به نام خدا

# بررسی Language Models

استاد لیلی محمد خانلی

محمدامین غنی زاده

## مقدمه

با ورود ChatGPT و تکنولوژی های مشابه به دنیای تکنولوژی، تب LLM ها داغ شده است. تکنولوژی های متنوعی داخل اینگونه مدل ها استفاده میشوند که به بررسی آن ها خواهیم پرداخت. اما ابتدا باید به طور کلی NLP مورد بررسی قرار گیرد تا بتوانیم تکنولوژی های مربوطه ی داخل LLM ها و PLM ها را درک کنیم. پس ابتدا به بررسی پردازش زبان طبیعی خواهیم پرداخت.

# مروری روی NLP

## تاریخچه

پردازش زبان طبیعی یا همان NLP، یک زیررشته از علوم کامپیوتر و هوش مصنوعی است که روی فعل انفعالات بین کامپیوتر ها و زبان های انسانی متمرکز است. NLP تاریخچه ی نسبتا طولانی را دارا است که از 1950 و با توسعه ی اولین سیستم ترجمه زبانی، شروع شده است. برخی از نقاط عطف در تاریخ NLP را میتوان بدینگونه بیان کرد:

- 1950: اولین سیستم ترجمه از روسی به انگلیسی توسط محققان دانشگاه جرج تاون توسعه داده شد.
- 1960: رشته computational linguistics برای اولین بار ظهور کرد. این رشته با استفاده از متد های محاسباتی، به زبانی شناسی می پردازد.
- 1970: اولین چک کننده کامپیوتری اشتباهات گرامری ظهور کرد.
- 1980: مفهوم یادگیری ماشین برای NLP معرفی شد که باعث شد مدل های زبانی پیچیده تری توسعه پیدا کند.
- 1990: شبکه جهانی وب ظهور کرد که این مسئله باعث تولید دیتای متنی در حجم های عظیم شد.
- 2000: فیلد تحلیل احساسات یا sentiment analysis ظهور پیدا کرد که به وسیله آن احساسات نویسنده متن، تحلیل میشود.
- 2010: در این دهه و با اوج گرفتن یادگیری عمیق، مدل های زبانی پیچیده تری به وسیله تکنیک های یادگیری عمیق مانند شبکه های عصبی ساخته شدند و انقلابی در حوزه پردازش زبان طبیعی پدید آمد.

امروزه NLP در بسیاری از کاربرد ها مورد استفاده قرار میگیرد. از جمله آن ها میتوان به ترجمه، تحلیل احساسات، خلاصه سازی و تشخیص گفتار اشاره کرد.

## زیر شاخه های NLP

در این بخش به بررسی برخی از تسک هایی که در پردازش زبان طبیعی انجام میشوند، میپردازیم:

- دسته بندی متن یا Text Classification: همان طور که از اسم آن پیداست، در این تسک سعی بر این است که یک برچسب یا label به یه متن نسبت بدهیم. برای مثال فرض کنید بازخورد کاربران برای یک محصول را در اختیار داریم و میخواهیم ببینیم چند درصد نظرات مثبت و چند درصد منفی است. به این منظور به هر بازخورد مقدار "مثبت" یا "منفی" نسبت میدهیم. این کار را با مدل های مختلفی میتوان انجام داد، از جمله شبکه های عصبی.
- شناسایی دسته بندی شده نام ها یا Named Entity Recognition: در این تسک تلاش بر این است که موجودیت های نام دار مانند نام اشخاص، مکان ها یا نهادها را از متن استخراج کنیم. برای مثال از جمله "محمد اهل تبریز است"، محمد به عنوان شخص و تبریز به عنوان یک محل شناسایی شود.
- برچسب گذاری بخش های گفتاری یا Part-of-Speech Tagging: این تسک شامل شناسایی ساختار دستوری جمله و برچسب گذاری به عنوان مثلا اسم، فعل، فاعل و ... میشود. این تسک میتواند برای تسک های دیگر مانند تبدیل متن، ترجمه و تشخیص گفتار مفید باشد.
- تحلیل احساسات یا Sentiment Analysis: همان طور که قبلا هم گفته شد، در این تسک سعی میشود تا احساسات نویسنده متن از روی متن استخراج شود. این تسک میتواند برای کارهایی مثل تحلیل بازخورد مشتریان مفید باشد.
- Machine Translation: در این تسک، یک متن از یک زبان به یک زبان دیگر ترجمه میشود. معماری Transformers که جلوتر با آن آشنا خواهیم شد، برای اولین بار برای حل این مسئله پیشنهاد شد.
- پاسخ به سوالات یا Question Answering: همان طور که از اسم پیداست، یه رشته به عنوان ورودی داده میشود و انتظار میشود جواب آن رشته خروجی داده می شود. این تسک میتواند شامل تسک های دیگری مانند بازیابی اطلاعات باشد، به طوری که سیستم از یک مجموعه ای از اسناد برای یافتن جواب سوال پرسیده شده استفاده می کند.
- خلاصه سازی متن یا Text Summarization: ورودی یک متن و خروجی خلاصه سازی شده آن متن است. شامل دو نوع است:
  - Extractive summarization: سیستم فقط جمله های مهم را از متن اصلی انتخاب می کند.
  - Abstractive summarization: سیستم جمله های جدیدی را به عنوان خلاصه متن اصلی تولید می کند.

برخی از تسک های پر تکرار درون NLP آورده شدند. البته حوزه NLP بسیار گسترده است و در ترکیب آن با حوزه های دیگر مانند بینایی ماشین، تسک های زیاد و متنوعی وجود دارد.

## مدل ها و الگوریتم های مورد استفاده در NLP

در این بخش به بررسی برخی از مدل ها و الگوریتم هایی که برای حل کردن تسک های NLP که در بخش قبلی معرفی شدند، پرداخته می شود. از مدل های ساده تر شروع کرده و به معرفی مدل های پیچیده تر خواهیم پرداخت.

### Bag of Words (BoW)

یکی از تکنیک های ابتدایی و ساده برای تبدیل دیتا های متنی به دیتای عددی است. فرض کنید جملات زیر را در اختیار داریم و هدف، تبدیل دیتا ها به داده های عددی است:

The quick brown fox jumped over the lazy dog. The dog chased the cat up the tree. The cat sat in the tree and watched the dog.

قدم اول تشکیل برداری از لغات منحصر به فرد است. بردار ما به این شکل خواهد بود:

$$[The \quad quick \quad brown \quad fox \quad jumped \quad over \quad lazy \quad dog \quad chased \quad cat \quad up \quad tree \quad and \quad watched \quad in]$$

حال برای مثال، برای نمایش دادن جمله اول، کلماتی که در جمله اول وجود دارند را با 1 و کلماتی که وجود ندارند را با 0 نمایش میدهیم. پس نمایش جمله اول به این شکل خواهد شد:

$$[1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

این یکی از تکنیک های ساده ای است که می توان برای تبدیل متن ها به داده عددی استفاده شود تا بعدا از این بردار ها در مدل ها استفاده شود.

### N-gram Model

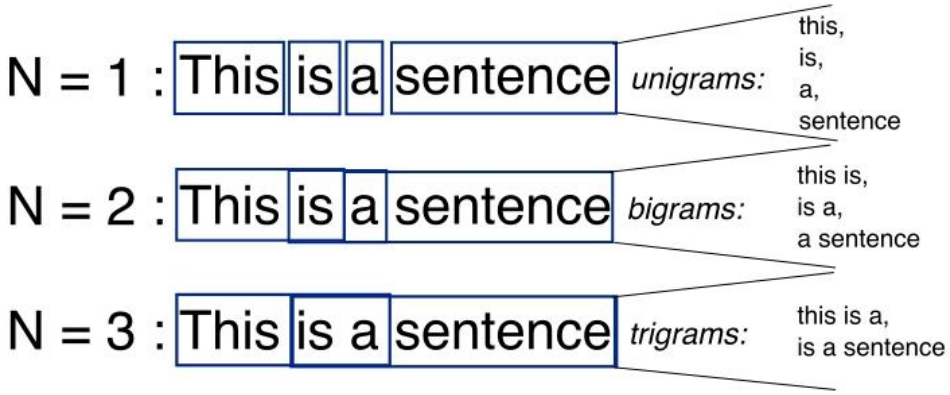
یک مدل احتمالاتی است که برای پیشبینی احتمال یک رشته از کلمات استفاده می شود. در این مدل این فرض وجود دارد که احتمال هر کلمه تنها به n-1 کلمه قبل از آن وابسته است. برای مثال فرض کنید این جمله را در اختیار داریم:"The cat sat on the mat"، مدل های مختلف به این شکل خواهند شد:

- Unigrams: The, cat, sat, on, the, mat
- Bigrams: The cat, cat sat, sat on, on the, the mat
- Trigrams: The cat sat, cat sat on, sat on the, on the mat
- 4-grams: The cat sat on, cat sat on the, sat on the mat

برای ساختن این مدل، ابتدا میزان تکرار هر یک از n-gram در متن اصلی مورد شمارش قرار میگیرد. سپس این تکرار ها مورد استفاده قرار میگیرد تا میزان احتمال هر یک از n-gram ها به دست آید. همچنین میزان احتمال هر کلمه به وسیله n-1 کلمه قبلی آن به دست می آید. برای مثال اگر میزان تکرار "cat sat" و تکرار "cat sat on" را در اختیار داشته باشیم، می توانیم این احتمال را محاسبه کنیم:

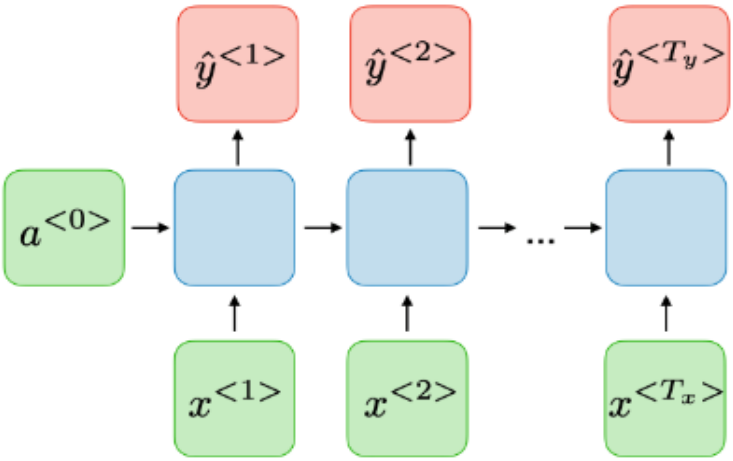
$$P(on \mid cat\ sat) = count(cat\ sat\ on) / count(cat\ sat)$$

این احتمال، احتمال کلمه "on" را بعد از "cat sat" محاسبه می کند. شکل زیر نحوه محاسبه با N های مختلف را نمایش می دهد.



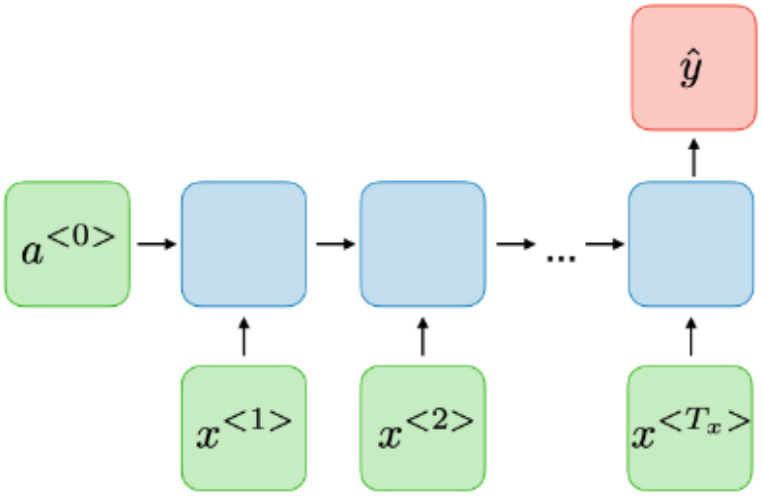

## شبکه های RNN

این شبکه ها، نوعی از شبکه های عصبی هستند که برای داده های ترتیبی طراحی شده اند. داده های ما در پردازش زبان طبیعی، داده های ترتیبی هستند پس این شبکه ها می توانند برای ما مفید واقع شوند. این شبکه ها بر خلاف شبکه های عصبی معمولی، دارای حافظه هستند که این حافظه این امکان را به آن ها میدهد که الگو ها را در یک رشته از داده ها چیدا کنند. از جمله شبکه های RNN می توان به Long Short Term Memory یا همان LSTM و GRU اشاره کرد. به عنوان یک نمونه از معماری شبکه RNN به شکل زیر دقت کنید:




با توجه به شکل، خروجی در هر مرحله، به ورودی در آن مرحله و همه ی ورودی های قبلی وابسته است. این معماری از RNN میتواند برای تسک Name entity recognition استفاده شود زیرا برای هر کلمه از ورودی، یک tag

نسبت داده میشود. حال فرض کنید میخواستیم از RNN ها برای تحلیل احساسات استفاده کنیم. در این تسک فقط به یک خروجی که همان احساسات نویسنده متن است نیاز خواهیم داشت. میتوان از این معماری برای این امر استفاده کرد:




همانطور که از شکل مشخص است، خروجی به همه ی اجزای رشته ورودی وابسته است.

## شبکه های Transformer

این معماری در یک مقاله به اسم "Attention Is All You Need" در سال 2017 معرفی شد. در این بخش فقط به مزایای این معماری در مقایسه با RNNs خواهیم پرداخت تا در بخش های بعدی این شبکه هارا با جزییات بیشتری بررسی کنیم. مزایای این معماری را به این صورت می توان لیست کرد:

- پردازش موازی: RNN ها داده ها را به صورت سری پردازش می کنند، به معنی که هر خروجی در هر زمان به ورودی های زمان های قبلی وابسته است. این امر موازی سازی پردازش ها در این معماری را دشوار می کند که در نتیجه سرعت محاسبات کاهش می یابد. در مقابل در معماری Transformers، همه توکن های رشته ورودی به صورت موازی پردازش می شود که این سرعت محاسبات را بالا می برد.
- وابستگی ها با برد زیاد: RNN ها در یاد گرفتن وابستگی های با برد زیاد در یک رشته، با مشکلاتی مواجه هستند، که یکی از دلایل این امر مشکل vanishing gradient است. به این معنی که تاثیر ورودی های قبلی به خروجی های با فاصله زیاد از آن ورودی، کمتر و کمتر می شود. در مقابل، Transformer ها از مکانیزم Attention استفاده می کنند که این مکانیزم، وابستگی بین همه توکن ها در یک رشته را بدون در نظر گرفتن فاصله آن دو اندازه می گیرد.
- مناسب برای Transfer learning: این نوع یادگیری شامل آموزش مدل روی یک دیتاست بزرگ و استفاده از وزن های آموزش دیده برای fine tune کردن مدل روی دیتای مورد نظر می شود. معماری Transformer برای این امر مناسب تر از RNN ها است.

قبل از بررسی معماری Transformer، مروری روی معیار های اندازه گیری دقت مدل ها در NLP خواهیم داشت. توجه داشته باشید که معیار هایی مثل دقت، precision و recall که در همه زیر شاخه های یادگیری ماشین مرسوم هستند مورد بررسی قرار نخواهند گرفت و تمرکز روی معیار های مدل های NLP خواهد بود. البته این به این معنی نیست که معیار های نامبرده در NLP استفاده نمی شوند.

## معیار های مورد استفاده در NLP

### ❖ معیار Perplexity

این معیار برای اندازه گیری کیفیت مدل های زبانی یا همان language models استفاده می شود. مدل زبانی یک توزیع احتمال روی جمله ها به ما می دهد. معیار perplexity در پردازش زبان طبیعی، میزان عدم قطعیت یک مدل زبانی را در پیش بینی متن اندازه میگیرد. پس هرچقدر که این معیار برای یک مدل کم تر باشد، عدم قطعیت کمتر و در نتیجه قطعیت مدل بیش تر است که این مطلوب ما است. در نتیجه مقدار کم تر این معیار، نشان دهنده مدل زبانی بهتری است. یک مدل زبانی بدون نقص، Perplexity برابر یک خواهد داشت که به این معنی است که همه ی کلمه های تست ست را بدون نقص پیشبینی می کند. فرمول این معیار اندازه گیری را به این صورت می توان بیان کرد:

$$PP(p) = 2^{-\sum_x p(x) \cdot \log_2 p(x)}$$

که در آن $p(x)$ نشان دهنده احتمال نسبت داده شده توسط مدل زبانی است. برای درک بهتر، مثالی برای محاسبه این معیار می آوریم. فرض کنید که مدل روی داده های انگلیسی آموزش دیده و داده های تست ما به این شکل هستند:

1. The cat sat on the mat
2. The dog chased the cat
3. The cat meowed and the dog barked

حال فرض کنید که از داده های تست کلمه آخر را حذف کرده، بقیه جمله را به مدل داده و از مدل می خواهیم که احتمال کلمه آخر را برایمان پیشبینی کند، مدل احتمالات زیر را برمی گرداند:

$$P(mat \mid The\ cat\ sat\ on\ the) = 0.8$$

$$P(cat \mid The\ dog\ chased\ the) = 0.9$$

$$P(Barked \mid The\ cat\ meowed\ and\ the\ dog) = 0.95$$

همانطور که مشخص است مدل با احتمال خوبی کلمات درست را پیشبینی می کند. با توجه به فرمول، ما احتمالات را در اختیار داریم و باید عبارت زیر را محاسبه کنیم:

$$\sum_x p(x) \cdot \log_2 p(x)$$

$$= 0.8 \cdot \ -0.3219 \ + \ 0.9 \cdot -0.152 + 0.95 \cdot -0.074 \ = \ -0.4646$$

$$\Rightarrow \ 2^{-\sum_x p(x) \cdot \log_2 p(x)} \ = \ 2^{0.4646} \ = \ 1.3799$$

پس معیار Perplexity برای این مثال، 1.3799 به دست آمد که با توجه به مقادیر احتمال مناسبی که مدل پیشبینی کرده، منطقی است و همان طور که قبلا اشاره شد، هرچقدر این معیار به یک نزدیک تر باشد، مدل مناسب تری در اختیار داریم. این معیار از جمله معیار های پر استفاده در language model ها و LLM ها است.

### ❖ BLEU score

معیار BLEU یا Bilingual Evaluation Understudy، معیاری برای اندازه گیری کیفیت سیستم های ترجمه یا همان machine translation است. این معیار شباهت بین ترجمه تولید شده توسط سیستم ترجمه و ترجمه های تولید شده توسط انسان ها را اندازه می گیرد. این معیار عددی بین 0 و 1 است که 0 بیان کننده بدترین ترجمه و 1 بیان کننده بهترین ترجمه تولید شده توسط سیستم ترجمه است. برای محاسبه این معیار ابتدا n-gram های ترجمه تولید شده توسط مدل و رفرنس هارا پیدا میکنیم. سپس n-gram هایی که در ترجمه تولید شده توسط مدل و رفرنس ها هستند را می شماریم. همچنین یک عبارت دیگر به اسم brevity penalty را محاسبه می کنیم که ترجمه هایی که کوتاه تر از رفرنس ها هستند را جریمه می کند.

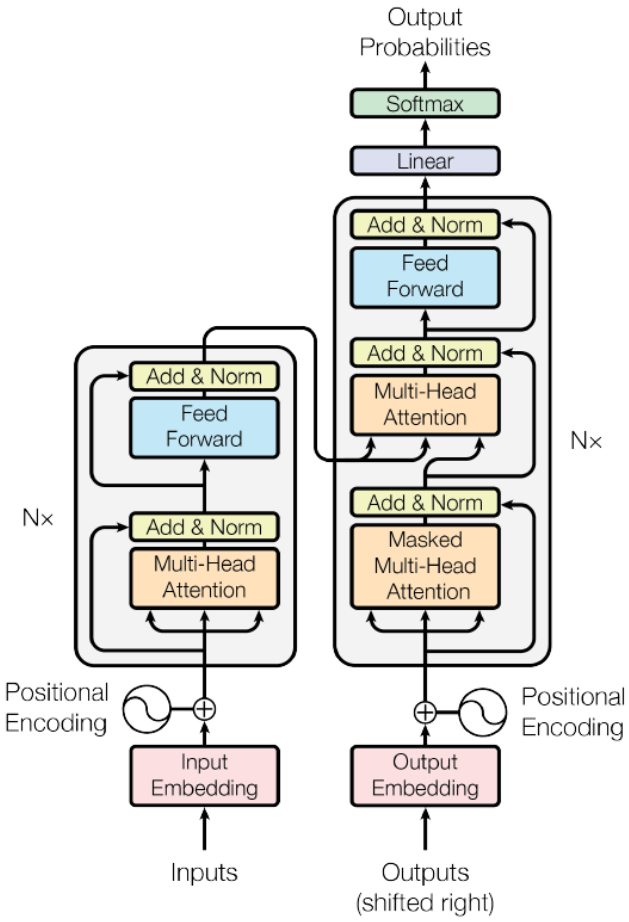
### ❖ ROUGE score

معیار ROUGE یا Recall-Oriented Understudy for Gisting Evaluation مجموعه ای از معیار ها برای اندازه گیری کیفیت سیستم های خلاصه سازی هستند. این معیار میزان همپوشانی بین خروجی تولید شده توسط سیستم خلاصه سازی و خلاصه ی فراهم شده توسط انسان را اندازه می گیرد. چندین نوع از این معیار از جمله ROUGE-1، ROUGE-2 و ROUGE-L وجود دارند که در اندازه گیری همپوشانی با هم تفاوت دارند.

سپس به بررسی معماری Transformer و اجزای تشکیل دهنده آن می پردازیم.

# معماری Transformer

همانطور که قبلا اشاره شد این معماری در مقاله ای در سال 2017 با نام "Attention Is All You Need" معرفی شد. در مقاله اصلی این معماری از دو بخش encoder و decoder تشکیل شده است و برای تسک Machine translation معرفی شده است. بعد ها این معماری برای تسک های دیگر Seq2Seq مانند خلاصه سازی و تولید متن و حتی در حوزه بینایی ماشین در ViT مورد استفاده قرار گرفته است. این معماری بر پایه مکانیزم Attention عمل می کند و روی بخش های مهم رشته ورودی تمرکز بیش تری میگذارد. به عبارت دیگر وزن های متفاوتی را برای هر بخش از رشته ورودی تخصیص می دهد. بخش encoder رشته ورودی را میگیرد و یک رشته hidden state تولید می کند. در حالی که decoder خروجی encoder را می گیرد و یک رشته خروجی را به ازای یک توکن در هر زمان تولید می کند. شکل مربوط به این معماری در مقاله اصلی به این صورت آمده است:




همانطور که از شکل مشخص است هر کدام از encoder و decoder از بخش های مختلفی تشکیل شده اند. اکنون به بررسی بخش های مختلف این معماری می پردازیم.

## Scaled dot-product Attection

می توان گفت که مهم ترین بخش معماری این بخش است. این مکانیزم در هردوی encoder و decoder مورد استفاده قرار گرفته است. مراحل محاسبه به این صورت است:

1. ابتدا بردارQuery یا Q، بردار Key یا K و بردار Value یا V برای هر توکن محاسبه می شوند. این بردار ها پارامتر های یادگیرنده مدل ها همان learned parameters هستند.
2. سپس ضرب داخلی بین بردار های Q و K محاسبه می شود که نتیجه آن ماتریسی است که نشان دهنده شباهت بین هر جفت از توکن ها است.
3. سپس این ماتریس به ریشه دوم اندازه سر attention یا attention head تقسیم می شود. این عمل برای جلوگیری از اندازه بیش از حد این مقادیر اعمال می شود.
4. سپس این مقادیر به یک تابع Softmax ورودی داده می شود تا normalize شوند.
5. در نهایت این وزن ها استفاده می شوند تا یک جمع وزن دار از بردار V برای هر توکن محاسبه شوند.

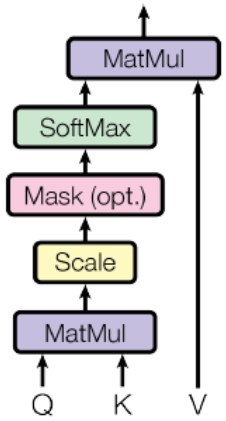
این مراحل در این فرمول که در مقاله اصلی آمده خلاصه می شود:

$$Attention(Q,\ K,\ V)\ =\ soft\max\left(\frac{Q\cdot K^T}{\sqrt{d_k}}\right)\cdot V$$

که تابع softmax به این صورت تعریف می شود:

$$\sigma(\vec{z})_i\ =\ \frac{e^{z_i}}{\sum_{j=1}^{K}\ e^{z_j}}$$

شکلی برای نمایش طرز کار Scaled dot-product attention:

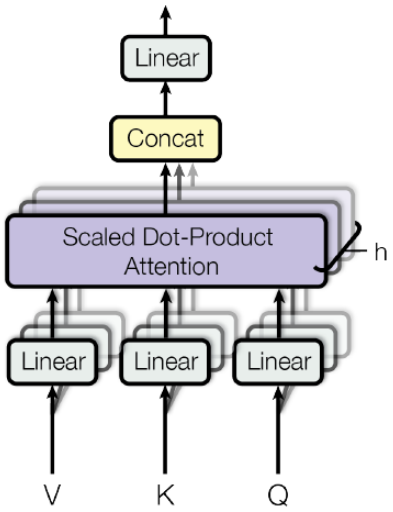


## Multi-head Attention

در مقاله اصلی پیشنهاد شده به جای استفاده از یک تابع attention تنها، از چندین تابع به صورت موازی استفاده کنیم تا هر کدام روی جنبه متفاوتی از داده ورودی تمرکز می کنند که این باعث می شود توانایی یاد گرفتن رابطه های پیچیده تری را داشته باشند. این عمل را می توان به این صورت نشان داد:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

همانطور که فرمول نشان می دهد، پس از محاسبه head های مختلف، آن ها را concatenate می کنیم. با این شکل می توان این عملیات را خلاصه کرد:




در فرمول بالا پس از Concatenate کردن، به یک $W^O$ هم ضرب شده که همان لایه Linear آخر در شکل بالا است.

## Feed-Forward Networks

علاوه بر زیر لایه های Attention، هر یک از encoder و decoder حاوی یک شبکه Feed-Forward هستند. این بخش شامل دو عدد تبدیل خطی همراه یک تابع فعال ساز ReLU بین آن ها است:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

## Embeddings

این لایه برای تبدیل توکن های ورودی به بردار استفاده می شود. این لایه نیز پارامتر های قابل یادگیری دارد که در مرحله یادگیری بهینه می شوند.

## Positional Encoding

در معماری transformer، بر خلاف RNN ها ساختار بازگشتی وجود ندارد که تا اطلاعات مربوط به موقعیت هر توکن به مدل وارد شود، از این جهت برای اعمال اطلاعات مربوط به ترتیب قرارگیری توکن ها از Positional Encoding استفاده می شود. بردار های مربوط به Positional Encoding هر توکن با بردار Embedding آن هم بعد هستند و پس از محاسبه Embedding، این بردار به آن اضافه می شود تا اطلاعات ترتیبی هر توکن را وارد کنیم. از معروف ترین توابع که برای Positional Encoding استفاده می شوند می توان به تابع سینوس و کسینوس اشاره کرد. فرمول این توابع برای محاسبه Positional Encoding به این صورت است:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

که pos نشان دهنده شماره ترتیب توکن مورد نظر در رشته و i نشان دهنده بعد است. به عنوان مثال، فرض کنید که یک رشته به اندازه 4 داریم و d_model برابر 4 است به این معنی که بردار Embedding و Positional Encoding 4 درایه خواهند داشت و با هم جمع خواهند شد. حال فرض کنید می خواهیم بردار Positional Encoding را به وسیله توابع سینوس و کسینوس برای توکن با pos=1 یا همان توکن اولی به دست آوریم. به این صورت محاسبه می کنیم:

$$PE(1,\ 0) = \sin\left(\frac{1}{10000^{\frac{0}{4}}}\right)$$
$$PE(1,\ 1) = \cos\left(\frac{1}{10000^{\frac{0}{4}}}\right)$$
$$PE(1,\ 2) = \sin\left(\frac{1}{10000^{\frac{2}{4}}}\right)$$
$$PE(1,\ 3) = \cos\left(\frac{1}{10000^{\frac{2}{4}}}\right)$$

اکنون همه ی درایه های مورد نیاز برای بردار Positional Encoding را در اختیار داریم، فقط کافی است که این بردار را به بردار Embedding به صورت درایه به درایه اضافه کنیم تا ورودی برای مدل مان آماده شود.

## مقایسه عملکرد Transformers با سایر معماری ها

ابتدا پیچیدگی ای معماری با سایر را مقایسه می کنیم. به جدول زیر که در مقاله اصلی آمده توجه کنید:

| Layer Type | Complexity per Layer | Sequential Operations | Maximum Path Length |
|---|---|---|---|
| Self-Attention | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| Recurrent | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| Convolutional | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(log_k(n))$ |
| Self-Attention (restricted) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

در این جدول n نشان دهنده طول رشته ورودی به مدل، d نشان دهنده ابعاد مدل، k نشان دهنده اندازه کرنل در Convolution و r نشان دهنده اندازه همسایگی در attention محدود شده است. همانطور که مشخص است برای n های کوچک تر از d، سرعت Self-Attention از بقیه بیش تر است.

اکنون به بررسی عملکرد این معماری در مقایسه با دیگر مدل ها در تسک Machine Translation می پرازیم.

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | **$3.3 \cdot 10^{18}$** | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |

معیار مقایسه BLEU score است که قبلا معرفی شد. همان طور که معلوم است این مدل با هزینه آموزش کم تر، به نتایج بهتری در مقایسه با دیگر مدل ها دست پیدا کرده است.

معماری Transformer در معرفی اول از هر دو بخش Encoder و Decoder تشکیل شده است. ولی مدل ها می توانند تشکیل شده از فقط Encoder یا فقط Decoder باشند:

- Encoder-only

این مدل ها یک رشته ورودی را به یک نمایش عددی تبدیل می کنند تا بعدا از این نمایش عددی بتوانند استفاده کنند. در text classification و NER می توان از آن استفاده کرد. به عنوان مثال این معماری می توان به BERT و انواع آن از جمله RoBERTa و DistilBERT اشاره کرد.

- Decoder-only

با ورودی دادن یک جمله، این مدل ها با پیشبینی محتمل ترین کلمه بعدی جمله را به صورت بازگشتی کامل می کنند. خانواده مدل های GPT به این معماری تعلق دارند.

## مرور Transformers

همانطور که دیدیم معماری Transformers در سال 2017 بر پایه مکانیزم Attention برای تسک Machine Translation معرفی شد و اجزای آن را بررسی کردیم. ورودی که ابتدا تبدیل به بردار های Embedding می شوند، سپس با بردار های Positional Ecoding مربوطه جمع می شوند و سپس وارد بخش encoder یا decoder می شوند. لایه های مختلف داخل هر کدام را نیز از جمله Multi-head Attention یا Feed-forward را بررسی کردیم. سپس به بررسی مدل های زبانی می پردازیم.

# GPT

## تاریخچه

برای اولین بار در سال 2018، محققان در openai مدل GPT یا Generative Pre-trained Transformer را معرفی کردند. مقاله مربوط به GPT-2 هم در سال 2019 در مقاله ای با عنوان "Language Models are Unsupervised Multitask Learners" معرفی شد. همانطور که از اسم آن مشخص است، معماری مورد استفاده در این مقاله ها معماری Transformer است که پیش تر بررسی شد. این مدل های در بخش مدل های PLM یا Pretrained language Models قرار می گیرند و مدلی چون GPT-3 که 175 میلیارد پارامتر دارد، در بخش مدل های LLM یا Large language Models قرار می گیرند. مدل GPT-2 دارای 1.5 میلیارد پارامتر است. سپس به بررسی ایده های مطرح شده در این مقاله ها و عملکرد آن ها می پردازیم.

### ایده

میدانیم که برای ساختن مدلی برای کشف کردن الگوهای پیچیده و یا برای تسک هایی پیچیده مثل یک چت بات که بتواند با انسان صحبت کند، نیاز به مدلی پیچیده با پارامتر های زیاد داریم. حال برای آموزش و استفاده از این مدل بزرگ، با چالش هایی رو به رو خواهیم شد. یکی از چالش ها آموزش این مدل است که نیاز به سخت افزار زیاد و قدرتمند و مدت آموزش زیادی دارد. مشکل دوم این است که با بزرگ تر شدن مدل، ما به دیتای زیادی برای آموزش خواهیم داشت و برای یک تسک با نظارت، این دیتا باید لیبل دار باشد که کار سخت، زمان بر و پر هزینه ای است. برای این منظور

روش مورد پیشنهاد مقاله "Improving Language Understanding by Generative Pre-Training" این گونه است که ابتدا مدل را روی یک دیتا ست بدون لیبل و به صورت unsupervised باید Pre-train کنیم، سپس مدل Pre-train شده و دیتای لیبل دار را استفاده کنیم و روی تسک مورد نظر fine-tuning انجام دهیم. در واقع در این روش قبل از نشان دادن دیتای لیبل دار به مدل، دیتا های بدون لیبلی را به مدل نشان می دهیم تا آشنایی نسبی با دیتا پیدا کند تا وقتی دیتای لیبل دار را به آن نشان می دهیم، بهتر و سریع تر یاد بگیرد. برای درک بهتر این ایده به این مثال توجه کنید:

مثال

فرض کنید دو شخص به سن 18 سال رسیده اند و اکنون می خواهیم به هر دو رانندگی یاد بدهیم. یکی از این افراد تا سن 18 سالگی نابینا بوده و به تازگی شفا پیدا کرده است، ولی یکی از بچگی بینا بوده و دنیای اطراف را در این 18 سال مشاهده کرده است. اکنون فرض کنید که می خواهیم دیتای لیبل دار زیر را به هر دو بدهیم:

$$Q(red\,light,\ pushing\,gas\,pedal) = -5$$

به عبارت در یادگیری تقویتی به این صورت خوانده می شود که گاز دادن پشت چراغ قرمز پاداش منفی 5 خواهد داشت که معنی آن این است که عمل بدی است و منفی 5 نیز دیتای لیبل دار ما است. حال شخصی که در این 18 سال دنیای اطراف را ندیده است، نمی داند گاز چیست، همچنین نمی داند چراغ چیست و حتی از اینکه به کدام رنگ قرمز می گویند اطلاع ندارد، از طرفی شخص دیگری همه این اطلاعات را در طول دوره زندگی کسب کرده است. حال روشن است که شخص دوم راحت تر و سریع تر با استفاده از این دیتای لیبل دار رانندگی را یاد خواهد گرفت.
پس فهمیدیم که آموزش از دو مرحله تشکیل خواهد شد: Unsupervised pre-training و Supervised fine-tuning.

## Unsupervised pre-training

این مقاله از یک تسک language modeling برای پیش آموزش مدل استفاده می کند. در language modeling سعی بر این است که با در اختیار داشتن یک بخش از داده های ترتیبی، بخش مبهم داده را حدس بزنیم. این آموزش به این صورت تعریف می شود که می خواهیم عبارت زیر را ماکزیمم کنیم:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta)$$

که در این عبارت، $\mathcal{U} = \{u_1, \ldots, u_n\}$ توکن های بدون لیبل ما هستند و $k$ اندازه context window است که مشخص می کند با توجه به چند توکن قبلی، توکن فعلی را پیش بینی می کنیم و $\Theta$ پارامتر های شبکه عصبی ما هستند که قرار است با الگوریتم stochastic gradient descent آموزش ببینند.

به عنوان مثال فرض کنید که این جمله درون دیتاست بدون لیبل ما قرار دارد:

- The cat sat on the mat

هم چنین فرض کنید که $k$ را 5 می گیریم، به این معنی که با در اختیار داشتن 5 توکن، توکن ششم را پیشبینی کنیم. برای آموزش مدل روی این یک جمله باید این احتمال را با آپدیت کردن $\Theta$ های شبکه عصبی، ماکزیمم کنیم:

$$\log P(mat \mid the\ cat\ sat\ on\ the)$$

بعد از ماکزیمم کردن این احتمال توسط شبکه عصبی، مدل یاد خواهد گرفت که بعد از دیدن آن 5 توکن ابتدایی، احتمال آمدن توکن mat، زیاد است. با تکرار این فرایند روی یک دیتاست خیلی بزرگ، مدل الگو ها و ساختار های به کار رفته در زبان را بدون دیدن دیتای لیبل دار، یاد خواهد گرفت.

همان طور که قبل تر اشاره شد، این مدل از معماری Transformer decoder استفاده می کند. هم چنین مکانیزم attention به کار رفته، multi-headed self attention است که قبلا در بخش معماری Transformer معرفی شد. در کل معماری Transformer مورد استفاده در این مرحله را می توان به این صورت خلاصه کرد:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned}$$

## Supervised fine-tuning

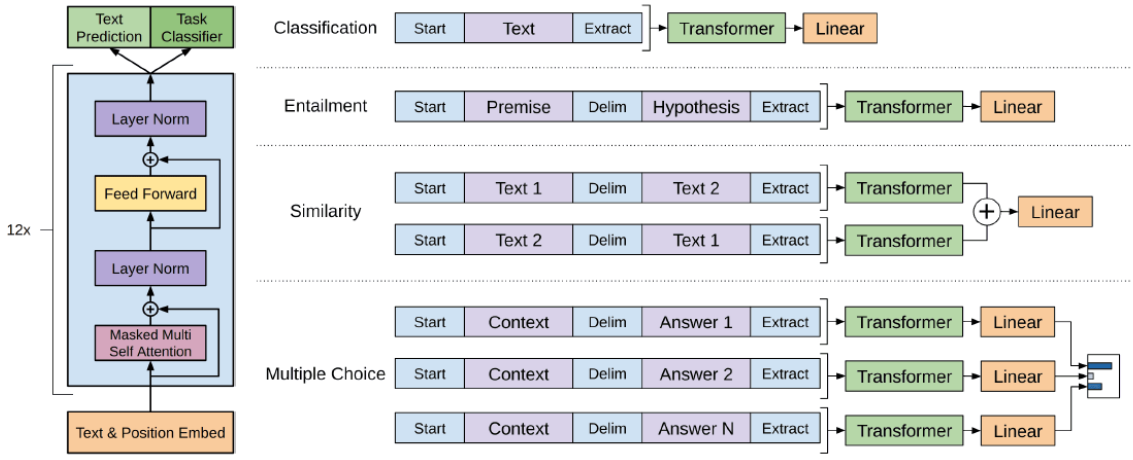
در این مرحله، از پارامتر های یاد گرفته شده در مرحله قبل استفاده کرده و مدل را روی یک تسک و دیتا ست لیبل دار خاص fine tune می کنیم. در این مرحله این عبارت را خواهیم داشت:

$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y).$$

$x^1$ تا $x^m$ همان رشته ورودی از توکن ها و $y$ لیبل مربوط به این رشته ورودی است. $h_l^m$ خروجی مدل pre-trained شده با دادن این دشته ورودی است و $W_y$ یک لایه خطی است که در آخر برای پیش بینی $y$ اضافه می شود. پس در نهایت هدف ما ماکزیمم کردن این عبارت خواهد بود:

$$\sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

برای استفاده از این مدل پیش آموزش دیده روی تسک های با لیبل، باید ورودی ها را برای بعضی از تسک ها تغییر دهیم. برای بعضی از تسک ها مثل text classification ما تغییر خاصی را روی رشته ورودی نیاز نداریم ولی برای تسک هایی مثل question answering یا textual entailment نیازمند تغییراتی هستند. شکل زیر معماری Transformer به کار رفته و تغییرات لازم روی تسک هارا نشان می دهد.




## تسک ها و دیتاست های به کار رفته

تسک ها و دیتاست های به کار رفته برای مرحله fine tuning به این صورت آمده است:

| Task | Datasets |
|---|---|
| Natural language inference | SNLI [5], MultiNLI [66], Question NLI [64], RTE [4], SciTail [25] |
| Question Answering | RACE [30], Story Cloze [40] |
| Sentence similarity | MSR Paraphrase Corpus [14], Quora Question Pairs [9], STS Benchmark [6] |
| Classification | Stanford Sentiment Treebank-2 [54], CoLA [65] |

## بررسی نتایج مدل پیشنهاد شده

در این بخش عملکرد روش پیشنهاد شده را روی تسک های language inference و question answering و commonsense reasoning و Semantic similarity و classification بررسی می کنیم.

## natural language inference

| Method | MNLI-m | MNLI-mm | SNLI | SciTail | QNLI | RTE |
|---|---|---|---|---|---|---|
| ESIM + ELMo [44] (5x) | - | - | 89.3 | - | - | - |
| CAFE [58] (5x) | 80.2 | 79.0 | 89.3 | - | - | - |
| Stochastic Answer Network [35] (3x) | 80.6 | 80.1 | - | - | - | - |
| CAFE [58] | 78.7 | 77.9 | 88.5 | 83.3 | | |
| GenSen [64] | 71.4 | 71.3 | - | - | 82.3 | 59.2 |
| Multi-task BiLSTM + Attn [64] | 72.2 | 72.1 | - | - | 82.1 | **61.7** |
| Finetuned Transformer LM (ours) | **82.1** | **81.4** | **89.9** | **88.3** | **88.1** | 56.0 |

همان طور که مشاهده می شود روش ارائه شده، روی همه ی دیتاست ها به جز یکی بهتر از بقیه ی مدل هایی که در زمان ارائه مقاله بودند، عمل کرده است.

## question answering and commonsense reasoning

| Method | Story Cloze | RACE-m | RACE-h | RACE |
|---|---|---|---|---|
| val-LS-skip [55] | 76.5 | - | - | - |
| Hidden Coherence Model [7] | 77.6 | - | - | - |
| Dynamic Fusion Net [67] (9x) | - | 55.6 | 49.4 | 51.2 |
| BiAttention MRU [59] (9x) | - | 60.2 | 50.3 | 53.3 |
| Finetuned Transformer LM (ours) | **86.5** | **62.9** | **57.4** | **59.0** |

روی همه ی دیتاست ها بهتر عمل کرده است.

## Semantic similarity and classification

| **Method** | **Classification** | | **Semantic Similarity** | | | **GLUE** |
|---|---|---|---|---|---|---|
| | CoLA (mc) | SST2 (acc) | MRPC (F1) | STSB (pc) | QQP (F1) | |
| Sparse byte mLSTM [16] | - | **93.2** | - | - | - | - |
| TF-KLD [23] | - | - | **86.0** | - | - | - |
| ECNU (mixed ensemble) [60] | - | - | - | 81.0 | - | - |
| Single-task BiLSTM + ELMo + Attn [64] | 35.0 | 90.2 | 80.2 | 55.5 | 66.1 | 64.8 |
| Multi-task BiLSTM + ELMo + Attn [64] | 18.9 | 91.6 | 83.5 | 72.8 | 63.3 | 68.9 |
| Finetuned Transformer LM (ours) | **45.4** | 91.3 | 82.3 | **82.0** | **70.3** | **72.8** |

### نتیجه

می توان نتیجه گرفت که با استفاده از این روش پیش آموزش و سپس fine tune کردن مدل روی دیتاست مورد نظر و همچنین با استفاده از معماری Transformers، به نتایج بهتری نسبت به مدل ها و روش های قبلی دست پیدا کرد. البته این روش در سال 2018 معرفی شد و مسلما اکنون مدل های بزرگ تر و بهتری موجور هستند.

# GPT-2

### مقدمه

مقاله ای که GPT-2 در آن معرفی شد "Language Models are Unsupervised Multitask Learners" نام دارد که در سال 2019 منتشر شد. معماری به کار رفته در این مقاله معماری Transformers است و مدل به کار رفته شبیه به مدل به کار رفته در مقاله قبلی با کمی تغییر است. تمرکز اصلی این مقاله روی Pre-train کردن مدل با تسک language modeling است که قبلا توضیح داده شد. همچنین تلاش این مقاله استفاده از تنظیمات zero-shot است، به این معنی که بعد از Pre-train کردن مدل روی یک دیتاست بزرگ، بدون اینکه مدل رو روی تسک و دیتاست خاصی fine tune کنیم، از آن در آن تسک استفاده کنیم. با توجه به این مطلب متوجه فرق این روش با روش پیشنهاد شده در مقاله قبلی مربوط به GPT می شویم: روش پیشنهاد شده در مقاله قبلی شامل دو مرحله Unsupervised pre-training و Supervised fine-tuning بود ولی در این مقاله فقط آموزش روی یک دیتاست به اسم WebText که دیتاست بزرگی است، انجام می شود و تمرکز روی zero-shot بودن مدل است. پس از Pre-train کردن مدل، آن را روی دیتاست های تسک های مختلفی تست کرده و گزارش هر کدام آمده است. همچنین همان طور که اشاره شد معماری مدل شبیه به مقاله قبلی با تغییرات جزیی از جمله اضافه شدن یک لایه layer normalization بعد از آخرین بلوک attention است. همچنین برای نمایش داده ورودی، روش Byte Pair Encoding پیشنهاد شده است.

### اندازه مدل های معرفی شده

در زیر اندازه و بعد و تعداد لایه مربوط به مدل های پیشنهاد شده آورده شده است که بزرگ ترین مدل تقریبا 1.5 میلیارد پارامتر دارد.

| Parameters | Layers | $d_{model}$ |
|---|---|---|
| 117M | 12 | 768 |
| 345M | 24 | 1024 |
| 762M | 36 | 1280 |
| 1542M | 48 | 1600 |

## Byte Pair Encoding

این مقاله برای نمایش داده های ورودی از Byte Pair Encoding استفاده می کند که در سال 2015 در مقاله "Neural machine translation of rare words with subword units" معرفی شده است. این الگوریتم به این صورت عمل می کند:

1. مقداردهی اولیه: ابتدا دیتاست مورد نظر را جمع آوری کرده و آماده می کنیم. فرض کنید که دیتاست جمع آوری شده به این صورت شد:
   - apple, banana, apply, app, bee, band
2. ساختن واژگان یا Vocabulary اولیه: در این مرحله با توجه به دیتاست، Vocabulary اولیه را می سازیم. مثلا برای دیتاستی که آورده شد، Vocabulary به این صورت خواهد شد:

$$Vocabulary = [apple, banana, apply, app, bee, band]$$

3. شمارش تکرار: در این مرحله تعداد تکرار ها را شمارش می کنیم.
4. ادغام زیر کلمه های پر تکرار: زیر کلمه هایی که بیش ترین تکرار را در Vocabulary دارند را ادغام می کنیم. این مرحله را تا زمانی ادامه می دهیم را Vocabulary به اندازه دلخواه برسد.
5. پس از تکرار مرحله 4 به اندازه مورد نظر، Vocabulary نهایی به دست می آید. مثلا بعد از تعدادی تکرار، این Vocabulary برای مثال بالا به دست خواهد آمد:

$$Final\,Vocabulary = [ap, ple, banana, apply, pl, e, bee, band]$$

استفاده از BPE مزایایی دارد که مطالعه آن به خواننده واگذار می شود.

سپس به بررسی عملکرد این مدل روی تسک های و دیتاست های مختلف می پردازیم. توجه شود همانطور که قبلا اشاره شد، این مدل روی تسک ها و دیتاست ها fine tune نشده و به صورت zero-shot عمل می کند.

# بررسی عملکرد

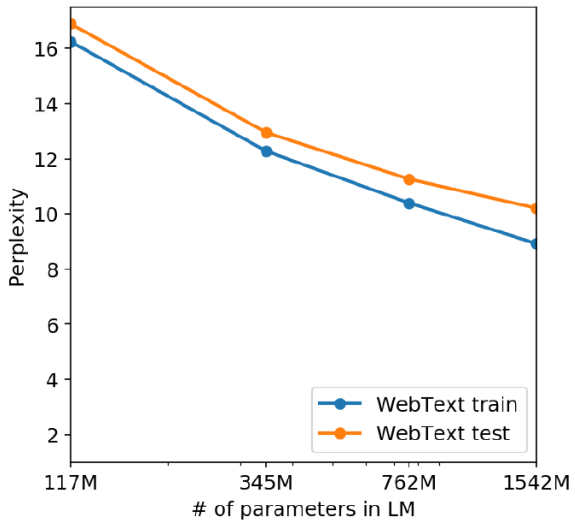
## Language modeling

در این بخش به بررسی عملکرد مدل Pre-train شده روی دیتاست WebText روی دیتاست های دیگر در تسک LM می پردازیم.

| | LAMBADA (PPL) | LAMBADA (ACC) | CBT-CN (ACC) | CBT-NE (ACC) | WikiText2 (PPL) | PTB (PPL) | enwik8 (BPB) | text8 (BPC) | WikiText103 (PPL) | 1BW (PPL) |
|---|---|---|---|---|---|---|---|---|---|---|
| SOTA | 99.8 | 59.23 | 85.7 | 82.3 | 39.14 | 46.54 | 0.99 | 1.08 | 18.3 | **21.8** |
| 117M | **35.13** | 45.99 | **87.65** | **83.4** | **29.41** | 65.85 | 1.16 | 1.17 | 37.50 | 75.20 |
| 345M | **15.60** | 55.48 | **92.35** | **87.1** | **22.76** | 47.33 | 1.01 | **1.06** | 26.37 | 55.72 |
| 762M | **10.87** | **60.12** | **93.45** | **88.0** | **19.93** | **40.31** | **0.97** | **1.02** | 22.05 | 44.575 |
| 1542M | **8.63** | **63.24** | **93.30** | **89.05** | **18.34** | **35.76** | **0.93** | **0.98** | **17.48** | 42.16 |

در جدول بالا SOTA نشان دهنده عملکرد مدل State Of The Art در دیتاست مورد نظر است و 4 ردیف پایین نشان دهنده مدل های مختلف معرفی شده در این مقاله با توجه به تعداد پارامتر ها است. همان طور که مشاهده می شود مدل با 1.5 میلیارد پارامتر روی همه ی دیتاست ها به جز 1BW از مدل SOTA بهتر عمل کرده است.

همینطور در نمودار زیر عملکرد مدل با افزایش پارامتر ها را نمایش می دهد:




همان طور که قبلا اشاره شده است، perplexity کمتر نشان دهنده عملکرد بهتر مدل است. پس مدل با 1.5 میلیارد پارامتر بهترین عملکرد را دارد.

## Summarization

در این بخش به بررسی مدل آموزش دیده در تسک Summarization می پردازیم.

| | R-1 | R-2 | R-L | R-AVG |
|---|---|---|---|---|
| Bottom-Up Sum | **41.22** | **18.68** | **38.34** | **32.75** |
| Lede-3 | 40.38 | 17.66 | 36.62 | 31.55 |
| Seq2Seq + Attn | 31.33 | 11.81 | 28.83 | 23.99 |
| GPT-2 TL;DR: | 29.34 | 8.27 | 26.58 | 21.40 |
| Random-3 | 28.78 | 8.63 | 25.52 | 20.98 |
| GPT-2 no hint | 21.58 | 4.03 | 19.47 | 15.03 |

پس مدل Botton-Up Sum معرفی شده در مقاله "Bottom-up abstractive summarization" در سال 2018 بهترین عملکرد را دارد و GPT-2 عملکرد خوبی در این تسک ندارد. البته توجه شود این مدل فقط Pre-train شده و Fine tune نشده است.

### نتیجه

ایده اصلی معرفی شده این بود که می توان با آموزش یک مدل بزرگ روی دیتای حجیم در تسک LM می توان به نتایج قابل قبولی برای تسک های دیگر دست پیدا کرد.

# BERT

### مقدمه

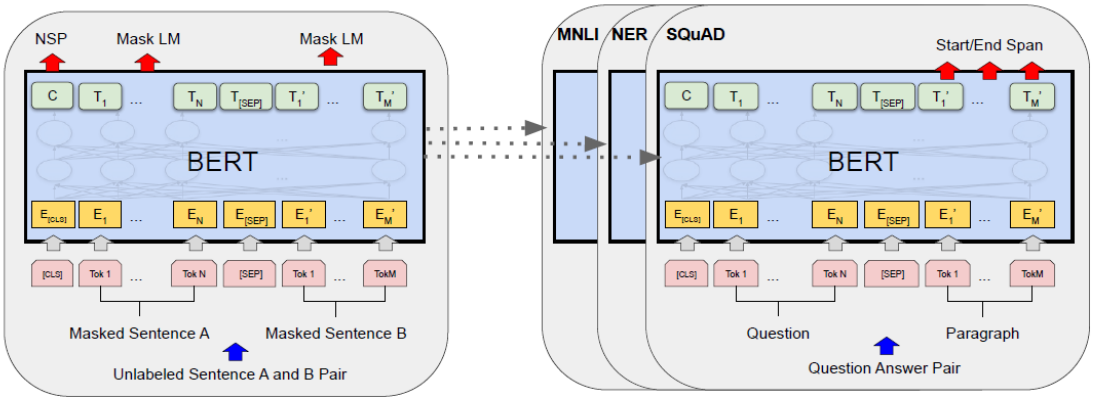
این مدل در مقاله به اسم "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" در سال 2018 معرفی شد. در روشی که برای Pre-train کردن GPT معرفی شد، برای پیشبینی کردن یک توکن در تسک LM فقط به طرف چپ آن توکن توجه می شد (در زبان انگلیسی). ایده مطرح در این مقاله این است که به جای تمرکز فقط روی طرف چپ توکن، باید به هردو طرف چپ و راست توجه کرد. همچنین در این مقاله به جای Pre-train کردن مدل فقط با تسک LM، مدل هم با تسک LM و هم با تسک Next Sentence Prediction مرحله Pre-train انجام می شود. قابل ذکر است که این مدل مانند مقاله اول GPT مرحله fine tuning روی تسک های مورد نظر را دارا می باشد.

برای Pre-train کردن مدل روی تسک LM، روشی به اسم MLM معرفی شده است که توضیح داده خواهد شد. ابتدا معماری مورد استفاده را بررسی می کنیم.

## معماری

معماری مورد استفاده در BERT، یک Transformer چند لایه ی دو طرفه است. در این مقاله دو مدل با تعداد پارامتر های مختلف معرفی شده است. اولی با اسم $BERT_{BASE}$ دارای 12 لایه، 12 تا Attention head و 110 میلیون پارامتر. تعداد پارامتر های این مدل نزدیک به تعداد پارامتر های مدل اول GPT انتخاب شده تا مقایسه شوند. مدل دوم $BERT_{LARGE}$ نام دارد و دارای 24 لایه، 16 Attention head و 340 میلیون پارامتر است.

همان طور که قبلا اشاره شد، BERT نیز شامل دو مرحله Pre-training و Fine-Tuning است. وزن های یاد گرفته شده در مرحله Pre-train به هر کدام از مدل تسک های مختلف کپی می شود. این مطلب در شکل زیر نمایش داده شده:




Pre-training Fine-Tuning

پس وزن ها از مرحله Pre-train برای Fine-Tune کردن روی دیتاست های مورد نظر مانند SQuAD و ... استفاده می شود. در مرحله بعد به چگونگی نمایش ورودی و خروجی خواهیم پرداخت.

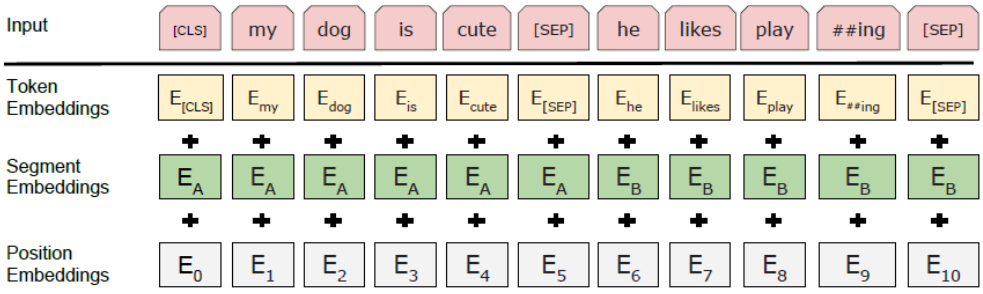
## نمایش ورودی و خروجی

برای این که بتوان از BERT در تسک های مختلف استفاده کرد، ورودی آن طوری طراحی شده تا بتواند هم یک و هم چند جمله را بپذیرد. مثلا فرض کنید می خواهیم روی تسک Question Answering مدل را Fine-Tune کنیم. ورودی را به صورت دو جمله (سوال، جواب) به مدل می دهیم. همچنین از توکن $[SEP]$ برای جدا کردن جمله ها و از توکن $[CLS]$ برای مشخص کردن ابتدای رشته استفاده می شود. برای ساختن نمایش پایانی ورودی، از جمع سه بردار استفاده می شود:

- Token Embeddings: ابتدا با استفاده از این روش، نمایش عددی مربوط به رشته را به دست می آوریم.
- Segment Embeddings: این بخش برای متمایز کردن جمله های مختلفی که در رشته جمع شدند، با کار گرفته می شود

- Position Embeddings: که مرحله هم در که در بخش Transformers به آن پرداخته شد، برای وارد کردن اطلاعات ترتیبی رشته به کار می رود.

خلاصه این مراحل در شکل زیر آورده شده است:




سپس به مرحله Pre-training می پردازیم.

## Pre-training

برای مرحله پیش آموزش BERT، از دو تسک MLM و NSP استفاده می شود. همچنین دیتاست های مورد استفاده در این مرحله، BooksCorpus با 800 میلیون کلمه و English Wikipedia با 2500 میلیون کلمه است.

### MLM

این روش برای Language Modeling با چیزی که برای Pre-train کردن GPT استفاده شد، کمی متفاوت است. در مورد GPT، ما با استفاده از تعدادی توکن، توکن بعدی را پیشبینی می کردیم. ولی در این روش تعدادی از توکن های رشته را به طور تصادفی حذف می کنیم و از مدل می خواهیم با توجه به توکن های اطراف توکن حذف شده، آن را پیشبینی کند. به مثال زیر توجه کنید:

فرض کنید جمله زیر را در اختیار داریم:

$$I want to buy a new car$$

درصدی از توکن ها را حذف کرده و با توکن $[MASK]$ جایگزین می کنیم:

$$I want to [MASK] a new [MASK]$$

سپس از مدل می خواهیم توکن های حذف شده را پیشبینی کند.

درصد توکن های حذف شده برای BERT، مقدار 15 درصد پیشنهاد شده است.

### NSP

بسیاری از تسک ها در NLP از جمله QA نیازمند این هستند که مدل بتواند روابط بین دو جمله را درک کند. برای همین برای Pre-train کردن BERT علاوه بر تسک LM، از تسک NSP نیز استفاده می شود.این تسک به این صورت است: ابتدا یک جمله را از دیتاست انتخاب می کنیم. سپس به احتمال 50 درصد جمله بعد از آن جمله و به احتمال 50 درصد یک جمله تصادفی را انتخاب می کنیم. سپس از مدل می خواهیم پیشبینی کند که آیا جمله دوم، جمله ی بعدی جمله اول است یا خیر و با توجه به مقدار پیشبینی شده، وزن های مدل را به روزرسانی می کنیم. فرمت داده ها برای این تسک به این صورت خواهد بود:

$$[InputSentence] - [NextSentence] - [Label]IsNext$$

پس با ورودی دادن Input Sentence و Next Sentence، از مدل می خواهیم IsNext را True یا False پیشبینی کند.

پس از این مرحله، نوبت به مرحله Fine-Tuning می رسد و عملکرد BERT را روی تسک ها و دیتاست های مختلف بررسی می کنیم.

## بررسی عملکرد

### GLUE Benchmark

ابتدا عملکرد BERT را روی بنچمارک GLUE که شامل تسک ها و دیتاست های مختلفی است، بررسی می کنیم.

| System | MNLI-(m/mm) 392k | QQP 363k | QNLI 108k | SST-2 67k | CoLA 8.5k | STS-B 5.7k | MRPC 3.5k | RTE 2.5k | **Average** - |
|---|---|---|---|---|---|---|---|---|---|
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.8 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| $BERT_{BASE}$ | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $BERT_{LARGE}$ | **86.7/85.9** | **72.1** | **92.7** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **82.1** |

همانطور که مشاهده می شود، مدل بزرگ تر BERT عملکرد بهتری نسبت به GPT-1 و دیگر مدل ها روی این بنچمارک را داراست.

### SQuAD

دیتاست Stanford Question Answering Dataset یک دیتاست برای تسک Question Answering است که شامل سوالات و جواب های مربوطه است. عملکرد BERT را روی دو ورژن متفاوت از این دیتاست اندازه گیری می کنیم. روی ورژن 1.1 به این صورت خواهد بود:

| System | Dev | | Test | |
|---|---|---|---|---|
| | EM | F1 | EM | F1 |
| Top Leaderboard Systems (Dec 10th, 2018) | | | | |
| Human | - | - | 82.3 | 91.2 |
| #1 Ensemble - nlnet | - | - | 86.0 | 91.7 |
| #2 Ensemble - QANet | - | - | 84.5 | 90.5 |
| Published | | | | |
| BiDAF+ELMo (Single) | - | 85.6 | - | 85.8 |
| R.M. Reader (Ensemble) | 81.2 | 87.9 | 82.3 | 88.5 |
| Ours | | | | |
| $\text{BERT}_{\text{BASE}}$ (Single) | 80.8 | 88.5 | - | - |
| $\text{BERT}_{\text{LARGE}}$ (Single) | 84.1 | 90.9 | - | - |
| $\text{BERT}_{\text{LARGE}}$ (Ensemble) | 85.8 | 91.8 | - | - |
| $\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA) | **84.2** | **91.1** | **85.1** | **91.8** |
| $\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA) | **86.2** | **92.2** | **87.4** | **93.2** |

هم چنین برای ورژن 2.0 نتایج به این صورت خواهند بود:

| System | Dev | | Test | |
|---|---|---|---|---|
| | EM | F1 | EM | F1 |
| Top Leaderboard Systems (Dec 10th, 2018) | | | | |
| Human | 86.3 | 89.0 | 86.9 | 89.5 |
| #1 Single - MIR-MRC (F-Net) | - | - | 74.8 | 78.0 |
| #2 Single - nlnet | - | - | 74.2 | 77.1 |
| Published | | | | |
| unet (Ensemble) | - | - | 71.4 | 74.9 |
| SLQA+ (Single) | - | | 71.4 | 74.4 |
| Ours | | | | |
| $\text{BERT}_{\text{LARGE}}$ (Single) | 78.7 | 81.9 | 80.0 | 83.1 |

## نتیجه

ایده اصلی معرفی شده در BERT، دوطرفه بودن آموزش مدل است که اشاره به این دارد که برای پیشبینی یک توکن در رشته باید به هردو طرف توکن توجه کنیم. علاوه بر این در BERT، مرحله Pre-training با دو عدد تسک MLM و NSP انجام شد در حالی که این مرحله در GPT تنها با LM سنتی انجام گرفته بود. در حالت کلی می توان نتیجه گرفت که دوطرفه بودن در آموزش می تواند برای به دست آوردن نتایج بهتر مفید باشد.

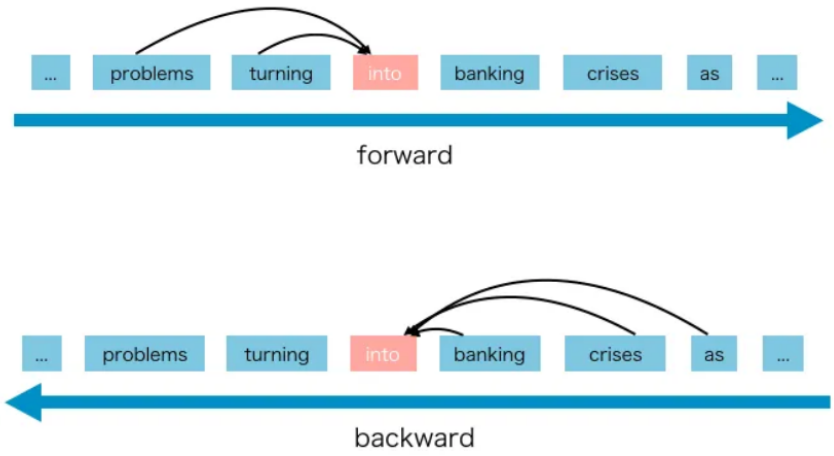
# XLNet

## مقدمه

این مقاله در سال 2019 معرفی شد و سعی دارد تا عیب و نقص های BERT را تصحیح کند.هدف معرفی یک روش Pre-training است که روشی بین متد مورد استفاده در GPT و BERT است. همان طور که دیدیم به دلیل اینکه MLM که در BERT استفاده شد به هردو طرف یک کلمه برای پیشبینی آن کلمه نگاه می کند، در نتیجه این روش نتایج بهتری نسبت به روش Autoregressive که در GPT استفاده شد و فقط به یک طرف کلمه نگاه می کند، دارد. با این حال روش MLM نیز معایب خود را دارد که جلوتر معرفی می شوند. XLNet تلاش می کند تا روشی را بین AR که در GPT استفاده شد و MLM که در BERT استفاده شد به کار گیرد و از مزایای هر کدام استفاده کند و معایب هر دو را برطرف کند.

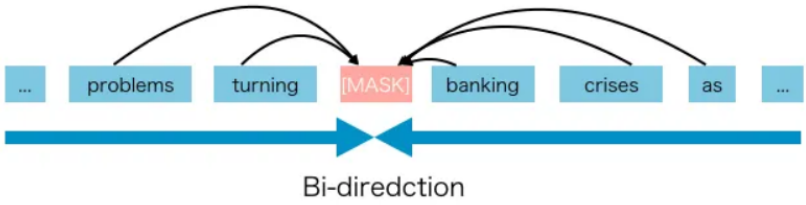
## مزایا و معایب AR و MLM

همان طور که پیش تر دیدیم روش مورد استفاده برای Pre-train کردن GPT، روش AR روی تسک Language modeling بود. در این روش با در اختیار داشتن $t-1$ کلمه قبلی یا بعدی، کلمه مربوط به جایگاه $t$ را پیشبینی می کردیم. این روش در شکل زیر خلاصه شده است:




عیب این روش این است که نمی تواند محتوای چپ و راست را همزمان ببیند و این باعث می شود نتواند مثل یک مدل دوطرفه مانند BERT نتیجه بدهد.

از طرفی دیدیم که روش مورد استفاده در BERT، روش MLM بود که بعضی از کلمات را MASK می کرد و سعی می کرد که کلمه MASK شده را با توجه به محتوای اطراف آن پیشبینی کند. این روش دی شکل زیر خلاصه شده است:




این روش چون که همزمان محتوای هردو طرف کلمه را می بیند، نتایج بهتری نسبت به روش AR دارد ولی معایبی نیز دارد که به این صورت بیان می شود:

- هنگام آموزش این مدل، ما به جای بعضی از کلمات توکن $[MASK]$ را قرار می دهیم، ولی این توکن در زمان fine-tuning وجود ندارد و این باعث کاهش عملکرد آن می شود که به این پدیده، pretrain-finetune discrepancy می گوییم.
- عیب دیگر روش MLM که باعث کاهش عملکرد آن می شود این است که فرض را بر این می گذارد که کلماتی که MASK شدند از یکدیگر مستقل هستند و ارتباطی باهم ندارند ولی این به دور از واقعیت است. برای درک بهتر به مثال زیر توجه کنید:

فرض کنید این جمله را در اختیار داریم:

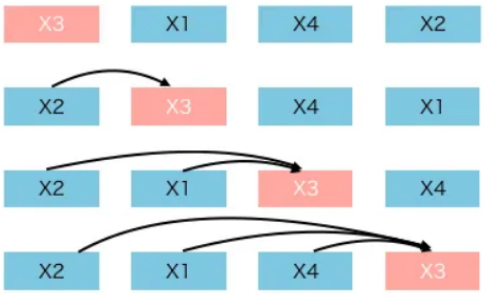
It shows that the housing crisis was turned into a banking crisis

حال کلمات banking و crisis را با MASK جایگزین می کنیم. از مفهوم جمله اصلی میتوان دریافت که این دو کلمه با یکدیگر رابطه دارند ولی وقتی که هردو را MASK می کنیم، مدل تلاش خواهد کرد که هردو کلمه را فقط با توجه به کلماتی که MASK نشده اند پیشبینی کند و این می تواند نتیجه مطلوبی نداشته باشد.

## ایده XLNet

روشی که XLNet استفاده می کند را می توان چیزی مابین AR و MLM دانست. نویسندگان اسم این روش را Permutation Language Modeling گذاشته اند. همانطور که از اسمش پیداست، این روش بر مبنای جایگشت کار می کند. برای شرح این روش از یک مثال کمک می گیریم:

فرض کنید یک رشته به این صورت داریم: $[x_1, x_2, x_3, x_4]$. تعداد جایگشت های این رشته به تعداد $4! = 24$ هستند. حال فرض کنید که می خواهیم توکن $x_3$ را پیشبینی کنیم. در تمام جایگشت های این رشته، $x_3$ یا در خانه اولی است یا دومی یا سومی و یا چهارمی. با توجه به این، $x_3$ را به این صورت پیشبینی می کنیم:




توجه کنید که پیشبینی $x_3$ به صورت Autoregressive است ولی از آنجایی که از جایگشت استفاده می کنیم، برای پیشبینی $x_3$ از محتوای هردو طرف آن کمک میگیریم. مثلا در شکل بالا و در ردیف آخر، برای پیشبینی $x_3$ از $x_4$ نیز کمک گرفتیم در حالی که در رشته اصلی $x_4$ بعد از $x_3$ آمده است.

همچنین XLNet از Two-Stream Self-Attention استفاده می کند.

### نتیجه

مقاله XLNet روشی جدید را معرفی می کند و سعی بر این دارد تا مزایای روش های Autoregressive و MLM را ترکیب کرده و روشی بهتر نسبت به هردوی آن ها ارائه کند.

# مقایسه XLNet و BERT

## IMDB review dataset

در بخش های قبلی، روش های مورد استفاده در BERT و XLNet را مورد بررسی قرار دادیم و دریافتیم که XLNet سعی دارد تا با ترکیب روش های مورد استفاده در GPT و BERT، نتایج بهتری را به دست آورد. در این بخش مدل های Pre-train شده مدل های BERT و XLNet را روی دیتاست IMDB review که یک دیتاست Classification است fine-tune می کنیم و نتایج را مقایسه می کنیم. همچنین برای استفاده از این مدل ها از کتابخانه transformers استفاده شده تا کار راحت تر انجام شود.

ابتدا کمی به بررسی دیتاست می پردازیم. ابتدا به نمونه ای از یک مثال مربوط به یک نظر منفی از دیتاست توجه کنید:

```
{'text': 'It was great to see some of my favorite stars of 30 years ago including John Ritter, Ben Gazarra and Audrey
Hepburn. They looked quite wonderful. But that was it. They were not given any characters or good lines to work with.
I neither understood or cared what the characters were doing.<br /><br />Some of the smaller female roles were fine,
Patty Henson and Colleen Camp were quite competent and confident in their small sidekick parts. They showed some
talent and it is sad they didn\'t go on to star in more and better films. Sadly, I didn\'t think Dorothy Stratten got
a chance to act in this her only important film role.<br /><br />The film appears to have some fans, and I was very
open-minded when I started watching it. I am a big Peter Bogdanovich fan and I enjoyed his last movie, "Cat\'s Meow"
and all his early ones from "Targets" to "Nickleodeon". So, it really surprised me that I was barely able to keep
awake watching this one.<br /><br />It is ironic that this movie is about a detective agency where the detectives and
clients get romantically involved with each other. Five years later, Bogdanovich\'s ex-girlfriend, Cybil Shepherd had
a hit television series called "Moonlighting" stealing the story idea from Bogdanovich. Of course, there was a great
difference in that the series relied on tons of witty dialogue, while this tries to make do with slapstick and a few
screwball lines.<br /><br />Bottom line: It ain\'t no "Paper Moon" and only a very pale version of "What\'s Up,
Doc".',
 'label': 0}
```

با توجه به نوشته های مربوط به این نظر متوجه می شویم که این یک نظر منفی درباره یک فیلم است، label که مساوی صفر است نیز این را تایید می کند. حال به یک نظر مثبت توجه کنید:

```
{'text': 'Budget limitations, time restrictions, shooting a script and then cutting it, cutting it, cutting it...
This crew is a group of good, young filmmakers; thoughtful in this script - yes, allegorical - clever in zero-dollar
effects when time and knowledge is all you have, relying on actors and friends and kind others for their time,
devotion, locations; and getting a first feature in the can, a 1-in-1000 thing. These guys make films. Good ones.
Check out their shorts collection "Heartland Horrors" and see the development. And I can vouch, working with them is
about the most fun thing you\'ll do in the business. I\'m stymied by harsh, insulting criticism for this film,
wondering if one reviewer even heard one word of dialogue, pondered one thought or concept, or if all that was
desired of this work was the visual gore of bashing and slashing to satisfy some mindless view of what horror should
mean to an audience. Let "The Empty Acre" bring itself to you. Don\'t preconceive what you expect it should be just
because it gets put in the horror/thriller genre due to its supernatural premise. It\'s a drama with depth beyond how
far you can stick a blade into someone with a reverence for a message that doesn\'t assault your brain\'s visual
center, but rather, draws upon one\'s empathetic imagination to experience other\'s suffering of mind and spirit.
mark ridgway, Curtis, "The Empty Acre"',
 'label': 1}
```

که label برابر یک است.

حال هر کدام از مدل ها را بارگذاری کرده و آموزش را انجام می دهیم. از آوردن کد آموزش خودداری می کنیم ولی این کد ها از طریق لینکی که در آخر آورده می شود قابل دسترس هستند. Fine-tuning به اندازه 4 epoch انجام می شود و نتایج مربوط به مدل BERT به این طرح هستند:

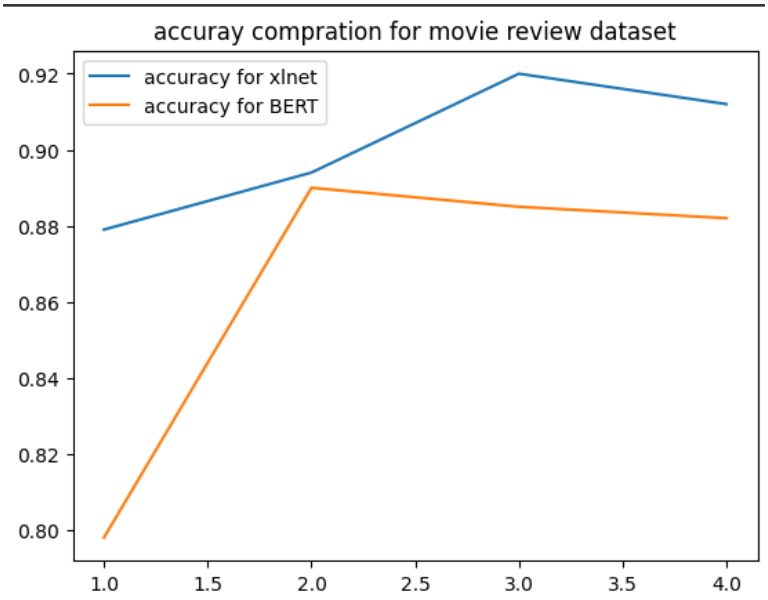
| Validation Loss | Accuracy |
|---|---|
| 0.575557 | 0.798000 |
| 0.411905 | 0.890000 |
| 0.575389 | 0.885000 |
| 0.617563 | 0.882000 |

سپس همین کار ها را برای XLNet تکرار می کنیم، نتایج به دست آمده به این شرح هستند:

| Validation Loss | Accuracy |
|---|---|
| 0.502717 | 0.879000 |
| 0.436439 | 0.894000 |
| 0.370760 | 0.920000 |
| 0.462966 | 0.912000 |

همان طور که می بینیم هر دو تا یک جایی دقتشان بهتر شده و از یک جا به بعد تر کمی بدتر شده و تابع loss مربوط به validation هم کمی بد تر شده است. این نشان دهنده overfit کردن مدل است و می توان برای رفع این از تکنیک های مختلفی از جمله regularization یا افزایش داده های دیتاست استفاده کرد ولی در اینجا این این تکنیک ها نمی پردازیم. از روی همین اعداد متوجه می شویم که عملکرد XLNet بهتر است.

برای مقایسه ی بهتر، دقت هر دو را در یک نمودار رسم می کنیم:




این نمودار هم فرض ما را تایید می کند و XLNet در کل عملکرد بهتری دارد.

## Amazon Review Dataset

این یک دیتاست مربوط به امتیازاتی است که کاربران به محصولات مختلف در سایت آمازون داده اند و یادداشت مربوط به این امتیاز است. امتیاز یک عدد از 1 تا 5 است و امتیاز 5 نشان دهنده رضایت مشتری از محصول است. این دیتاست شامل زبان های مختلفی است که ما به خاطر سادگی فقط روی زبان انگلیسی fine-tuning را انجام می دهیم.

به یک نمونه از دیتای این دیتاست توجه کنید:

```
{'review_id': 'en_0964290',
 'product_id': 'product_en_0740675',
 'reviewer_id': 'reviewer_en_0342986',
 'stars': 1,
 'review_body': "Arrived broken. Manufacturer defect. Two of the legs of the base were not completely formed, so
there was no way to insert the casters. I unpackaged the entire chair and hardware before noticing this. So, I'll
spend twice the amount of time boxing up the whole useless thing and send it back with a 1-star review of part of a
chair I never got to sit in. I will go so far as to include a picture of what their injection molding and quality
assurance process missed though. I will be hesitant to buy again. It makes me wonder if there aren't missing
structures and supports that don't impede the assembly process.",
 'review_title': "I'll spend twice the amount of time boxing up the whole useless thing and send it back with a 1-
star review ...",
 'language': 'en',
 'product_category': 'furniture'}
```

این دیتاست بر خلاف دیتاست قبلی که فقط text و label داشت، feature های دیگری نیز دارد که می توان از آن ها نیز استفاده کرد مانند review_title ولی برای سادگی کار فقط از review_body به عنوان ورودی و stars به عنوان label استفاده می کنیم.  با توجه به مثال بالا می بینیم که مشتری در review_body از محصول گلایه می کند و این امر از یک ستاره ای که به محصول داده مشخص است.

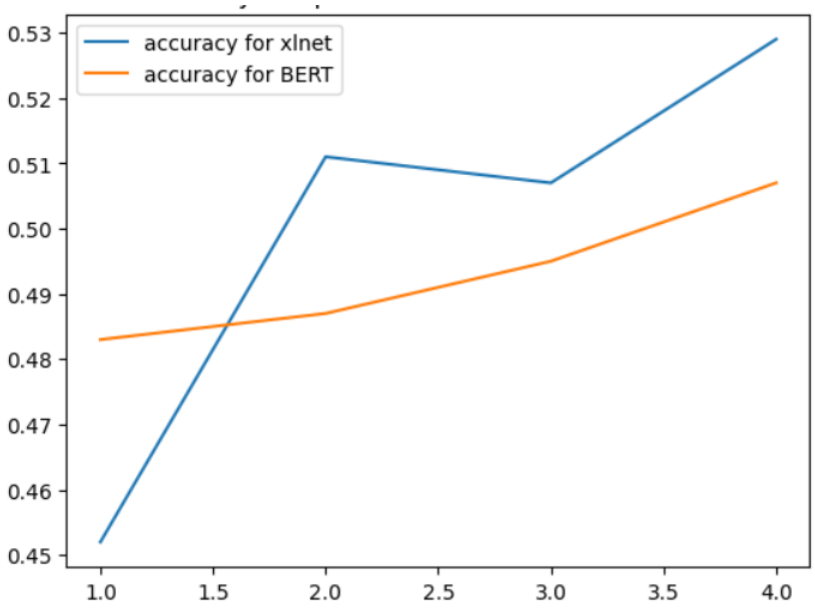
بعد از مرحله Preprocessing، مدل هارا fine-tune می کنیم، نتایج برای مدل BERT به این شرح هستند:

| Validation Loss | Accuracy |
|---|---|
| 1.201904 | 0.483000 |
| 1.201369 | 0.487000 |
| 1.363648 | 0.495000 |
| 1.466130 | 0.507000 |

همچنین برای XLNet نتایج به این شرح خواهند بود:

| Validation Loss | Accuracy |
|---|---|
| 1.292777 | 0.452000 |
| 1.146919 | 0.511000 |
| 1.439580 | 0.507000 |
| 1.643538 | 0.529000 |

برای مقایسه بهتر، هردو را در یک نمودار رسم می کنیم:




همان طور که از نمودار معلوم است، BERT شروع بهتری نسبت به XLNet دارد ولی بعد از 4 ایپاک XLNet دقت بهتری نسبت به BERT دارد. البته که دقت 0.53 قابل قبول نیست و می توان به تعداد ایپاک های بیشتری آموزش را انجام داد تا نتایج بهتری به دست آید.

## Reuters-21578

این مورد هم مربوط به کلاس بندی مربوط به اخبار است. برای مثال اخبار مربوط به غلات با برچسب 3 مشخص می شود. به یک نمونه از این دیتاست توجه کنید:

```
{'text': 'computer language research in clri th qtr shr loss cts vs loss cts net loss vs loss revs mln vs mln qtly
div three cts vs three cts prior year shr profit two cts vs profit cts net profit vs profit revs mln vs mln note
dividend payable april one to shareholders of record march reuter',
 'label': 2}
```

هر نمونه از یک بخش text و یک بخش label که مربوط به برچسب آن است مشخص می شود. این دیتاست نسبت به دیتاست های قبلی نمونه های کمتری دارد.

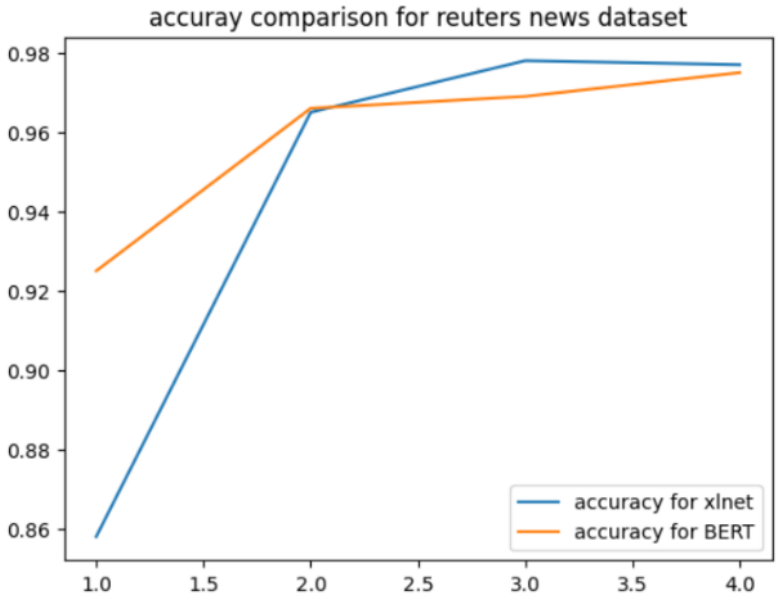
نتایج مربوط به مدل BERT به این شرح خواهند بود:

| Validation Loss | Accuracy |
|---|---|
| 0.279036 | 0.925000 |
| 0.135686 | 0.966000 |
| 0.131494 | 0.969000 |
| 0.114013 | 0.975000 |

نتایج برای XLNet نیز بدین شرح خواهند شد:

| Validation Loss | Accuracy |
|---|---|
| 0.356312 | 0.858000 |
| 0.140455 | 0.965000 |
| 0.101059 | 0.978000 |
| 0.113867 | 0.977000 |

برای مقایسه بهتر، به این نمودار توجه کنید:




مدل BERT شروع بهتری داشته و در پایان نیز تقریبا عملکرد مشابهی دارند.

### نتیجه

مقایسه های صورت گرفته، ادعای XLNet مبنی بر بهتر بودن نسبت به BERT را تایید می کنند. پس روش به کار گرفته شده در XLNet باعث پیشی گرفتن این مدل از BERT که از روش پیش آموزش MLM استفاده می کند، می شود.

# Large Language Models

### مقدمه

در این بخش می خواهیم به طور مختصر به بررسی LLM ها و تفاوت آن ها با Pre-trained Language Model یا PLM ها بپردازیم. محققان متوجه شده اند که با افزایش اندازه مدل و دیتای PLM ها، علمکرد آن ها به طور قابل توجهی بهتر می شود. در این مدل های بزرگ که به آن ها LLM گفته می شود، قابلیت های جدیدی به اسم توانایی ها یا قابلیت های نوظهور یا همان emerging capabilities پدیدار می شوند که باعث متمایز شدن این مدل ها از PLM ها می شوند. در واقع با گذشتن تعداد پارامتر های مدل از یک آستانه ای، این قابلیت ها در مدل ها پدیدار می شوند. برخی از مثال ها برای LLM ها عبارت اند از GPT-3 با 175B پارامتر و PaLM با 540B پارامتر اشاره کرد. بزرگ ترین مدلی که در این مقاله بررسی شد GPT-2 بود که 1.5B پارامتر داشت، پس می بینیم که اندازه LLM ها چقدر با PLM ها فاصله دارد. اکنون به بررسی برخی از قابلیت های نوظهور در LLM ها می پردازیم.

## Emerging Capabilities

همان طور که اشاره شد، این قابلیت ها با افزایش تعداد پارامتر ها ظهور پیدا می کنند. برخی از این قابلیت ها عبارت اند از:

- instruction following: این توانایی به این صورت است که LLM ها می توانند تسک هایی که از قبل ندیده اند و به صورت دستورالعمل فراهم شده است را پردازش و اجرا کنند.
- Step-by-step reasoning: مدل های کوچک تر در حل مسئله هایی که نیاز به استدلال های پیچیده دارند، ناتوان هستند. این امر در LLM ها بهتر می شوند و این مدل های بزرگ می توانند مسئله هایی که نیاز به استدلال های پیچیده دارند را حل کنند.

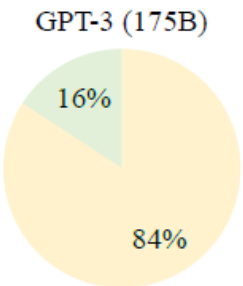
به عنوان مبحث آخر، به دیتاست های به کار رفته برای آموزش مدل های LLM می پردازیم.

### دیتاست ها

عمده ی دیتاست های به کار رفته در LLM ها و اندازه آن ها به این صورت است:

| Corpora | Size | Source | Latest Update Time |
|---|---|---|---|
| BookCorpus [122] | 5GB | Books | Dec-2015 |
| Gutenberg [123] | - | Books | Dec-2021 |
| C4 [73] | 800GB | CommonCrawl | Apr-2019 |
| CC-Stories-R [124] | 31GB | CommonCrawl | Sep-2019 |
| CC-NEWS [27] | 78GB | CommonCrawl | Feb-2019 |
| REALNEWs [125] | 120GB | CommonCrawl | Apr-2019 |
| OpenWebText [126] | 38GB | Reddit links | Mar-2023 |
| Pushift.io [127] | 2TB | Reddit links | Mar-2023 |
| Wikipedia [128] | 21GB | Wikipedia | Mar-2023 |
| BigQuery [129] | - | Codes | Mar-2023 |
| the Pile [130] | 800GB | Other | Dec-2020 |
| ROOTS [131] | 1.6TB | Other | Jun-2022 |

که البته برای آموزش مدل ها میتوان از ترکیبی از دیتاست ها استفاده کرد. برای مثال برای آموزش GPT-3 دیتای آموزش از 16 درصد کتاب ها و اخبار و 84 درصد صفحات وب استفاده شده است.



# نتیجه

دراین مقاله ابتدا مروری روی NLP و الگوریتم ها و تسک ها و معیار های به کار رفته در آن پرداختیم. سپس معماری Transformers که معماری به کار رفته در اکثر PLM های امروزی است را بررسی کردیم و نحوه کار آن را شرح دادیم. سپس به بررسی مقاله های GPT و GPT-2 پرداختیم که روشی را برای Pre-train کردن معرفی کردند. همچنین در این مقاله ها از معماری Transformers که پیش تر معرفی شد، استفاده شده بود. سپس BERT را بررسی کردیم که سعی می کرد با استفاده از MLM یا همان Masked Language Modeling، عملکرد بهتری نسبت به GPT ارائه دهد. بعد از آن XLNet را بررسی کردیم که سعی داشت با ترکیب روش های به کار رفته در GPT و BERT و استفاده از مزایای هردو، نتایج بهتری را به دست آورد. سپس مدل های از پیش آموزش داده شده BERT و XLNet را روی 3 دیتاست Classification با تعداد کلاس های مختلف، fine-tune کردیم و نتایج را با یکدیگر مقایسه کردیم. در پایان نیز نگاهی کلی به مدل های با پارامتر های زیاد یا همان LLM ها کردیم و برخی از دیتاست های به کار رفته در آن ها را نام بردیم.

کد های مقایسه های صورت گرفته از لینک زیر قابل دسترسی هستند:

https://github.com/aminghani/LM

# منابع


- Attention Is All You Need
- Improving Language Understanding by Generative Pre-Training
- Language Models are Unsupervised Multitask Learners
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- XLNet: Generalized Autoregressive Pretraining for Language Understanding
- A Survey of Large Language Models
- https://stanford.edu
- https://towardsdatascience.com