به نام خدا

“Wave to Vector Embedding Models”

پروژه درس یادگیری ماشین آماری

استاد درس: دکتر امینی

ارائه دهنده: حسام افشار

شماره دانشجویی: ۶۱۰۳۰۳۰۲۲

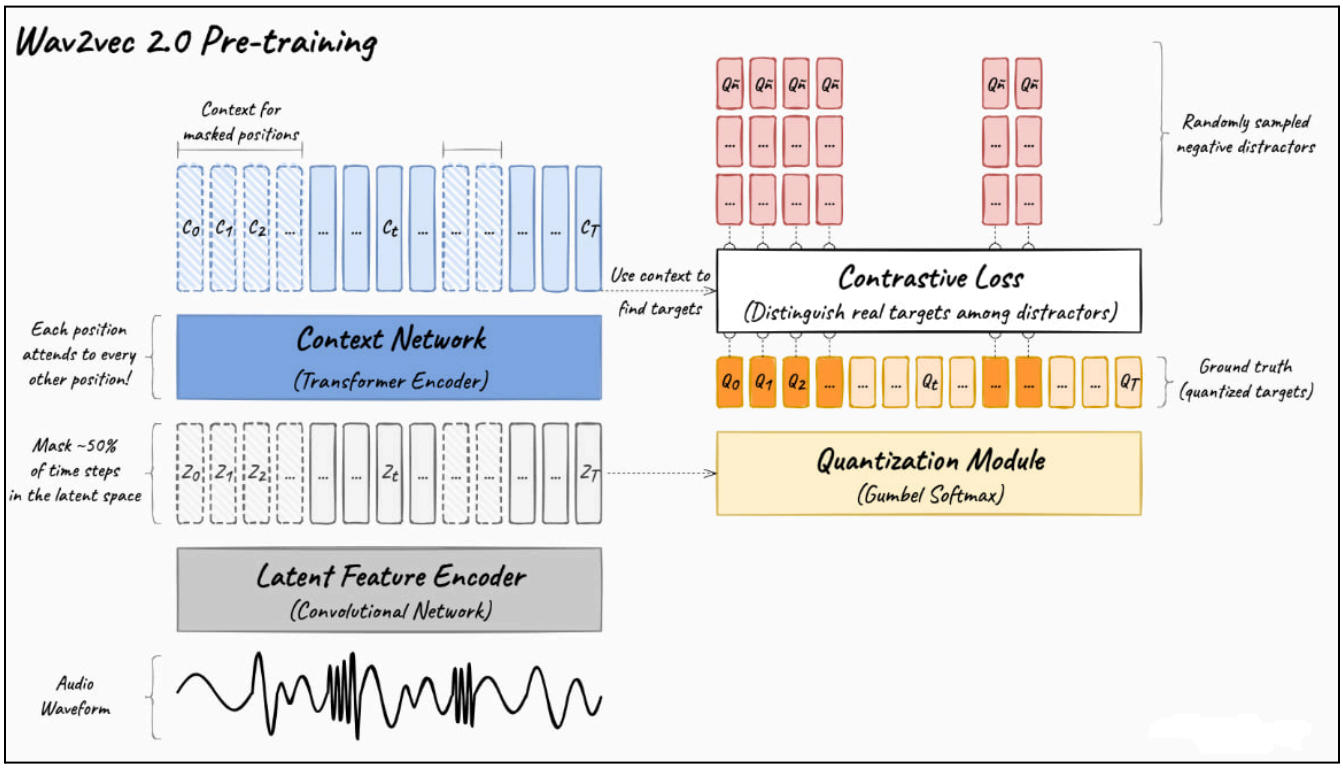
۱. چکیده

در این پروژه، مدل Wav2Vec 2.0 برای بازشناسی خودکار گفتار (ASR) بر روی مجموعه داده Common Voice 11.0 پیاده سازی و ارزیابی شده است. این مدل با استفاده از یادگیری خودنظارتی (Self-Supervised Learning) ابتدا ویژگی‌های گفتاری را از داده های خام صوتی استخراج کرده و سپس با مقدار محدودی از داده های دارای برچسب Fine-Tune میشود. برای آموزش مدل، 1500 نمونه صوتی برای آموزش و 300 نمونه برای آزمون انتخاب و پردازش شدند. کدها شامل سه بخش اصلی آماده سازی داده ها، پیکربندی مدل، و آموزش و ارزیابی هستند. مقدار WER نهایی مدل روی مجموعه آزمایشی 98% گزارش شده است که نشان دهنده دقت مناسب مدل با توجه به حجم داده و قدرت سیستم پردازش کننده میباشد. این پروژه نشان میدهد که حتی با داده های خام و بدون تنظیمات پیچیده، Wav2Vec 2.0 قادر به یادگیری ویژگی های گفتاری است.

۲. مدل wav2vec 2.0

در سال‌های اخیر، رویکردهای خودنظارتی (Self-Supervised) در حوزه‌های مختلف یادگیری عمیق، از جمله پردازش زبان طبیعی (مانند BERT و GPT) و بینایی کامپیوتر، پیشرفت چشمگیری داشته اند. ایده اصلی این رویکردها، استفاده از حجم زیادی از داده های خام (بدون برچسب) برای آموزش مدل است. این مدل، با طراحی وظایف پیشبینی بخش‌های گمشده داده یا تشخیص تناسب بین قسمت های مختلف آن، بازنمایی هایی قدرتمند می‌آموزد. در مرحله بعد، با مقدار اندکی داده دارای برچسب، مدل میتواند برای وظایف انتهایی (مثل طبقه بندی، استخراج ویژگی یا تشخیص گفتار) Fine-Tune شود و عملکردی رقابتی یا حتی بهتر از مدل‌های با نظارت داشته باشد. در حوزه گفتار، داده‌های خام (سیگنال‌های صوتی) زیادی وجود دارند، ولی برچسب گذاری آن‌ها فرآیندی پرهزینه و زمان بر است. پژوهشگران فیسبوک (FAIR) در سال‌های گذشته مدل های مختلفی را تحت نام کلی Wav2Vec ارائه کرده اند. اولین نسخه، یعنی Wav2Vec نشان داد که میتوان از سیگنال خام صوتی با روش‌های خودنظارت ویژگی‌های مفیدی استخراج کرد. سپس، نسخهٔ VQ-Wav2Vec ایده گسسته سازی ویژگی‌ها (Quantization) را مطرح کرد. سرانجام، در سال 2020، نسخهٔ Wav2Vec 2.0 معرفی شد که ترکیبی پیشرفته از شبکهٔ کانولوشنی، معماری ترنسفورمر و مکانیزم گسسته سازی چند کدبوکه (Multi-Quantizer) را در بر دارد. این مدل با استفاده از داده‌های بدون برچسب، سیگنال های صوتی پیوسته را به بردارهای معنایی (Contextualized Representations) تبدیل میکند. ویژگی برجسته این معماری، توانایی آن در استفاده از داده‌های بدون برچسب برای یادگیری اولیه است، که سپس می‌تواند با استفاده از یک مجموعه داده کوچکتر و دارای برچسب برای وظایف خاصی مانند بازشناسی خودکار گفتار (ASR)، تشخیص احساسات صوتی و موارد مشابه تنظیم (fine-tune) شود. مدل wav2vec 2.0 داده‌های صوتی خام را با استفاده از یک شبکه عصبی کانولوشنال چندلایه پردازش کرده و ویژگی های پنهان صوتی را استخراج می‌کند. سپس، بخشی از این ویژگی‌ها به صورت تصادفی ماسک می‌شوند، مشابه روش Masked Language Modeling در پردازش زبان طبیعی (NLP). این ویژگی‌های ماسک شده به یک شبکه ترنسفورمر (Context Network) داده میشوند تا بردارهای معنایی تولید شوند. در طول فرآیند آموزش، مدل با استفاده از مکانیزم Gumbel Softmax واحدهای صوتی گسسته (Discrete Speech Units) را برای نمایش بهتر ویژگی‌های صوتی یاد می‌گیرد و در نهایت، از این واحد ها به عنوان متغیر هدف استفاده میکند تا بردار های معنایی که همان embedding های

مورد نظر هستند را با رویکرد Contrastive Learning یاد بگیرد. در این رویکرد مدل یاد میگیرد تا واحد گسسته متناسب با هر ویژگی پنهان ماسک شده را با واحد های گسسته نادرستی که به صورت تصادفی از بخش های دیگر دنباله نمونه گیری میشوند را تمایز دهد. تصویر زیر شهود قابل درکی درباره عملکرد مدل در مرحله آموزش نشان میدهد.



Feature Encoder .۱.۲

سیگنال صوتی خام اغلب با نرخ نمونه برداری 16kHz یا بالاتر ذخیره می‌شود. اگر این توالی طولانی مستقیماً وارد ترنسفورمر شود، هزینهٔ محاسباتی مکانیزم توجه که با $O(T^2)$ (بر حسب طول توالی) رشد می‌کند، بسیار بالا خواهد بود. بنابراین، یک شبکه Feature Encoder کانولوشنال طراحی شده است تا هم نرخ نمونه برداری را کاهش دهد (Subsampling) و هم ویژگی‌های محلی (طیفی-زمانی) را در قالب بردارهایی فشرده استخراج نماید. در پیکربندی اصلی Wav2Vec 2.0 (نسخهٔ Base یا Large) از هفت لایهٔ متوالی کانولوشنال یک بعدی (1D CNN) استفاده می‌شود که هر کدام 512 فیلتر با استراید (Stride) های به ترتیب (5,2,2,2,2,2,2) , اندازه (10,3,3,3,3,2,2) هستند. فرض کنیم $x \in R^T$ سیگنال های خام اولیه باشد. در خروجی آخرین لایه شبکه کانولوشنال ، بردار ویژگی $z \in R^{T' \times d}$ بدست میاید که:

- $T' \approx \frac{T}{\prod_{l=1}^{7} stride_l}$: طول زمانی فشرده شده.
- d : تعداد فیلتر یا ابعاد ویژگی لایه آخر

با ضرب تمام استرایدها فشرده سازی زمانی قابل توجهی اعمال می‌شود.به عبارت دیگر، اگر ورودی با نرخ 16kHz باشد، در خروجی این هفت لایه، نرخ نمونه برداری مؤثر به حدود 49 کاهش مییابد. این یعنی هر بردار خروجی نمایندهٔ حدود 20 میلی‌ثانیه از سیگنال صوتی خام است. در نتیجه، طول توالی به مقدار قابل توجهی کوچک‌تر می‌شود، اما تعداد فیلتر ها (512 در نسخهٔ Base) ابعاد ویژگی را تشکیل میدهد. هر لایه کانولوشنال پس از عملیات کانولوشن از تابع فعالساز GLU و Layer Normalization استفاده میکند :

$$z^{(l)} = \sigma(Norm(W^{(l)} * z^{(l-1)} + b^{(l)}))$$

که در آن $x = z^{(0)}$ سیگنال خام اولیه، $W^{(l)}$ فیلترهای لایهٔ lام و $Norm$ همان Layer Normalization است که در آن، هر نمونه به صورت مجزا و صرفاً در طول مؤلفه های ویژگی نرمال می‌شود. در این روش، برخلاف نرمالسازی دسته ای (Batch Normalization) که میانگین و واریانس را بر حسب کل mini-batch محاسبه می‌کند، میانگین و واریانس هر بردار ورودی صرفاً برحسب ویژگی های همان نمونه محاسبه شده و سپس اعمال می‌شود. اگر بردار $a^{(l)} \in R^d$ خروجی پس از یک لایه کانولوشن باشد، ابتدا میانگین μ و واریانس σ^2 مؤلفه‌های $a^{(l)}$ از روابط زیر به دست می‌آید:

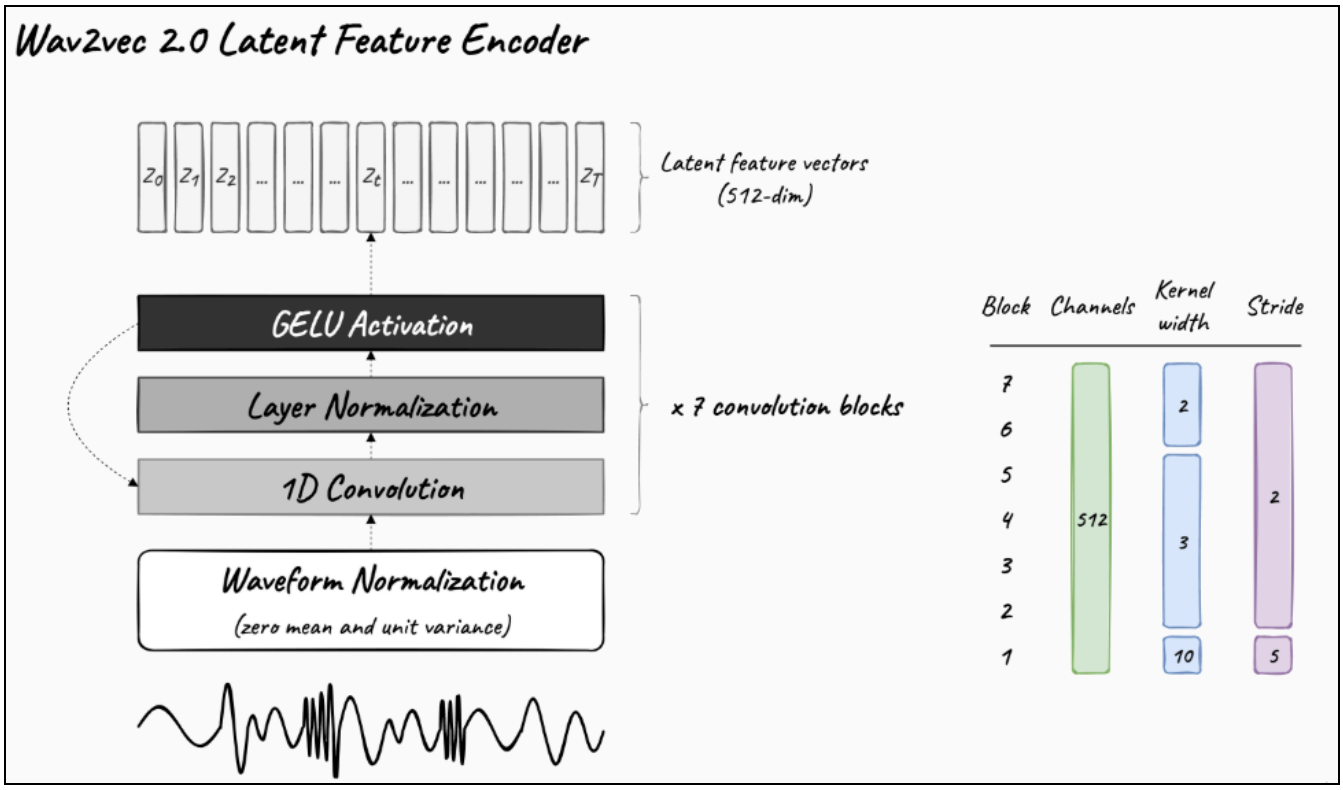
$$\mu = \frac{1}{d}\sum_{i=1}^{d} {a^{(l)}}_i \quad , \quad \sigma^2 = \sum_{i=1}^{d} ({a^{(l)}}_i - \mu)2$$

$$LayerNorm(a^{(l)})_i = \frac{{a^{(l)}}_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma_i + \beta_i, \forall i = 1, \dots, d$$

γ و β پارامترهای قابل آموزش با بُعد d هستند که برای تغییر مقیاس (Scaling) و جابجایی (Shifting) بعد از نرمالسازی به کار می‌روند که در طول فرآیند آموزش مدل بدست میایند و ϵ ثابت کوچک (مانند 10^{-5}) است که برای پایداری عددی به واریانس در مخرج اضافه میشود. همچنین σ تابع فعالسازی GELU میباشد و اگر $o^{(l)}$ خروجی لایه lام باشد خروجی تابع فعالساز به صورت زیر محاسبه میشود:

$$GELU(o^{(l)}) = o^{(l)}x \cdot \Phi(o^{(l)}) = 2o^{(l)}[1 + erf(2o^{(l)})] \approx 0.5o^{(l)}[1 + tanh(\pi 2(o^{(l)} + 0.044715{o^{(l)}}^3))]$$

که استفاده از این تابع فعالساز به جای ReLU باعث میشود گرادیان در نقاط ورودی منفی به تابع فعالساز صفر نشود و خطر ناپدید شدن گرادیان (Vanishing Gradient) را کاهش دهد. تصویر زیر به صورت شهودی عملکرد بخش feature encoder را نمایش میدهد:



۲.۲. ماژول گسسته سازی (Quantization)

بخش کلیدی معماری Wav2Vec 2.0، گسسته سازی است که بازنمایی پیوسته را به واحد های گسسته تبدیل میکند. زیرا یکی از موانع مهم در استفاده از ترنسفورمر برای پردازش گفتار، پیوستگی سیگنال صوتی است. هدف اصلی این ماژول، تبدیل ویژگی‌های پیوسته گفتار به واحدهای گسسته ای است که بتوانند نقش «واژگان گفتاری» را برای مدل ایفا کنند، بدون آنکه نیاز به برچسب‌های فونتیک یا گذراندن دیتاست از یک فرایند برچسب گذاری دستی باشد. در پردازش زبان نوشتاری، متن را میتوان به سادگی به واحدهای گسسته (نظیر کلمات یا زیرکلمات) شکست. این واحدها یک واژگان (Vocabulary) متناهی تشکیل میدهند که مدل میتواند آنها را به بردارهای Embedding نگاشت کند. اما در گفتار، چنین سیستم گسسته طبیعی ای وجود ندارد. سیگنال آکوستیک با گذر مداوم در زمان، پیوسته است و امکان برش یا تجزیه آن به واحد های کوچک لزوماً روشن و ازپیش تعیین شده نیست. استفاده از فونم ها به عنوان واحد های گسسته، راهکاری کلاسیک است. اما برای این کار، لازم است از ابتدا کل مجموعه داده توسط متخصصان برچسب گذاری فونتیک شود. این فرآیند پرهزینه و زمان بر است و ما را از توانایی پیش آموزش خودنظارت (روی داده‌های عظیم و بدون برچسب) محروم میکند. به همین دلیل، درWav2Vec 2.0 ایده طراحی یک ماژول گسسته سازی مطرح شده است که بتواند به صورت کاملاً خودکار و انتها-به-انتها (End-to-End) واحدهای گسسته صوتی را بیاموزد. Wav2Vec 2.0 راهکاری را پیشنهاد میکند تا واحدهای گسسته گفتاری را به صورت خودکار بیاموزد. این کار از طریق نمونه برداری از توزیع Gumbel-Softmax صورت میگیرد. واحدهای ممکن از کد واژه هایی (Codewords) که از کدبوک‌ها (Codebooks یا گروه) نمونه برداری میشوند، ساخته میشوند. سپس این کدواژه‌ها با هم ادغام (Concatenate) میگردند و واحد نهایی Unit (Speech) را شکل می‌دهند. در Wav2Vec 2.0 از ۲ کدبوک

(گروه) استفاده می‌شود، که هر کدام ۳۲۰ کدواژه دارند. فایده استفاده از چند کدبوک (به جای یک کدبوک بزرگ) این است که مدل میتواند ظرفیت بیشتری برای کد گذاری جنبه های مختلف گفتار داشته باشد (مثلاً یک گروه بخشی از ویژگی‌ها را مدل کند و گروه دیگر بخش دیگری از ویژگی‌ها را).

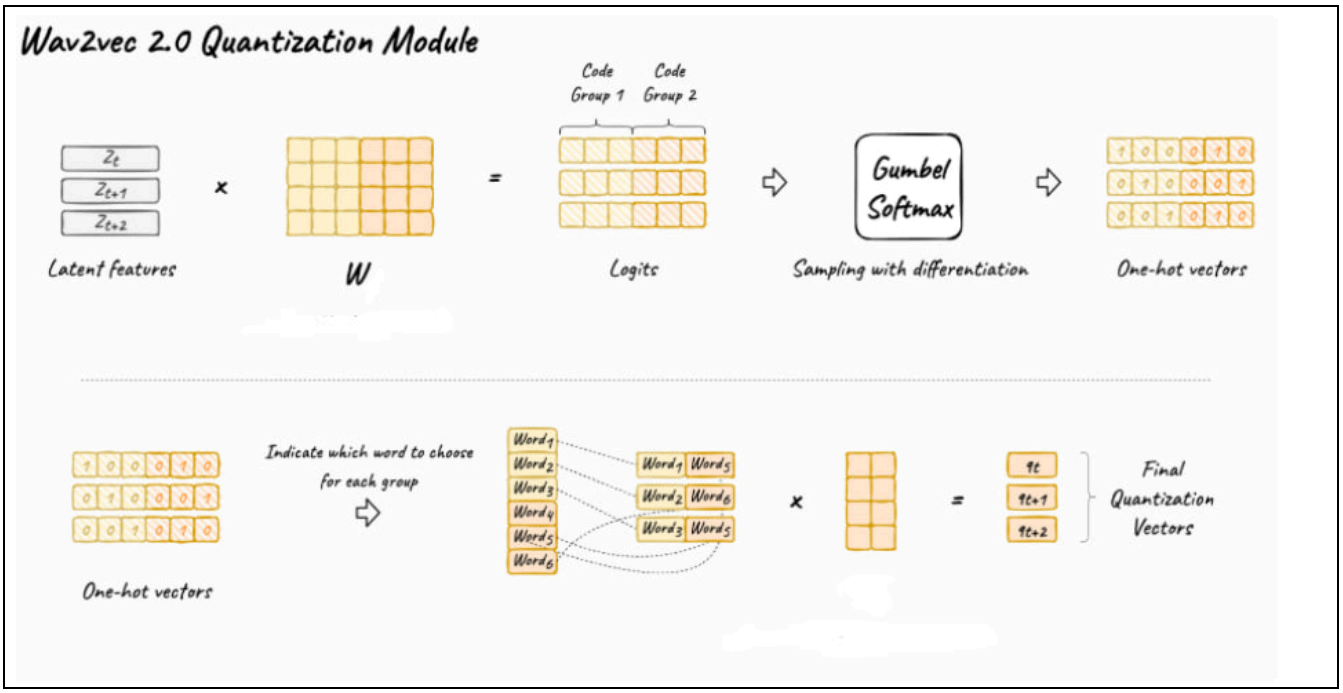
هر کدبوک شامل تعدادی بردار کد واژه e_v است و به طور کلی هر کد واژه $e_v^{(g)}$ یک بردار قابل یادگیری در فضای R^d است (d همان بعد دنباله خروجی Feature Encoder یعنی z_t است). اگر z_t خروجی شبکه Feature Encoder باشد، برای هر کدبوک (گروه) g، ابتدا یک تبدیل خطی (مثلاً ضرب در ماتریس وزنی $W^{(g)}$ روی z_t اعمال میشود تا به تعداد کد واژه های آن کدبوک (یعنی 320) یک امتیاز (Logit) اختصاص یابد:

$$l_t^{(g)} = z_t W^{(g)} \in R^{320}$$

این لاجیت‌ها نشان می‌دهند که کدام کدواژه ممکن است مناسب تر باشد. انتخاب صریح ($argmax$) کد واژه باعث می‌شود گرادیان از این مرحله عبور نکند. برای حفظ مشتق پذیری در یادگیری انتها-به-انتها، از روش Gumbel-Softmax استفاده می‌شود. در این روش، به هر لاجیت نویز Gumbel افزوده شده و با تابع Softmax (همراه با پارامتر دما τ) توزیع احتمالی روی کد واژه‌ها ایجاد می‌گردد.

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

که در آن $n = -log(-log(u))$ یک نویز گامبل است که u از توزیع یکنواخت $Uniform(0,1)$ نمونه برداری میشود. τ پارامتر دما است تعیین میکند که توزیع احتمال خروجی تا چه اندازه تیز (Sharp) یا نرم (Smooth) باشد. اگر τ به سمت صفر برود، تقریباً همه جرم احتمال روی یکی از کد واژه‌ها متمرکز می‌شود (حالتی نزدیک به $argmax$). اگر τ بزرگ باشد، احتمال بین کد واژه‌های مختلف پخش می‌شود. در عمل، معمولاً طی آموزش τ را کم کم کاهش میدهند تا مدل ابتدا تنوع کد واژه‌ها را کاوش کند و سپس به انتخاب های قاطع تری برسد. در مرحله Forward کد واژه با بالاترین $p_{g,v}$ (یا معادلش $i = argmax_v(p_{t,v}^{(g)})$) انتخاب می‌شود و بدین ترتیب، برای کدبوک g در فریم زمانی t، کد واژه $e_i^{(g)}$ انتخاب میشود. این کد واژه ها به صورت افقی با هم ادغام شده و خروجی نهایی q_t را شکل میدهند. در backpropagation برای محاسبهٔ گرادیان، خروجی Gumbel-Softmax همان خود احتمال ها (که مشتق پذیر است) استفاده می‌شود تا مدل بتواند کل سازوکار را انتها به انتها یاد بگیرد، ولی در Forward ما یک انتخاب ($argmax$) را نگه می‌داریم تا واحد گسسته واقعاً شبیه سازی شود. تصویر زیر سازوکار مرحله گسسته سازی را به صورت شهودی نمایش میدهد:



۳.۲. بلوک ترنسفورمر (Context network)

شبکهٔ ترنسفورمر در معماری Wav2Vec 2.0، که گاه از آن با عنوان Context Network یاد میشود، پس از مرحله Feature Encoder قرار دارد و نقش اصلی آن مدل سازی روابط بلند مدت در سیگنال صوتی و تولید بازنمایی های محتوایی از گفتار است. لازم به ذکر است در بخش Context Network تنها از بخش ecoder شبکه ترنسفورمر استفاده میشود. ابتدا خروجی شبکه کانولوشنی با بعد 512، از یک لایه خطی ساده (Feature Projection) عبور می‌کند تا با بعد داخلی ترنسفورمر (نظیر 768 در نسخه Base یا 1024 در نسخه Large) هماهنگ شود. اگر $z_t \in R^{512}$ نشان‌دهنده بردار ویژگی در لحظه زمانی t باشد و d_{tf} بعد مورد انتظار ترنسفورمر، این لایه را می‌توان به شکل زیر نوشت:

$$\tilde{z}_t = z_t W_{projection}$$

که $\tilde{z}_t \in R^{d_{tf}}$ است و $W_{projection} \in R^{512 \times d_{tf}}$ وزن های قابل یادگیری میباشند.

پس از افزایش بعد، از $\tilde{z}_t$ استفاده میشود تا جاسازی مکانی (Positional Embedding) آنها مشخص شود. یکی از تفاوت های مهم Wav2Vec 2.0 با ترنسفورمر استاندارد در شیوه اعمال جاسازی مکانی نمایان میشود. در ترنسفورمرهای زبان (مانند BERT یا Transformer اصلی)، موقعیت هر توکن با بردارهای سینوسی یا بردارهای قابل یادگیری ثابت مشخص میشود. اما در Wav2Vec 2.0 از یک لایه کانولوشن یک بعدی گروه بندی شده (Grouped Convolution) برای یادگیری مستقیم جاسازی مکانی استفاده شده است. در کانولوشن استاندارد (Normal Convolution)، برای هر ویژگی ورودی (زمانی یا مکانی)، یک فیلتر به طور کامل روی تمام ابعاد ورودی اعمال می‌شود. در اینجا، اندازه فیلتر، تعداد فیلترها و گام ها تعیین میکند که ویژگی های محلی چطور از سیگنال استخراج شوند. اما در کانولوشن گروه بندی شده، به جای اعمال یک فیلتر مشترک به همه ویژگی ها، ورودی به گروه های کوچکتر تقسیم

میشود و برای هر گروه از ویژگی ها، یک فیلتر مجزا اعمال میشود. این امر موجب می‌شود که کانولوشن گروه بندی شده قادر باشد اطلاعات مکانی را بطور مستقل برای هر گروه از ویژگی‌ها بیاموزد، در حالی که همزمان درک بهتری از وابستگی های زمانی و محلی در سیگنال ایجاد میکند. در واقع، این نوع کانولوشن به مدل این امکان را میدهد که پویایی بیشتری در یادگیری موقعیت زمانی هر فریم داشته باشد، در حالی که از پیچیدگی محاسباتی کمتری نسبت به روش‌های سنتی برخوردار است.

در کانولوشن گروه بندی شده، ورودی به G گروه تقسیم میشود و بر روی هر گروه بطور مستقل فیلتر های اختصاصی اعمال می‌شود. فیلتر ها نیز دارای ابعاد ($k \times d_{group}$) هستند که در آن k طول فیلتر و d_{group} تعداد ویژگی‌ها در هر گروه است. در پیکر بندی اصلی Wav2Vec 2.0 تعداد گروه‌ها $G = 16$ و طول فیلتر $k = 128$ در نظر گرفته شده است، بنابراین، در هر گروه، هر فیلتر به اندازه $128 \times \frac{d_{tf}}{16}$ از ویژگی‌ها را پردازش میکند از آنجایی که میخواهیم بعد جاسازی های مکانی با بعد ترنسفورمر یعنی $T' \times d_{tf}$ برابر باشد به همین تعداد فیلتر ها در هر گروه برابر $\frac{d_{tf}}{16}$ میباشد. و چون اندازه کرنل ها 128 است و stride برابر یک است باید به تعداد مناسب در دو طرف دنباله ورودی یعنی $\tilde{z}_t$ پدینگ صفر (zero padding) داشته باشیم. که با توجه به فرمول کلی تعداد پدینگ در کانولوشن یک بعدی یعنی $(1 - \frac{kernel\ size}{2})$ لازم است در یک طرف دنباله 63 پدینگ صفر و در طرف دیگر 64 پدینگ صفر داشته باشیم (دلیل عدم تقارن پدینگ ها زوج بودن اندازه فیلتر است). در هر در این بخش هم پس از اعمال لایه کانولوشن گروه بندی شده تابع فعالساز GELU و Layer Normalization اعمال میشود.

در ادامه اگر جاسازی های مکانی را با P_t نمایش دهیم ورودی نهایی شبکه ترنسفورمر ازجمع $\tilde{z}_t$ با P_t بدست میاید و آن را $z_t^{(final)}$ نام گذاری میکنیم. دنباله $Z = \left\{z_1^{(final)}, ..., z_{T'}^{(final)}\right\}$ اکنون شامل هم اطلاعات محتواییِ هر فریم و هم اطلاعات مکانیِ قابل یادگیری است و آماده می‌شود تا وارد بلوک‌های ترنسفورمر شود. در نسخهٔ BASE، معمولاً ۱۲ لایهٔ ترنسفورمر به صورت متوالی قرار دارند و در نسخهٔ LARGE تعداد این لایه‌ها به ۲۴ می‌رسد. هر لایهٔ ترنسفورمر، طبق ساختار استاندارد، مکانیزم Multi-Head Self-Attention و یک شبکه Feed-Forward دارد. در مکانیزم Multi-Head Self-Attention، ابتدا سه ماتریس Q، K، و V از دنباله ورودی Z استخراج می‌شود:

$$V = ZW_V \ , \ K = ZW_K \ , \ Q = ZW_Q$$

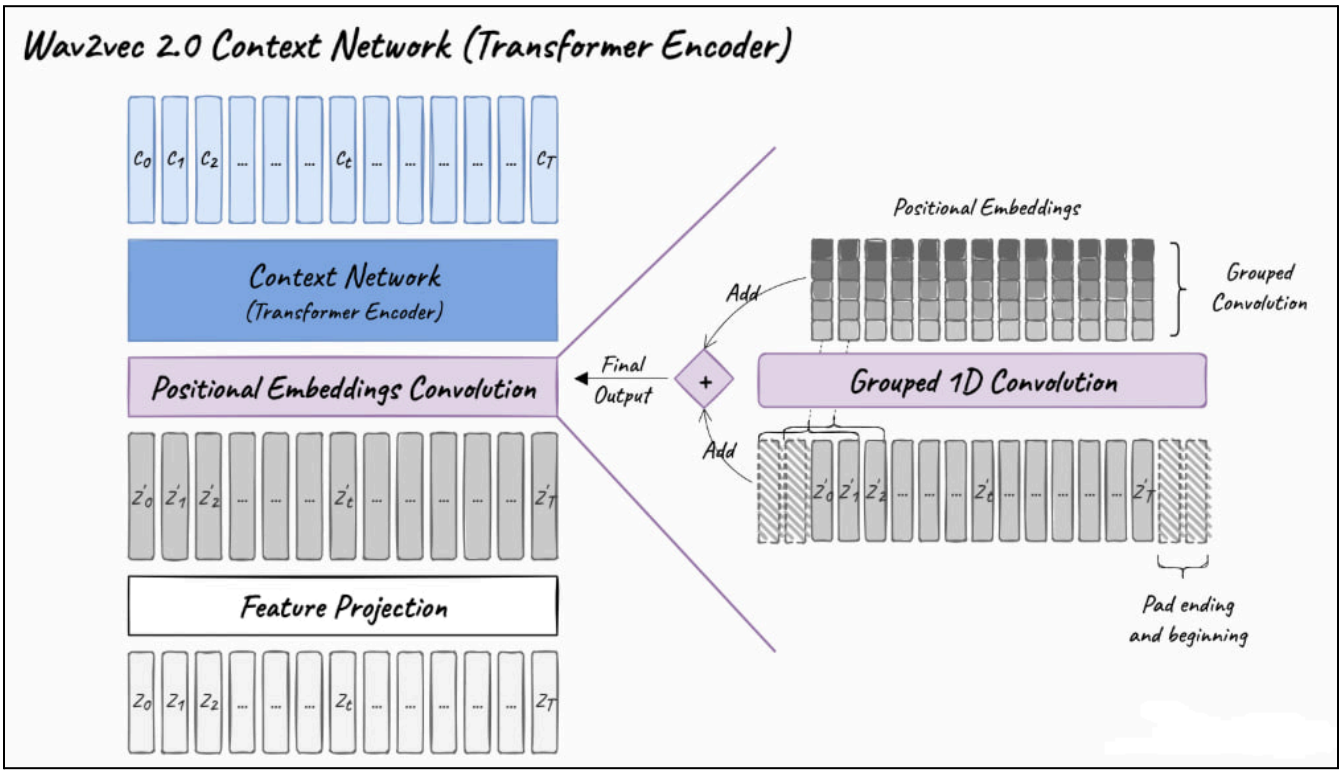
که $Z \in R^{T' \times d_{tf}}$ ماتریس ورودی هر لایه و $W_Q, W_Q, W_Q \in R^{d_{tf} \times d_{tf}}$ وزن‌های قابل یادگیری هستند. در هر head از self-Attention، ابعاد Q، K، و V معمولاً $\frac{d_{tf}}{h}$ در نظر گرفته می‌شود که h تعداد head های Multi-Head Self-Attention است. مکانیزم Attention با محاسبهٔ زیر تعریف می‌شود:

$$softmax(\frac{KQ^T}{\sqrt{d_k}})V = Attention(Q, K, V)$$

که در آن $d_k = \frac{d_{tf}}{h}$ ابعاد کلید در هر Attention head است. خروجی head های مختلف سپس باهم ادغام (Concatenate) می‌شود و از لایه خطی دیگری عبور میکند تا دوباره به بعد d_{tf} بازگردد. در ادامه، این خروجی با ورودی لایه (به اصطلاح به اینکار Residual Connection میگویند) جمع و بر آن لایه نرمالسازی اعمال میگردد. سپس، یک شبکه Feed-Forward دولایه روی خروجی مرحله Attention اعمال می‌شود:

$$FFN(a) = \sigma(aW_1 + b_1)W_2 + b_2$$

که $a \in R^{d_{tf}}$ خروجی مرحله Attention و ورودی شبکه FFN است و W_1، W_2، b_1 و b_2 وزن ها و بایاس های قابل یادگیری هستند. σ نیز تابع فعالساز ReLU میباشد. خروجی این بخش نیز دوباره با ورودی اش جمع میشود و از لایه نرمال‌سازی عبور می‌کند. مجموع این مراحل (Self-Attention + FFN) یک بلوک ترنسفورمر را شکل می‌دهد و در معماری Wav2Vec 2.0 معمولاً ۱۲ یا ۲۴ بلوک پشت سر هم قرار میگیرند و در نهایت، پس از عبور از تمام لایه‌های ترنسفورمر، برای هر فریم زمانی یا گام $z_t^{(final)}$ بردار خروجی c_t به دست میاید که اغلب از آن با عنوان Context Vector یاد میشود. در مرحله پیش آموزش (Self-Supervised)، این بردار مستقیماً برای یادگیری گسسته سازی و تابع هزینه تضاد (Contrastive Loss) مورد استفاده قرار میگیرد تا مدل بتواند فریم های ماسک شده را بر اساس بافت زمانی بازسازی کرده و ضمن نزدیک کردن نمونه‌های مثبت، نمونه های منفی را دور سازد. همچنین در مرحلهٔ Fine-Tuning، همین بردار c_t مبنای پیشبینی برچسب های کاراکتری قرار می‌گیرد (با اضافه کردن مثلا یک لایه Feed Forward در خروجی بلوک ترنسفورمر) و مدل را قادر می‌سازد با مقدار اندکی داده دارای برچسب نیز به دقت بالایی در تشخیص گفتار برسد. تصویر زیر به طور خلاصه فرآیند این بخش را نمایش میدهد:



۴.۲. مرحله پیش آموزش تابع هدف

در فرایند پیش‌آموزش (Pre-Training) مدل Wav2Vec 2.0، حدود ۵۰٪ از گام‌های زمانی خروجی Feature Encoder به صورت تصادفی ماسک می‌شوند. موقعیت‌های ماسک شده با یک بردار آموزشی ثابت z'_M جایگزین می‌شوند که برای تمامی این موقعیت‌ها مشترک است. این بردار اطلاعات خاصی از محتوای سیگنال ندارد و تنها به عنوان یک جایگزین عمل میکند تا مدل مجبور شود بازنمایی های گسسته صحیح را تنها از طریق اطلاعات محتوایی یعنی بردار c_t بازسازی کند. خروجی ماسک شده سپس وارد شبکه ترنسفورمر میشود که وابستگی های بلند مدت در سیگنال گفتار را مدل‌سازی میکند. تابع هدف پیش آموزش شامل دو بخش اصلی یادگیری تضاد (L_m Contrastive Loss) و تنوع کدبوک (L_d Diversity Loss) است که این دو بخش برای هر دسته (Batch) داده به صورت زیر ترکیب میشوند و در آن α یک ابرپارامتر است که میزان اهمیت هر بخش را تنظیم می‌کند:

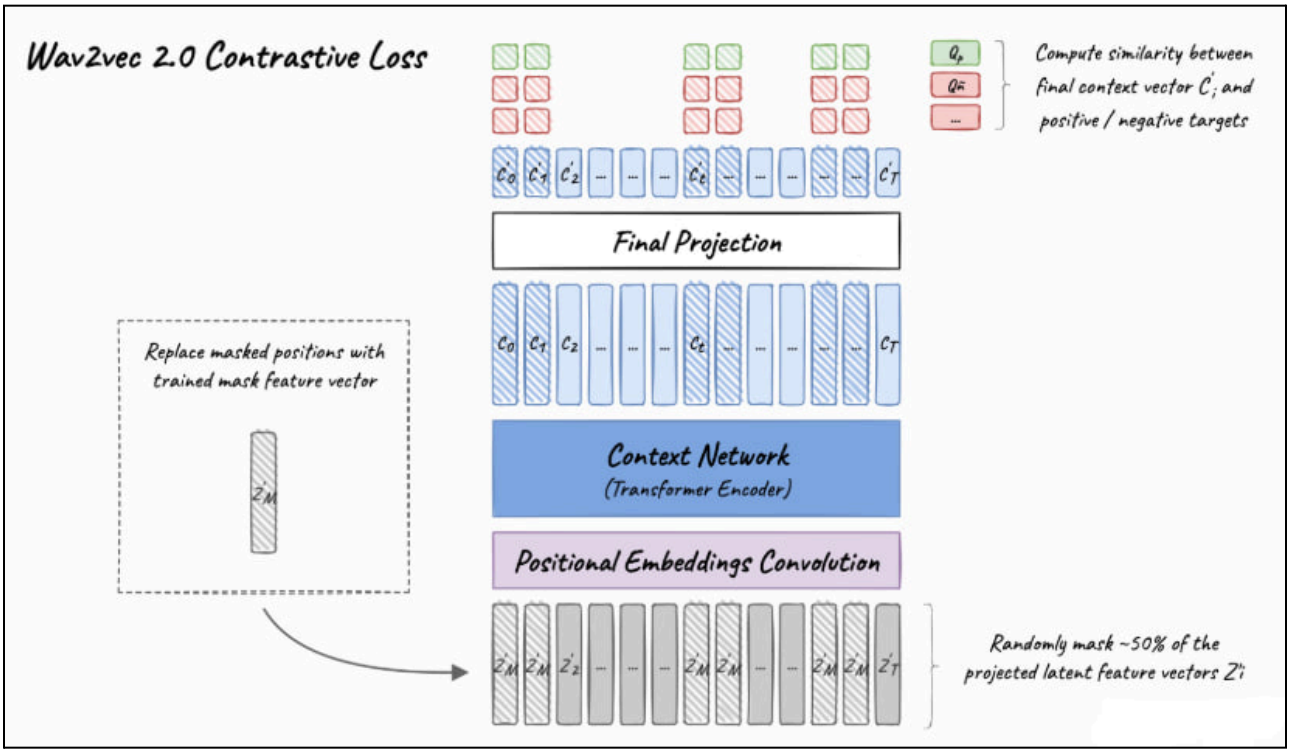
$$L = L_m + \alpha L_d$$

بخش اصلی تابع هدف پیش‌آموزش، یادگیری تضاد است که مدل را وادار میکند بازنمایی گسسته صحیح یک موقعیت ماسک شده را از میان نمونه های گمراه کننده تشخیص دهد. برای هر موقعیت ماسک شده t، مدل باید بازنمایی گسسته صحیح q_t را از مجموعه ای K+1 عضوی $\tilde{q} \in Q_t$ که شامل q_t و K نمونه گمراه کننده (Negative Distractors) است، شناسایی کند. نمونه های گمراه کننده به صورت تصادفی و با توزیع یکنواخت از موقعیت‌های ماسک شده دیگر انتخاب میشوند.

در پیکر بندی اصلی مدل wav2vec 2.0 تعداد K برابر 100 در نظر گرفته شده است و بردار c_t که خروجی شبکه ترنسفورمر برای موقعیت ماسک شده t است، از یک لایه خطی عبور میکند تا ابعاد آن با بازنمایی های گسسته Q_t برابر شود و آن را c'_t نام گذاری میکنیم. سپس شباهت کسینوسی (Cosine Similarity) c'_t و q_t محاسبه شده و با شباهت بین c'_t و نمونه های گمراه کننده مقایسه میشود.در نهایت تابع یادگیری تضاد به صورت زیر تعریف می‌شود:

$$L_m = -\ log\left(\frac{exp\ (sim(c'_t, q_t)/\kappa)}{\sum_{\tilde{q}\sim Q_t} exp(sim(\tilde{q}, c'_t)/\kappa)}\right)$$

که در آن $sim(a,b) = \frac{a^T b}{\|a\|\ \|b\|}$ شباهت کسینوسی، κ پارامتر دما که شدت توزیع شباهت را تنظیم می‌کند و Q_t مجموعه ای شامل q_t و نمونه های گمراه کننده در گام t ام است. هدف این تابع، افزایش شباهت c'_t با نمونه هدف مثبت q_t و کاهش شباهت با نمونه های گمراه کننده است. به این ترتیب، مدل یاد میگیرد بازنمایی های خوب و عمیقی از سیگنال صوتی تولید کند که اطلاعات ماسک شده را به درستی بازسازی کنند. تصویر زیر به صورت شهودی درک بهتری از این فرآیند ارائه میدهد:



همانطور که در بخش های قبلی اشاره شد مدل Wav2Vec 2.0 از یک ماژول گسسته سازی استفاده میکند که شامل چندین کدبوک موازی G است و هر کدبوک شامل V کد واژه ورودی است. مدل برای یادگیری تضاد به این کدبوک ها وابسته است، زیرا نمونه های مثبت و منفی را در این فضا بازنمایی میکند. اگر مدل تنها از تعداد محدودی از ورودی های کدبوک استفاده کند، ظرفیت بازنمایی آن

کاهش می یابد و در نتیجه، مدل به خوبی قادر به یادگیری سیگنال های گفتاری متنوع نخواهد بود. برای جلوگیری از این مسئله، تابع هزینهٔ تنوع (L_d Diversity Loss) طراحی شده است تا از استفاده یکنواخت از تمامی ورودی های کدبوک اطمینان حاصل شود. این تابع بر اساس بیشینه سازی آنتروپی توزیع کدبوک‌ها عمل میکند. در این روش، مدل تشویق میشود تا تمامی ورودی‌های موجود در هر کدبوک را با توزیع یکنواخت به کار گیرد. این توزیع احتمال میانگین سافت مکس (Softmax) را برای هر کدبوک $\overline{p}_g$ در یک دسته (batch) از داده‌ها بهینه سازی میکند. برخلاف سایر بخش‌های مدل، در این مرحله توزیع سافت مکس از نویز گامبل و پارامتر دما استفاده نمیکند، تا مدل دچار انحراف تصادفی نشود. فرمول این تابع هزینه به صورت زیر است:

$$L_d = \frac{1}{GV}\sum_{g=1}^{G} - H(\overline{p}_g) = \frac{1}{GV}\sum_{g=1}^{G}\sum_{v=1}^{V} p_{g,v} log(p_{g,v})$$

که در آن $p_{g,v}$ توزیع میانگین استفاده از ورودی v در کدبوک g در یک دسته از داده‌ها است.

با استفاده از این مکانیزم، مدل یاد می‌گیرد که از تمامی ورودی‌های کدبوک به‌طور مساوی استفاده کند و در نتیجه، ظرفیت بازنمایی آن افزایش می‌یابد. این امر به مدل امکان می‌دهد که طیف گسترده‌ای از الگو های گفتاری را پوشش دهد و بازنمایی‌های آن به سیگنال های صوتی مختلف بهتر تعمیم پیدا کند.

۵.۲. مرحله Fine-Tuning و کاربرد در تشخیص گفتار (ASR)

پس از پیش‌آموزش مدل Wav2Vec 2.0 با یادگیری خودنظارت‌شده، مرحلهٔ Fine-Tuning انجام می‌شود تا مدل برای وظایف خاصی مانند تشخیص گفتار (ASR) تنظیم شود. این فرآیند شامل افزودن یک لایه خطی پیش بینی برای تبدیل خروجی مدل به واحدهای متنی و بهینه سازی با تابع هزینه Connectionist Temporal Classification یا به اختصار CTC است. ابتدا خروجی ترنسفورمر که به شکل دنباله ای از بردارهای ویژگی $c_t \in R^{d_{tf}}$ برای هر گام زمانی t است. برای تبدیل این بازنمایی‌ها به توزیع احتمالاتی بر روی واژگان متنی، یک **لایه خطی** به بالای شبکه افزوده می‌شود:

$$l_t = Wc_t + b$$

که در آن $W \in R^{C \times d}$ ماتریس وزنی است که ویژگی‌های خروجی ترنسفورمر را به یک فضای C بعدی (که C تعداد کلاس‌های خروجی است) تبدیل میکند و $b \in R^C$ بردار بایاس است. همچنین l_t مقدار خام (logit) است که نشان میدهد مدل چقدر به هر کاراکتر از واژگان در لحظه t اطمینان دارد. برای تبدیل logit ها به توزیع احتمال، از تابع Softmax استفاده میشود:

$$P(k_t|x) = \frac{exp(l_{t,k})}{\sum_{k' \in C} exp(l_{t,k'})}$$

که در آن k_t کلاس پیشبینی شده در گام t، x ورودی صوتی خام و $l_{t,k}$ مقدار خروجی برای کلاس k در لحظه t است. اما چالش اصلی این است که مدل دنباله ای از پیشبینی ها را با طول متغیر تولید میکند که مستقیماً قابل تبدیل به متن خروجی نیست. برای حل این مشکل، از تابع هزینه CTC استفاده می‌شود.

در مدل های معمولی Seq2Seq، برای هر ورودی x یک خروجی y با طول مشخص و هم تراز شده وجود دارد. اما در CTC، مدل بدون داشتن هم ترازی مشخص بین ورودی و خروجی آموزش می بیند و خودش یاد میگیرد که متن را از روی بازنمایی های زمانی استخراج کند. CTC برای حل مسئله هم ترازی از یک توکن خالی (Blank Token) که معمولاً با ϵ نمایش داده می‌شود، استفاده میشود. این توکن به مدل اجازه میدهد تا پیشبینی هایش را بدون نیاز به هم ترازی مستقیم بین ورودی و خروجی، توزیع کند. اگر مدل چندین فریم متوالی یک کاراکتر را پیش‌بینی کند، مشخص نیست که این تکرارها یک حرف کشیده هستند (بر اساس سرعت صحبت کردن افراد) یا چندین نمونه از همان حرف (تکرار کاراکتر ها در یک کلمه). CTC این مشکل را حل میکند، زیرا تکرارهای متوالی را به یک حرف منفرد کاهش می‌دهد، مگر اینکه بین آنها یک توکن خالی باشد.

به عنوان مثال، اگر مدل دنباله خروجی $\{\epsilon, A, \epsilon, \epsilon, B, \epsilon, C, C, \epsilon\}$ را تولید کند. سپس با اعمال عملگر فشرده‌سازی B، این دنباله به خروجی "ABC" تبدیل می‌شود ($B(\epsilon, A, \epsilon, \epsilon, B, \epsilon, C, C, \epsilon) = "ABC"$). در این فرآیند توکن های ϵ خالی حذف میشوند، تکرارهای متوالی از یک کاراکتر (مانند CC) به یک حرف کاهش مییابد و اگر دو حرف مشابه بدون جداسازی ϵ باشند، آن‌ها یک حرف منفرد در نظر گرفته میشوند. بنابراین، مدل می‌تواند دنباله‌های گفتاری را بدون نیاز به هم ترازی مشخص، به توالی های متنی معنی دار فشرده کند. تابع هزینه CTC برای محاسبه احتمال رخداد هر جفت ورودی x و متن هدف y، مجموعه ای از تمام مسیرهای ممکن π را که میتوانند به متن y منجر شوند، محاسبه کرده و احتمال کلی آنها را در نظر می‌گیرد:

$$P(y|x) = \sum_{\pi \in B^{-1}(y)} P(\pi|x)$$

که در این فرمول $B^{-1}(y)$ مجموعهٔ تمامی مسیرهایی است که می‌توانند به متن هدف y نگاشت شوند و $P(\pi|x)$ احتمال مسیر π است که به‌صورت حاصل ضرب احتمالات در گام های زمانی مختلف محاسبه میشود:

$$P(\pi|x) = \prod_{t=1}^{T} P(k_t^{\pi}|x)$$

که در این فرمول $P(k_t^{\pi}|x)$ احتمال پیشبینی کلاس k_t^{π} در گام زمانی t برای مسیر π است و هر k_t^{π} به معنای کلاس (یا توکن) تولید شده در گام زمانی t برای مسیر π است. همچنین T طول دنباله زمانی ورودی x میباشد. در اینجا، k_t^{π} شامل توکن های خالی یا همان Blank Token ها باشد که نشان دهنده فضا های خالی یا وقفه ها در داخل دنباله هستند.معمولاً، این توکن ها به طور خاص به عنوان

ورودی مدل در نظر گرفته میشوند و به مدل کمک میکنند تا بتواند پیشبینی های دقیق تری در طول دنباله انجام دهد. این توکن ها میتوانند باعث شوند که مسیرهای متفاوتی از کلاس ها برای رسیدن به متن هدف مشابه y در نظر گرفته شوند.

در نهایت تابع هزینه CTC برای بهینه سازی مدل، مقدار منفی لگاریتم احتمال متن هدف هر دسته از داده که به نحوی همان لگاریتم تابع درستنمایی میباشد را محاسبه میکند:

$$L_{CTC} = \quad - \log P(y|x)$$

محاسبه این مقدار با الگوریتم پیش رو-پس رو (Forward-Backward Algorithm) انجام میشود که با روشی مشابه الگوریتم ویتربی (Viterbi Algorithm) مسیرهای معتبر را محاسبه کرده و احتمال نهایی را تخمین میزند و پس از آموزش، مدل دنباله ای از پیشبینی های CTC را تولید میکند و دو روش برای استخراج خروجی متنی نهایی وجود دارد:

۱. جستجوی حریصانه (Greedy Decoding): در هر فریم زمانی، محتمل ترین خروجی را انتخاب میکند و سپس عملگر فشرده سازی B را اعمال میکند که این روش سریع است اما ممکن است همیشه بهترین خروجی را تولید نکند.

۲. جستجوی پرتو (Beam Search Decoding): به جای انتخاب فقط یک مسیر، چند مسیر پر احتمال را دنبال میکند و در نهایت معتبرترین خروجی را انتخاب می‌کند. اگر یک مدل زبانی (Language Model) نیز در کنار این روش استفاده شود، خروجی متن نهایی به جملات روان تر و دقیق تر تبدیل خواهد شد.

پس از انجام Fine-Tuning مدل Wav2Vec 2.0، لازم است که دقت و عملکرد آن در وظیفه تشخیص گفتار خودکار (ASR) ارزیابی شود. برای این منظور، از معیار های استاندارد استفاده می‌شود که میزان خطای مدل را در تبدیل سیگنال گفتاری به متن اندازه گیری میکنند. یکی از رایج ترین این معیار ها، نرخ خطای کلمه (WER - Word Error Rate) است که در بیشتر پژوهش ها به عنوان معیار اصلی گزارش میشود. WER معیار ارزیابی اصلی برای سنجش کیفیت خروجی یک مدل تشخیص گفتار است و به صورت زیر تعریف میشود:

$$WER = \frac{S+D+I}{N}$$

که در آن S (Substitutions) تعداد کلمات جایگزین شده در خروجی مدل، D (Deletions) تعداد کلمات حذف شده که مدل در پیشبینی نهایی آنها را از دست داده است، I (Insertions) تعداد کلمات اضافه شده که مدل به اشتباه در خروجی تولید کرده است و N تعداد کل کلمات در متن مرجع (Ground Truth) میباشند. WER مقیاسی بدون بعد و همیشه غیرمنفی است که مقدار کمتر آن نشان دهنده عملکرد بهتر مدل است. فرض کنید متن مرجع (Ground Truth) به صورت زیر باشد:

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

و خروجی مدل ASR به صورت زیر تولید شود:

THE FAST BROWN FOX JUMP OVER A LAZY DOG

با مقایسه این دو جمله، 2 جایگزینی (S) "JUMPS" → "JUMP"، "QUICK" → "FAST"، حذف (D) 1 کلمه "THE" در انتهای جمله و افزودن (I) کلمه "A" قبل از "LAZY DOG" مشاهده میشود. با توجه به اینکه تعداد کل کلمات در متن مرجع $N = 9$ است، مقدار WER برابر با $WER = \frac{1+1+2}{9} \approx 0.444$ یا $44.4\ \%$ خواهد بود.

علاوه بر WER، برخی معیارهای دیگر نیز برای سنجش دقت مدل در نظر گرفته میشوند. به عنوان مثال Character Error Rate یا به اختصار CER مشابه WER است اما در سطح کاراکترها محاسبه میشود. این معیار در زبان‌هایی مانند چینی، ژاپنی و کره‌ای که تقسیم بندی کلمه ای مشخصی ندارند، کاربرد دارد.

$$CER = \frac{S_C + D_C + I_C}{N_C}$$

که در آن S_C، D_C و I_C به ترتیب تعداد جایگزینی ها، حذف ها و افزوده های کاراکترها هستند.

یکی از مهمترین نتایج گزارش شده در مقاله Wav2Vec 2.0، توانایی مدل در رسیدن به نتایج رقابتی حتی با داده بسیار کم دارای برچسب (مثل ۱۰ ساعت یا ۱۰۰ ساعت از مجموعهٔ LibriSpeech) میباشد. این نشان می‌دهد که مرحل پیش‌آموزش خودنظارت، بخش اعظم ویژگی های لازم برای تشخیص گفتار را قبلاً یاد گرفته است.

۳. مجموعه دادگان

در این پروژه از مجموعه داده Common Voice 11.0 که توسط Mozilla Foundation منتشر شده است، استفاده شده است. این مجموعه یکی از بزرگترین مجموعه های دادگان متن باز برای آموزش و ارزیابی مدل‌های باز شناسی گفتار محسوب میشود. داده های این مجموعه شامل نمونه های صوتی ضبط شده توسط کاربران داوطلب از سراسر جهان است که به زبان های مختلف، از جمله انگلیسی، در دسترس قرار گرفته اند. هر نمونه صوتی شامل یک فایل صوتی ضبط شده و رونوشت متنی متناظر آن است، که این ویژگی آن را برای آموزش مدل‌های تبدیل گفتار به متن (ASR) بسیار مناسب میکند.

فرآیند جمع آوری داده ها به این صورت انجام شده که کاربران داوطلب به پلتفرم Common Voice مراجعه کرده و جملات کوتاهی که توسط Mozilla تهیه شده اند را مشاهده میکنند. این کاربران می‌توانند جملات را با صدای خود ضبط کنند یا به نمونه های صوتی سایر

کاربران گوش دهند و تأیید کنند که آیا متن ضبط شده با گفتار مطابقت دارد یا خیر. بعد از ضبط صدا، سایر کاربران با گوش دادن به نمونه‌ها، صحیح بودن آنها را تایید یا اصلاح میکنند. نمونه های تایید شده به مجموعه داده نهایی اضافه شده و همراه با اطلاعات جانبی مانند جنسیت، سن و لهجه گوینده منتشر میشوند. داده‌های پردازش شده در قالب مجموعه داده‌ای ساختاریافته در دسترس قرار می‌گیرند که شامل فایل‌های صوتی در فرمت WAV و متن‌های مرتبط در قالب JSON یا TSV است. علاوه بر این، اطلاعات اضافی مانند زبان، کیفیت ضبط، مدت زمان و لهجه گوینده نیز در این مجموعه موجود است.

مجموعه داده Common Voice برای این پروژه انتخاب شده است زیرا به دلیل متناظر بودن صوت و متن، مدل Wav2Vec2 میتواند از این داده‌ها برای یادگیری موثر تر استفاده کند. همچنین تنوع بالای گویندگان باعث می‌شود که مدل روی نمونه های واقعی عملکرد بهتری داشته باشد و از وابستگی بیش از حد به الگوهای صوتی خاص جلوگیری شود. این مجموعه داده به صورت متن باز و رایگان در دسترس است، که یک مزیت بزرگ برای تحقیقات دانشگاهی و مدل‌سازی محسوب میشود. علاوه بر این، مجموعه Common Voice دارای حجم بالایی از داده‌های صوتی است که به بهبود دقت مدل و جلوگیری از بیش‌برازش (Overfitting) کمک میکند. برخلاف بسیاری از مجموعه‌های داده صوتی که فاقد رونوشت‌های دقیق هستند و نیاز به پردازش‌های اضافی دارند، داده‌های این مجموعه از قبل برچسب گذاری شده و پردازش شده‌اند که باعث سهولت استفاده برای fine-tuning میشود.

برای این پروژه، ۱۵۰۰ نمونه صوتی برای آموزش و ۳۰۰ نمونه برای آزمون از مجموعه Common Voice 11.0 انتخاب شده است. این نمونه‌ها از مجموعه داده به صورت تصادفی بارگیری شده و به صورت محلی ذخیره شده اند. به طور کلی، مجموعه داده مورد استفاده در این پروژه به دلیل کیفیت بالای نمونه های صوتی، برچسب گذاری دقیق، حجم مناسب و تنوع زیاد گویندگان، انتخاب شده تا مدل wav2vec 2.0 با تابع زیان CTC و با استفاده از این داده ها برای این داده های fine-tune شود.

۴. توضیحات پیاده سازی

۱.۴. آماده‌سازی داده‌ها (Data Preparation)

در این بخش، ابتدا مجموعه داده Common Voice 11.0 بارگیری شده است. به دلیل حجم بالای این مجموعه، داده‌ها به صورت استریمینگ دریافت شده‌اند و تعداد مشخصی از نمونه‌های صوتی برای پردازش انتخاب شده است. در این پروژه 1500 نمونه برای آموزش و 300 نمونه برای آزمایش مورد استفاده قرار گرفته است.

بعد از دریافت داده ها، ستون های غیرضروری شامل سن، جنسیت، لهجه و شناسه کاربر حذف شده اند تا تنها اطلاعات مورد نیاز برای یادگیری مدل باقی بماند. سپس، متون متناظر با داده های صوتی پاکسازی شده اند تا کاراکترهای غیرضروری مانند نقطه، ویرگول، نقل قول‌ها و علائم نگارشی حذف شوند و تمامی متن ها به حروف کوچک تبدیل شوند.

واژگان استفاده شده در این مجموعه داده استخراج شده و یک واژگان سفارشی ساخته شده است که برای مدل Wav2Vec2 بهینه باشد. این واژگان شامل تمامی کاراکترهای موجود در مجموعه داده، همراه با توکن های [UNK] و [PAD] است که برای کاراکتر های شناخته نشده و توکن های خالی است. فایل واژگان در یک فایل JSON ذخیره شده و در ادامه برای پردازش داده های ورودی استفاده شده است.

در نهایت، داده‌های صوتی به قالب PyTorch Tensor تبدیل شده اند تا بتوانند در مدل مورد استفاده قرار گیرند. این تبدیل شامل پردازش فایل های صوتی و آماده‌سازی آنها برای ورود به مدل است.

۲.۴. پیکربندی مدل (Model Configuration)

پس از آماده سازی داده‌ها، در این بخش مدل Wav2Vec2 برای یادگیری تنظیم شده است. ابتدا توکنایزر (Tokenizer) و استخراج کننده ویژگی (Feature Extractor) به صورت شخصی شده برای داده های مسئله این پروژه تعریف شده اند. از `Wav2Vec2CTCTokenizer` برای پردازش متون و از `Wav2Vec2FeatureExtractor` برای تبدیل فایل های صوتی خام به فرمت قابل استفاده برای مدل استفاده شده است. این دو بخش در قالب یک پردازشگر مشترک (Wav2Vec2Processor) ترکیب شده اند تا بتوانند فایل های صوتی و متون را با هم پردازش کنند.

در ادامه، مدل `Wav2Vec2ForCTC` از مجموعه مدل های Facebook AI بارگیری شده است. این مدل قبلاً روی مجموعه داده های گسترده ای آموزش دیده، اما برای این پروژه دوباره تنظیم (Fine-tune) میشود.

سپس، پارامترهای آموزشی مدل مانند نرخ یادگیری (`learning_rate`)، تعداد ایپاک‌ها (`num_train_epochs`) و سایر تنظیمات بهینه سازی در قالب یک شئ TrainingArguments تنظیم شده اند. در این نسخه از کد، نرخ یادگیری 5e-4 تنظیم شده که مقدار بهینه ای برای جلوگیری از نوسانات مدل در طول آموزش است.

۳.۴. آموزش و ارزیابی مدل (Model Training and Evaluation)

در این مرحله، داده های آموزشی به دو بخش آموزش (90%) و اعتبارسنجی (10%) تقسیم شده اند. برای مدیریت داده های ورودی در طول آموزش، یک Data Collator سفارشی تعریف شده است که داده‌های صوتی و برچسب‌ها را به صورت پد شده (Padded) آماده میکند. در این مرحله، توکن‌های [PAD] که برای پر کردن فضای خالی استفاده شده اند، به مقدار -100 تبدیل شده اند تا در طول یادگیری نادیده گرفته شوند.

سپس مدل با استفاده از کلاس Trainer آموزش داده شده است. در این مرحله، مدل داده‌های پردازش‌شده را دریافت کرده و بر اساس تابع هزینه CTC آموزش میبیند. Gradient Checkpointing نیز فعال شده است تا مصرف حافظه در حین یادگیری بهینه شود.

برای ارزیابی مدل، از معیار WER استفاده شده است که میزان اختلاف بین متن پیش‌بینی شده توسط مدل و متن اصلی را محاسبه میکند. برای این منظور، ابتدا پیشبینی های مدل استخراج شده و پس از پردازش، با متون واقعی مقایسه شده اند. در نهایت، WER محاسبه شده و نتایج ارزیابی نمایش داده شده اند و پس از پایان آموزش، مدل روی مجموعه آزمون اجرا شده و عملکرد نهایی آن بررسی شده است.

۵. تحلیل نتایج

نتایج آموزش مدل نشان‌دهنده‌ی روند یادگیری تدریجی Wav2Vec2 است. با مقایسه‌ی Training Loss و Validation Loss مشاهده می‌شود که مقدار Training Loss از 6.53 در اپیاک اول به 5.75 در اپیاک پنجم کاهش یافته است. این کاهش نشان میدهد که مدل به درستی در حال یادگیری الگوهای صوتی از داده‌های آموزشی است. همچنین، مقدار Validation Loss در ابتدای آموزش 5.55 بوده و با نوساناتی، در حدود 5.54 در اپیاک پنجم ثابت مانده است. این نشان میدهد که مدل توانسته است روی داده‌های اعتبارسنجی نیز عملکرد مناسبی از خود نشان دهد .

Epoch	Training Loss	Validation Loss	Wer
1	6.537300	5.558956	0.977926
2	6.140600	5.629985	0.977926
3	5.927200	5.478723	0.977926
4	5.767700	5.580047	0.977926
5	5.751400	5.542346	0.977926

مقدار WER ثابت در طول اپیاک ها می‌تواند نشان دهنده ی یک مرحله اولیه از آموزش باشد که مدل هنوز در حال همگرایی است. در بسیاری از پروژه‌های Fine-Tuning مدل‌های خودنظارتی مانند Wav2Vec2، مقدار WER در ابتدا بالاست و کاهش آن نیاز به تعداد اپیاک های بیشتری دارد. که با توجه به حجم پردازش بالای مدل و اینکه جمع آوری داده بیشتر از سایت common voice نیازمند سیستم قوی تر است، موفق به اجرای مدل در اپیاک های بالاتر نشدیم. همچنین، روند نزولی در Training Loss نشان میدهد که مدل به تدریج در حال یادگیری ویژگی های جدید از داده های صوتی است.

پس از آموزش مدل، برای 300 نمونه‌ی آزمایشی مورد ارزیابی قرار گرفت که مقدار WER نهایی برابر با 0.98 به دست آمده است. این مقدار اگرچه هنوز بالا است، اما نکته‌ی مهم این است که مدل توانسته است یک ساختار اولیه‌ی یادگیری را ایجاد کند. در مقایسه با مدل‌هایی که بدون Pretraining آموزش داده می‌شوند، این مقدار WER نشان می‌دهد که مدل حداقل توانسته است برخی از الگوهای گفتاری را یاد بگیرد.

همچنین عملکرد مدل را میتوان با گذراندن پیشبینی ها از یک مدل پردازش زبان بهبود داد که در مقاله اصلی اینکار صورت گرفته و WER را کاهش داد.

۶. منابع

کد های پیاده سازی از مقاله زیر ایده گرفته شده که نحوه‌ی تنظیم و آموزش مدل Wav2Vec2 را توضیح داده است:

https://huggingface.co/blog/fine-tune-wav2vec2-english

تصاویر مربوط به توضیح مدل Wav2Vec2 از وب سایت زیر برداشته شده:

https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html

مقاله اصلی Wav2Vec 2.0 که مبانی نظری و ریاضی این مدل را توضیح داده:

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