Parameter Efficient Fine Tuning with LORA
Attention is all you need

استاد : دکتر مهدی افتخاری*

مباحث ویژه در هوش مصنوعی (یادگیری عمیق)

سپهر بزمی

بخش مهندسی کامپیوتر، دانشگاه شهید باهنر کرمان، کرمان، ایران

Introduction

در این پروژه قصد داریم مدل از قصد آموزش داده شده GPT2 را بر روی دیتاست SQuaD که مجموعه ای از پرسش و پاسخ هاست Fine-Tune کنیم. ولی به جای Fine-Tune کردن تمام پارامتر ها، از low rank adaptation استفاده میکنیم و تعداد پارامتر های قابل آموزش را به طور قابل توجهی کاهش میدهیم.

Attention is all you need

هدف اصلی پشت مکانیسم توجه این است که به جای ساخت word embed های مستقل از معنی و متن، contextualized word embedding انجام دهیم، چون کلمات در context های متفاوت معانی متفاوتی خواهند داشت، و برای درک و generate کردن متون این موضوع از اهمیت بسیار زیادی برخوردار است. پس این کلمات مشابه با معانی متفاوت نیاز دارند که در embedding space موقعیت های متفاوتی به خود اختصاص دهند. (در RNN ها حتی با اضافه کردن حافظه نمیتوانستیم این مشکل رو کاملا حل کنیم.)

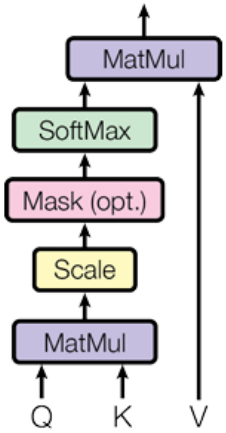
ایده اصلی در این مقاله این است که به مدل اجازه دهیم خودش یاد بگیرد که چه قسمت هایی از input برای ساخت output با معنی مهم تر هستند.

در مکانیسم توجه، query که کلمه فعلی (مورد بررسی) میباشد، به تمام key های دیتابیس (سایر کلمات) به صورت soft، match میشود. با وزنی بین 0 و 1 (از softmax استفاده میکنیم). انگار یک معدل وزن دار از value ها میگیرم، عملا بردار هایی که بین query و سایر کلمات ایجاد میشه همون میزان توجه کلمه به سایر کلمات متن است.

مشکلی که مدل های قدیمی داشتند این بود که encoder همه input هارا به یک vector با سایز ثابت تبدیل میکند و output را بر حسب همان vector محاسبه میکند. مکانیسم توجه به نوعی به خروجی اجازه میده که قبلی هارو هم چک کند.

• Scaled dot product Attention

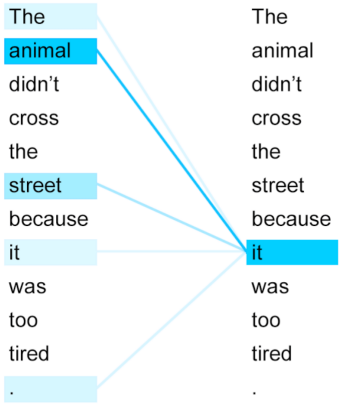
Scaled Dot-Product Attention



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

برای محاسبه self attention ابتدا dot product بین query و key را محاسبه میکنیم سپس آن را تقسیم بر جذر ابعاد فضای embedding میکنیم. (d) به این عمل scaling میگوییم که از ملزومات است زیرا هرچه فضا بزرگ تر میشود، dot product کلید و کوعری بزرگ تر میشود و باید به نحوی اندازه آن را کنترل کنیم، که با این عملیات scaling میتوانیم این کار را به خوبی انجام دهیم.

در self attention مفهموهی به نام global connectivity داریم، به این معنا که هر کلمه به همه کلمات دیگر متصل است، و نزدیک و دوری کلمات به هم در جمله بی تاثیر است، محاسبه میزان توجه برای هر کلمه نسبت به کلمات دیگر نیز کاملا به صورت موازی انجام میشود.



$$s_{ij} = similarity(l_{i-1}, h_j)$$

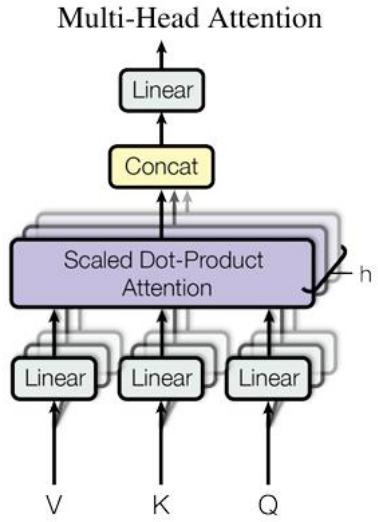
$$a_{ij} = \frac{e^{s_{ij}}}{\sum_{k=1} e^{s_{ik}}}$$

$$c_i = \sum_{j=1} a_{ij} h_j$$

• Multi-Head Attention

نویسندگان این مقاله باور دارند که اگر به جای یک attention head از چندین تا استفاده کنند، خروجی مفید تری خواهند داشت، هر head میتونه از جنبه های متفاوتی به کلمات توجه کنه، محاسبه attention، h بار با linear projection های یاد گرفته شده متفاوت به صورت موازی انجام شده و در انتها همه در کنار هم قرار میگیرند.

توجه از جنبه های متفاوت را به طور دقیق تر میتوان اینگونه بیان کرد که مدل میتواند به صورت همزمان به اطلاعات representation subspaces های متفاوت در کنار هم توجه کند.



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

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

• Transformer

پس از آشنایی با مفهوم attention میتوانیم به تحلیل ساختار transformer ها بپردازیم. این مدل ها دو قسمت اصلی Decoder و Encoder دارند. ورودی قسمت Encoder یک sequence است که پس از اعمال positional encoding که جلو تر مورد بررسی قرار میدهیم با استفاده از مکانیسم attention به context vector تبدیل میشود.

قسمت Encoder دو بخش اصلی دارد : (6 عدد)

- Self attention + addnorm
- Feed forward + addnorm

قسمت decoder سه بخش اصلی دارد : (6 عدد)

- Masked multi-head attention + addnorm
- Decoder Encoder cross attention + addnorm
- Feed forward + addnorm

که پس از این سه بخش یک linear projection وجود دارد که برای تبدیل ابعاد خروجی های FFN به ابعاد منطبق با vocabulary است.

Decoder حالت auto regressive دارد یعنی برای تولید کلمه i ام از کلمات ...i-1,i-2, استفاده میکند، mask کردنی که در این بخش هنگام محاسبه توجه اتفاق میفتد باعث میشود که از توجه به token های جلوتر جلوگیری شود و خاصیت auto regressive بودن حفظ شود.

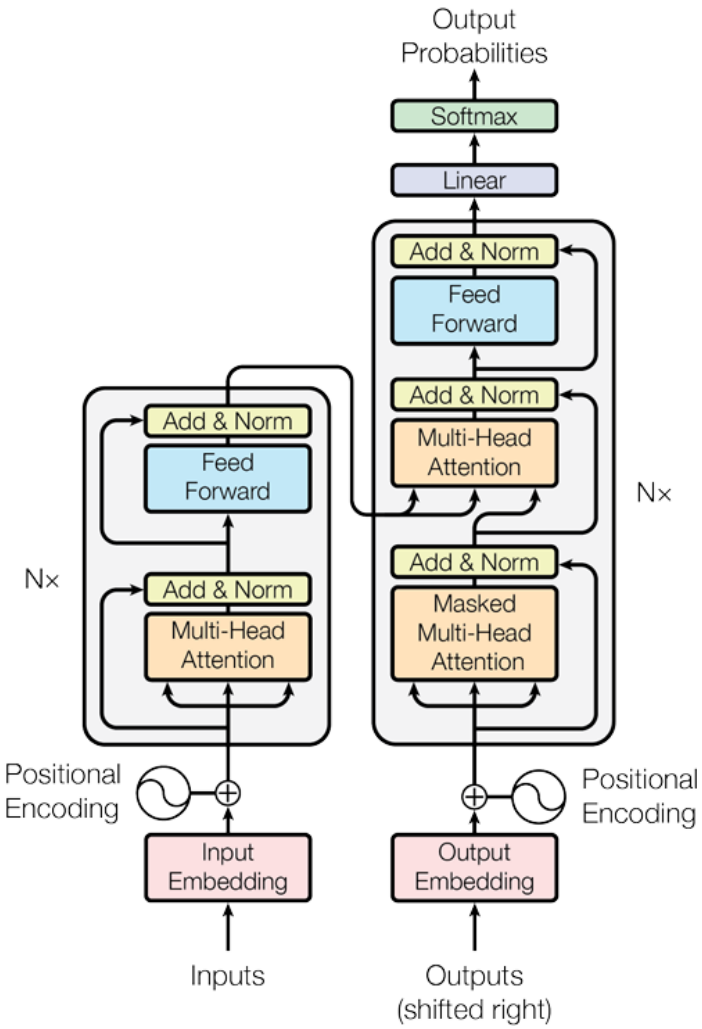
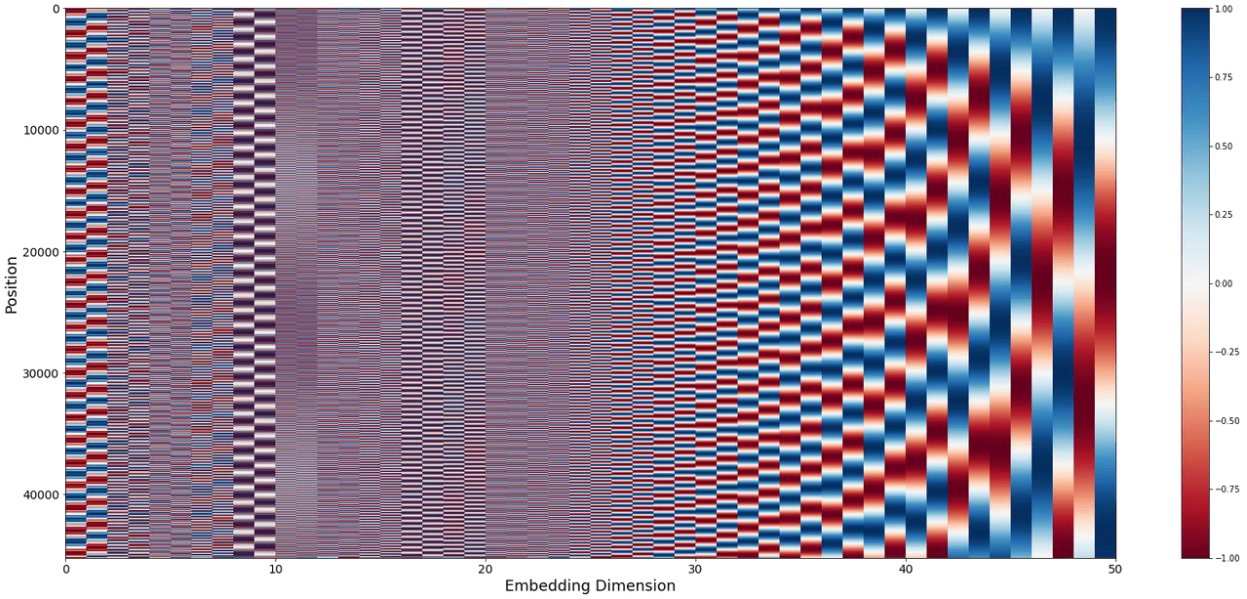


Figure 1: The Transformer - model architecture.

- **Positional encoding**

ترانسفورمر ها از ترتیب کلمات و position آن ها در جملات مطلع نیستند، بدون PE ورودی مثل یک bag of words میماند و ساختار sequence اش را از دست میدهد، برای جلوگیری از این اتفاق ورودی ها با یک positional encoding جمع میشوند. تا اطلاعات درباره موقعیت کلمات در متن را به اطلاعتمان inject کنیم.

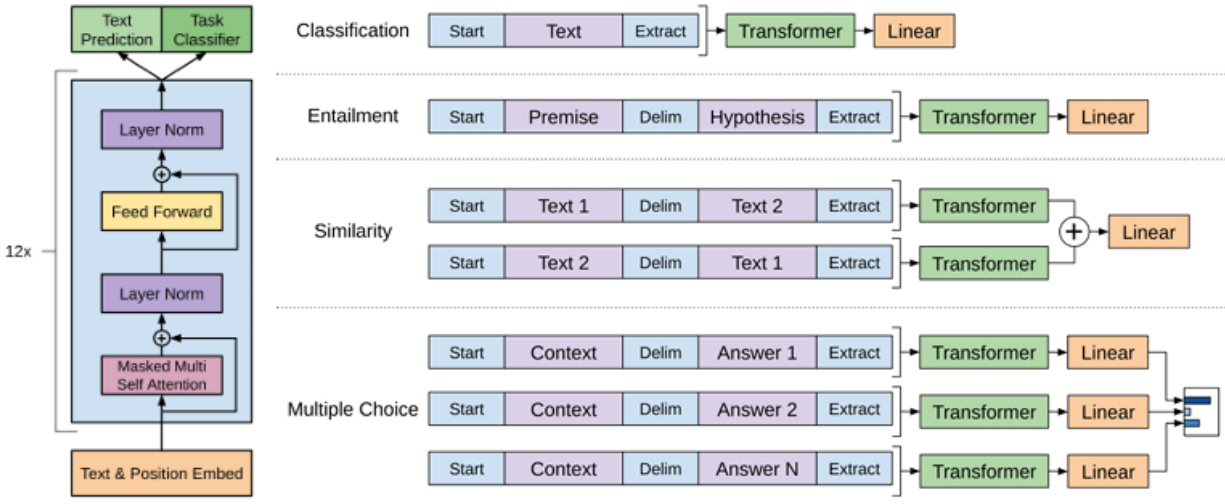
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Generative Pre-Trained Transformer (GPT)

GPT یک مدل left to right است که وظیفه next token prediction را دارد. این مدل فقط از بلاک های decoder ترنسفورمر استفاده میکنه و دیگه نیازی به قسمت cross attention هم به علت نبود قسمت encoder ندارد.

GPT از masked multi-head attention استفاده میکند چون میخواهد یکی یکی کلمات را generate کند و فقط بتواند از کلمات قبلی استفاده کند.



Feature	GPT-1	GPT-2
Release Date	June 2018	February 2019 (full in Nov)
Parameters	~117 million	124M (smallest) to 1.5 billion
Layers	12	Up to 48
Context Window	512 tokens	1024 tokens
Dataset Size	BooksCorpus (7,000+ books)	WebText (~8 million docs)

در این پروژه میخواهیم مدل از قبل tune شده GPT2 را با استفاده از دیتاست The Stanford Question Answering Dataset، fine-tune کنیم. این دیتاست به یک benchmark تبدیل شده برای مدل هایی که نیاز دارند توانایی نوشتن و درک و reason روی متن هارو درک کنند، مدل های زبانی مثل GPT و BERT معمولا روی این دیتاست fine tune میشوند. این دیتا ست سعی میکند با دادن یک context و سوال بتواند مدب را به درکی برساند که بتواند جواب صحیح را استخراج و برگرداند.

Dataset Pre-processing

Datapoint های این دیتاست به صورت JSON هستند. یک نمونه را در پایین مشاهده میکنید.


```
{
  "title": "Super_Bowl_50",
  "paragraphs": [
    {
      "context": "Super Bowl 50 was an American football game to determine the champion...",
      "qas": [
        {
          "id": "56be4db0acb8001400a502ec",
          "question": "Which NFL team won Super Bowl 50?",
          "answers": [
            {
              "text": "Denver Broncos",
              "answer_start": 177
            }
          ],
          "is_impossible": false
        }
      ]
    }
  ]
}
```


ابتدا این اطلاعات باید به context و question و answer به صورت منظم، ساده و قابل فهم تبدیل شوند. سپس tokenization با توجه به طول دنباله ها انجام شود. و با label masking خود سوال ها از محاسبه loss دور شوند.

با ست کردن label tokens به 100- مطمعن میشیم که loss فقط طبق answer ها محاسبه شود و نه کل prompt. این باعث میشه که مدل تغییرات گرادیان پرامپت را کامل دریافت نکند و یاد بگیره جواب generate کنه نه اینکه دوباره prompt رو بازسازی کنه، اگه این کارو انجام ندیم مدلی مثل GPT2 تمام توانایی یادگیری ش را صرف یادگرفتن recreate کردن پرامپت میکنه.

موضوع مهم دیگه ای که باید مورد بررسی قرار بگیره max_length هست که تعداد کل token هایی که مدل از input میبینه رو تعیین میکنه، این یک پارامتر خیلی مهم هست چون هم رویperformance و هم efficiency مدل تاثیرگذار هست.

Max Length	Implication
Too short	Context, question, or answer may be **truncated**, leading to **incomplete or wrong outputs**
Optimal	Enough room for full context + question + answer; best performance
Too long	Wastes computation on padding; can degrade training speed or cause OOM (Out of Memory) errors

Truncate کرد ن میتونه ریسک اینو داشته باشه که یه جواب ناقص بمونه، دیفالت GPT2 مقدار 1024 توکن هست اما ما این مقدار رو به 512 برای ساده تر شدن تغییر دادیم.

بهترین عمل اینه که به صورت پویا فقط context رو truncate کنیم و به سوال و جواب دست نزنیم. بیشترین الویت برای حفظ شدن رو جواب داره سپس سوال و در نهایت context. که از همین روش در این پروژه استفاده شده است.

انتخاب بین dynamic و static padding یک trade off بین computation cost و performance است. حالت استاتیک همیشه به یک اندازه محاسبات برای هر batch داره و مقدار memory مورد استفاده ش کاملا deterministic است.

Feature	Static Padding	Dynamic Padding
Padding Amount	Always to `max_length`	Up to longest in batch
Memory Usage	Constant, but potentially wasteful	Efficient and adaptive
Compute Efficiency	Predictable, often underutilized	Higher, less wasted computation
Hardware Compatibility	Better for TPUs, accelerators	Better on GPUs with dynamic support
Batch Size Flexibility	Fixed	Variable, can be larger
Implementation Complexity	Simple	Slightly more complex (needs collator)

Low-Rank Adaptation (LoRA)

LoRA تعداد پارامتر های trainable شبکه رو کاهش میدهد. در این روش میخواهیم `W را یاد بگیریم که ساختشه شه از W که وزن های از قبل tune شده مدل هستند به علاوه یک ماتریکس وزن دیگه که به دو ماتریکس کوچک تر A و B تبدیل میشود است.

$$W' = W + \Delta W = W + BA$$

where:

- $W \in \mathbb{R}^{d \times k}$ is the original pre-trained weight matrix
- $\Delta W = BA$ with $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and $r \ll \min(d, k)$
- Only matrices A and B are trained while W remains frozen

The rank r and scaling factor α control the adaptation capacity:

$$h = Wx + \frac{\alpha}{r} BAx$$

همینطور که مشاهده میشود، ماتریکس A و B ابعاد بسیار کوچکتری دارند (r اینجا مهم ترین پارامتر هست) که این باعث میشود محاسبات بسیار کمتری در فرایند یادگیری داشته باشیم. توجه داشته باشید که وزن های اولیه مدل که W باشد حقظ و freeze میشود. و دیگه لازم نیست همه وزن های مدل رو یاد بگیریم.

در جدول زیر میزان تاثیری که استفاده از LoRA بر روی تعداد پارامتر های trainable مدل GPT3 میگذارد را مشاهده میکنیم.

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

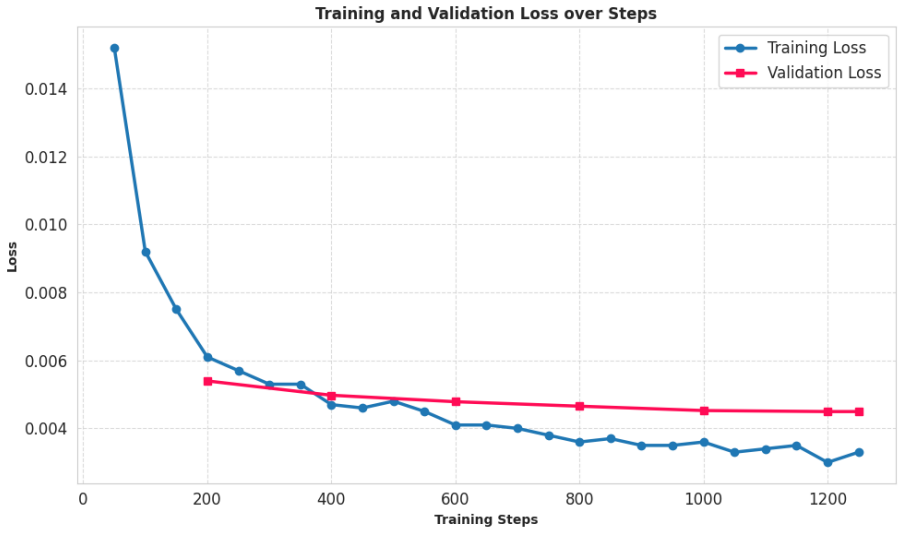
LoRA configurated models

برای این پروژه configuration 5 متفاوت مورد آزمایش و مقایسه قرار خواهد گرفت. تفاوت این آزمایش ها در مقدار r، اندازه learning rate و حضور یا عدم حضور mlp خواهد بود.

Experiment	Name	r	α (alpha)	Dropout	LR	Target Modules	Trainable Params	All Params	Trainable %
1	exp_r4_lr1e-4_attn	4	16	0.1	0.0001	['c_attn']	147,456	124,587,264	0.1184%
2	exp_r8_lr2e-4_attn	8	16	0.1	0.0002	['c_attn']	294,912	124,734,720	0.2364%
3	exp_r16_lr5e-4_attn	16	16	0.1	0.0005	['c_attn']	589,824	125,029,632	0.4717%
4	exp_r8_lr2e-4_attn_mlp	8	16	0.1	0.0002	['c_attn', 'mlp.c_proj']	663,552	125,103,360	0.5304%
5	exp_r32_lr2e-4_attn_mlp	32	32	0.1	0.0002	['c_attn', 'mlp.c_proj']	2,654,208	127,094,016	2.0884%

هرکدام از این مدل ها به اندازه epoch 10، fine-tune شدن.

نمودار train-val error آزمایش اول نشان میدهد که این تعداد epoch برای جلوگیری از overfitting کافی است.



در نهایت با انجام همه آزمایش ها به نتیجه زیر میرسیم.

`exp_r4_lr1e-4_attn`

Step	Training Loss	Validation Loss
200	0.0044	0.005062
400	0.0035	0.005025
600	0.0034	0.004739
800	0.0032	0.004759
1000	0.0033	0.004645
1200	0.0028	0.004613

`exp_r8_lr2e-4_attn_mlp`

Step	Training Loss	Validation Loss
200	0.0093	0.007652
400	0.0070	0.006079
600	0.0056	0.005521
800	0.0058	0.005297
1000	0.0055	0.005176
1200	0.0048	0.005070

`exp_r8_lr2e-4_attn`

Step	Training Loss	Validation Loss
200	0.0044	0.005062
400	0.0035	0.005025
600	0.0034	0.004739
800	0.0032	0.004759
1000	0.0033	0.004645
1200	0.0028	0.004613

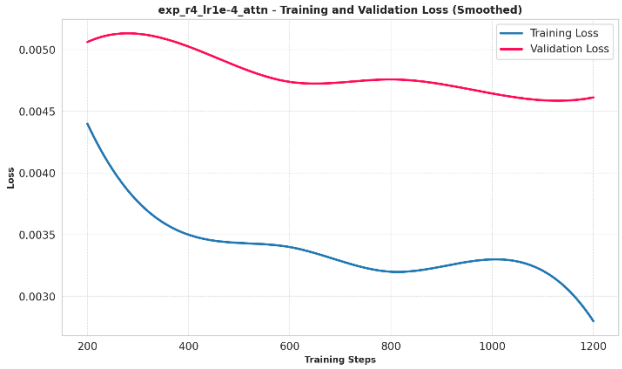
`exp_r32_lr2e-4_attn_mlp`

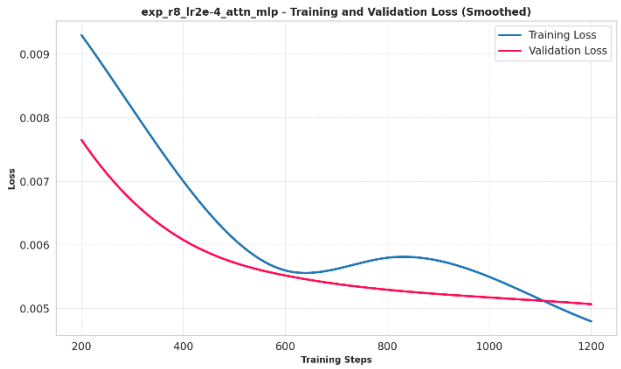
Step	Training Loss	Validation Loss
200	0.0085	0.007029
400	0.0066	0.005809
600	0.0053	0.005320
800	0.0055	0.005107
1000	0.0052	0.005002
1200	0.0045	0.004883

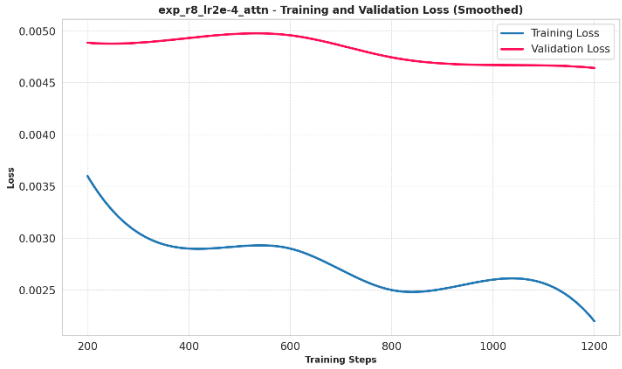
`exp_r16_lr5e-4_attn`

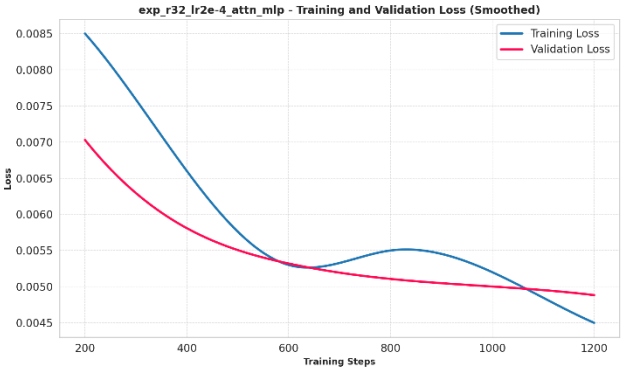
Step	Training Loss	Validation Loss
200	0.0031	0.004810
400	0.0022	0.004820
600	0.0019	0.005309
800	0.0015	0.005265
1000	0.0015	0.005287
1200	0.0011	0.005291

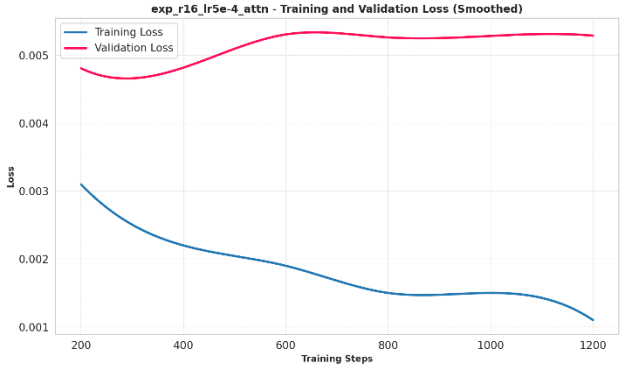
با بررسی نتایج میتوانیم نتیجه بگیریم که افزایش r تا زمانی داره به بهبود یادگیری کمک میکنه، وقتی از 4 به 8 میریم یادگیری بهتر شده اما وقتی به 16 رفتیم پس رفت میکنیم. نتیجه دیگه ای که میتونیم بگیریم اینه که اضافه کردن Multi Layer Perceptron به افزایش قدرت representation مدل کمک میکنه اما اضافه کردن MLP و بزرگ تر کردن مدل لزوما validation loss رو کاهش نمیده مگه اینکه مدل به خوبی regularized شده باشه.











یک نتیجه دیگه که از این آزمایش ها میگیرم اینه که مدل هایی که MLP دارند کمتر overfit شده اند و قابل اعتماد ترند، اما مدل هایی که ندارند، فاصله بین training loss و validation loss شان زیاد است و رو به overfit شدند هستند.

یک نتیجه دیگر آن است که مدل های بزرگ تر(یعنی مدل هایی که از rank بزگتری استفاده میکنند) به مدت زمان بیشتری برای یادگیری نیازدارند که این برایمان قابل پیشبینی بود.

پس از training مدل ها ذخیره شده و بهترین مدل برای ارزیابی استفاده میشود و با gpt2 base-model مقایسه میشود.

Model Evaluation

متریک های F1 و exact match برای ارزیابی مدل ها استفاده شده اند، هرچند متریک exact match خیلی مناسب جواب های طولانی نبود و بری اینکه مقدار بالایی دریافت کند لازم بود که max token هایی که خروجی میسازد را کاهش دهیم که خودش باعث کاهش F1 میشد، اما در نهایت به یک trade off رسیدیم که در آن هر دو متریک به 40% رسیدند.

```
SQuAD Metrics:
Exact Match: 40.00 | F1: 40.00
```

نمونه ای از خروجی های zero shot تولید شده توسط این مدل :

```
[ ] context = (
        "Barack Obama served as the 44th President of the United States. "
        "His story is the American story – values from the heartland, a middle-class upbringing in a strong family, "
        "hard work and education as the means of getting ahead, and the conviction that a life so blessed should be lived in service to others."
    )
    question = "Who is the 44th president of the USA?"
    print(generate_answer(ft_model, tokenizer, context, question))

⇒ Barack Obama.

[ ] context = (
        "The Mona Lisa is a half-length portrait painting by Italian artist Leonardo da Vinci. "
        "Considered an archetypal masterpiece of the Italian Renaissance, it has been described as the best known, "
        "most visited, most written about, and most parodied work of art in the world."
    )
    question = "Who painted the Mona Lisa?"
    print(generate_answer(ft_model, tokenizer, context, question))

⇒ Leonardo da Vinci

[ ] context = (
        "Mount Everest is Earth's highest mountain above sea level, located in the Himalayas on the border between Nepal and the Tibet Autonomous Region of China. "
        "Its peak is 8,848 meters (29,029 feet) above sea level."
    )
    question = "Where is Mount Everest located?"
    print(generate_answer(ft_model, tokenizer, context, question))

⇒ the Himalayas on the border between Nepal and the Tibet Autonomous Region of China. Mount Everest is 8,848 meters (29,029 feet) above sea level.
```

جواب های تولید شده و metric های به دست آمده برتری چشم گیری نسبت به base model داشته است.

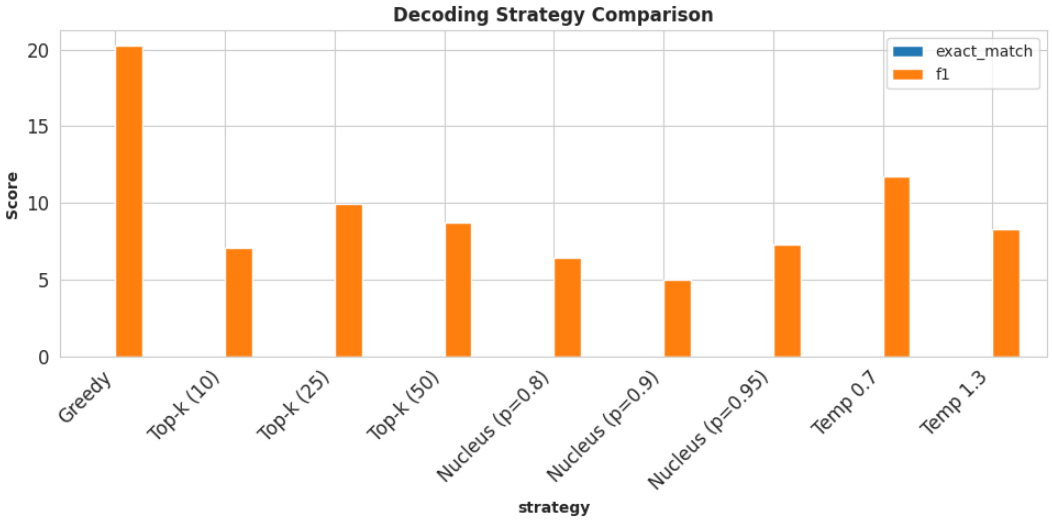
Model	Exact Match (%)	F1 Score (%)
Baseline GPT-2	0.0	2.55
Fine-tuned LoRA GPT-2	20.0	34.40

Decoding Strategies

1. **Greedy decoding :**
مدل در هر گام، محتمل‌ترین کلمه را انتخاب می‌کند.
2. **Top-K sampling :**
مدل فقط از بین k کلمه محتمل تر انتخاب میکنه که باعث میشه خلاقیت یکم بیشتر بشه ولی احتمال خطا هم به وجود میاد.
3. **Nucleus sampling :**
به جای تعداد ثابت، از تمام کلمات با احتمال تجمعی تا p انتخاب می‌شود که این یک کنترل تطبیقی ایجاد میکنه بر اساس تنوع کلمات.
4. **Temperature Scaling :**
با تنظیم عددی به نامTemperature ، میزان تصادفی بودن انتخاب کنترل می‌شود، اعداد بالا خلاقیت بیشتر و اعداد پایی نخلاقیت کمتر به همراه دارند.

با اعمال این استراتژی ها در تولید جواب به نتایج زیر رسیدیم. (درباره 0 بودن exact match صحبت کرده ایم.)

strategy	EM	F1
Greedy	0.00	20.22
Top-k (10)	0.00	7.06
Top-k (25)	0.00	9.93
Top-k (50)	0.00	8.76
Nucleus (p=0.8)	0.00	6.46
Nucleus (p=0.9)	0.00	5.05
Nucleus (p=0.95)	0.00	7.33
Temperature 0.7	0.00	11.72
Temperature 1.3	0.00	8.28



اگر هدف، دقت بالا و پاسخ دقیق باشد، Greedy decodingهمچنان بهترین انتخاب است.

برای افزایش تنوع بدون افت شدید کیفیت، Temperature 0.7گزینه مناسبی است.

Top-k و Nucleus sampling در این سناریو، باعث افت دقت شده‌اند و مناسب سوالات fact-based نیستند.