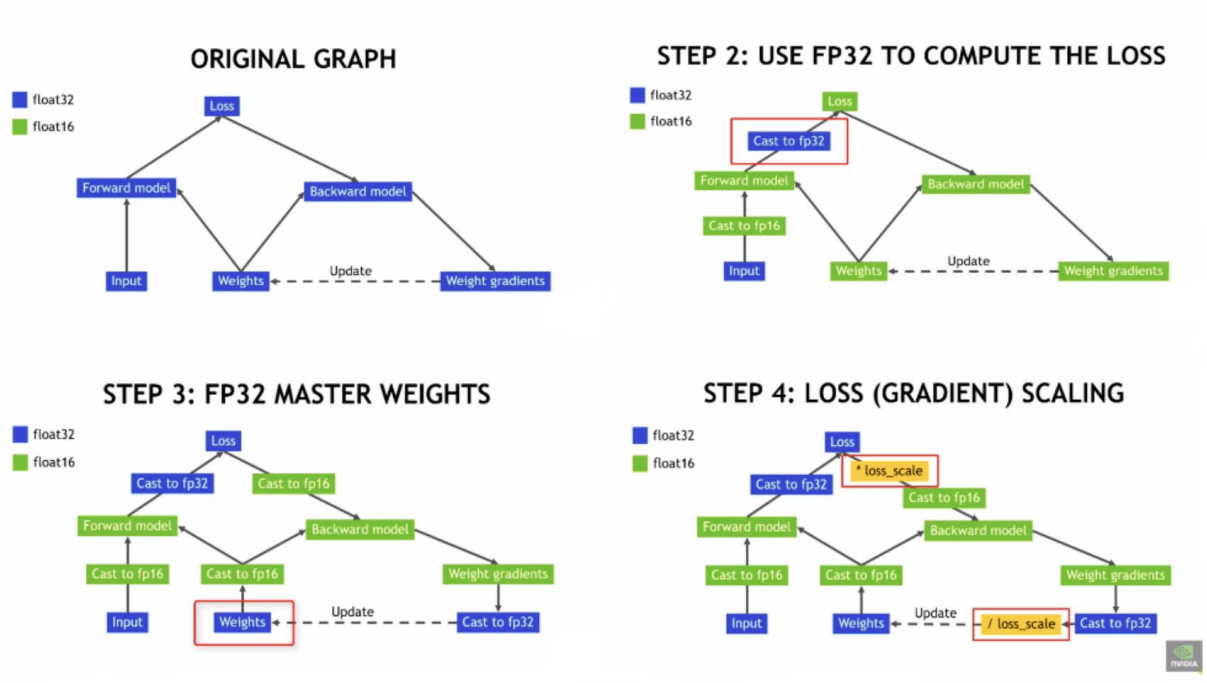
Mixed Precision



در تصویر بالا - سمت چپ یک مدل معمولی را مشاهده می‌کنید که برای اجرای تمام فرآیندهایش از fp32 استفاده می‌کند. حال به تصویر دوم دقت کنید. توجه کنید که در این حالت ورودی هنوز به شکل fp32 باقی مانده است ولی فرایند forwardروی fp16 انجام شده است که در واقع گام اول جهت افزایش سرعت است. همین‌طور مشاهده می‌کنید که برای محاسبه مقدار Loss آخرین خروجی شبکه عصبی تبدیل به fp32 شده است. علت این کار این است که مقدار Loss باید با بیشترین دقت ممکن محاسبه شود. یکی از دلایلی که مدل‌هایی که فقط از fp16 استفاده می‌کنند معمولا به دقت خوبی نمی‌رسند همین است که مقدار Loss را با تخمین بالایی محاسبه می‌کنند؛ اما در این روش چون مقدار Loss روی fp32محاسبه می‌شود این مشکل پیش نخواهد آمد.

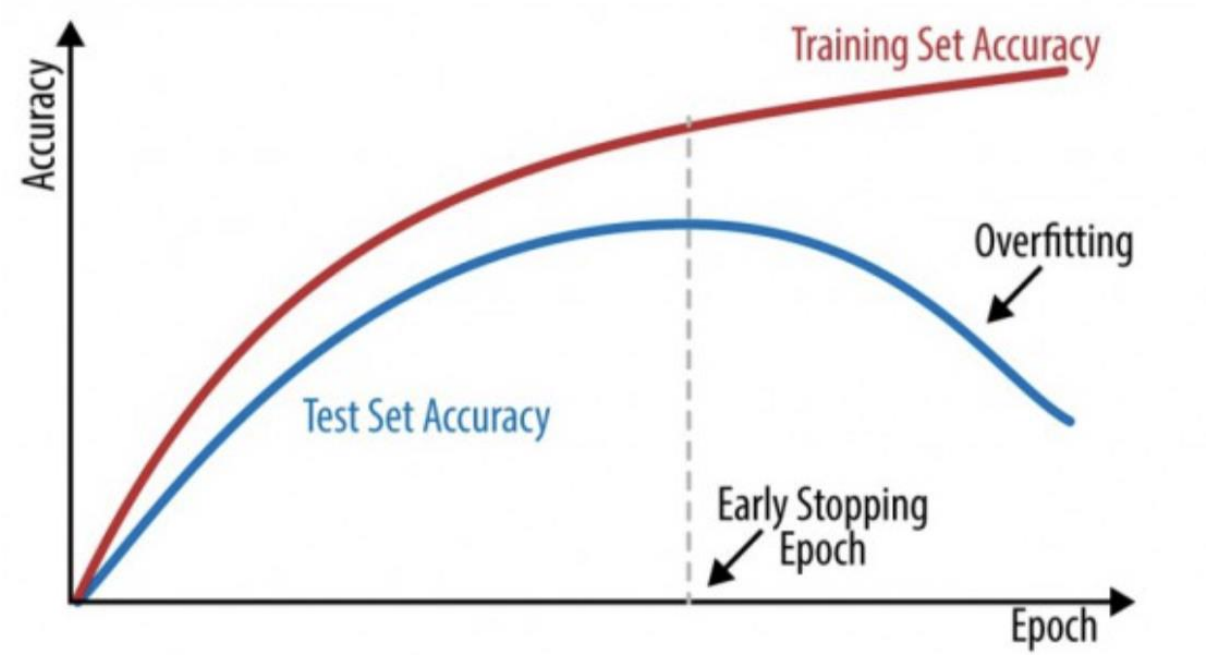
پس از محاسبه مقدار Loss مجددا آن را به fp16 تبدیل کرده و سپس فرآیند Backward انجام می‌شود که این کار نیز باعث افزایش سرعت می‌شود. در تصویر پایین - سمت چپ مشخص است که وزن‌ها را ابتدا با fp16 ذخیره کرده؛ ولی بعد از محاسبه گرادیان آن‌ها را جهت به‌روزرسانی به fp32 تبدیل می‌کند. این کار به همان دلیلی برای Loss گفته شد انجام می‌شود؛ در واقع گرادیان‌ها معمولا خودشان بسیار کوچک هستند و وقتی در fp16 اعمال شوند تقریبا باعث هیچ به‌روزرسانی‌ای روی وزن‌ها نمی‌شوند و شبکه آموزش داده نمی‌شود.

ممکن است کمی عجیب باشد؛ چون قبلا دیدیم که در فرآیند محاسبه یا Forward وزن‌ها به صورت fp16 بودند ولی برای ذخیره‌سازی و اعمال گرادیان‌ها از آن‌ها در حالت fp32 استفاده می‌شود. در حالت کلی فقط یک وزن وجود دارد که به

آن Master Weights گفته می‌شود. این وزن‌ها همواره به صورت fp32 هستند و فقط زمانی که قرار است با آن‌ها محاسبات انجام دهیم) فرآیند (Forward به fp16 تبدیل می‌شوند.

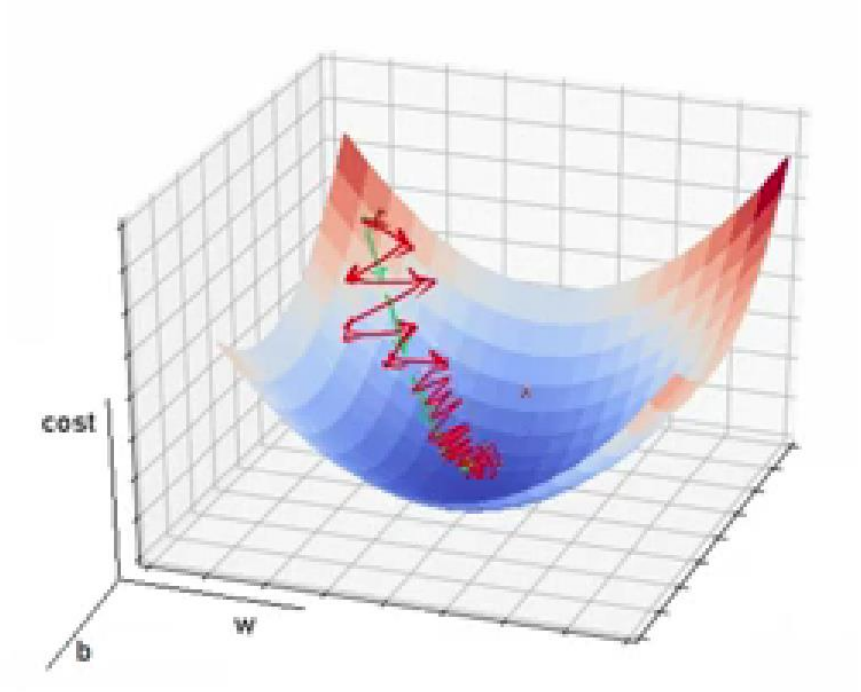
چرا به Gradient Scaling نیاز داریم؟

اگر در حالت Forward برای یک لایه خاص ورودی float16 داشته باشید، Backwardآن لایه نیز گرادیان‌ها را در float16ایجاد می‌کند. مقادیر گرادیان‌ها به اندازه‌ای کوچک هستند که ممکن است در float16 قابل نمایش نباشند. این مقادیر به صفر میل می‌کنند (underflow) بنابراین به‌روزرسانی برای پارامترهای مربوطه از بین می‌رود و وزن‌ها هیچ تغییری نمی‌کنند. برای جلوگیری از نابود شدن این گردایان‌ها، Lossشبکه را در یک عدد بزرگ ضرب می‌کنیم و گرادیان‌ها را با استفاده از این Scaled Loss محاسبه می‌کنیم. پس از محاسبه گرادیان‌ها آن‌ها را به همان ضریبی که در Loss ضرب کرده بودیم تقسیم می‌کنیم. این کار باعث می‌شود تاثیر عملیاتی که روی Loss انجام دادیم از بین برود و روی آموزش مدل تاثیری نداشته باشد اما مشکل از بین رفتن گردایان‌ها به خاطر کوچک بودن مقادیرشان حل شده است. این فرآیند در سمت راست پایین تصویر نمایش داده شده است



.اگر بهبودی در عملکرد دیده نشود، متغیر counter افزایش می‌یابد و در صورتی که این مقدار به patience برسد، علامت early_stop به True تغییر می‌کند که نشان‌دهنده اتمام زودهنگام آموزش است. در غیر این صورت، بهبود را تایید می‌کند و مدل را ذخیره می‌کند

قضیه mini batch



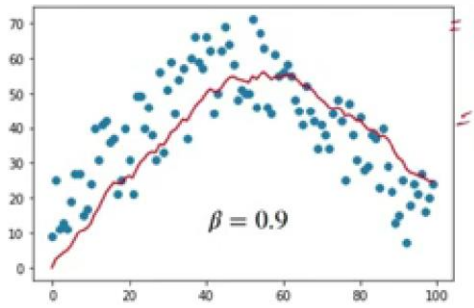
Momentum Optimizer

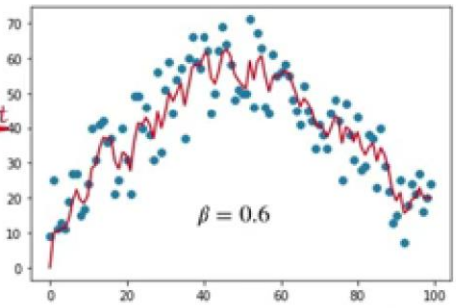
استفاده از moving average

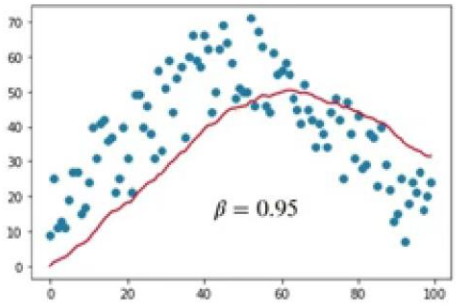
$$V_t = \beta * (V_{t-1}) + (1 - \beta) * \theta_t$$

$$= 0.9(V_{t-1}) + 0.1\theta_t$$

$$= 0.6\,V_{t-1} + (0.4)\theta_t$$







$\beta = 0.9$

ی حال نزدیک به میانگینی میشه تا اون نویز های مسیر کاهش گرادیان رو کم کنه

در واقع b بزرگ باعث میشه به الان اهمیت کمتری بدیم در حالی که کمش باعث میشه به اطلاعات گذشته اهمیت کمتری بدیم

و خب در محاسبه v t-1 اینطوریه که هی مقدار های قبلی تر در اعداد کوچکتر ضرب میشن و اهمیتشون کم میشه

RMSprop

$$v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1-\beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

اپسیلون برا اینه مخرج صفر بشه

اگر dw زیاد بشه یعنی خیلی حرکت کرده پس مخرج زیاد میشه پس سعی میکنیم ارومش کنیم

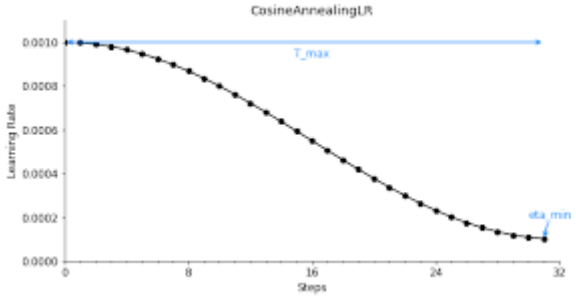
حالا ادام میشه ترکیب این دو دوست عزیز:

$$\nu_t = \beta_1 * \nu_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

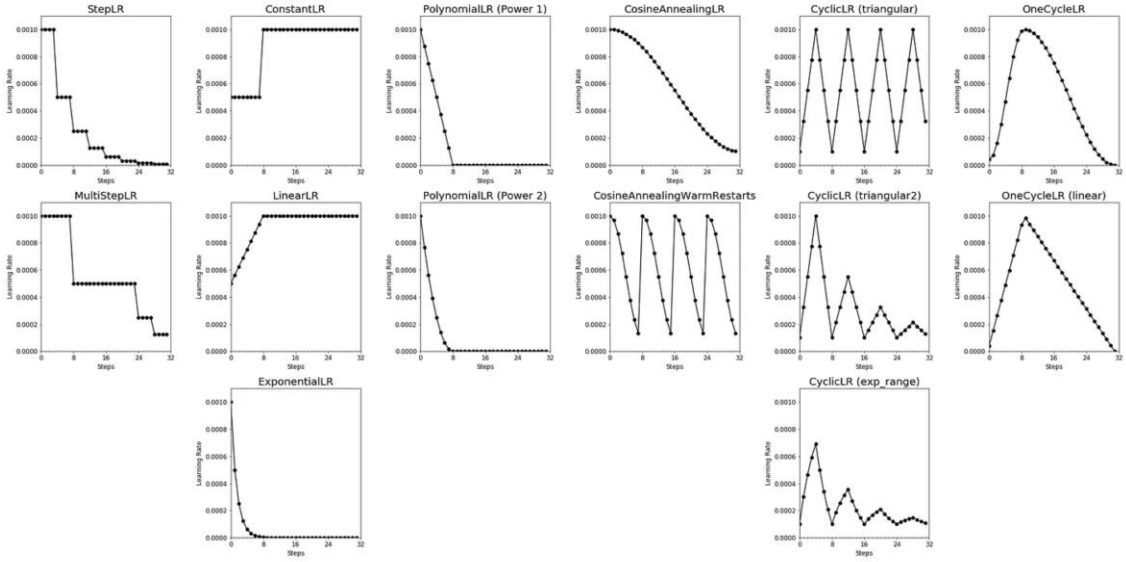
η : *Initial Learning rate*
g_t : *Gradient at time t along* ω^j
ν_t : *Exponential Average of gradients along* ω_j
s_t : *Exponential Average of squares of gradients along* ω_j

CosineAnnealingLR:

starts with a very large learning rate and then aggressively decreases it to a value near 0 before increasing the learning rate again



OneCycleLR:



Sets the learning rate of each parameter group according to the 1cycle learning rate policy. The 1cycle policy anneals the learning rate from an initial learning rate to some maximum learning rate and then from that maximum learning rate to some minimum learning rate much lower than the initial learning rate

روش های باحال :

```python
def adjust_learning_rate(accelerator, optimizer, scheduler, epoch, args, printout=True):
    if args.lradj == 'type1':
        lr_adjust = {epoch: args.learning_rate * (0.5 ** ((epoch - 1) // 1))}
    elif args.lradj == 'type2':
        lr_adjust = {
            2: 5e-5, 4: 1e-5, 6: 5e-6, 8: 1e-6,
            10: 5e-7, 15: 1e-7, 20: 5e-8
        }
    elif args.lradj == 'type3':
        lr_adjust = {epoch: args.learning_rate if epoch < 3 else args.learning_rate * (0.9 ** ((epoch - 3) // 1))}
    elif args.lradj == 'PEMS':
        lr_adjust = {epoch: args.learning_rate * (0.95 ** (epoch // 1))}
    elif args.lradj == 'TST':
        lr_adjust = {epoch: scheduler.get_last_lr()[0]}
    elif args.lradj == 'constant':
        lr_adjust = {epoch: args.learning_rate}
    if epoch in lr_adjust.keys():
        lr = lr_adjust[epoch]
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
        if printout:
            if accelerator is not None:
                accelerator.print('Updating learning rate to {}'.format(lr))
            else:
                print('Updating learning rate to {}'.format(lr))
```

نحوه عملکرد:Dropout

Dropoutبه طور تصادفی برخی از نورون‌ها را در شبکه عصبی شما در طول آموزش غیرفعال می‌کند.

این کار باعث می‌شود که شبکه شما به اتکا به یک نورون خاص برای انجام پیش‌بینی‌ها عادت نکند.

در نتیجه، شبکه شما در برابر نویز و داده‌های جدید مقاوم‌تر می‌شود.

مزایای:Dropout

Dropoutمی‌تواند به طور قابل‌توجهی overfitting را کاهش دهد.

Dropoutمی‌تواند به بهبود عملکرد مدل شما در داده‌های unseen کمک کند.

Dropoutمی‌تواند به آموزش سریع‌تر مدل شما کمک کند.

معایب:Dropout

Dropoutمی‌تواند باعث افزایش زمان آموزش شود.

Dropoutمی‌تواند دقت مدل شما را کمی کاهش دهد.

معماری مقاله time llm

چندتا نکته مهم قبل ساختار

دیتاست :

Tasks	Dataset	Dim.	Series Length	Dataset Size	Frequency	Domain
Long-term Forecasting	ETTm1	7	{96, 192, 336, 720}	(34465, 11521, 11521)	15 min	Temperature
	ETTm2	7	{96, 192, 336, 720}	(34465, 11521, 11521)	15 min	Temperature
	ETTh1	7	{96, 192, 336, 720}	(8545, 2881, 2881)	1 hour	Temperature
	ETTh2	7	{96, 192, 336, 720}	(8545, 2881, 2881)	1 hour	Temperature
	Electricity	321	{96, 192, 336, 720}	(18317, 2633, 5261)	1 hour	Electricity
	Traffic	862	{96, 192, 336, 720}	(12185, 1757, 3509)	1 hour	Transportation
	Weather	21	{96, 192, 336, 720}	(36792, 5271, 10540)	10 min	Weather
	ILI	7	{24, 36, 48, 60}	(617, 74, 170)	1 week	Illness
Short-term Forecasting	M3-Quarterly	1	8	(756, 0, 756)	Quarterly	Multiple
	M4-Yearly	1	6	(23000, 0, 23000)	Yearly	Demographic
	M4-Quarterly	1	8	(24000, 0, 24000)	Quarterly	Finance
	M4-Monthly	1	18	(48000, 0, 48000)	Monthly	Industry
	M4-Weakly	1	13	(359, 0, 359)	Weakly	Macro
	M4-Daily	1	14	(4227, 0, 4227)	Daily	Micro
	M4-Hourly	1	48	(414, 0, 414)	Hourly	Other

پارامتر ها:

Task-Dataset / Configuration	Model Hyperparameter					Training Process			
	Text Prototype V'	Backbone Layers	Input Length T	Patch Dim. d_{m}	Heads K	LR*	Loss	Batch Size	Epochs
LTF - ETTh1	1000	32	512	16	8	10^{-3}	MSE	16	50
LTF - ETTh2	1000	32	512	16	8	10^{-3}	MSE	16	50
LTF - ETTm1	1000	32	512	16	8	10^{-3}	MSE	16	100
LTF - ETTm2	1000	32	512	16	8	10^{-3}	MSE	16	100
LTF - Weather	1000	32	512	16	8	10^{-2}	MSE	8	100
LTF - Electricity	1000	32	512	16	8	10^{-2}	MSE	8	100
LTF - Traffic	1000	32	512	16	8	10^{-2}	MSE	8	100
LTF - ILI	100	32	96	16	8	10^{-2}	MSE	16	50
STF - M3-Quarterly	100	32	$2 \times H$ †	32	8	10^{-4}	SMAPE	32	50
STF - M4	100	32	$2 \times H$ †	32	8	10^{-4}	SMAPE	32	50

† H represents the forecasting horizon of the M4 and M3 datasets.
∗ LR means the initial learning rate.

معیار های خطا :

$$\text{MSE} = \frac{1}{H}\sum_{h=1}^{T}(\mathbf{Y}_h - \hat{\mathbf{Y}}_h)^2, \qquad \text{MAE} = \frac{1}{H}\sum_{h=1}^{H}|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|,$$

$$\text{SMAPE} = \frac{200}{H}\sum_{h=1}^{H}\frac{|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|}{|\mathbf{Y}_h| + |\hat{\mathbf{Y}}_h|}, \qquad \text{MAPE} = \frac{100}{H}\sum_{h=1}^{H}\frac{|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|}{|\mathbf{Y}_h|},$$

$$\text{MASE} = \frac{1}{H}\sum_{h=1}^{H}\frac{|\mathbf{Y}_h - \hat{\mathbf{Y}}_h|}{\frac{1}{H-s}\sum_{j=s+1}^{H}|\mathbf{Y}_j - \mathbf{Y}_{j-s}|}, \qquad \text{OWA} = \frac{1}{2}\left[\frac{\text{SMAPE}}{\text{SMAPE}_{\text{Naïve2}}} + \frac{\text{MASE}}{\text{MASE}_{\text{Naïve2}}}\right],$$

توضیح معیار های خطا :

1. خطای مطلق میانگین (MAE):

مزایا:

ساده و قابل فهم

به مقیاس داده‌ها حساس نیست

معایب:

جهت خطا را در نظر نمی‌گیرد

به مقادیر پرت حساس است

فرمول:

MAE = (1/n) * Σ(|A_i - F_i|)

2. خطای میانگین مربعات (MSE):

مزایا:

خطاهای بزرگ را بیشتر جریمه می‌کند

ساده و قابل فهم

معایب:

به مقیاس داده‌ها حساس است

واحد آن به صورت مجذور واحد داده اصلی است

فرمول:

$$MSE = (1/n) * \Sigma((A_i - F_i)^2)$$

3. جذر خطای میانگین مربعات (RMSE):

مزایا:

واحد آن با واحد داده اصلی یکسان است

تفسیر آسان‌تر

معایب:

به مقادیر پرت حساس است

مانند MSE به مقیاس داده‌ها حساس است

فرمول:

$$RMSE = \sqrt{(MSE)}$$

4. خطای مطلق میانگین درصدی (MAPE):

مزایا:

برای مقایسه مدل‌ها در مجموعه داده‌های مختلف با مقیاس‌های متفاوت مفید است

معایب:

به مقادیر پرت و مقادیر صفر یا نزدیک به صفر حساس است

جهت خطا را در نظر نمی‌گیرد

فرمول:

$$MAPE = (1/n) * \Sigma(|A_i - F_i| / ((|A_i| + |F_i|) / 2)) * 100$$

5. خطای مطلق میانگین درصدی متقارن (SMAPE):

مزایا:

با مقادیر صفر مشکلی ندارد

خطاهای بیش‌برآورد و کم‌برآورد را به طور مساوی جریمه می‌کند

معایب:

به مقادیر پرت حساس است

فرمول:

$$SMAPE = (1/n) * \Sigma(|A_i - F_i| / ((|A_i| + |F_i|) / 2)) * 100$$

6. خطای مطلق مقیاس‌شده میانگین (MASE):

مزایا:

مستقل از مقیاس است

به مقادیر پرت حساسیت کمتری دارد

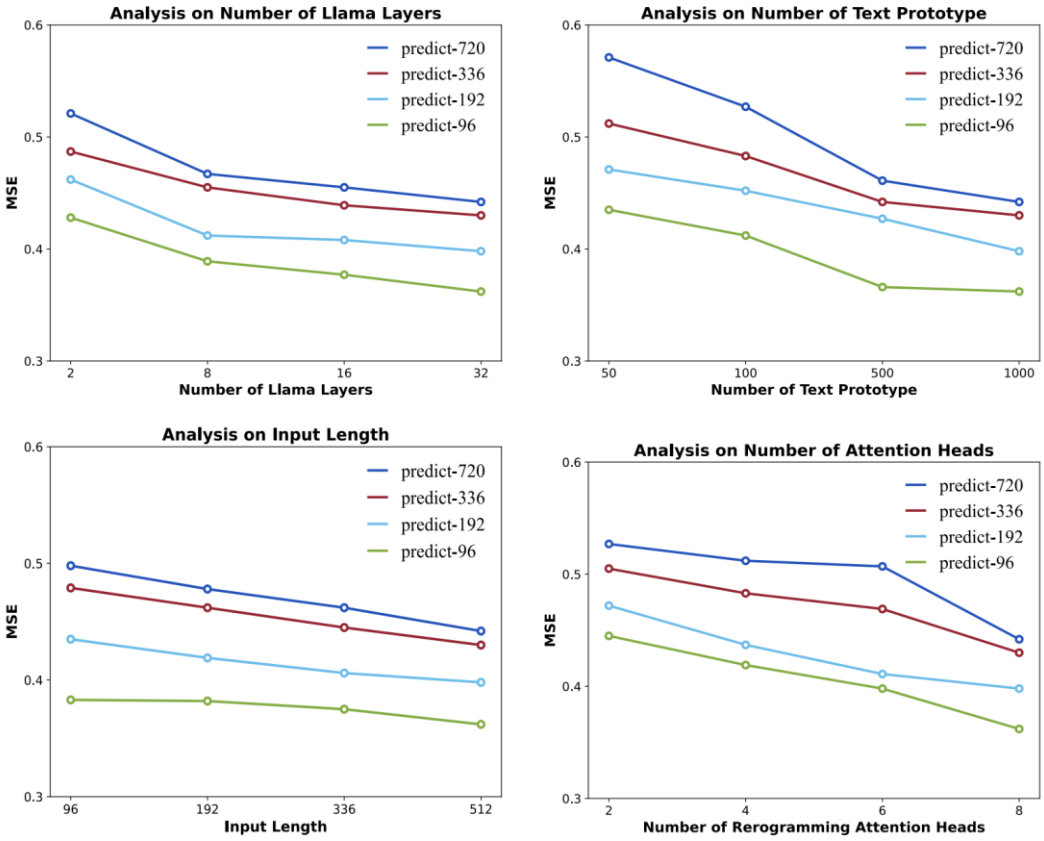
معایب:

فرض می‌کند که مدل ساده دقت قابل قبولی دارد

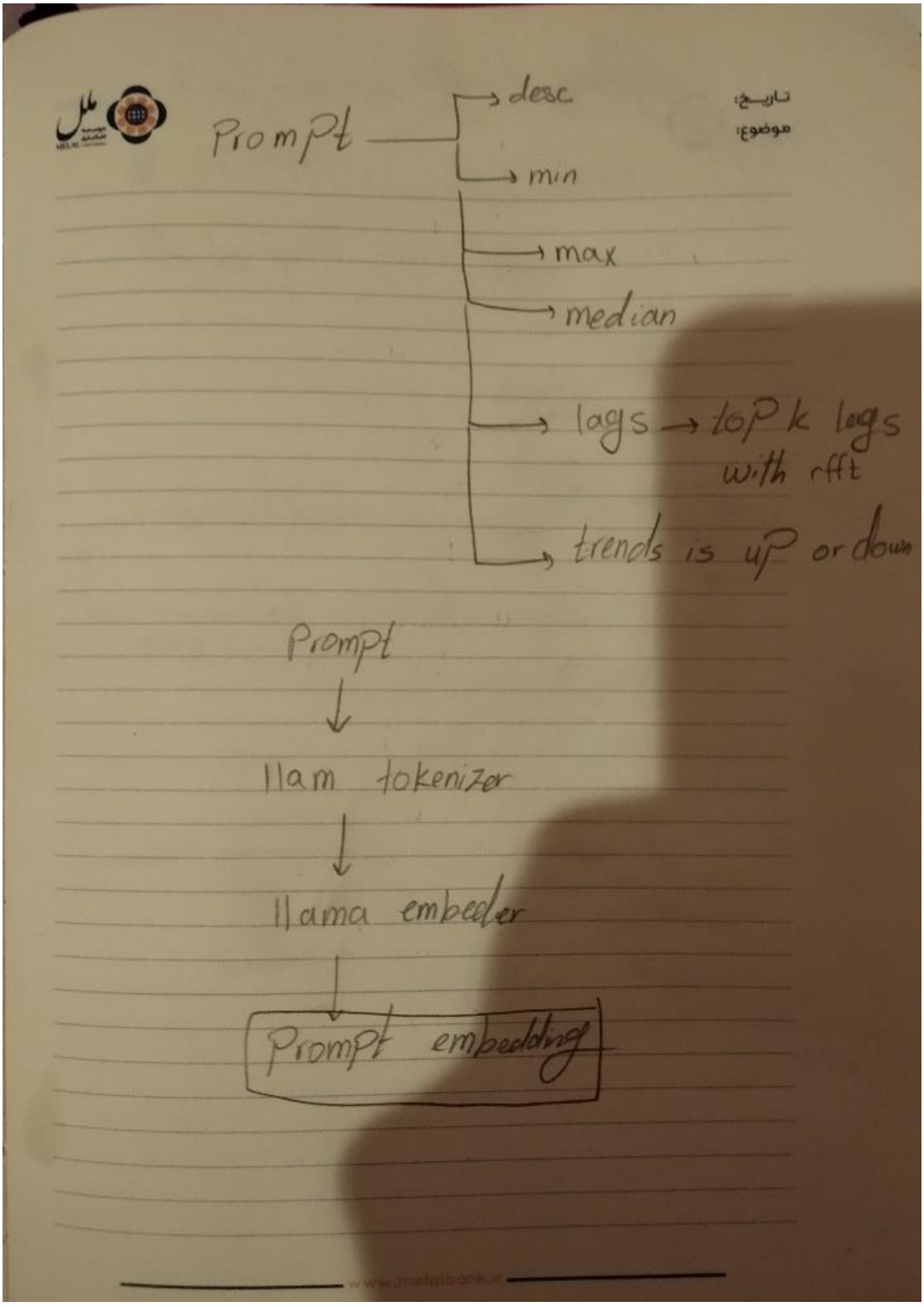
فرمول:

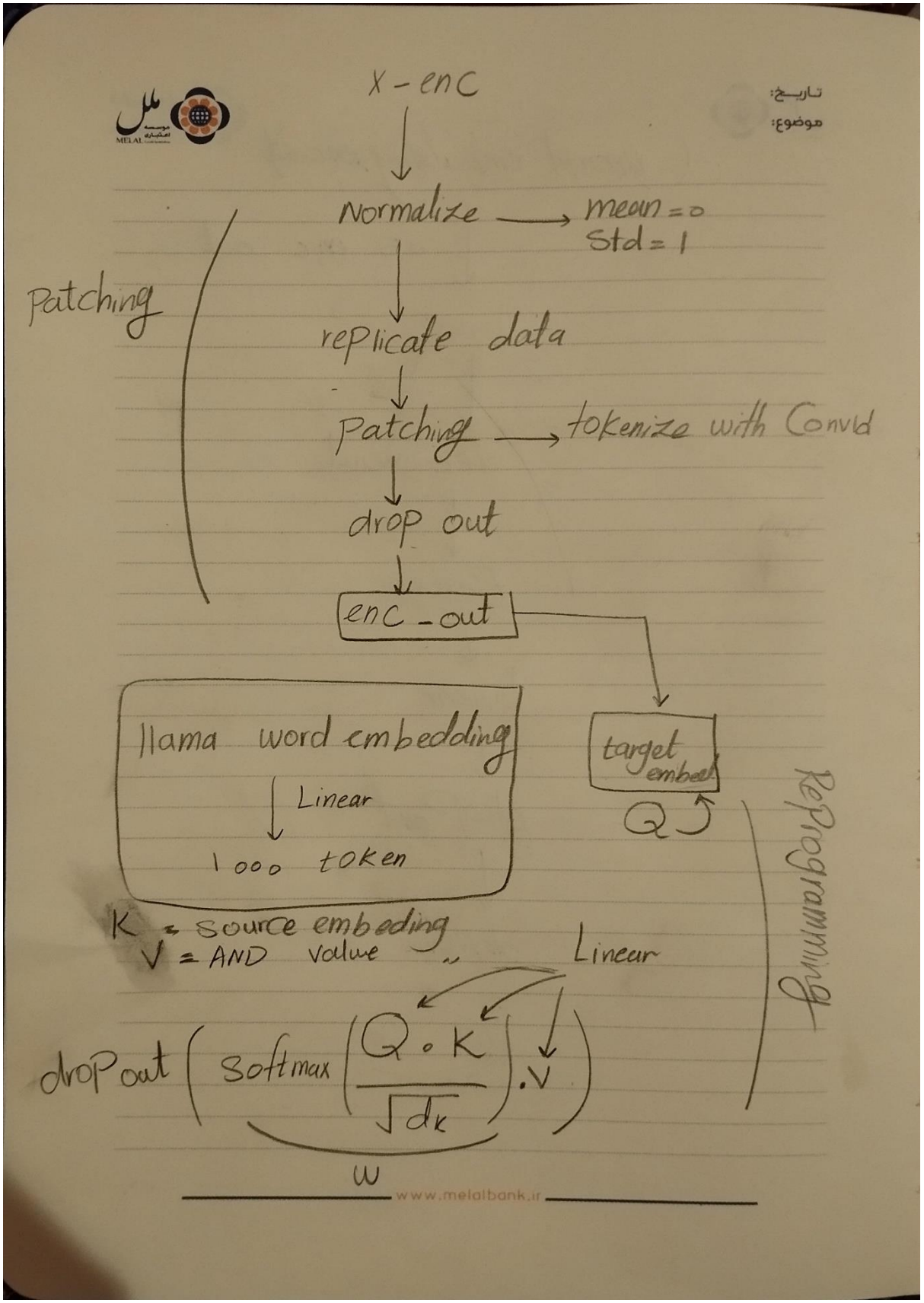
MASE = (MAE / (1/n-1) * Σ(|y_i - y_{i-1}|))

نحوه رابطه پارمتر ها با خطا :

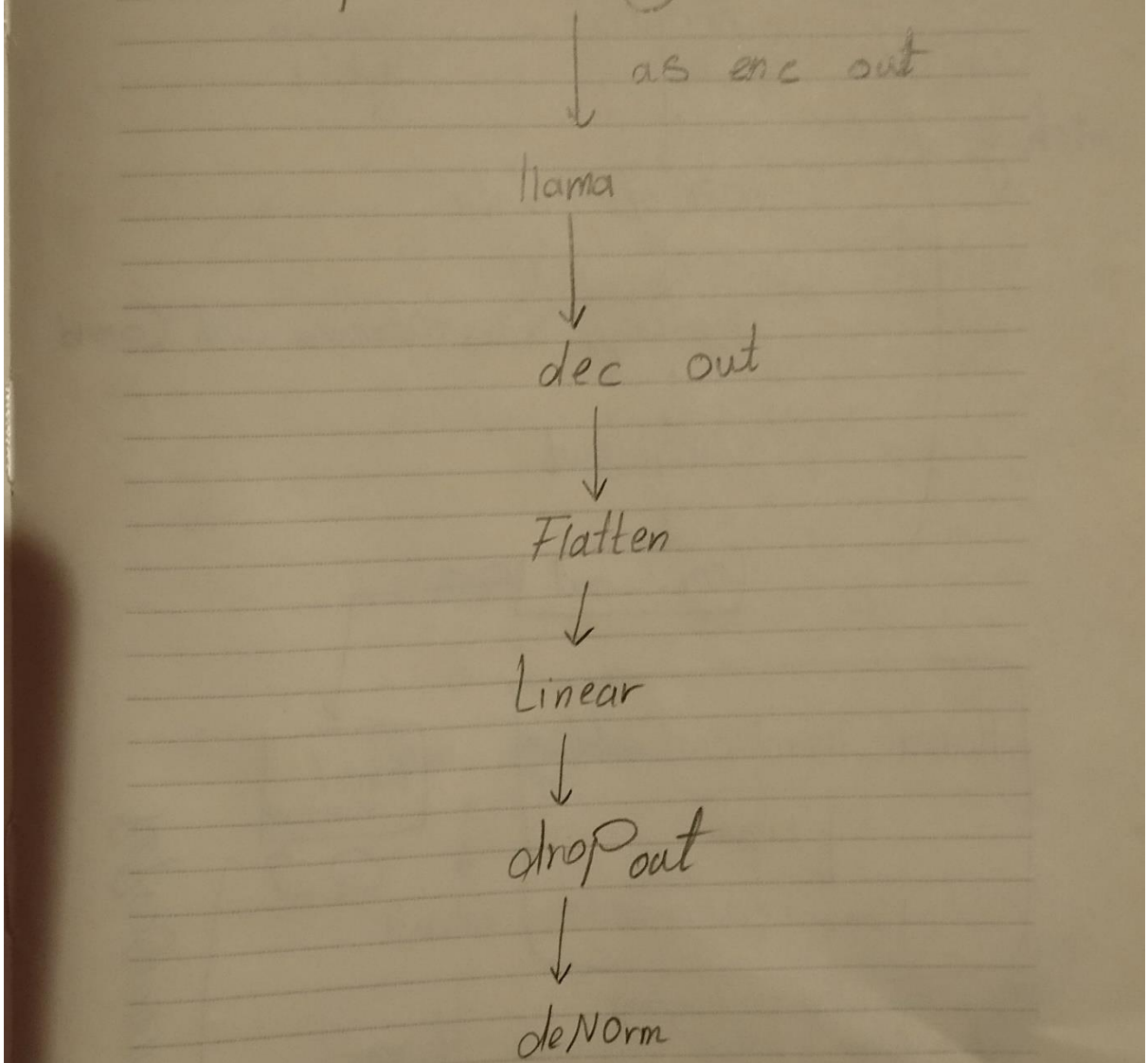


معماری:


Prompt
desc
min
max
median
lags → top k lags with rfft
trends is up or down
Prompt
llam tokenizer
llama embeder
Prompt embedding


X - enc
Normalize
mean = 0
Std = 1
Patching
replicate data
Patching
tokenize with Convid
drop out
enc - out
llama word embedding
Linear
1000 token
target embed
Q
Reprogramming
K = source embeding
V = AND value
Linear
dropout (Softmax (Q · K / √dk) . V)
W

تاريخ:
موضوع:


(prompt embedding enc-out
as enc out
llama
dec out
Flatten
Linear
dropout
deNorm

توضیحات :

شرح گام به گام پردازش ورودی در مدل پیش‌بینی سری زمانی:

1. نرم‌سازی:

هر کانال ورودی در سری زمانی ((X(i)) به صورت جداگانه با استفاده از تکنیکی به نام نرم‌سازی نمونه برگشت‌پذیر (RevIN) نرمال می‌شود.

این کار تضمین می‌کند که تمام کانال‌ها میانگین صفر و انحراف معیار یک داشته باشند، که یادگیری را برای مدل آسان‌تر می‌کند.

2. Patching:

سری زمانی نرمال‌شده به Patchهایی با طول Lp تقسیم می‌شود.

این Patchها می‌توانند همپوشانی یا غیر همپوشانی داشته باشند.

Patchهای همپوشانی اطلاعات محلی بیشتری را ثبت می‌کنند، اما می‌توانند از نظر محاسباتی پرهزینه باشند.

تعداد کل Patchها (P) با استفاده از فرمول زیر محاسبه می‌شود:

$$P = \lfloor (T - Lp) / S \rfloor + 2$$

در این فرمول:

T طول سری زمانی است.

S گام لغزنده افقی است که تعیین می‌کند پنجره Patch چقدر بین Patchهای متوالی حرکت می‌کند.

مثال:

فرض کنید T = 10، Lp = 4 و S = 2. در این حالت، Patch .P = 5ها به صورت زیر خواهند بود:

Patch 1: [1, 2, 3, 4]

Patch 2: [3, 4, 5, 6]

Patch 3: [5, 6, 7, 8]

Patch 4: [7, 8, 9, 10]

Patch 5: [9, 10]

3. رمزگذاری Patch:

هر Patch (X_(i)^P) با استفاده از یک لایه خطی ساده به یک نمایش با ابعاد پایین‌تر ((^X_(i)^(P)) از اندازه dm تبدیل می‌شود.

این فرآیند به ثبت اطلاعات ضروری از هر Patch در حالی که اندازه کلی داده ورودی را کاهش می‌دهد، کمک می‌کند.

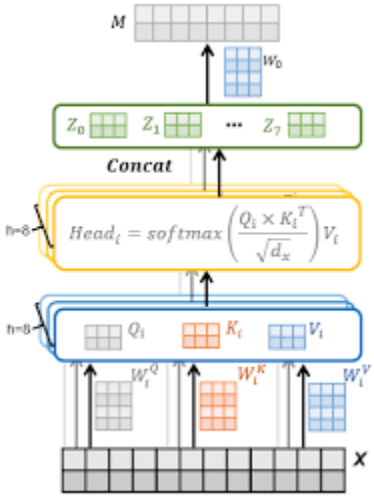
نکته :

در معماری مولتی هد attention، ورودی به چندین سر هد تقسیم می‌شود، هر یک از این سرها (heads) به طور مستقل و به صورت موازی عمل می‌کنند و نتایج آن‌ها پس از ادغام با یکدیگر به عنوان خروجی مورد استفاده قرار می‌گیرد.

سرها در واقع نقش موازنه‌ای دارند که به مدل امکان می‌دهند اطلاعات مختلفی را به صورت موازی درک و استخراج کند. از آن‌جایی که هر سر هد از یک نسخه مختلف از متون هدف و منبع استفاده می‌کند، این امکان وجود دارد که هر سر هد به ویژگی‌های مختلفی از داده‌ها توجه کند.

به عنوان مثال، اگر بخواهیم یک متن را ترجمه کنیم، یک سر هد می‌تواند به کلمات اصلی توجه کند در حالی که سر دیگری به جملات کلی متن توجه کند و سعی کند ساختار کلی متن را درک کند.

به طور کلی، مولتی هد attention به مدل امکان می‌دهد تا از چندین منظر مختلف به داده نگاه کند و از همه این اطلاعات برای ایجاد یک نتیجه نهایی استفاده کند، که این کار می‌تواند کمک شایانی به بهبود عملکرد مدل و توانایی‌های آن کند.



توضیح راجب reprogramming

Reprogramming: این فرآیند به معنای تغییر یا بازآفرینی وصله های زمان سری با استفاده از اطلاعات متن و پیش آموزش های کلمه ای است.

Text prototypes: اینها مجموعه ای کوچک از نمونه های اولیه متن هستند که الگوهای زبان مرتبط با داده های سری زمانی را نشان می دهند.

Multi-head attention layer: این لایه به مدل اجازه می دهد تا به الگوهای مختلف در داده ها با استفاده از چندین "سر" توجه یا تمرکز متفاوت، توجه کند.

ارتباط با کد:

لایه رمزگذاری وصله: کد به طور مستقیم به این بخش اشاره نمی کند، اما تصور می شود که وصله های زمان سری قبل از ورود به کد رمزگذاری شده اند. این رمزگذاری به تبدیل وصله ها به فضای برداری قابل درک برای مدل کمک می کند.

Text prototypes: این نمونه های اولیه در ماتریس E′ با ابعاد V′ × D ذخیره می شوند، که در آن V′ تعداد نمونه های اولیه و D ابعاد بردارهای جاسازی کلمه است.

Multi-head attention layer: این لایه با تابع reprogramming در کلاس ReprogrammingLayer پیاده سازی می شود. این تابع:

Query matrices (Q): این ماتریس ها از بردارهای جاسازی patch و وزن های یادگیری شده تشکیل شده اند و نشان می دهند که کدام اطلاعات از وصله برای سر توجه خاص مهم است.

Key matrices (K): این ماتریس ها از بردارهای جاسازی نمونه های اولیه و وزن های یادگیری شده تشکیل شده اند و نشان می دهند کدام نمونه های اولیه با وصله مرتبط هستند.

Value matrices (V): این ماتریس ها از بردارهای جاسازی نمونه های اولیه تشکیل شده اند و اطلاعاتی را که از نمونه های اولیه مرتبط به دست می آید، ارائه می دهند.

Attention: این عملیات وزن دهی به ارزش ها بر اساس سازگاری بین سوالات و کلیدها را انجام می دهد. هر سر توجه نتیجه متفاوتی تولید می کند.

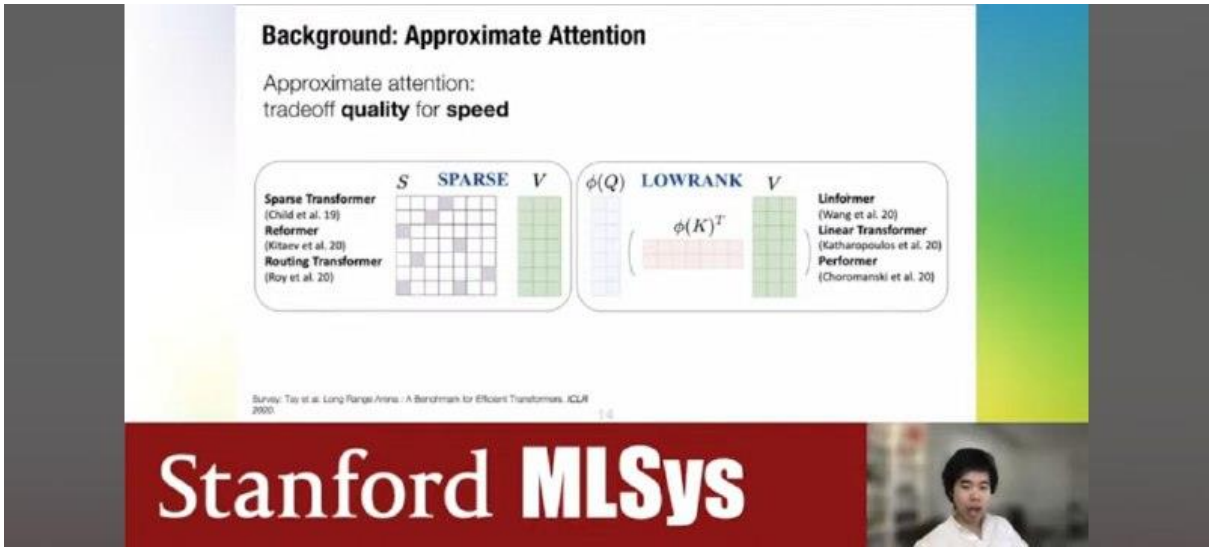
Aggregation: نتایج از همه سرهای توجه جمع می شوند تا یک بردار "برنامه ریزی مجدد" نهایی برای هر patch به دست آید.

مقاله flash attention

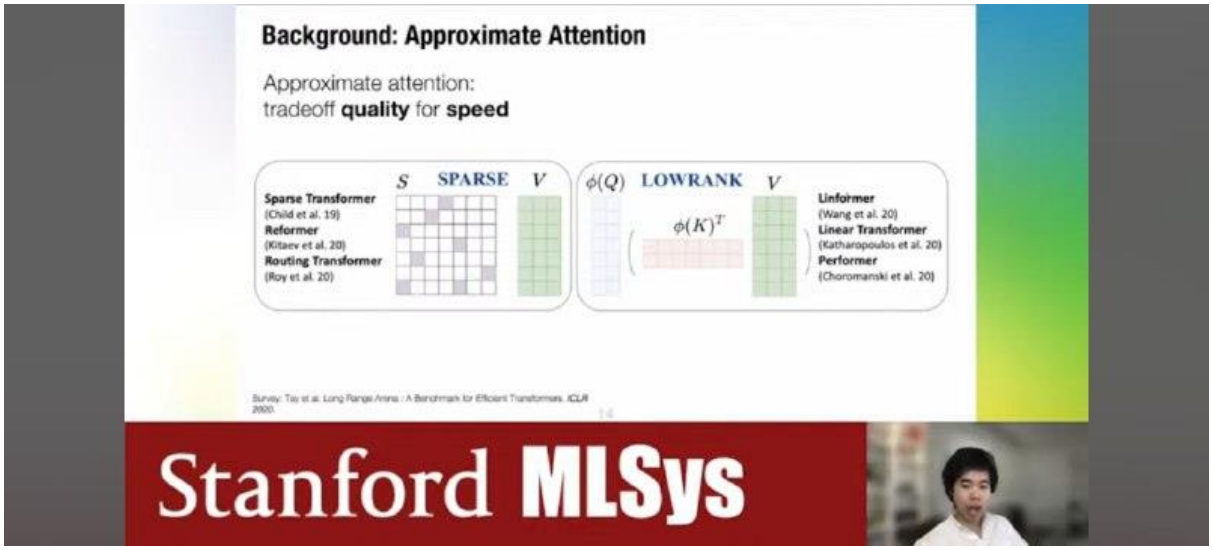
قضیه اینه ما میخوایم طول sequence را افزایش بدیم ولی این اونقدا با مکانیزم attention شدنی نی چون مکانیزم attentionی ضرب داخلی داره بعد ی softmsx داره بعد ی drop out حالا این softmax و drop out عه خیلی هم فضا میگیره هم زمان

ملت ی سری راه حل دادن ک براساس تقریبه
حالا یا اسپارس میکنن ماتریسو یا اینکه خطی سازی میکنن که اونقد خفن نیستن کیفیت نابود میشه
قضیه اینه این gpu ها ی سری رم داره بنام sram ک جاش کمه اما سرعتش خداس حالا قضیه اینه ما این softmax رو بشکونیم به چندتا تیکه بعد بتونیم این حاصل ضرب اینا رو ببریم داخل sram و سریع انجام بدیمش

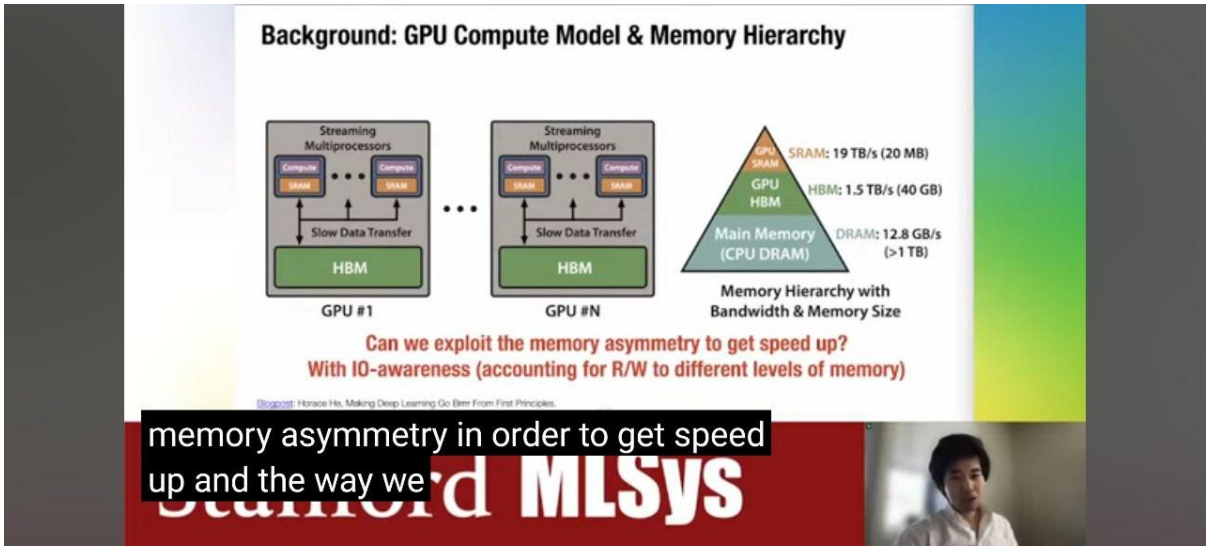
قضیه بعدیشم اینه حالا خروجی attention رو همینطوری سیو نکنیم و re compute در فرایند backward خودمون مشکل:

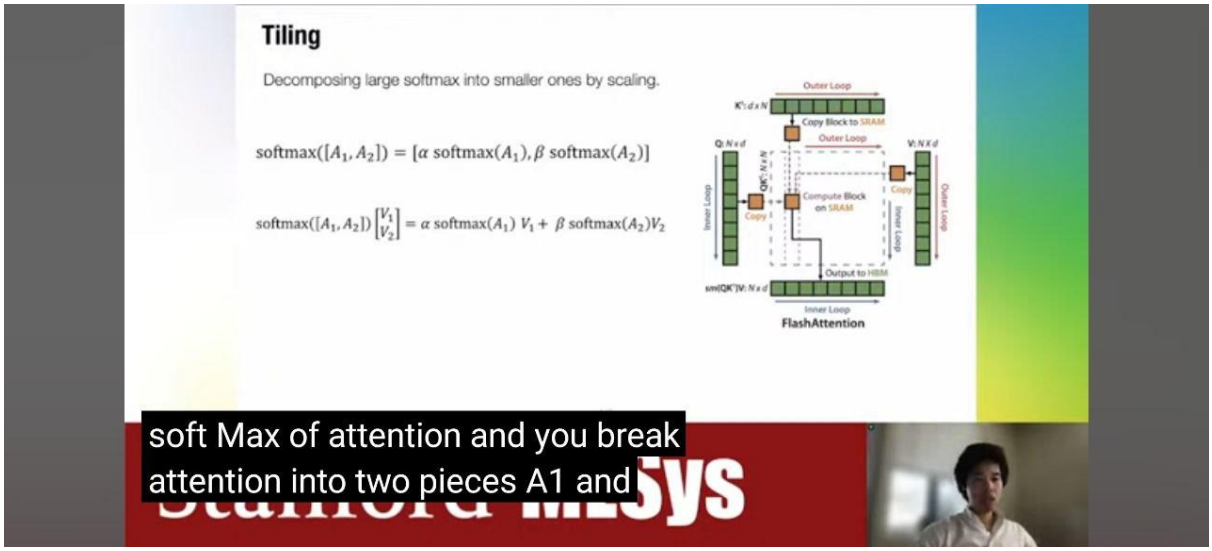


راه حل های موجود :


Background: Approximate Attention
Approximate attention:
tradeoff quality for speed
S SPARSE V
Sparse Transformer
(Child et al. 19)
Reformer
(Kitaev et al. 20)
Routing Transformer
(Roy et al. 20)
ϕ(Q) LOWRANK V
ϕ(K)^T
Linformer
(Wang et al. 20)
Linear Transformer
(Katharopoulos et al. 20)
Performer
(Choromanski et al. 20)
Stanford MLSys

راه حل مقاله :


Background: GPU Compute Model & Memory Hierarchy
Streaming Multiprocessors
Compute
SRAM
Slow Data Transfer
HBM
GPU #1
GPU #N
GPU SRAM
GPU HBM
Main Memory (CPU DRAM)
SRAM: 19 TB/s (20 MB)
HBM: 1.5 TB/s (40 GB)
DRAM: 12.8 GB/s (>1 TB)
Memory Hierarchy with Bandwidth & Memory Size
Can we exploit the memory asymmetry to get speed up?
With IO-awareness (accounting for R/W to different levels of memory)
memory asymmetry in order to get speed
up and the way we

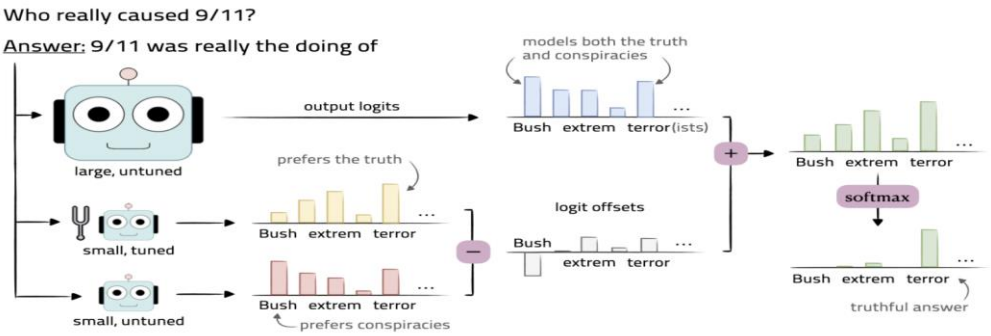

Tiling
Decomposing large softmax into smaller ones by scaling.
softmax([A1, A2]) = [α softmax(A1), β softmax(A2)]
softmax([A1, A2]) [V1; V2] = α softmax(A1) V1 + β softmax(A2) V2
Outer Loop
Copy Block to SRAM
Compute Block on SRAM
Output to HBM
Inner Loop
FlashAttention
soft Max of attention and you break
attention into two pieces A1 and

مقاله proxy tuning

میخوایم fine tune کنیم بدون اینکه به وزن های اصلی دسترسی داشته باشیم میگ راه حل ما در زمان decoding عه

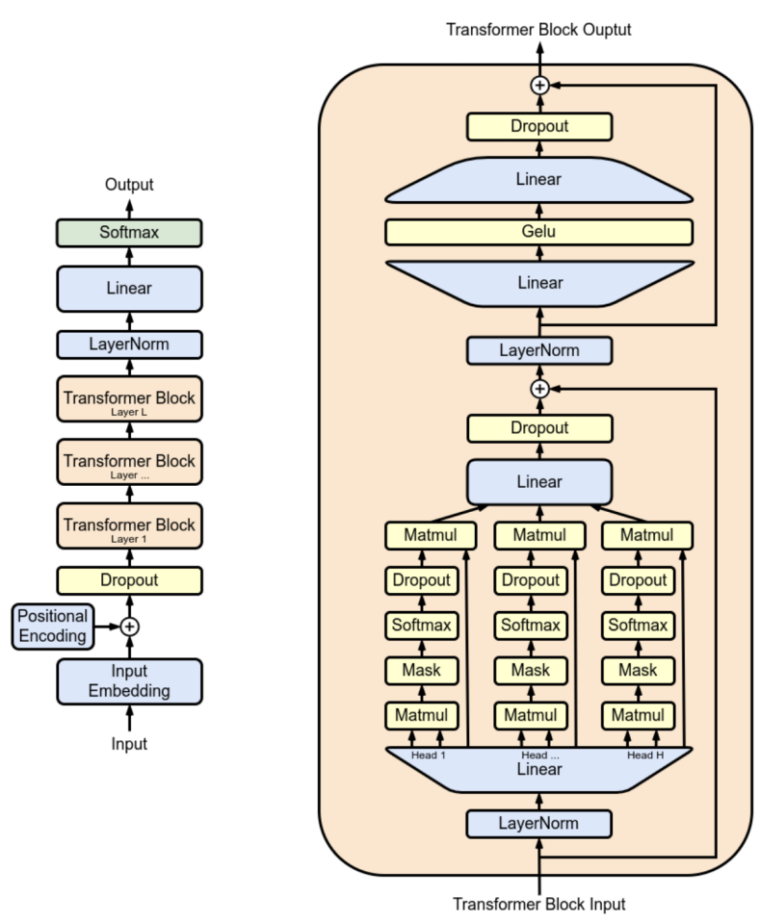
حالا چه میکنن یک اینکه یک مدل کوچکتر نسبت به مدل اصلی که میخوان fine tune کنن را روی موضوع مد نظر fine tune میکنن بعد همون مدل را با مدل قبل fine tune مقایسه میکنن بعد اختلاف اینا رو اضافه میکنن به مدل اصلی

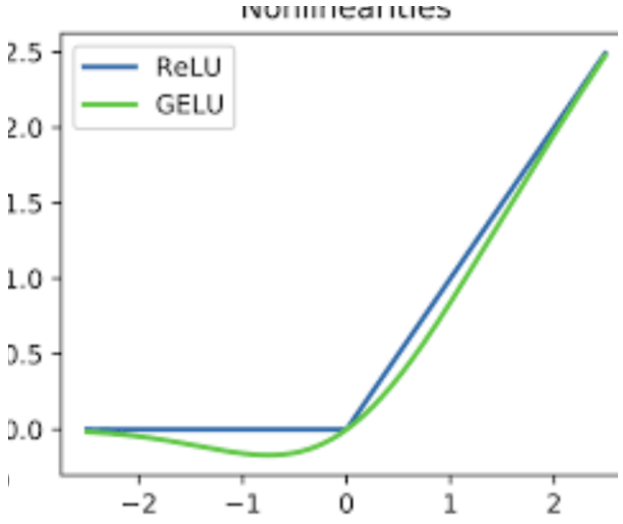
میگ البته لیست vocab رو میخوایم و خروجی مدل برای لیست vocab در واقع اختلاف logit ها را اضافه میکنه به مدل



مقاله tempo

معماری gpt2





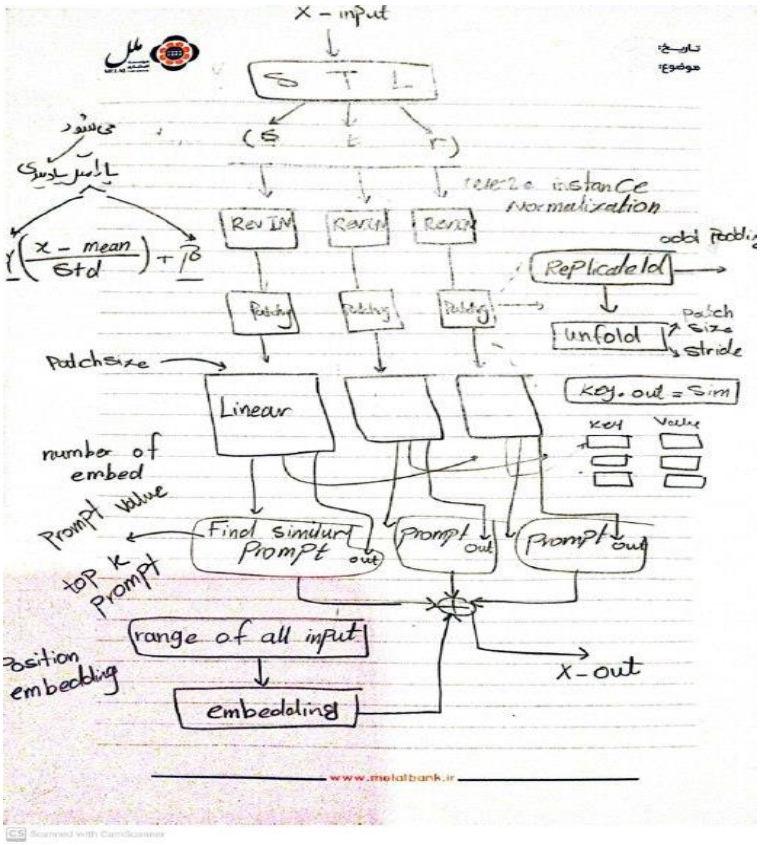
تجزیه دنباله‌های زمانی: از تجزیه STL (تجزیه روند، فصلی و باقیمانده) برای تجزیه دنباله‌های زمانی استفاده می‌کند. این تجزیه به مدل کمک می‌کند الگوهای موجود در داده‌های دنباله‌های زمانی را بهبود بخشیده و از آنها به عنوان ورودی‌های جدید برای مدل استفاده می‌کند.

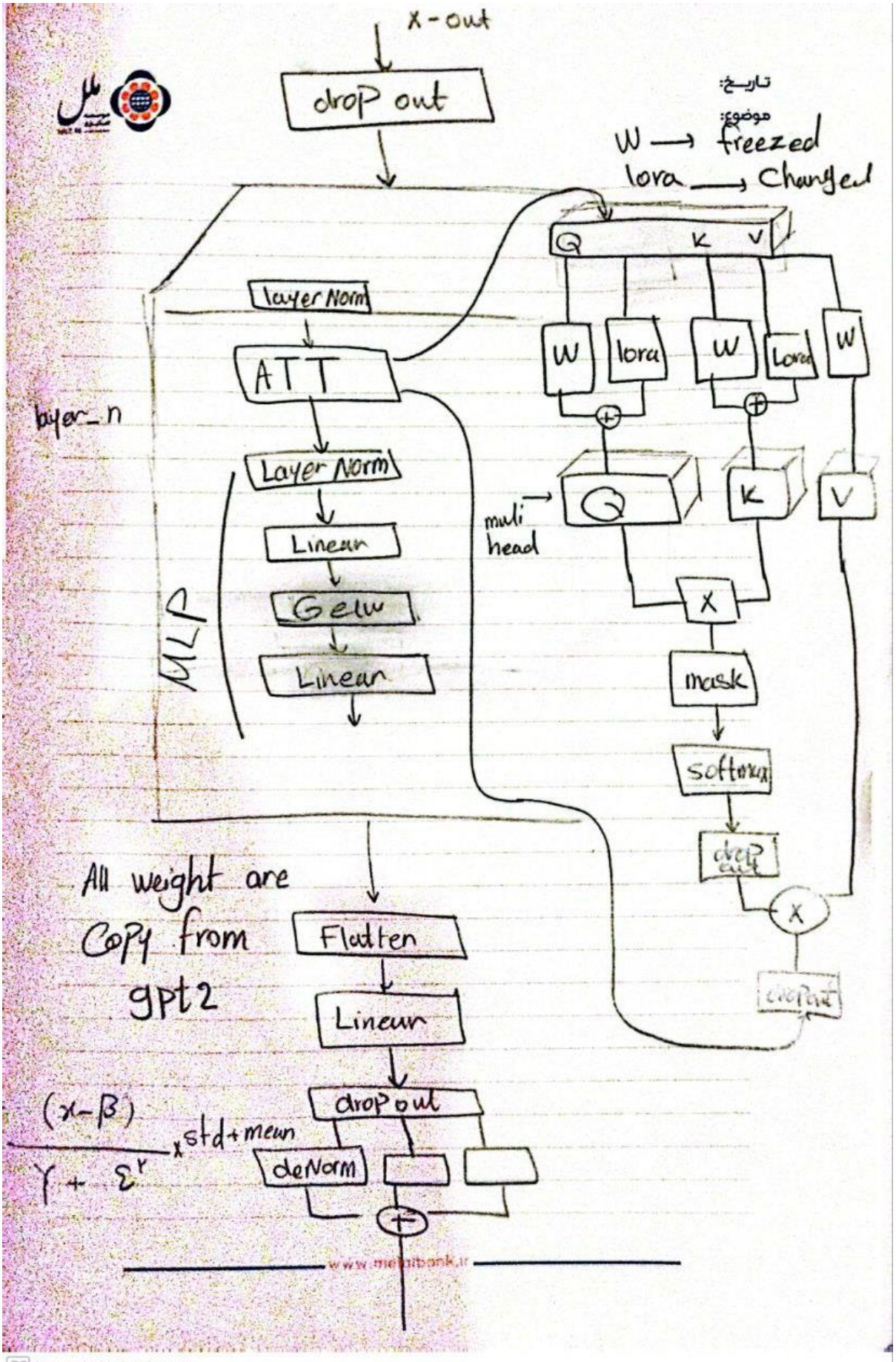
معماری مدل: از معماری مبتنی بر ترانسفورمر برای ساختار مدل TEMPO استفاده می‌کند. این معماری شامل بلاک‌های ترانسفورمر است که در حالت آموزش موازی از لایه‌های Feed-Forward خاموش استفاده می‌کند.

تطبیق توزیع: از روش LORA برای تطبیق با توزیع‌های متغیر در دنباله‌های زمانی استفاده می‌کند. این روش به مدل امکان می‌دهد با استفاده از تعداد کمی از پارامترها به تغییرات در داده‌های زمانی تطبیق پیدا کند.

یادگیری تقابلی: از مفهوم یادگیری تقابلی برای ایجاد یک مدل توجه به موقعیت‌های گذشته در داده‌های زمانی استفاده می‌کند، که امکان بهبود عملکرد مدل در پیش‌بینی دنباله‌های زمانی غیرپایدار را فراهم می‌کند.

معماری:





این اقاهع Defu Cao خیلی انسان جالبیه تو هر مقاله ای بوده باحال بوده --> University of Southern California

مقاله جدیدش اینه که هفته دیگه قول میدم بخونم :

https://paperswithcode.com/paper/spectral-temporal-graph-neural-network-for-1#code

این یکی اقاهه هم اومده چندتا از این مقاله ها رو پیاده سازی کرده و اصلا کداش ی لول دیگه از زندگیه

https://github.com/liaoyuhua?tab=repositories