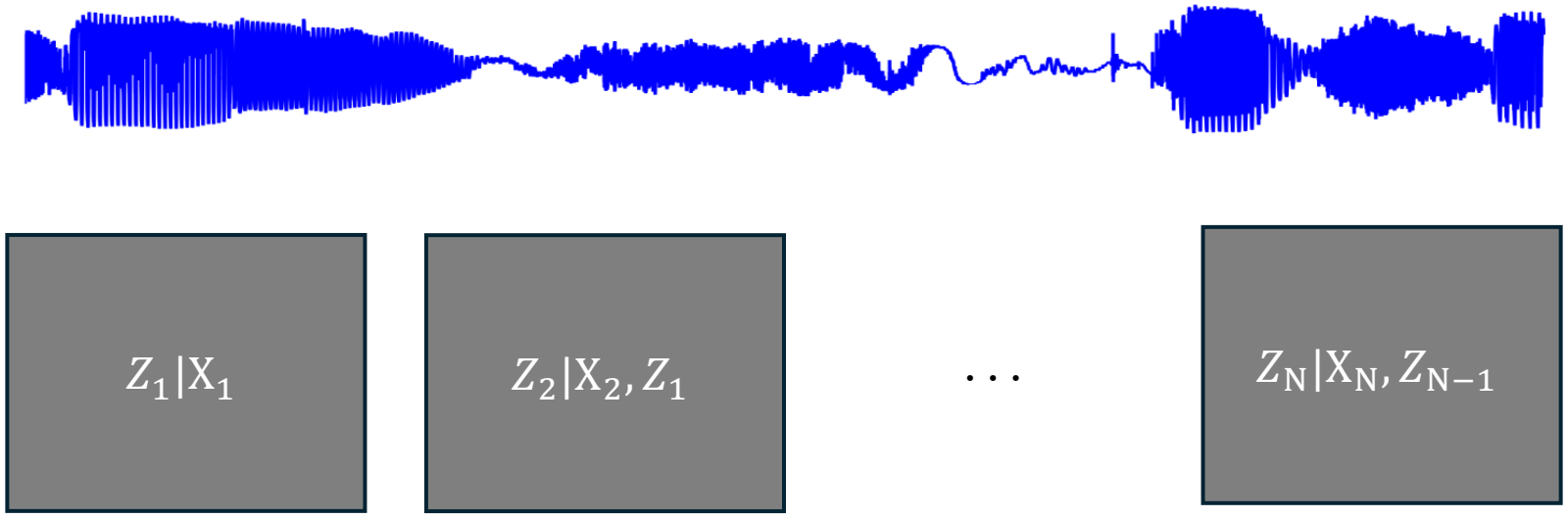
$$L_t(\theta) = -\log p_\theta(q_t = \mathrm{i}|q_{t-R}, \dots, q_{t-1}),$$

which is the negative log likelihood loss.

WaveNet


Feature Values
Feature Weights
Gated Activation
$z = \tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x)$
Residual
1×1
tanh
σ
Dilated Conv
k Layers
Causal Conv
Skip-connections
ReLU
1×1
ReLU
1×1
Softmax

Wav2Vec



Our goal is to learn a latent representation of the waveform, $\boldsymbol{z}$, that encodes information governing overall speech structure such as linguistics, phonetics, and identity.

Wav2Vec

Before we jump into what they do, lets think about what a useful embedding space might look like.

One useful aspect would be to conserve the temporal structure, such that:

$$p(\mathrm{x}_{t+\mathrm{k}}|z_{<t}) \approx p(x_{t+\mathrm{k}}|x_{<t})$$

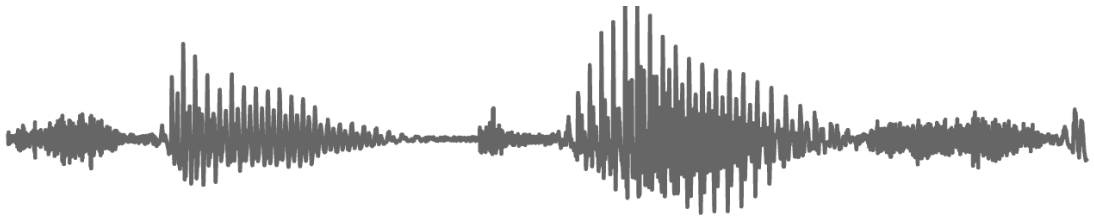
Additionally, we would want groups to naturally form because of speech structure:

$$s\left(z_i, z_j\right) > s\left(z_j, z_k\right),$$

where z_i and z_j share similar speech content, while z_k has very different speech content, and $s(\mathrm{A}, \mathrm{B}) = \frac{A \cdot B}{||A|| \cdot ||B||}$.

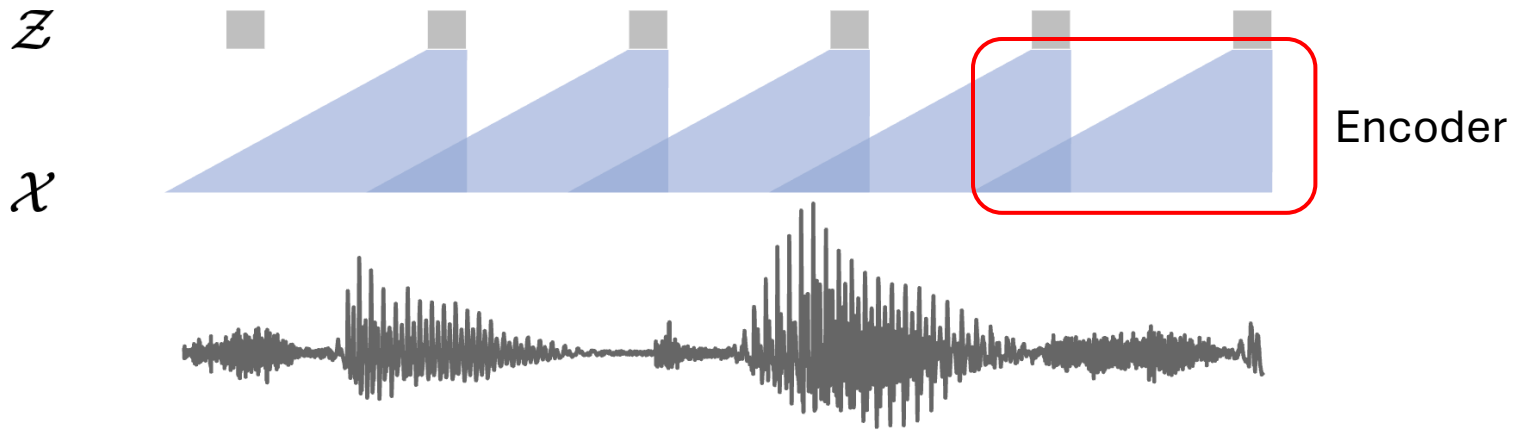
Wav2Vec

The raw waveform after normalization is fed into model. **No quantization!**



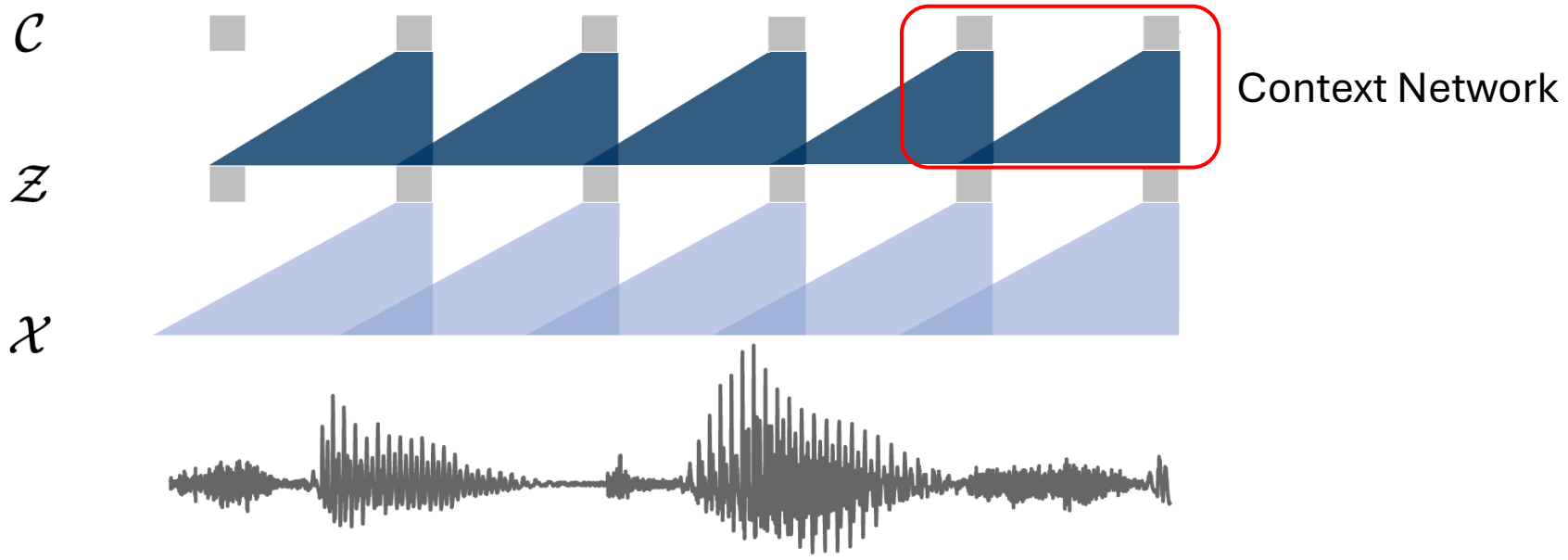
Wav2Vec

The encoder maps the audio signal $\boldsymbol{X}$ into a latent space $\boldsymbol{Z}$, such that $\boldsymbol{f}: \boldsymbol{X} \rightarrow \boldsymbol{Z}$.
It also uses **causal convolutions**.



Wav2Vec

The context network maps the latent $\boldsymbol{Z}$ to an an aggregate representation **C**, such that g : $\boldsymbol{Z} \rightarrow \boldsymbol{C}$.



Wav2Vec

Thinking back to how we wanted to organize the latent space, we wanted temporal prediction and speech structure. We can now define the loss as such:

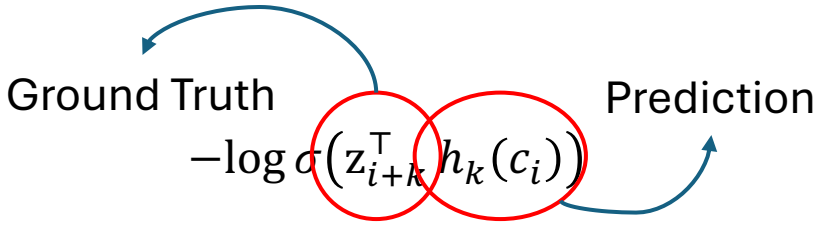
$$L_k = -\sum_{i=1}^{T-k} \log \sigma\left(z_{i+k}^{\top} h_k(c_i)\right) + \lambda E_{\tilde{z}\sim p_n}\left[\log \sigma\left(-\tilde{z}^{\top} h_k(c_i)\right)\right]$$

Prediction Discrimination

where σ is the sigmoid function, h_k is a step specific linear transformation, k is the number of steps to evaluate over, c_i is the output for the context network, and $\tilde{\mathrm{z}}$ are latents where $\mathrm{i} + \mathrm{k} \neq \mathrm{j}, \forall \mathrm{j} \in \{1, \mathrm{T}\}$.

This objective stems from **Contrastive Predictive Coding** where the goal is to maximize the mutual information between historical context and future representations.

Wav2Vec



The prediction term is the positive class binary log-likelihood between the prediction for timestep k, $h_k(c_i)$, and the encoded future timestep k, z_{i+k}.

σ is a sigmoid function. Therefore, maximizing the function means a large inner product between the two vectors.

Wav2Vec

$$-E_{\tilde{z}\sim p_n}\left[\log \sigma\left(-\tilde{z}^{\top} h_k(c_i)\right)\right]$$

The discriminator term is the negative class binary log-likelihood between the prediction for timestep k, $h_k(c_i)$, and latents sampled from the distractor set p_n, z̃.

In practice, 10 latents are sampled to estimate the expectation.

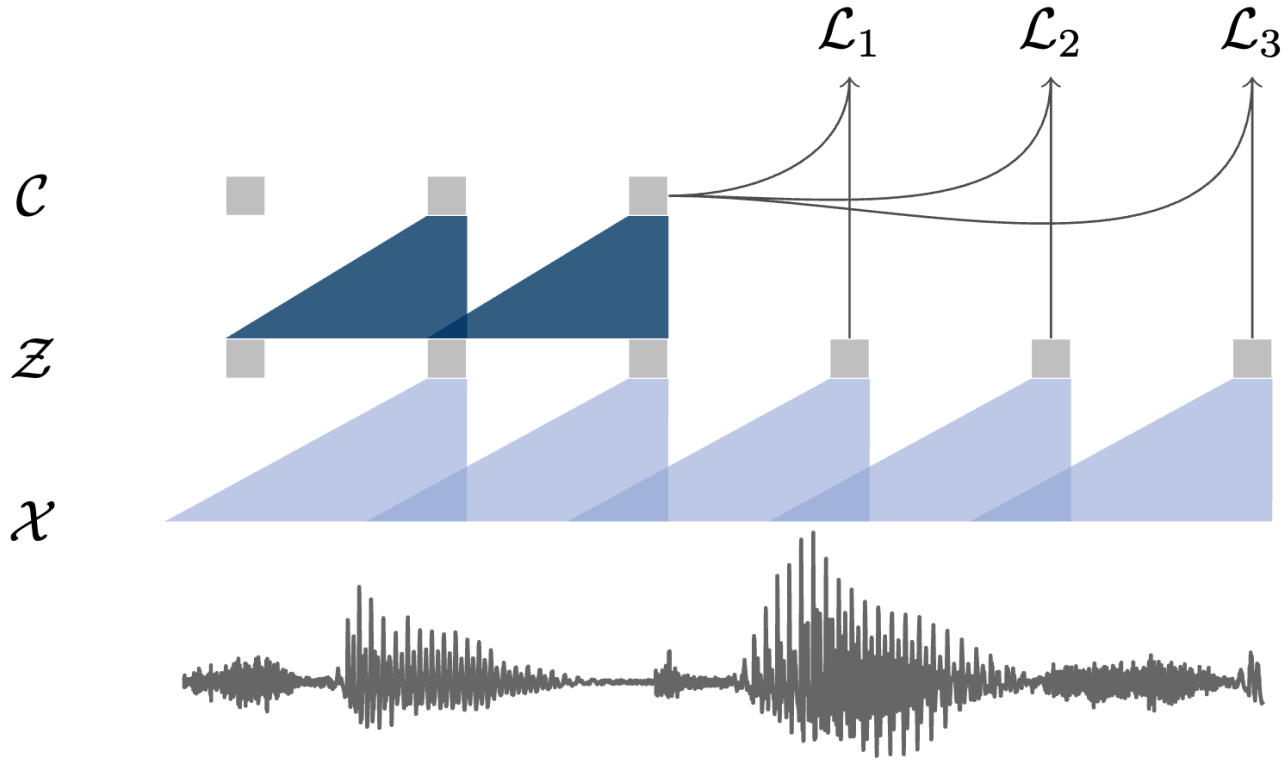
z̃ can be any other latent besides the current timestep, e.g.

$$i + \mathrm{k} = 145, T = 500 \rightarrow \boldsymbol{X} \sim p_n, \qquad \mathbf{index}(\boldsymbol{X}) \in \{120,5,6,490, \dots\}$$

Again, σ is a sigmoid function. Since we compute the negative inner product, we want to make the inner product as small as possible.

Wav2Vec

$$L_k = -\sum_{i=1}^{T-k} \log \sigma\left(z_{i+k}^{\top} h_k(c_i)\right) + \lambda E_{\tilde{z}\sim p_n}\left[\log \sigma\left(-\tilde{z}^{\top} h_k(c_i)\right)\right]$$



L_1 : $h_1(c_3) \rightarrow$ $\leftarrow z_4$
$h_1(c_3) \leftarrow$ $\rightarrow z_1, z_5, z_6$

L_2 : $h_2(c_3) \rightarrow$ $\leftarrow z_5$
$h_2(c_3) \leftarrow$ $\rightarrow z_2, z_4, z_6$

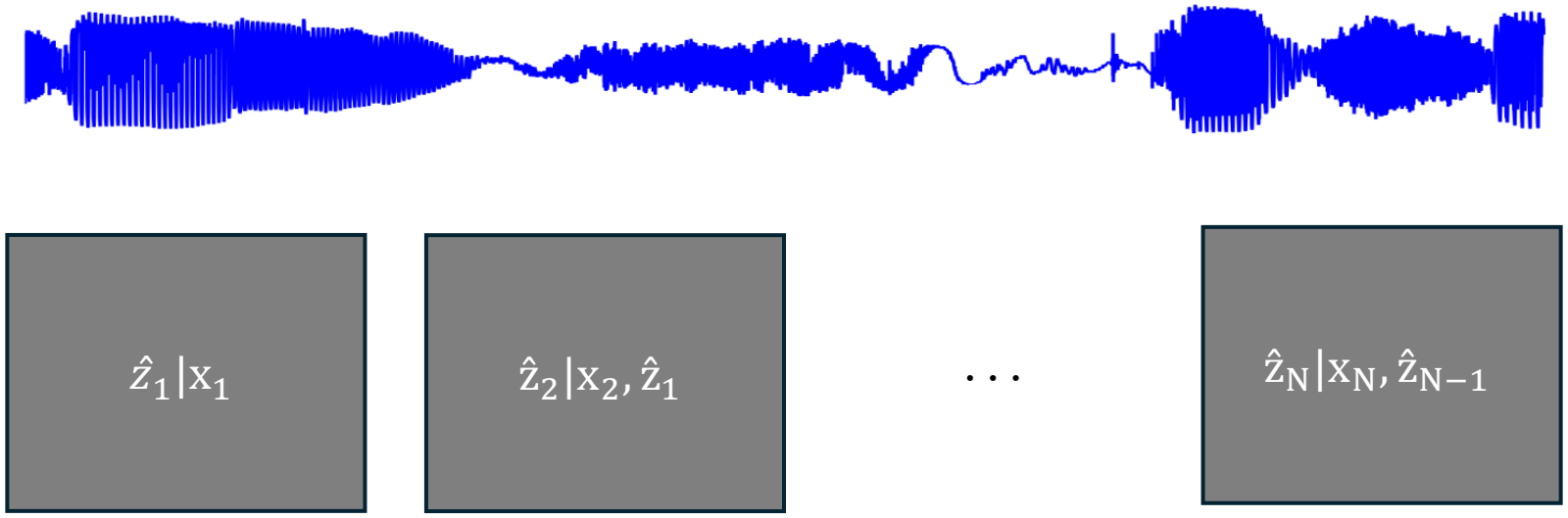
L_3 : $h_3(c_3) \rightarrow$ $\leftarrow z_6$
$h_3(c_3) \leftarrow$ $\rightarrow z_1, z_5, z_3$

Wav2Vec

NOTE:

The inner products determine which representations are pulled together or pushed apart, while the binary log-loss converts those scores into a stable classification objective with diminishing gradients for already-correct pairs.

VQ-Wav2Vec



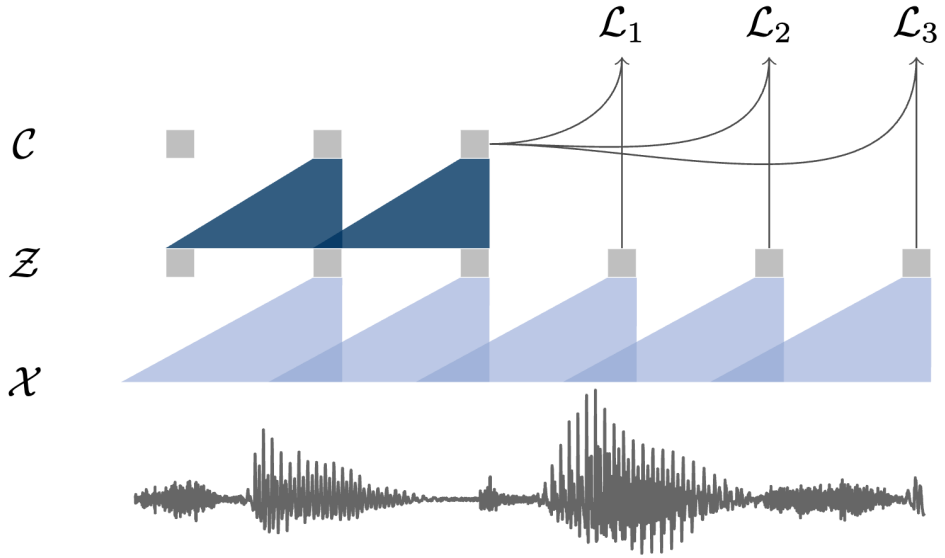
The goal of **VQ-Wav2Vec** is the same, learning a representation that contains sufficient temporal information for prediction, and various components of speech.

Surprisingly, it is the same method as **Wav2Vec** but with discretized latents $\hat{Z}$.

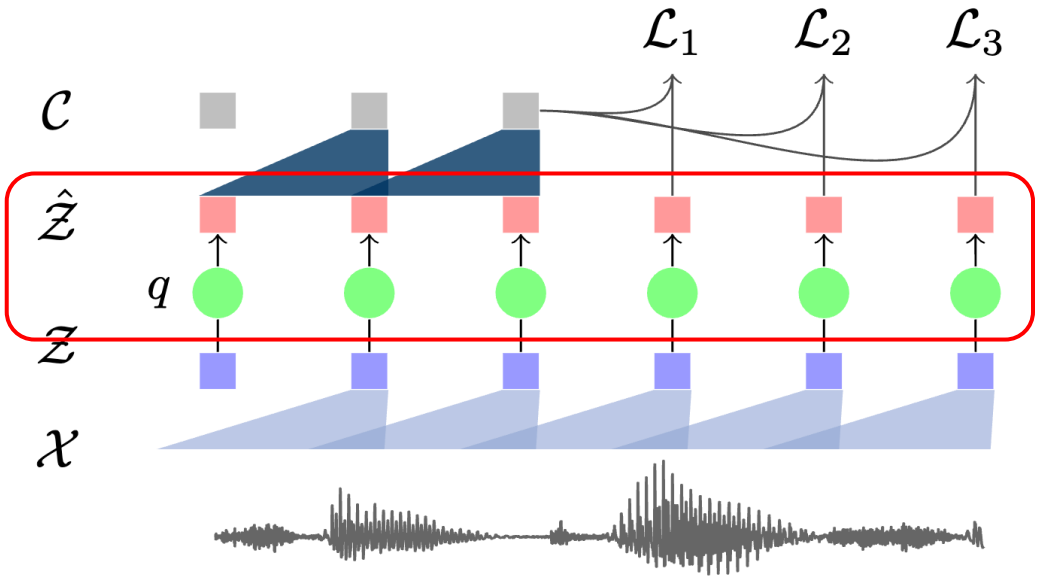
VQ-Wav2Vec

Architectures

Wav2Vec



VQ-Wav2Vec



Objectives

$$L_k = -\sum_{i=1}^{T-k} \log \sigma\left(z_{i+k}^{\top} h_k(c_i)\right) + \lambda E_{\tilde{z}\sim p_n}\left[\log \sigma\left(-\tilde{z}^{\top} h_k(c_i)\right)\right]$$

$$L_k = -\sum_{i=1}^{T-k} \log \sigma\left(\hat{z}_{i+k}^{\top} h_k(c_i)\right) + \lambda E_{\tilde{z}\sim p_n}\left[\log \sigma\left(-\tilde{\hat{z}}^{\top} h_k(c_i)\right)\right]$$

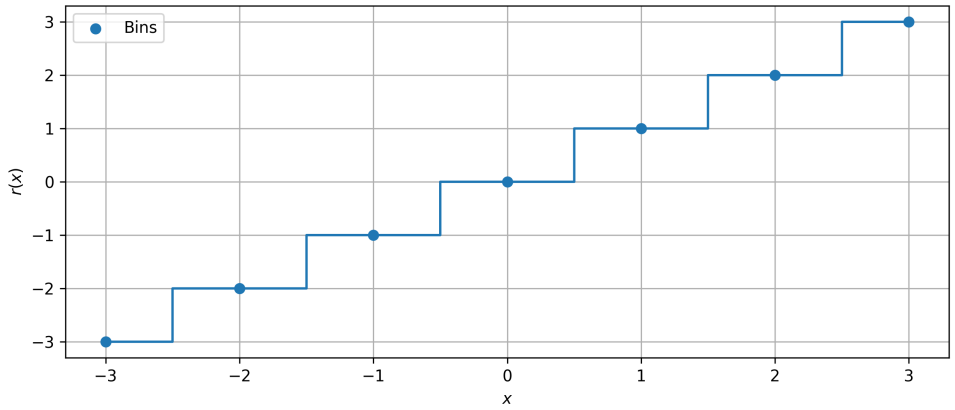
VQ-Wav2Vec

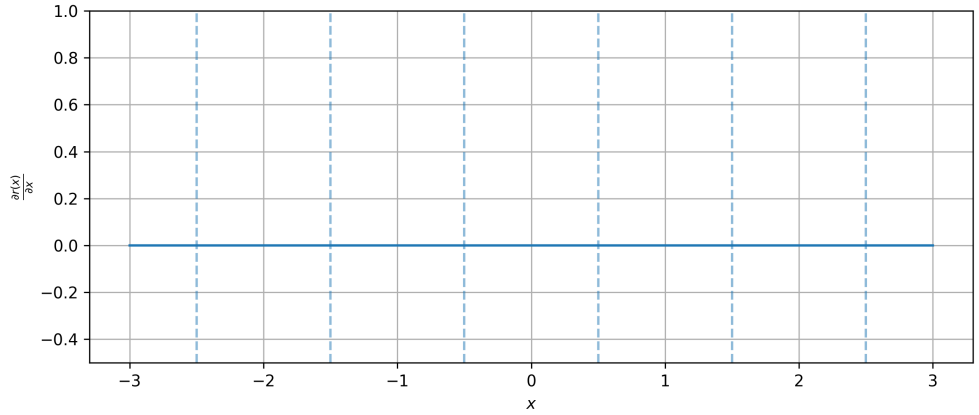
The key to VQ-Wav2Vec is the discretization module. Two quantization modules are introduced, but we will discuss “hard” Gumbel-Softmax.

First, lets detail a problem that occurs with quantization in models trained with gradient descent.

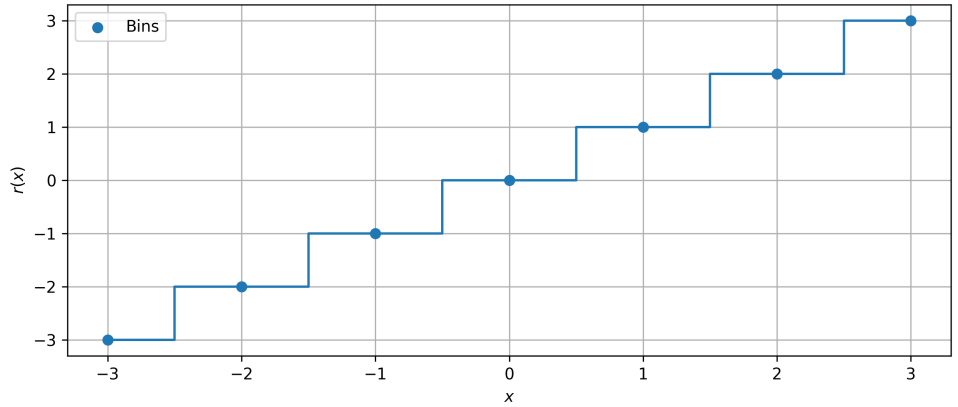
Let's consider what happens with a simple quantizer such as the round function, $r(x) = \text{round}(x)$.

Plotting r(x) and $\frac{\partial r(x)}{\partial x}$ we see:





VQ-Wav2Vec



The function $r(x)$ is defined everywhere.

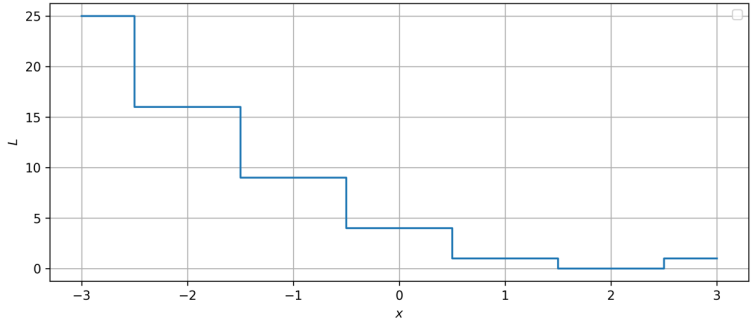
Let's compute the forward pass of this module. Our objective function will be $L = (r(x) - 2)^2$.

For the following values of x:

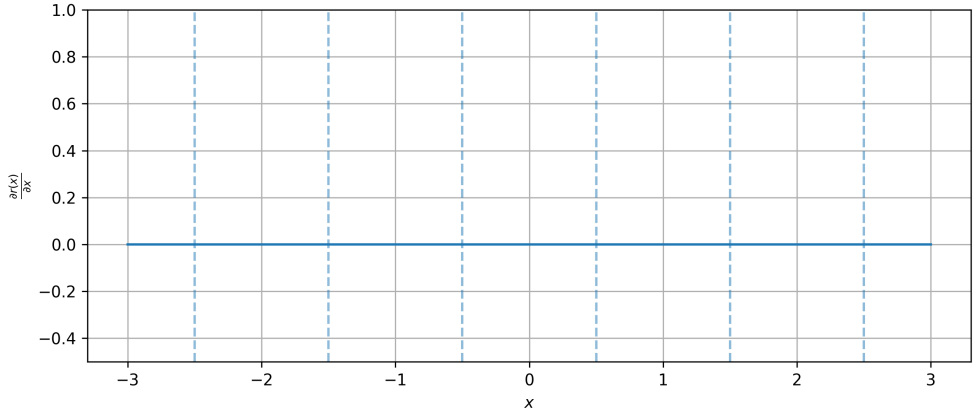
$$x_1 = -1.5 \rightarrow r(-1.5) = -1 \rightarrow L_1 = (-1 - 2)^2 = 9$$

$$x_2 = 0.7 \rightarrow r(0.7) = 1 \rightarrow L_2 = (1 - 2)^2 = 1$$

$$x_3 = 3 \rightarrow r(3) = 3 \rightarrow L_3 = (3 - 2)^2 = 1$$



VQ-Wav2Vec



That looked okay, we have loss values at least. Now let's compute the backward pass.

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial r}\frac{\partial r}{\partial x}, \qquad \frac{\partial L}{\partial r} = 2 \cdot (r(x) - 2), \qquad \frac{\partial r}{\partial x}$$

And finally looking at some values:

$$x_1 = -1.5 \rightarrow \frac{\partial L}{\partial r} = 2 \cdot (-1 - 2) = -6, \rightarrow \frac{\partial r}{\partial x} = \text{UNDEFINED} \rightarrow \frac{\partial L}{\partial x} = -6 \cdot \text{UNDEFINED} = \text{UNDEFINED}$$

$$x_2 = 0.7 \rightarrow \frac{\partial L}{\partial r} = 2 \cdot (1 - 2) = -2, \rightarrow \frac{\partial r}{\partial x} = 0 \rightarrow \frac{\partial L}{\partial x} = -2 \cdot 0 = 0$$

$$x_3 = 3 \rightarrow \frac{\partial L}{\partial r} = 2 \cdot (3 - 2) = 2, \rightarrow \frac{\partial r}{\partial x} = 0 \rightarrow \frac{\partial L}{\partial x} = 2 \cdot 0 = 0$$

VQ-Wav2Vec

Forward

$$x_3 = 3 \rightarrow r(3) = 3 \rightarrow L_3 = (3-2)^2 = 1$$

Backward

$$x_2 = 0.7 \rightarrow \frac{\partial L}{\partial r} = 2 \cdot (1-2) = -2, \rightarrow \frac{\partial r}{\partial x} = 0 \rightarrow \frac{\partial L}{\partial x} = -2 \cdot 0 = 0$$

As we have seen, with a quantization in a model that is updated via gradient descent, the forward pass works functionally the same but due to the quantization functions being constant everywhere except at discontinuities, the gradient is therefore 0, meaning we cannot update any parameters that occur **BEFORE** the quantization operation.

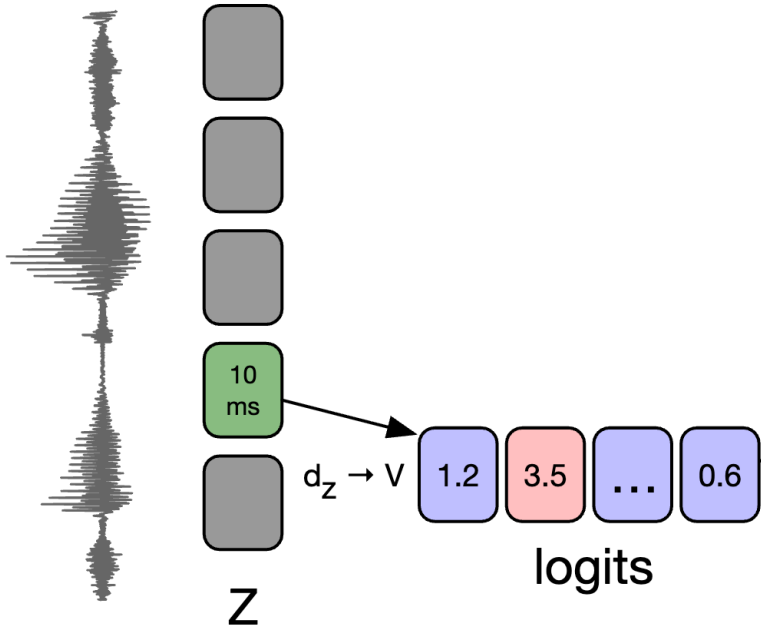
This is the **discretization problem**.

VQ-Wav2Vec

Now let's see how ”hard” Gumbel Softmax aims to relax this problem.

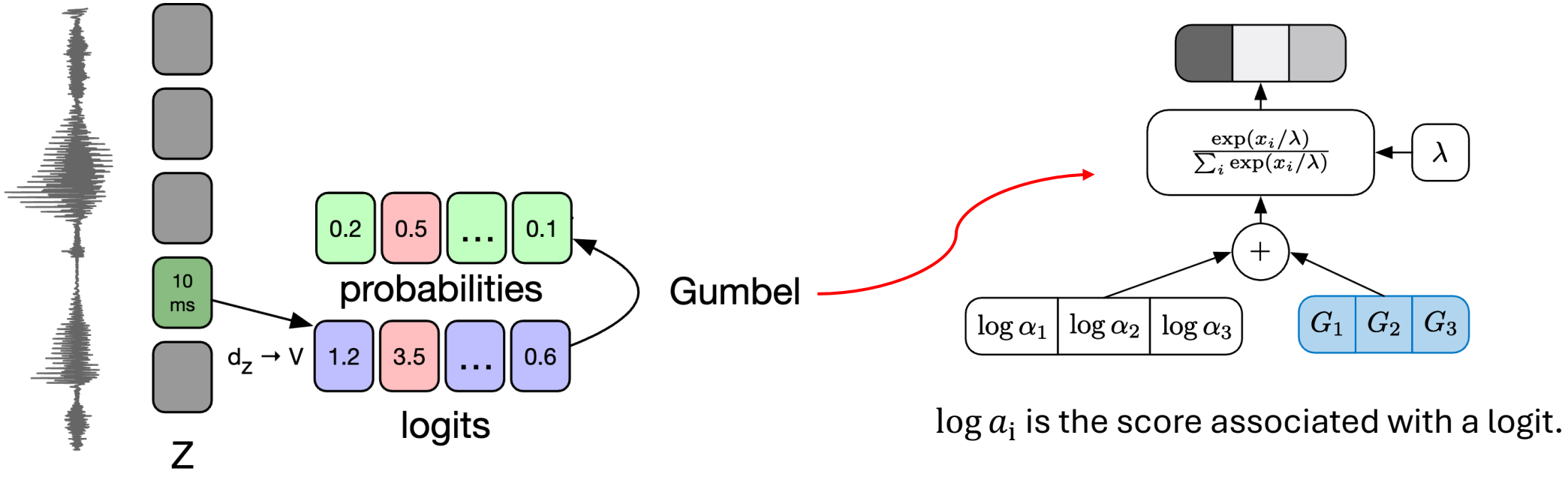
VQ-Wav2Vec

We first start from the continuous latent space.



VQ-Wav2Vec

Next, we apply Gumbel-Softmax to the continuous vectors to get a probability distribution for each codeword.

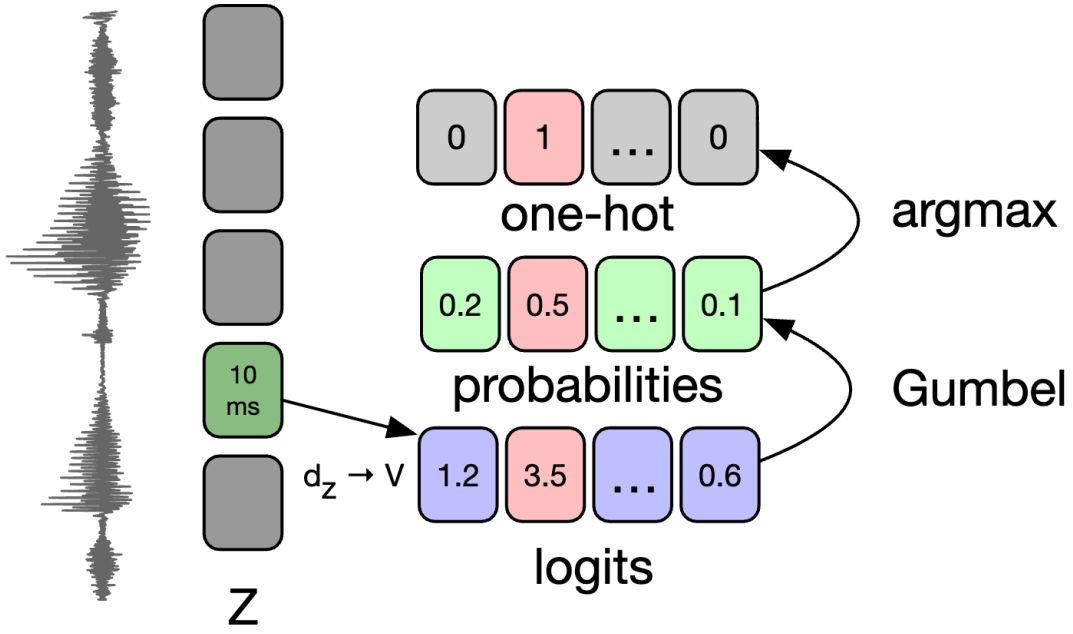


$\log a_i$ is the score associated with a logit.

G_i is Gumbel noise added to the score.

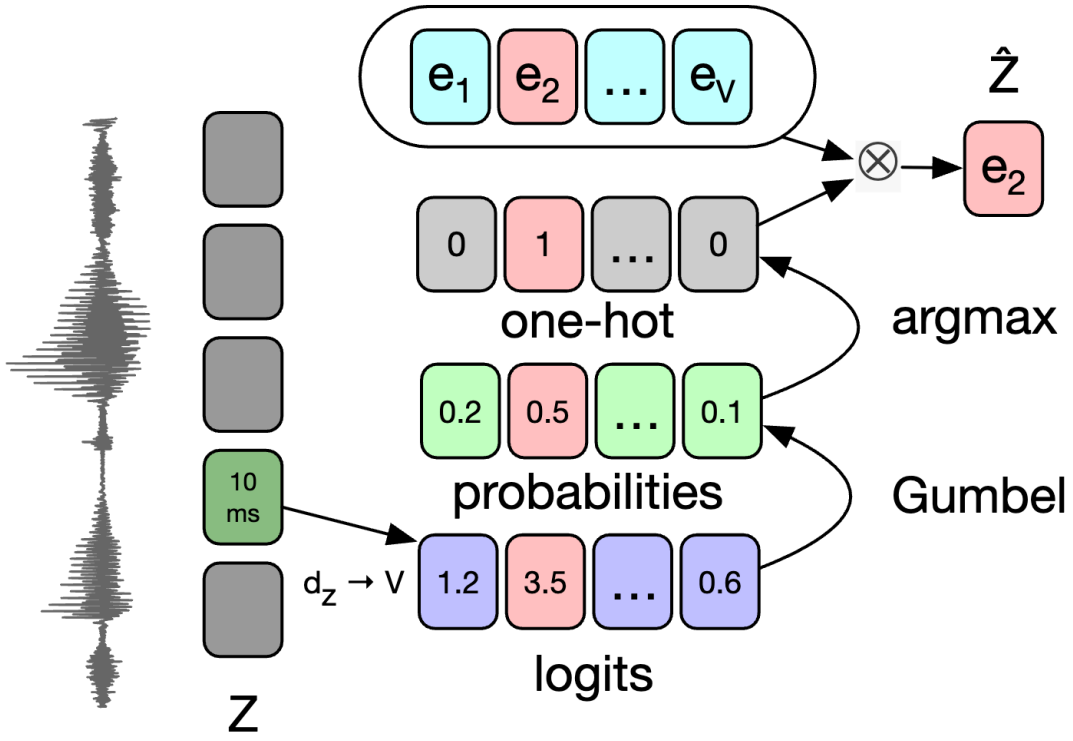
VQ-Wav2Vec

Using ”hard” Gumbel-Softmax, we compute the argmax of the distribution (also achieved as $\lambda \rightarrow 0^{+}$).



VQ-Wav2Vec

Using the one-hot vector we compute the codeword using an embedding layer.



VQ-Wav2Vec

But wait, didn't we show that the quantization (i.e., the argmax operation) is non-differentiable.

The solution to this problem is a **straight through estimator**.

```
def straight_through_estimator(self):
    return y_hard - y_soft.detach() + y_soft
```

VQ-Wav2Vec

But wait, didn't we show that the quantization (i.e., the argmax operation) is non-differentiable.

The solution to this problem is a **straight through estimator**.

```
def straight_through_estimator(self):
    return y_hard - y_soft
```

During the **forward pass**, both the second and third terms cancel, so the argmax is computed correctly.

VQ-Wav2Vec

But wait, didn't we show that the quantization (i.e., the argmax operation) is non-differentiable.

The solution to this problem is a **straight through estimator**.

```
def straight_through_estimator(self):
    return y_hard - y_soft.detach() + y_soft
```

During the **backward pass**, both the first and second terms have gradients that are zero everywhere, but the third term has well defined gradients which correspond to the Gumbel-Softmax distribution of the encoded vectors.

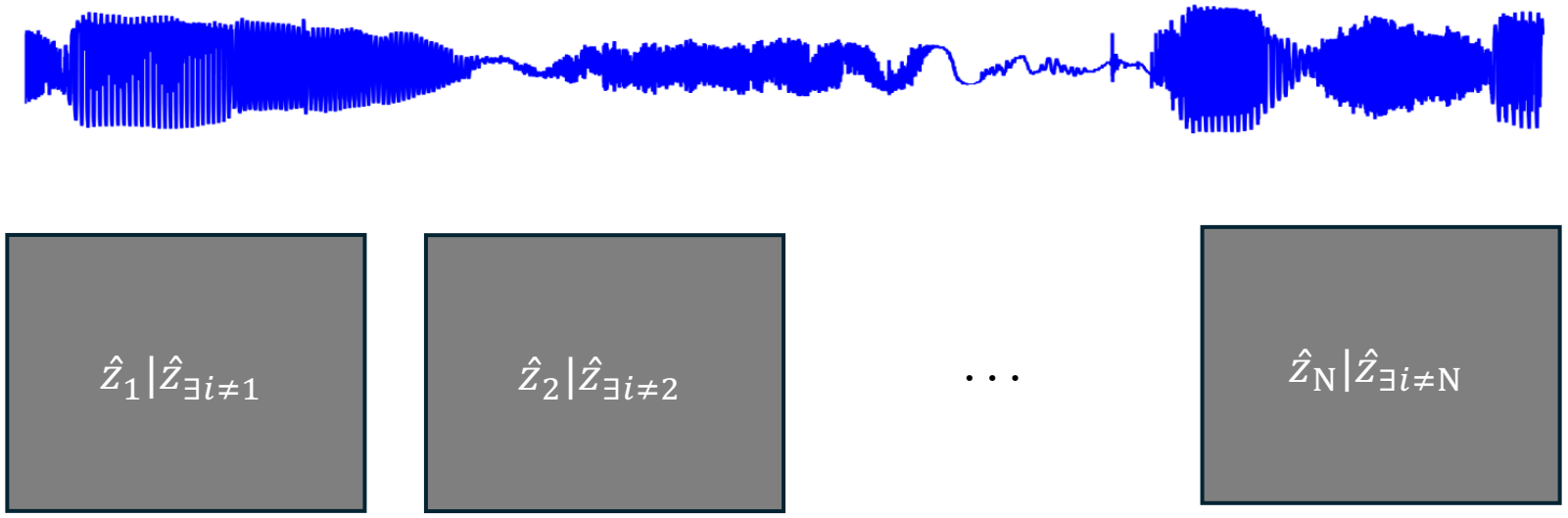
Although the gradient signal can be a little noisy, given that the gradient is w.r.t the one-hot vector and not the dense representation, overtime it is fairly stable.

VQ-Wav2Vec

We now know, (1) how the Gumbel-Softmax method approximates categorical distributions and (2) how straight through estimators can enable quantization in gradient descent optimization.

The rest of **VQ-Wav2Vec** remains the exact same as **Wav2Vec**.

Wav2Vec 2.0



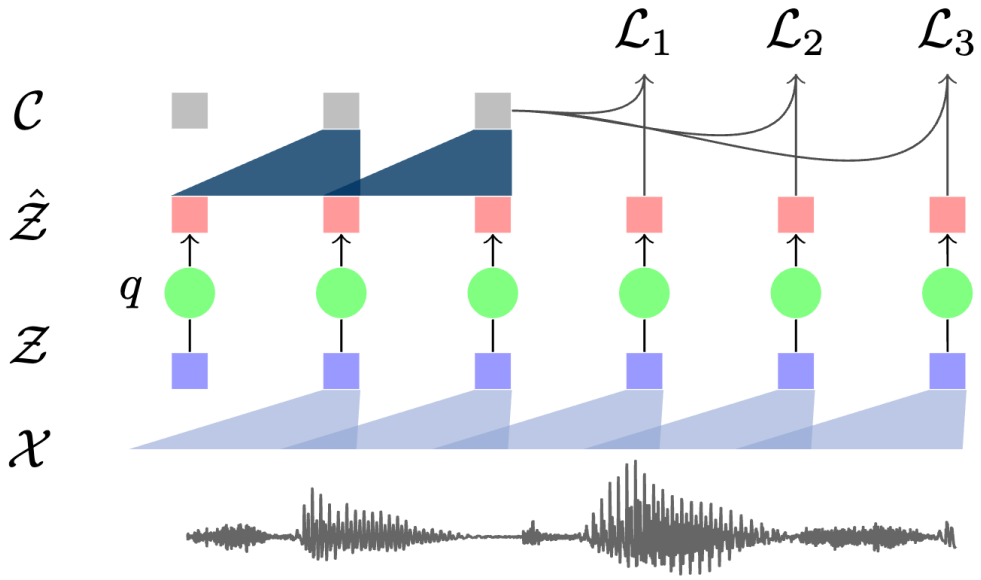
The goal of **Wav2Vec 2.0** is the same, learning a representation. We now aim to identify the latent quantized targets from the other latents.

Wav2Vec 2.0 builds upon **VQ-Wav2Vec** by (1) changing the objective, (2) changing the context network architecture, and (3) using masked prediction.

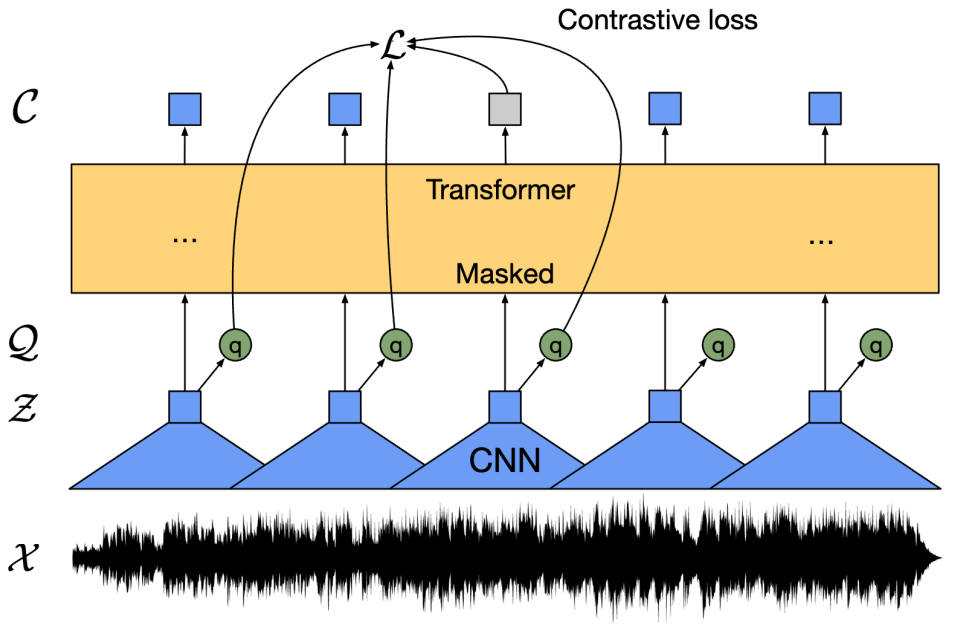
Wav2Vec 2.0

Architectures

VQ-Wav2Vec



Wav2Vec 2.0



Objectives

$$L_k = -\sum_{i=1}^{T-k} \log \sigma\left(\hat{z}_{i+k}^{\top} h_k(c_i)\right) + \lambda E_{\tilde{z}\sim p_n}\left[\log \sigma\left(-\tilde{\hat{z}}^{\top} h_k(c_i)\right)\right]$$

$$\mathrm{L} = -\log \frac{e^{s(c_t,q_t)/\kappa}}{\sum_{\tilde{q}\sim Q_t} e^{s(c_t,\tilde{q})/\kappa}} + \frac{\alpha}{GV}\sum_{g=1}^{G}\sum_{v=1}^{V} \bar{p}_{g,v}\, log\, \bar{p}_{g,v}$$

Wav2Vec 2.0

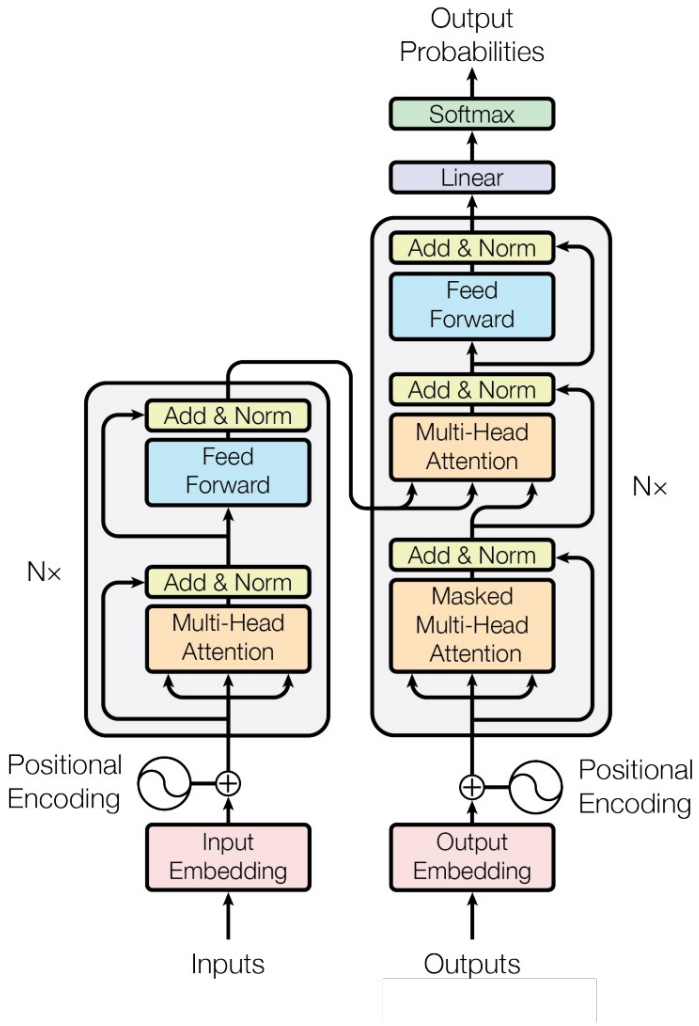
The first change is the architecture. We do not use **causal convolution** blocks in the context network anymore.

Wav2Vec 2.0

We now use a **Transformer**.

BERT

Encoder

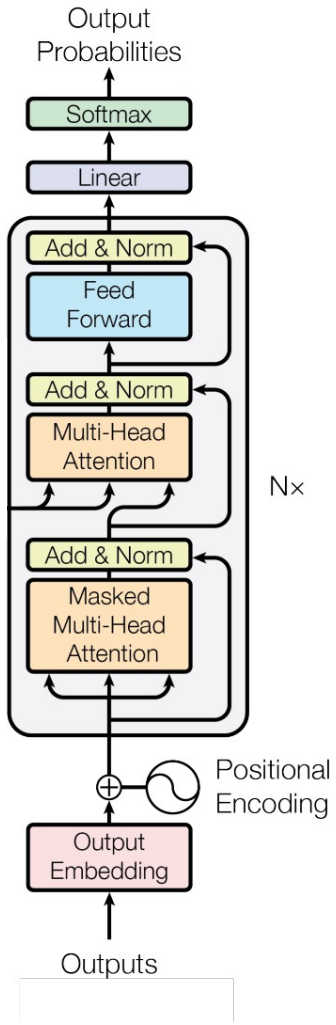


GPT

Decoder

Wav2Vec 2.0

First, throw away the decoder as this is a generative branch that we do not need/have use for.



GPT

Decoder

Wav2Vec 2.0

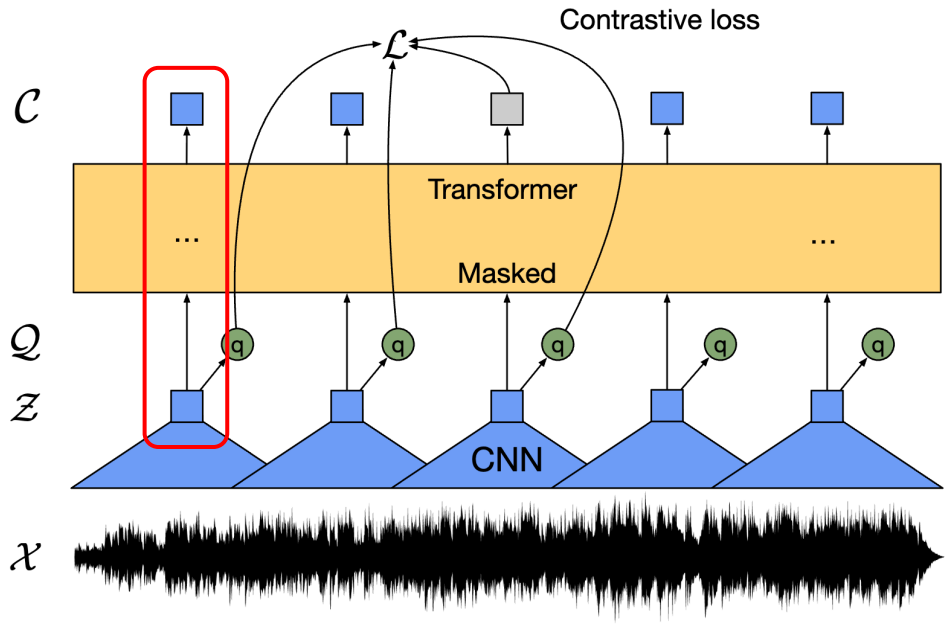
Now we are left with the encoder only, which was popularized by the method **BERT**.

The goal of this model is to take in a sequence of tokens and produce a sequence of embeddings.

In this scenario, our input is the **continuous latent space *z*** (output of the encoder).

BERT

Encoder



Wav2Vec 2.0

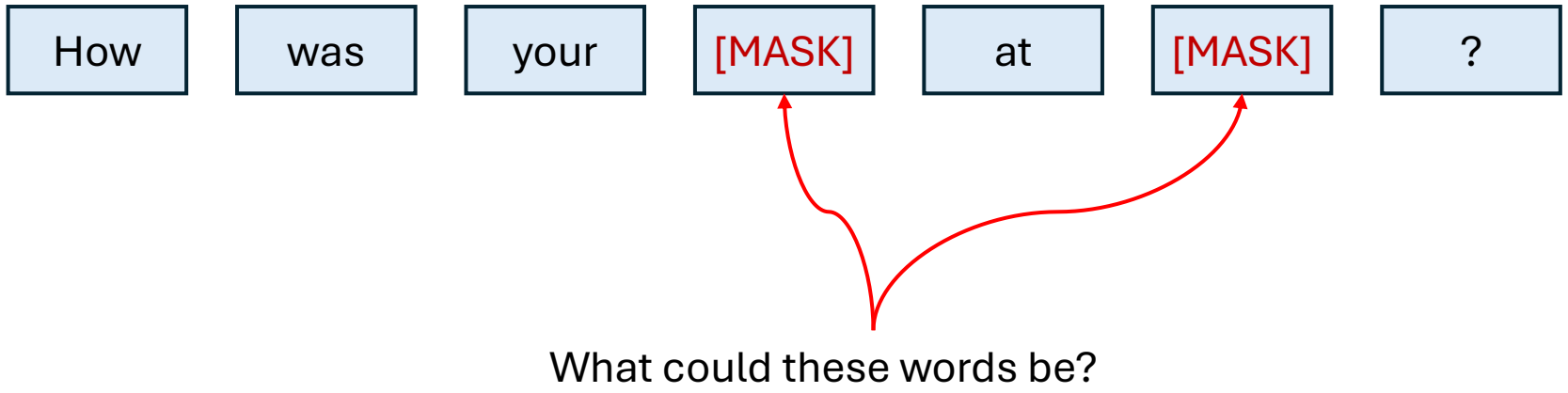
Next, we need to detail **masking**.

The idea of masking is to use the context to fill in gaps.

Wav2Vec 2.0

Next, we need to detail **masking**.

The idea of masking is to use the context to fill in gaps.



Wav2Vec 2.0

Next, we need to detail **masking**.

The idea of masking is to use the context to fill in gaps.

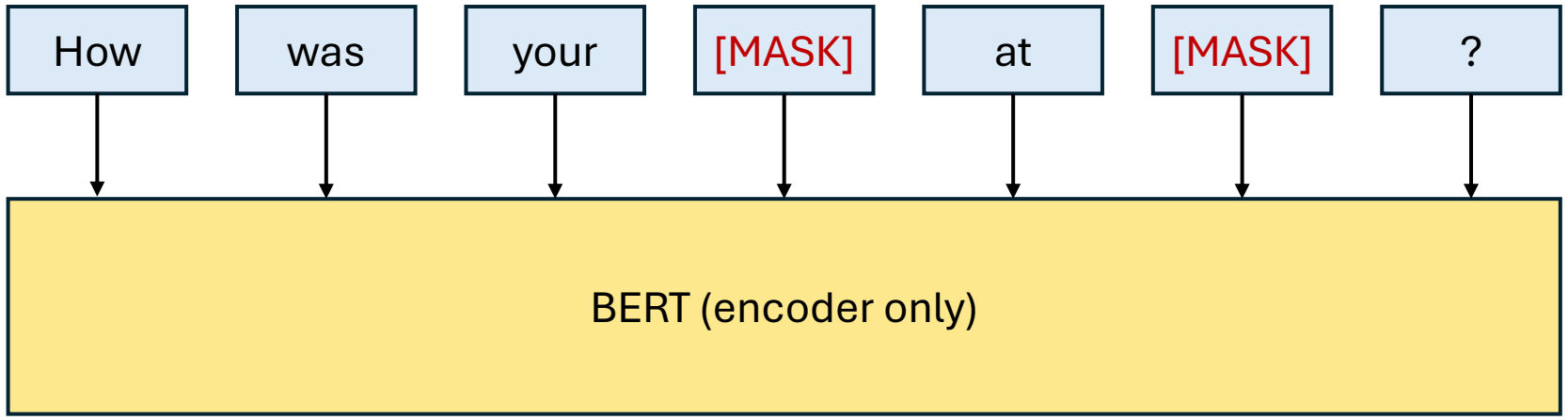
1. day
2. ride
3. flight

1. work
2. NYC
3. Vegas

Wav2Vec 2.0

Next, we need to detail **masking**.

The idea of masking is to use the context to fill in gaps.

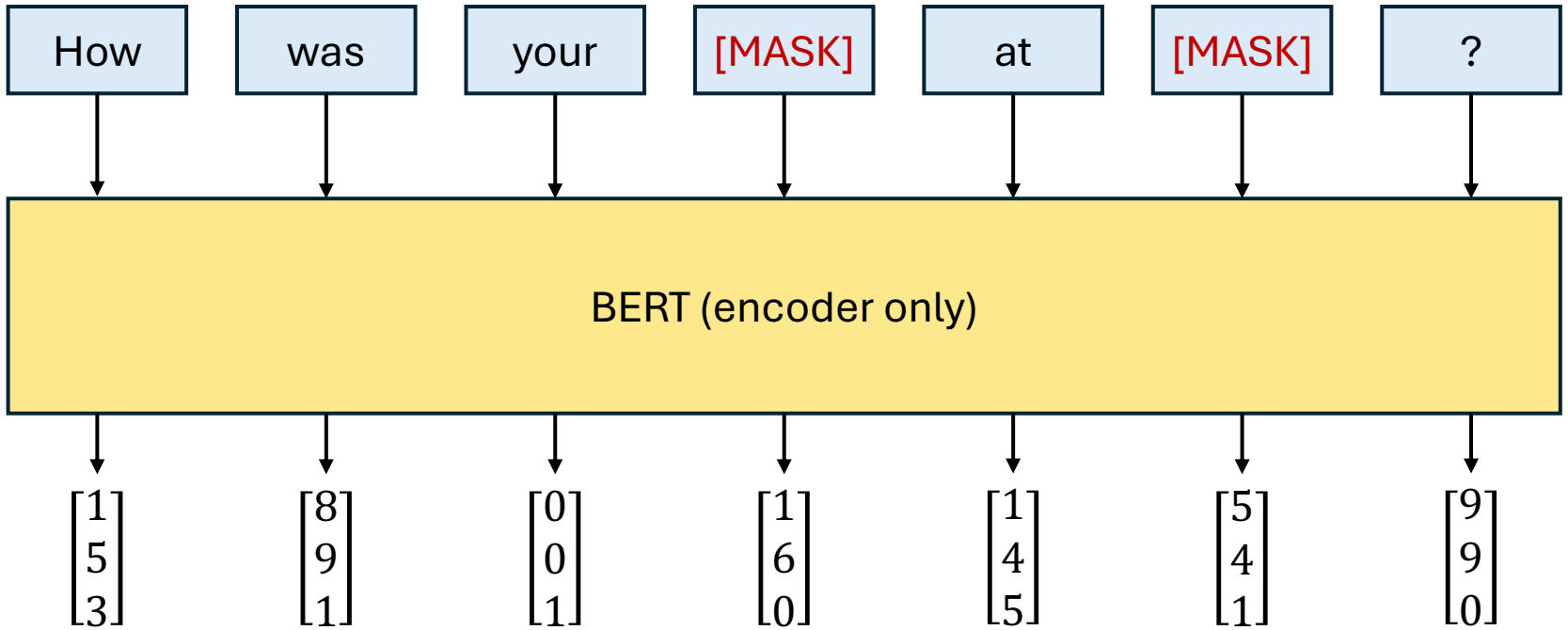


NOTE: Transformer models (such as BERT) have a fixed context length. The benefit of BERT is we get an embedding for all inputs in parallel.

Wav2Vec 2.0

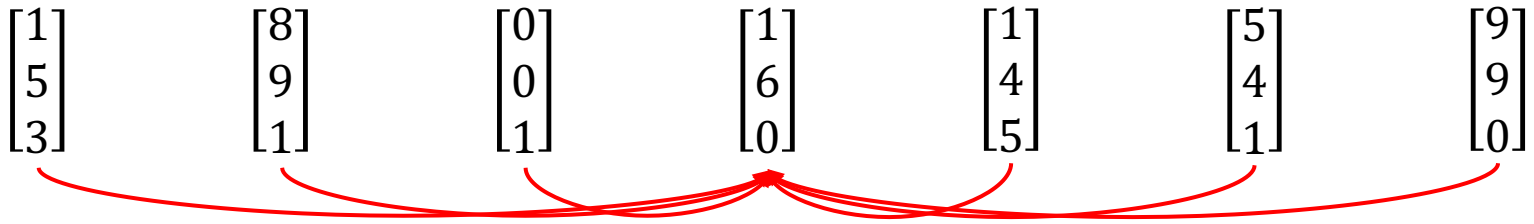
Next, we need to detail **masking**.

The idea of masking is to use the context to fill in gaps.



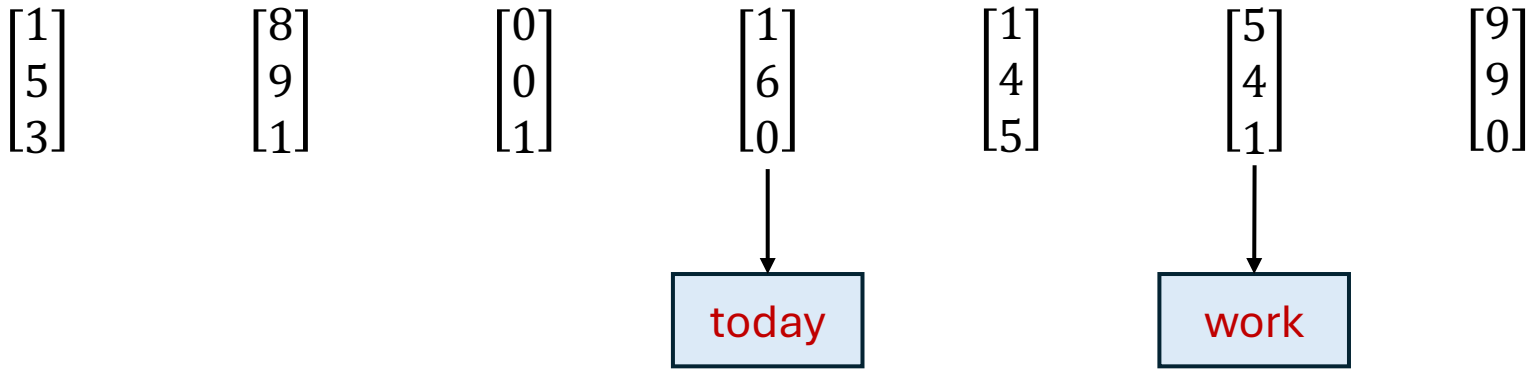
Wav2Vec 2.0

The goal of these embeddings is to embed information from the context.



Wav2Vec 2.0

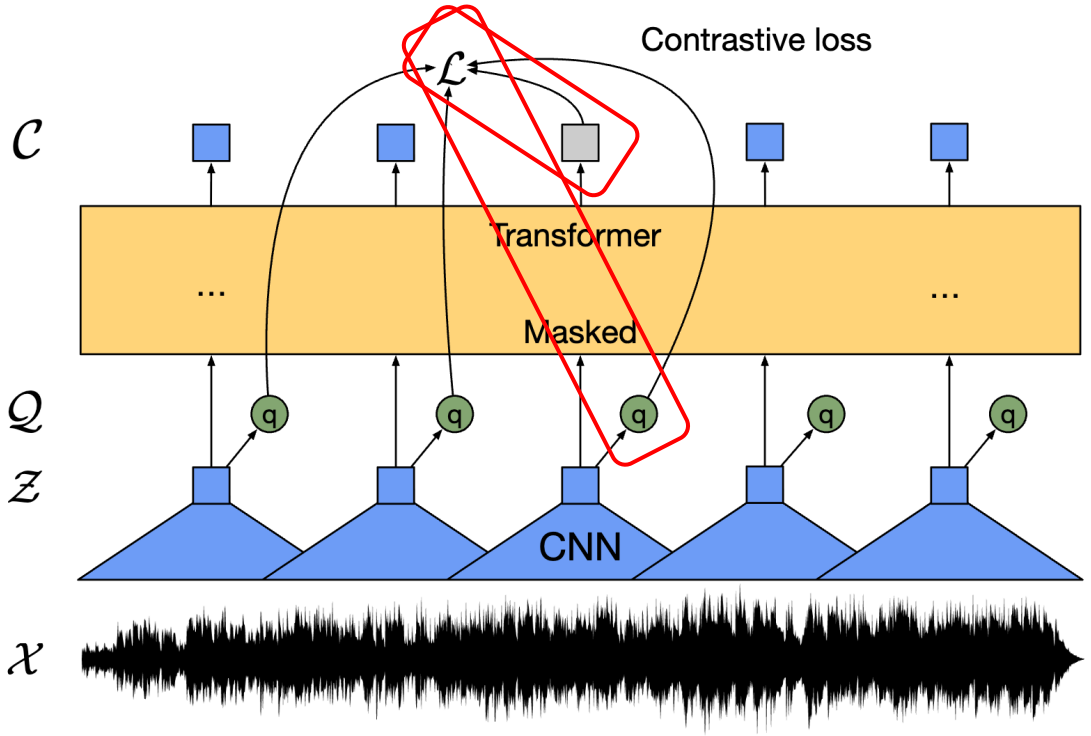
The goal of these embeddings is to embed information from the context.



We predict the masked tokens from the context.

Wav2Vec 2.0

We use the quantized target as the ground truth embedding.



Wav2Vec 2.0

Note:

The masked vector is just a single learnable vector. It is not a function of the input. It could be fixed, such as $\vec{0}$ or $\vec{1}$, but this may not align with the range learned by the encoder, so it is more stable to use a learnable vector.

Wav2Vec 2.0

Finally, we can cover the objective.

The first component is the **contrastive loss**. Maximizing the probability of the target vector:

$$L_m = -\log \frac{e^{s(c_t, q_t)/\kappa}}{\sum_{\tilde{q} \sim Q_t} e^{s(c_t, \tilde{q})/\kappa}}$$

where $s(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||}$, c_t is the masked token context embedding, q_t is the quantized target (premask) embedding, $\tilde{q}$ are randomly sampled quantized latents (including q_t among distractors), and κ is the temperature variable for Softmax distributions.

Minimizing this loss means (1) increasing the similarity with the quantized target and (2) decreasing the similarity with other quantized latents.

Wav2Vec 2.0

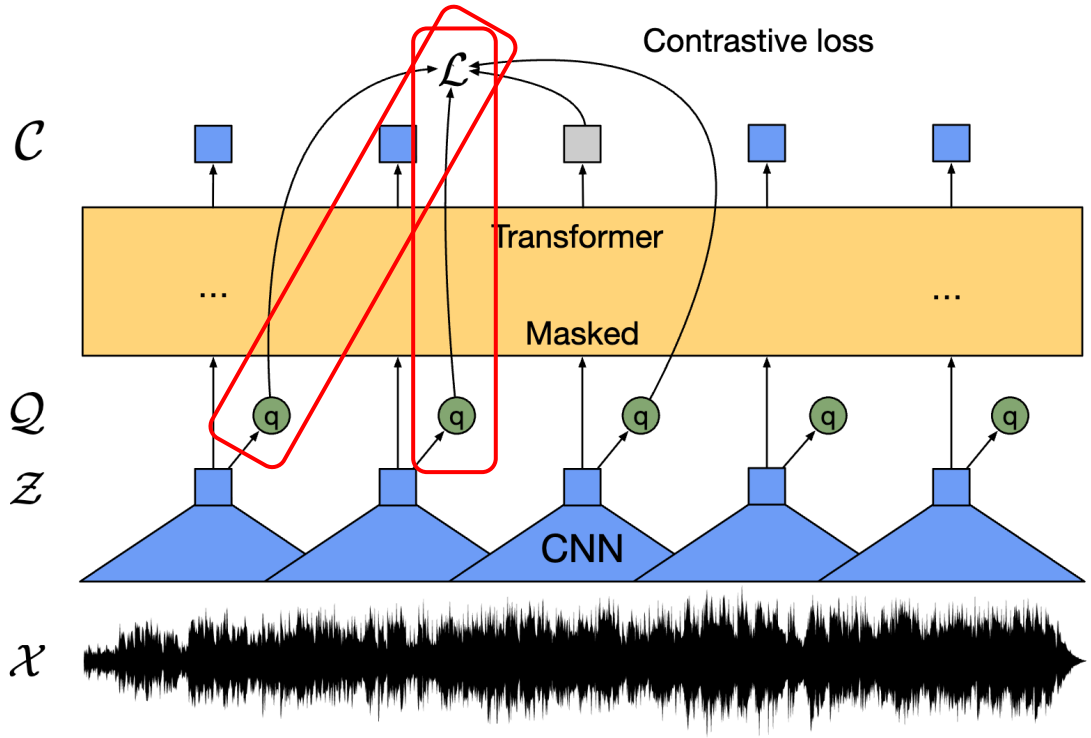
Recap:

Maximize the probability of predicting the target vector from the masked context embedding, $p(q_t|c_t)$, whilst lowering the probability of predicting the other non-target vectors, $p(\tilde{q}|c_t)$.

Wav2Vec 2.0

Recap:

We do not sample all the codewords for the set of distractors.



Wav2Vec 2.0

Next is the **diversity loss**.

$$L_d = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \, log \, \bar{p}_{g,v}$$

where G is the number of codeword groups, V is the number of codeword entries (for each group), $\bar{p}_{g,v}$ is the average Softmax selection probability for entry v in group g.

This equation is the negative entropy over each group. Therefore, minimizing this loss means **maximizing entropy of the average codeword usage**.

More entropy, more uniform.

Wav2Vec 2.0

Finally, our objective is:

$$\mathrm{L} = L_m + \alpha L_d = -\log \frac{e^{s(c_t, q_t)/\kappa}}{\sum_{\tilde{q} \sim Q_t} e^{s(c_t, \tilde{q})/\kappa}} + \frac{\alpha}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \, log \, \bar{p}_{g,v}$$

Altogether, we want to (1) maximize the probability of predicting the quantized target from the masked content embedding and (2) ensure the quantization has high entropy with uniform probability usage for each codeword.

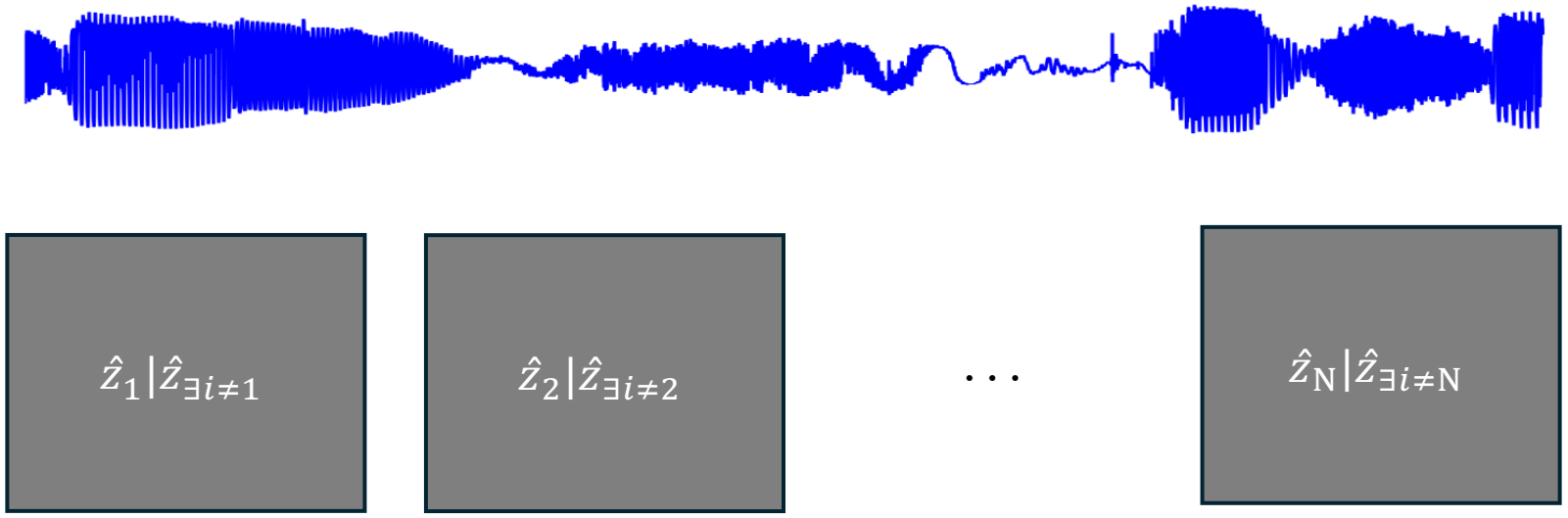
Wav2Vec 2.0

Note:

This objective changes things up a decent amount compared to **Wav2Vec/VQ-Wav2Vec**. This masked prediction objective has been shown to be correlated highly with learning structured latent spaces (*which is our goal*). To evaluate each of these representation learning method, these embeddings are typically used as input for downstream tasks: classification, **automatic speech recognition**, **automatic speaker verification**, phenome recognition, etc.

Wav2Vec 2.0 showed major empirical gains over prior methods. It is still used as a baseline/component of many modern methods.

HuBERT

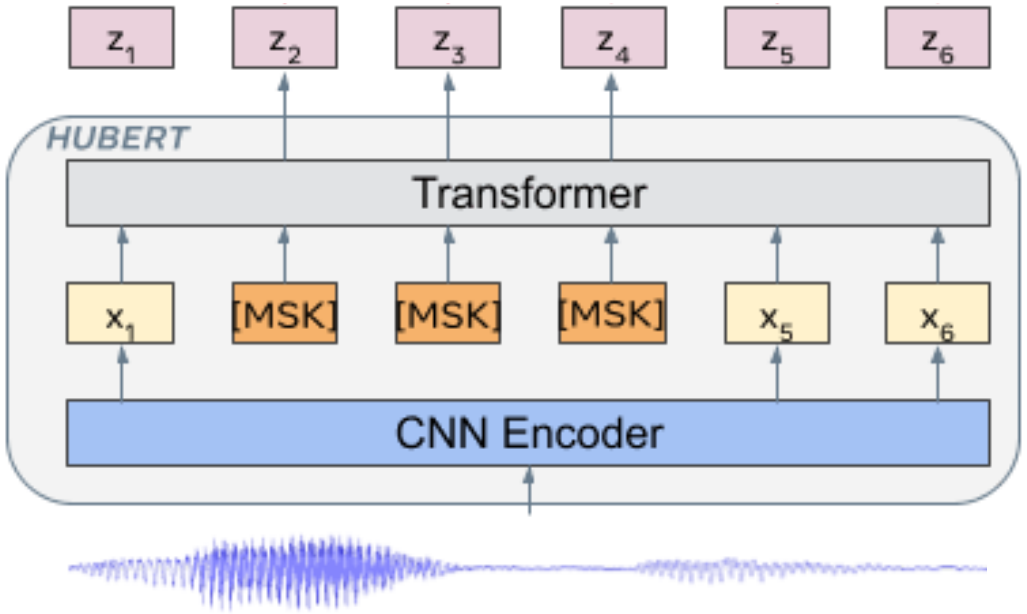


This model model builds off **BERT/Wav2Vec 2.0**, utilizing the same masked prediction task.

Compared to **Wav2Vec 2.0**, **HuBERT** (1) removes the differentiable vector quantization module and (2) adopts a multi-stage clustering task as target embeddings.

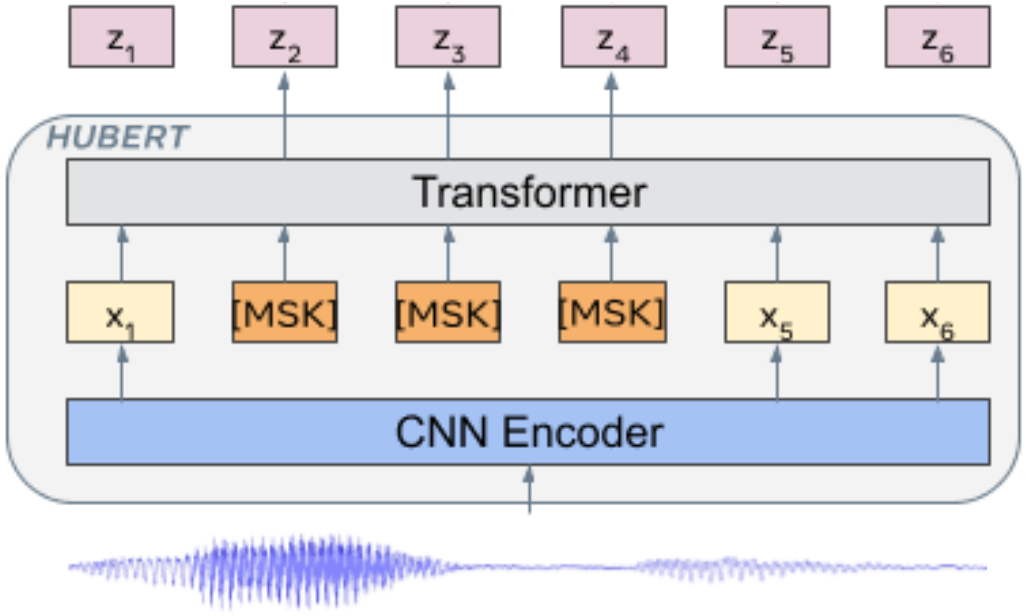
HuBERT

The architecture of **HuBERT** is nearly identical to **Wav2Vec 2.0**. The encoder is standard (**non-causal**) convolution blocks. The transformer block is again the BERT model. The mask is again a learnable vector.



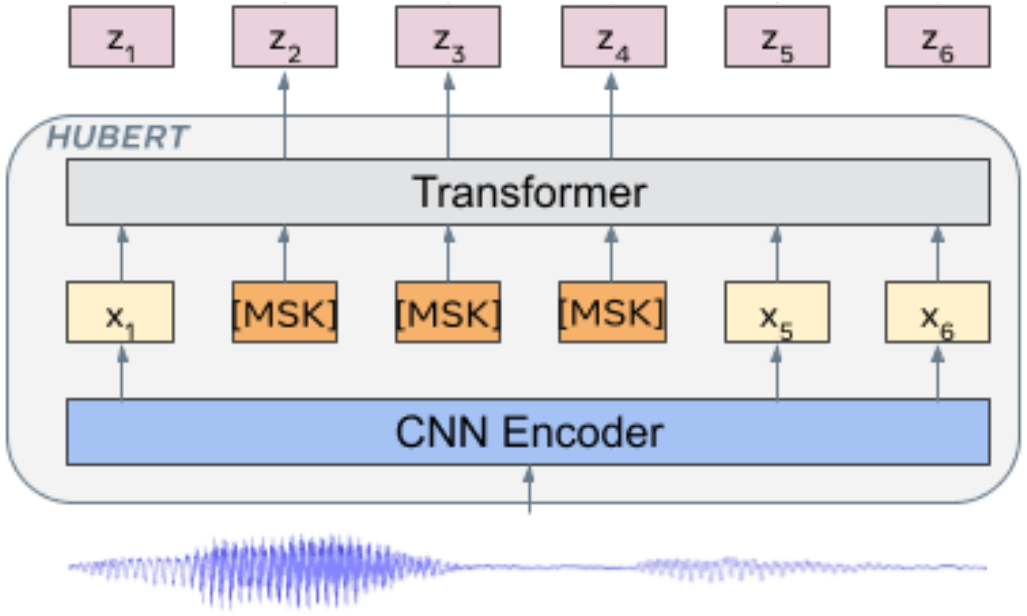
HuBERT

However, notice there is no quantization block. That brings us to the next point...



HuBERT

However, notice there is no quantization block. That brings us to the next point… **offline clustering**.



HuBERT

Offline clustering operates in 2 stages.

Stage 1 occurs before any training has begun. The goal is to do K-means clustering on the speech components.

First, from each utterance, $\boldsymbol{x_T}$, we compute the ***N* Mel-Frequency Cepstral Coefficients**, $\boldsymbol{m}_{\mathrm{N,t}}$, in frames of roughly 20-25ms. Along with this, we include the first and second order derivatives (computed via finite differences w.r.t time) of $\boldsymbol{m}_{\mathrm{N,t}}$, which is $\boldsymbol{\Delta m}_{\boldsymbol{N},\mathrm{t}}$ and $\boldsymbol{\Delta^2 m}_{\boldsymbol{N},\mathrm{t}}$ repspectively.

Next, we create the following feature vectors:

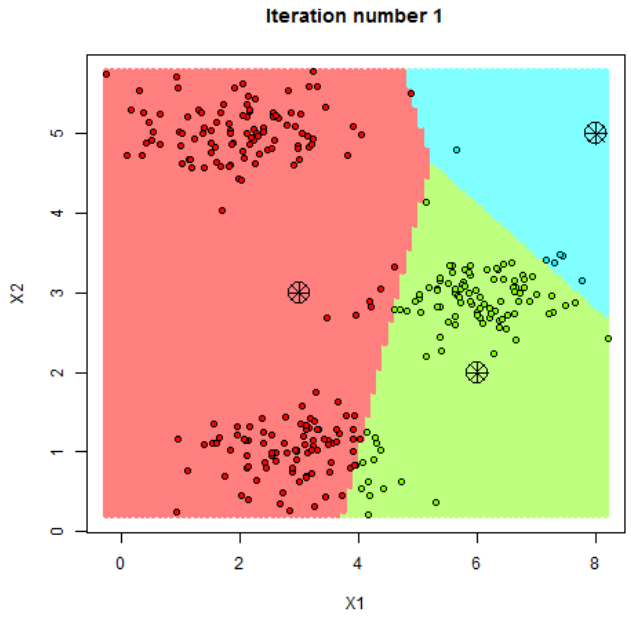
$$\mathbf{m}_{\mathrm{K,t}} = \begin{bmatrix} \boldsymbol{m}_{N,\mathrm{t}} \\ \boldsymbol{\Delta m}_{N,\mathrm{t}} \\ \boldsymbol{\Delta^2 m}_{N,\mathrm{t}} \end{bmatrix}, \forall x \in \mathbf{X}, \forall \mathrm{t} \in \boldsymbol{T}_x$$

where K is the number of features, $\boldsymbol{X}$ is the dataset of utterances, and $\boldsymbol{T}$ is the frames per utterance.

HuBERT

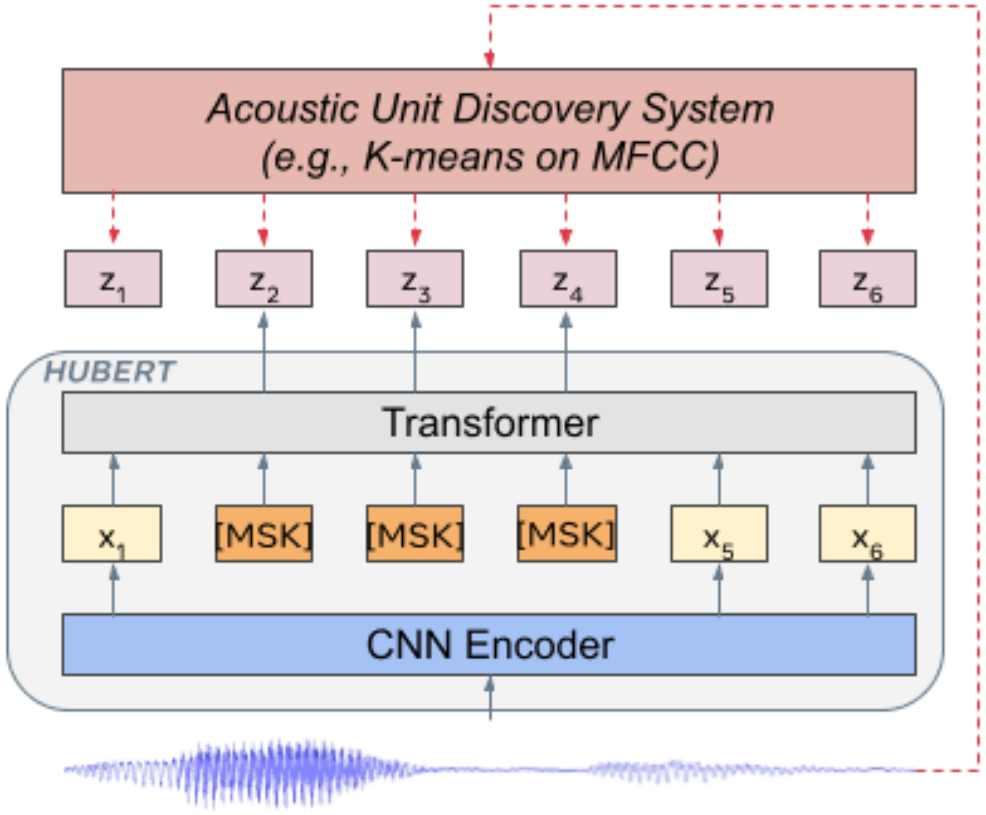
The resulting feature vectors are now all the same size, K. Some utterances get more representation than others depending on total length of utterance.

Next, we do **K-means** clustering on this feature space with K clusters (in this work they use $K = \mathbf{100}$).



HuBERT

Our objective has now become to predict the cluster for each masked frame. We now learn discrete embeddings **FROM** these clusters.



HuBERT

Written out, the objective function now becomes:

$$\mathrm{L} = -\log \frac{e^{s(e_j, c_t)/\kappa}}{\sum_{k=1}^{K} e^{s(e_k, c_t)/\kappa}}$$

where c_t the predicted embedding, e_j is the ground truth embedding, K is the number of classes/clusters, κ is the temperature for a Softmax-distribution, and $s(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||}$.

e_j and e_k are embeddings associated with the cluster labels, e.g.,

$x_1 \rightarrow z_1$ had a feature vector that was clustered into the 93rd class. We would create a learnable embedding, e_{93}, associated only with the 93rd class and that would be the target vector, e_j. We therefore want the predicted vector to be aligned with the learned embedding, $s(e_j, c_t) \uparrow$.

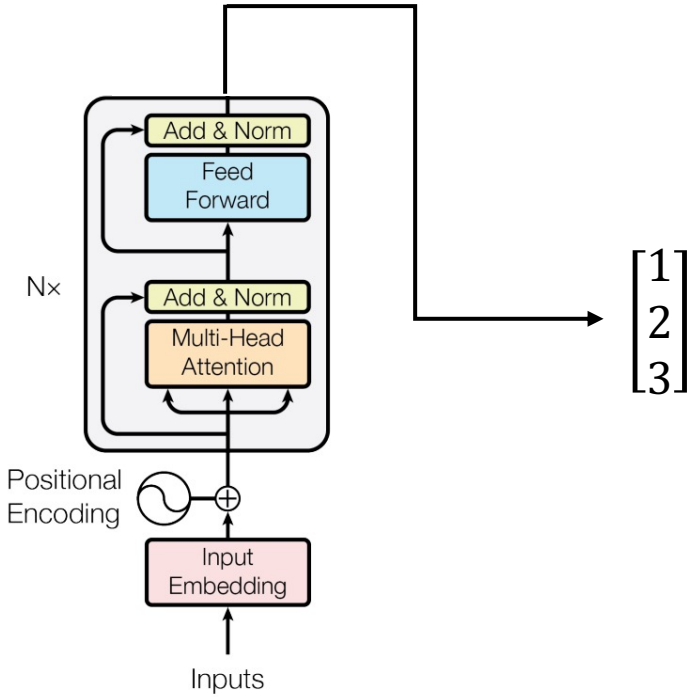
HuBERT

Note:

They also experiment with another term that predicts the non-masked token classes (the same loss but for every token not masked), i.e., $L = \alpha L_m + (1 - \alpha) L_u$, where L_m is the masked prediction and L_u is non-masked prediction.

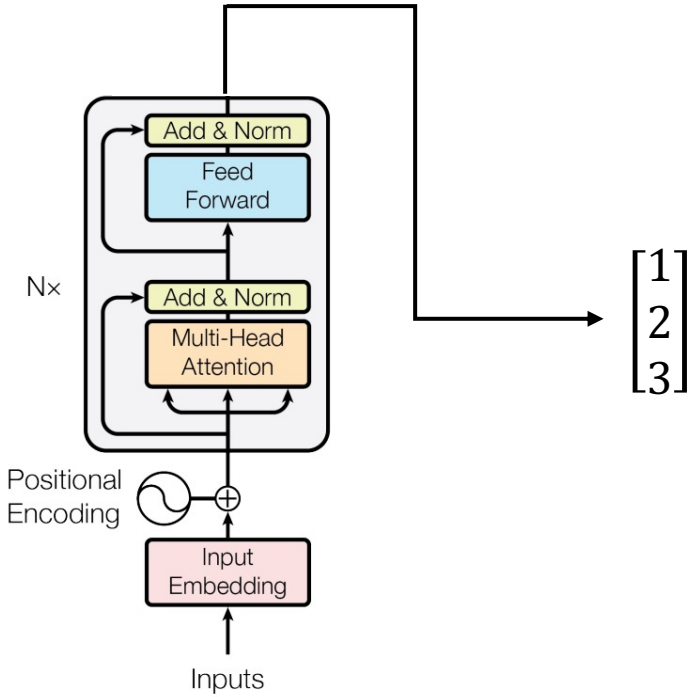
HuBERT

Stage 2 occurs after some training during stage 1 (in their work this is after 250k steps). The goal is to refine the cluster assignments.



HuBERT

To do so, all we do is extract an embedding from the transformer model $\forall x \in \mathbf{X}, \forall t \in \boldsymbol{T}_x$. This is typically from a shallower layer (in their work they use layer 6).



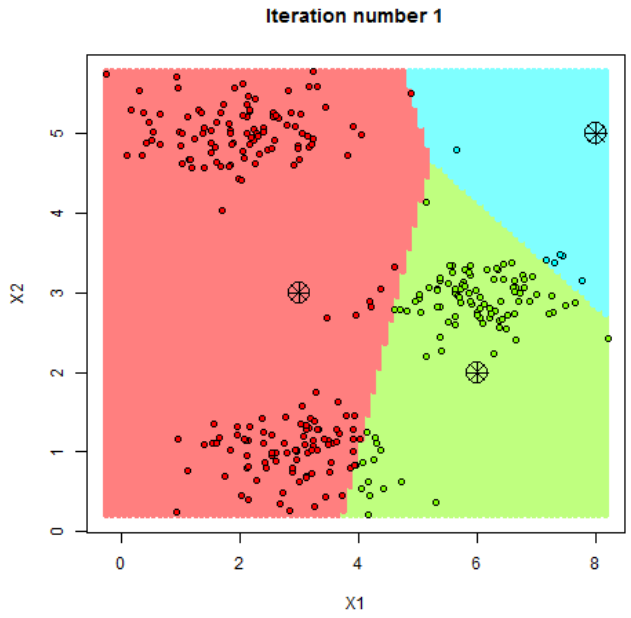
HuBERT

These embeddings are now our new feature vectors with size E (this size can get fairly large, i.e., $\mathrm{E} \in \{384, 2048\}$).

HuBERT

Given that the size of the features vectors nearly grow by $20\times$ (in their work they had $K = 39$ and $E = 768$), we now only use 10% of the total training dataset to do clustering.

We again do K-Means clustering on them with $K = 500$ clusters.



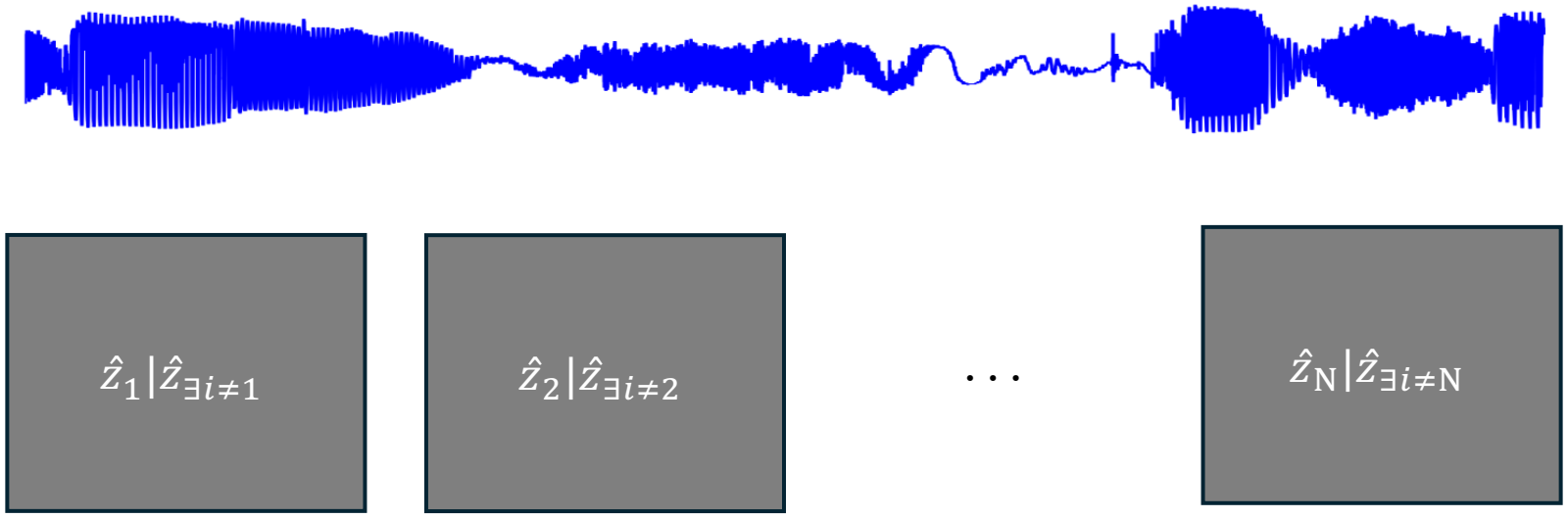
HuBERT

Given that we (1) have more clusters than stage 1 and (2) likely have different semantic meaning, we also need to update the learnable embeddings for each class.

We simply instantiate a new embedding matrix for the learnable embeddings e_j and e_k. Our prediction c_t will also have to instantiate a new transformation to this embedding size as well.

Training begins for stage 2, the same as stage 1.

WavLM



This model is essentially **HuBERT**, but with some slight modifications that work well.

Compared to **HuBERT**, **WavLM** (1) mixes in other utterances/noise and (2) adds a gated relative position bias.

WavLM

Note:

It also scales up data **a ton**, but we are just here to discuss methodology.

WavLM

The first change is the **gated relative position bias**.

This operates as an offset for the weights $a_{i,j}$ based off the “key” and ”query” in the self-attention mechanism.

First, lets consider some latent space h_i.

We con compute the “key”, “query”, and “values” of the attention mechanism as such:

$$q_i = h_i W^Q, \qquad k_i = h_i W^K, \qquad v_i = h_i W^V$$

where W is separate learnable matrices.

WavLM

Note:

We use learned transformations for the queries, keys, and values so that the original hidden representation is not required to use a single geometry for both determining relevance and transmitting information.

WavLM

We can compute the self attention output as:

$$a_{i,j} \propto e^{\frac{q_i \cdot k_j}{\sqrt{d_k}} + r_{i-j}},$$

$$\bar{h_i} = \sum_{j=1}^{T} a_{i,j} v_j$$

where q_i is the vector representing the current position, k_i is a vector representing another position, d_k is the dimension of the vectors (used for normalizing), and v_j is a vector representing another position.

The idea of the summation is to encode the current sequence with a weighted sum of information of all other positions (including itself).

WavLM

We can compute the self attention output as:

$$a_{i,j} \propto e^{\frac{q_i \cdot k_j}{\sqrt{d_k}} + r_{i-j}}$$

$$\bar{h}_i = \sum_{j=1}^{T} a_{i,j} v_j$$

where q_i is the vector representing the current position, k_i is a vector representing another position, d_k is the dimension of the vectors (used for normalizing), and v_j is a vector representing another position.

This is the **gated relative position bias**.

Without it, the calculation would be the standard computation of the self-attention mechanism.

WavLM

We can compute the **gated relative position bias** as follows:

$$g_i^{\text{update}} = \sigma(q_i \cdot u), \qquad g_i^{\text{reset}} = \sigma(q_i \cdot w)$$
$$\bar{r}_{i-j} = \omega g_i^{\text{reset}} d_{i-j}$$
$$r_{i-j} = d_{i-j} + g_i^{\text{update}} d_{i-j} + \left(1 - g_i^{\text{update}}\right) \bar{r}_{i-j}$$

where σ is the sigmoid function, q_i is again the query vector encoding the current sequence, u and w are learnable vectors with the same size as embeddings d_k, d_{i-j} is a relative position bias, and ω is a learnable scalar value.

The idea of all this math is similar to the older **gated activation** we saw.

WavLM

We can compute the **gated relative position bias** as follows:

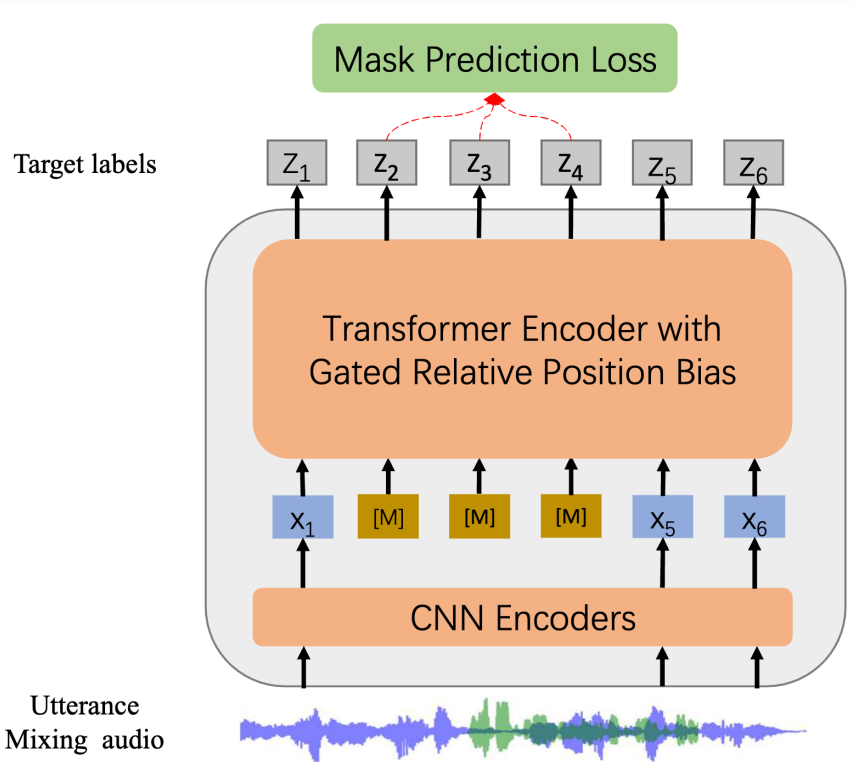
$$g_i^{\text{update}} = \sigma(q_i \cdot u), \qquad g_i^{\text{reset}} = \sigma(q_i \cdot w)$$
$$\bar{r}_{r-j} = \omega g_i^{\text{reset}} d_{i-j}$$
$$r_{i-j} = d_{i-j} + g_i^{\text{update}} d_{i-j} + \left(1 - g_i^{\text{update}}\right) \bar{r}_{i-j}$$

where σ is the sigmoid function, q_i is again the query vector encoding the current sequence, u and w are learnable vectors with the same size as embeddings d_k, d_{i-j} is a relative position bias, and ω is a learnable scalar value.

The temporal bias d_{i-j} allows the model to have information like "*this vector is 50 positions away from the current*". The first gate, g_i^{update}, determines whether to to downweight, or upweight this bias ($d_{i-j} \rightarrow 2d_{i-j}$). The second gate, g_i^{reset}, has an alternative learnable weight that can downweight, or upweight, more strongly.

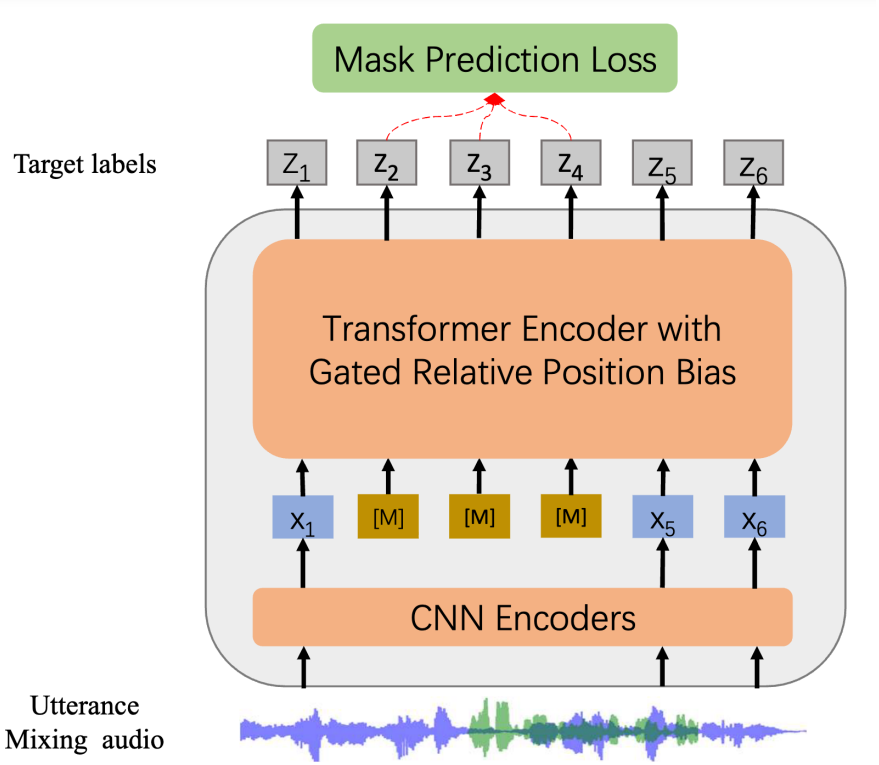
WavLM

The rest of the model remain untouched.



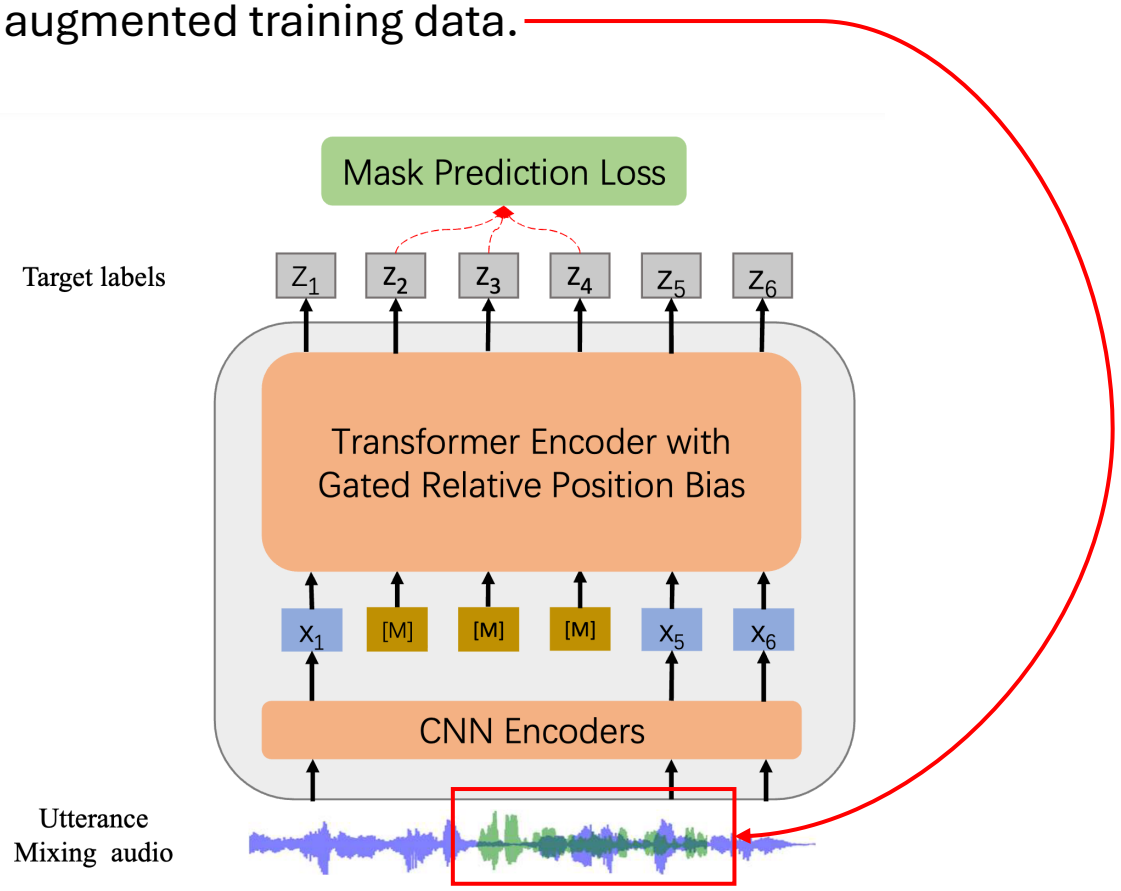
WavLM

The objective is the same. The targets are from the same multi-stage offline clustering as in HuBERT.



WavLM

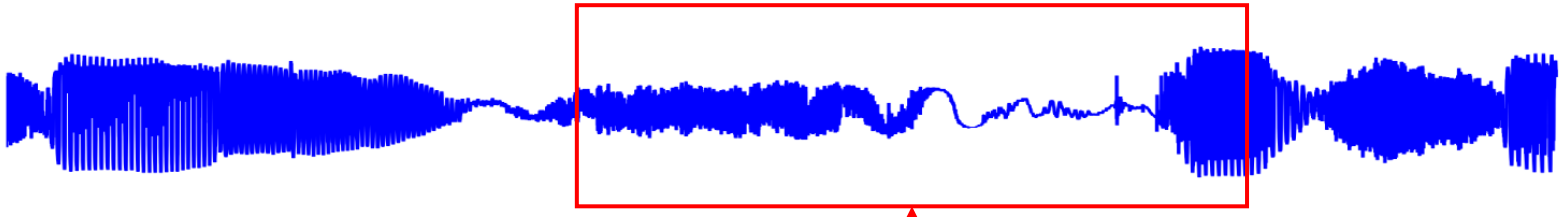
The final difference is the augmented training data.



WavLM

The algorithm is as follows:

(1) Sample a training utterance.

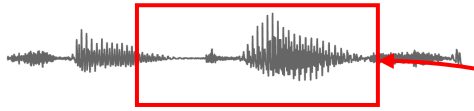


(2) Select a component to corrupt.

(3) Sample a corruption type.

(i) Noise.

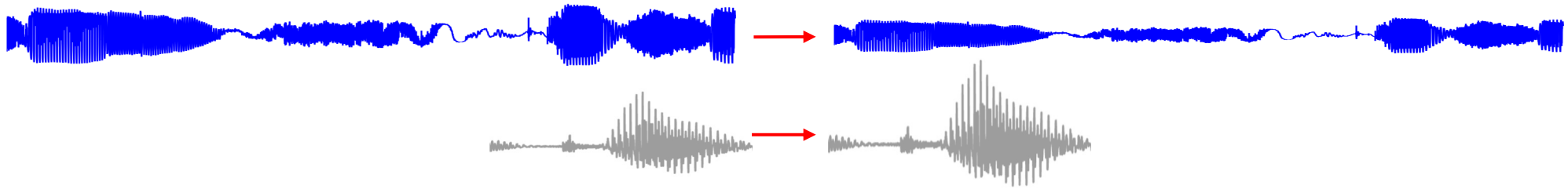
(ii) Another training utterance.



(4) Crop it to be the same size as the component to corrupt.

WavLM

(5) Scale the energy of the source and corruption.



(6) Add the corruption to the source.

(7) Use this an input to the model. Ensure normalization **AFTER** the corruption.

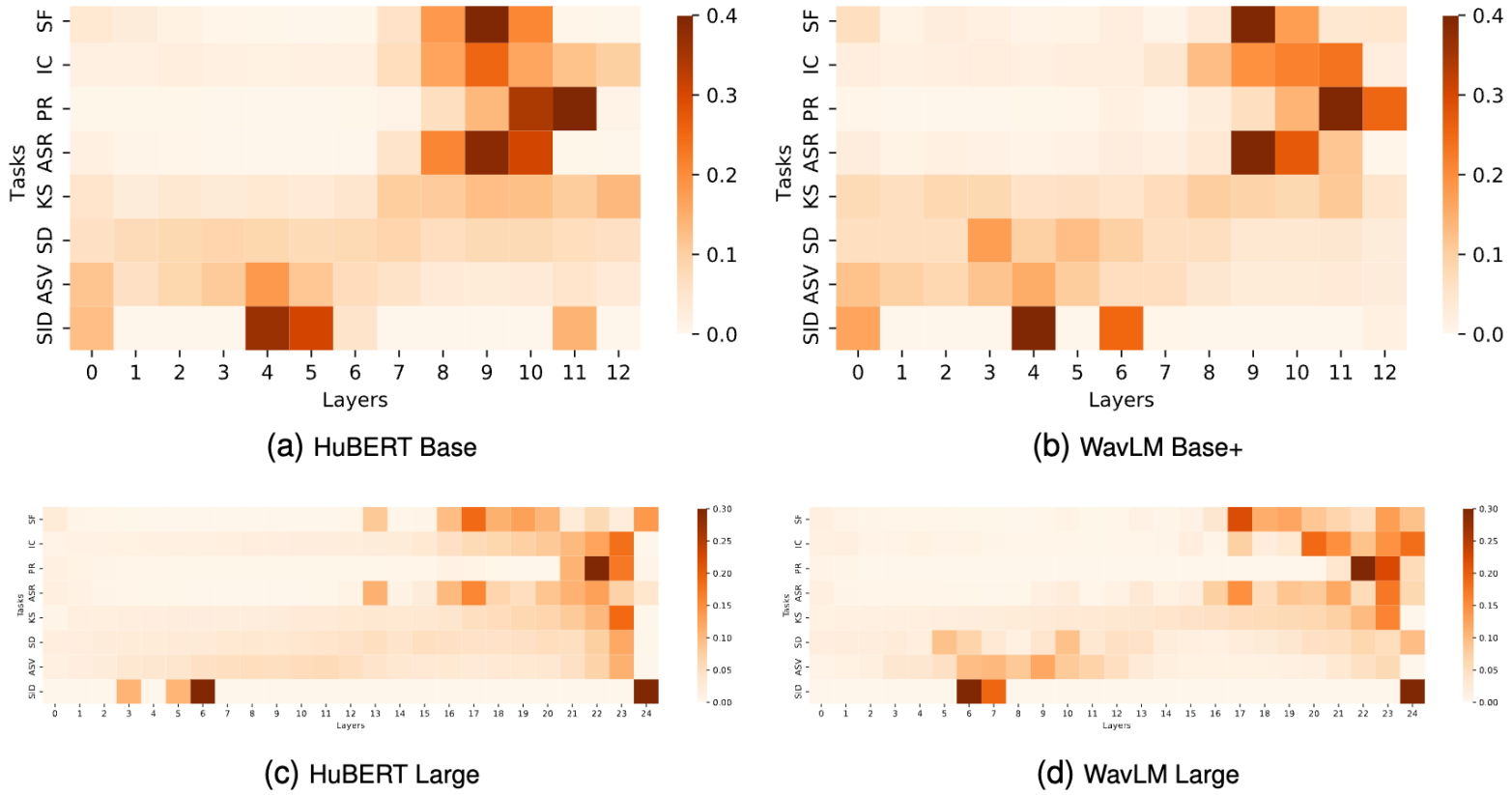
WavLM

Note:

The offline clustering is computed on the non-corrupted data. The targets for the masked instances (if the masks happen to fall within the corrupted region) are the source utterance. If the mask happens to be over a token with corruption, the model must also be able to denoise.

WavLM

The only time we will include results. **WavLM** likely became so popular simply because of this plot.



(a) HuBERT Base (b) WavLM Base+

(c) HuBERT Large (d) WavLM Large

References

1. WaveNet: A Generative Model for Raw Audio - https://arxiv.org/abs/1609.03499
2. WAV2VEC: UNSUPERVISED PRE-TRAINING FORSPEECH RECOGNITION - https://arxiv.org/pdf/1904.05862
3. Representation Learning with Contrastive Predictive Coding - https://arxiv.org/pdf/1807.03748
4. VQ-WAV2VEC: SELF-SUPERVISED LEARNING OFDISCRETE SPEECH REPRESENTATIONS - https://arxiv.org/pdf/1910.05453
5. THE CONCRETE DISTRIBUTION: A CONTINUOUS RELAXATION OFDISCRETE RANDOM VARIABLES - https://arxiv.org/pdf/1611.00712
6. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations - https://arxiv.org/pdf/2006.11477
7. Attention Is All You Need - https://arxiv.org/pdf/1706.03762
8. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - https://arxiv.org/pdf/1810.04805
9. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units - https://arxiv.org/pdf/2106.07447
10. WavLM: Large-Scale Self-Supervised Pre-Training for Full Stack Speech Processing - https://arxiv.org/pdf/2110.13900