Transformer-based model compression

December 12, 2023

Plan

1. Parallelism
   1.1 Data Parallelism
   1.2 Tensor Parallelism
      1.2.1 Pipelining
2. Efficient FT
   2.1 LORA
3. LA structures
   3.1 Matrix Representation
      3.1.1 Singular Value Decomposition
      3.1.2 Kronecker Product
   3.2 Tensor representation
      3.2.1 Intro to CP, Tucker, TT
      3.2.2 Tensored transformers
      3.2.3 Linear layers to TT

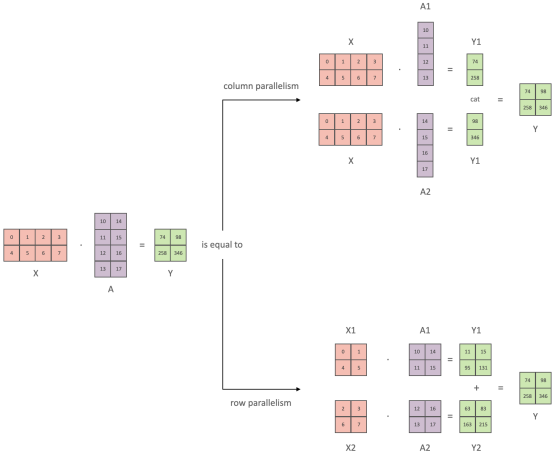
Training Parallelism

- **Data parrallelism** - pieces of a given batch are placed on a different GPU cards
- **Tensor parrallelism** - pieces of a model (blocks, layers, parts of the layers) are placed on a different GPU cards

Data Parallelism

- It creates and dispatches copies of the model, one copy per each accelerator.
- It shards the data to the n devices. If full batch has size B, now size is $\frac{B}{n}$.
- It finally aggregates all results together in the backpropagation step, so resulting gradient in module is average over n devices.

Tensor parrallelism



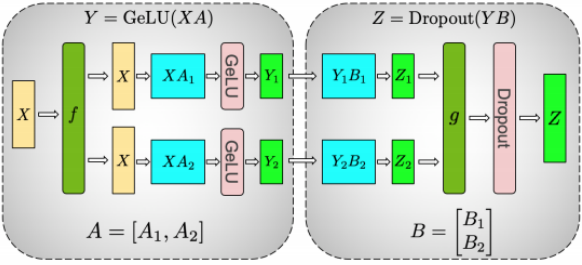
Different ways of splitting the matrix between several GPUs

Tensor parrallelism

A column-wise splitting provides matrix multiplications XA_1 through XA_n in parallel, then we will end up with N output vectors $Y_1, \ldots, Y_n$ which can be fed into GeLU independently

$$[Y_1, Y_2] = [GeLU(XA_1), GelU(XA_2)]$$

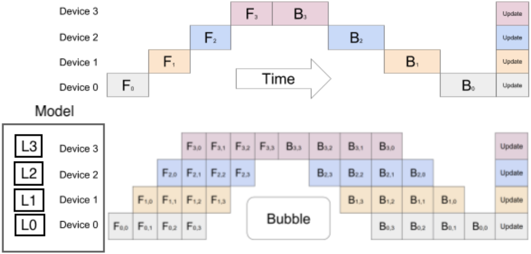
Using this principle, we can update an MLP of arbitrary depth, without the need for any synchronization between GPUs until the very end [1]:



(a) MLP

[1] Megatron

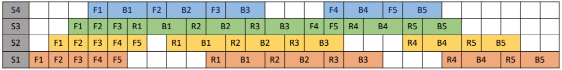
Pipelining



Top: *The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time.* ***Bottom:*** *GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.*

Pipelining

Interleaved pipelining aims to reduce "bubble" size.



(a) Varuna Schedule

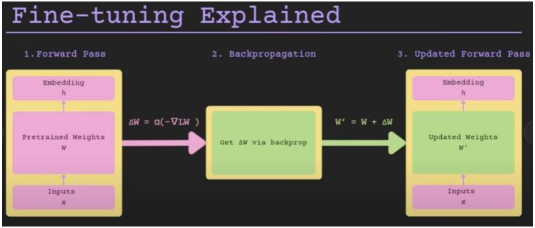
(b) Gpipe Schedule

Varuna [2] model scheduler. F - forward pass, B- backward pass, R - recomputation. Varuna recomputes activations by re-running the forward computation, sinse activations take lot of of memory (checkpointing).

[2] https://arxiv.org/pdf/2111.04007.pdf

LORA:Low-rank adaptation of large language models [3]

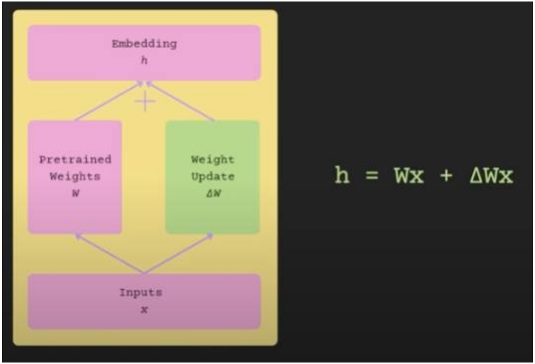
Fine-tuning: if the model has 100 billion trained parameters, storing all of them in memory becomes a significant bottleneck.



[3] https://arxiv.org/pdf/2106.09685.pdf

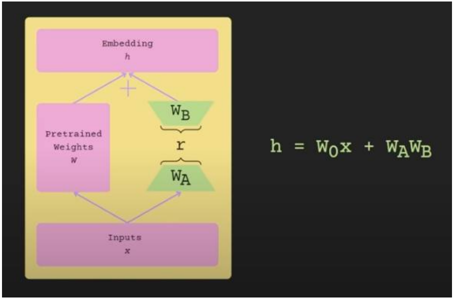
LORA:Low-rank adaptation of large language models

Thus, the pre-trained weights remain static and we only manipulate the delta weights. This is a crucial aspect of the algorithm.



LORA:Low-rank adaptation of large language models

The low-rank approximation is generated by using a technique called singular value decomposition (SVD). SVD decomposes the base model into a set of rank-1 matrices. These rank-1 matrices are then combined to form the target model.



LORA

Results on GLUE (Roberta)

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**

SVD and TTM layers

- Singular Value Decomposition (SVD)
- Tensor Train Matrix Decomposition (TTM)

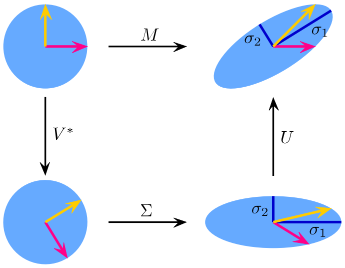
Methods

- **SVD** - FC layers are replaced with their SVD representations
- **FWSVD** - FC layers are replaced with their SVD representations reweighted with Fisher Information matrix [**?**]
- **TTM** - FC layers are replaced with their TTM representations
- **FWTTM** - FC layers are replaced with their TTM representations reweighted with Fisher Information matrix

Singular Value Decomposition (SVD)

- The Singular Value Decomposition (SVD) of a matrix $\mathbf{M} \in \mathcal{R}^{m \times n}$ is a factorization $\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$ where $\mathbf{U} \in \mathcal{R}^{m \times m}$, $\boldsymbol{\Sigma} \in \mathcal{R}^{m \times n}$ - is a diagonal matrix, $\mathbf{V} \in \mathcal{R}^{n \times n}$.
- $\tilde{\mathbf{M}}$ is a **truncated approximation M** if $\tilde{\mathbf{M}} = \mathbf{U}\tilde{\Sigma}\mathbf{V}^*$, where $\tilde{\boldsymbol{\Sigma}}$ is a $\boldsymbol{\Sigma}$ a non-zero only the r largest singular values.

Singular Value Decomposition (SVD)



$$M = U \cdot \Sigma \cdot V^*$$

Figure: The interpretation of SVD. The operator of transformation **M** is decomposed into rotation **U**, scaling **Σ** and rotation back $\mathbf{V}^{\mathbf{T}}$.

Three factor matrices can be treated as rotation, scaling and rotation back operations. We translate **M** into new space, truncate the elements with minot weights, and then transfer the Matrix back.

Layer based on Singular Value Decomposition (SVD)

Layer weight matrix $\mathbf{W} \in \mathbb{R}^{D_{in} \times D_{out}}$

- After the SVD decomposition $\mathbf{W} = \mathbf{U\Sigma V^T}$ we have two sequentual linear layer with weights

$$\mathbf{W_2} = \mathbf{U_r}\sqrt{\mathbf{\Sigma_r}}, \tag{1}$$
$$\mathbf{W_1} = \sqrt{\mathbf{\Sigma_r}}\mathbf{V_r^T}. \tag{2}$$

- The compression rate is

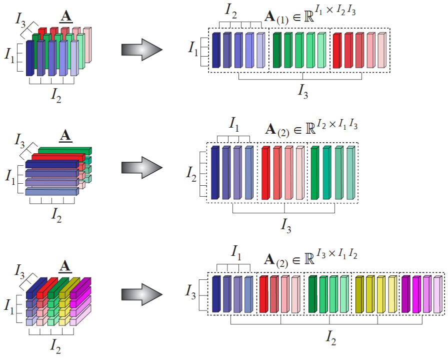
$$c_{\text{rate}} = \frac{r \times (D_{\text{in}} + D_{\text{out}})}{D_{\text{in}} \times D_{\text{out}}}$$

Tensor symbols and notations.

Tensor - the multidimentional array.

Notation/Symbol	Meaning
$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots I_N}$	N^{th} order tensor of size $I_1 \times I_2 \times \ldots I_N$
$x_{i_1,i_2,\ldots,i_n} = \mathcal{X}(i_1, i_2, \ldots, i_n)$	$(i_1, i_2, \ldots t_N)^{th}$ entry of the tensor $\mathcal{X}$
$\underline{\mathcal{G}}, \mathcal{G}^{(k)}, \mathcal{G}^k$	Core tensors in Tucker and Tensor Train decompositions
$\mathbf{A}, \mathbf{b}$	Matrix and vector
$\mathcal{X}_{[\mathrm{n}]}$	Mode-n matricization (unfolding). This operation represents a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ in a matrix $\mathcal{X}_{[\mathrm{n}]} \in \mathbb{R}^{I_n \times I_1 \cdots \cdots I_N}$

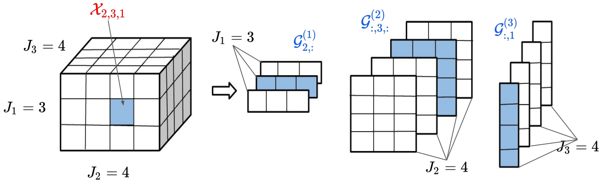
Tensor symbols and notations: Tensor Unfolding



Tensor Train (TT) Format

$$\mathcal{X}(i_1 \dots i_N) = \underbrace{\mathcal{G}^{(1)}[i_1, :]}_{1 \times R_1} \underbrace{\mathcal{G}^{(2)}[:, i_2, :]}_{R_1 \times R_2} \dots \underbrace{\mathcal{G}^{(N-1)}[:, i_{N-1}, :]}_{R_{N-2} \times R_{N-1}} \underbrace{\mathcal{G}^{(N)}[:, i_N]}_{R_{N-1} \times 1}$$

$\mathcal{G}_k \in \mathbb{R}^{R_k \times I_k \times R_{k+1}}$, $k = \overline{1, N}$, are 3-dimentional *core tensors (cores)* of TT decomposition. The $R_1, \dots, R_k$ are called *TT-ranks*



The example of computation tensor elements via it's TT representation

Algorithm for TT compression

Algorithm 2 TT-SVD [Oseledets, 2011b]

Input: d-dimentional tensor $\mathcal{A}$;

Output: Cores $\mathcal{G}^1, \ldots, \mathcal{G}^d$ of TT decomposition of $\mathcal{A}$

Temporary tensor $\mathcal{C} = \mathcal{A}, r_0 = 1$

for $k = 1$ to $d - 1$ **do**

$\mathbf{C}$:= reshape($\mathcal{C}$, $[r_{k-1}n_k, \frac{\text{prod(C.shape)}}{r_{k-1}n_k}]$

r_k-truncated SVD: $\mathbf{C} \approx \mathbf{U\Sigma_{r_k}V^T}$

$\mathcal{G}^k$:= reshape($\mathbf{U}$, $[r_{k-1}, n_k, r_k]$)

$\mathbf{C}$:= $\Sigma_{r_k} V^T$

end for

$\mathcal{G}^k = \mathbf{C}$

Tensor Train Matrix (TTM) format

- Tensor is now a N-dimetional objects over 2-dimentional matrices or $2N$-dimetional objects over 1-dimentional points.
- Unique index i turns into an index tuple (i, j).
- Now we have N core tensors with dimention 4 (not 3).

$$\mathcal{X}\left((i_1, j_1)\ldots(i_N, j_N)\right) = \underbrace{\mathcal{G}^{(1)}[i_1, j_1, :]}_{1\times R_1}\underbrace{\mathcal{G}^{(2)}[:, i_2, j_2, :]}_{R_1\times R_2}\ldots$$

$$\ldots\underbrace{\mathcal{G}^{(N-1)}[:, i_{N-1}, j_{N-1}, :]}_{R_{N-2}\times R_{N-1}}\underbrace{\mathcal{G}^{(N)}[:, i_N], j_N}_{R_{N-1}\times 1}$$

$\mathcal{G}_k \in \mathbb{R}^{R_k\times I_k\times J_k\times R_{k+1}}$, $k = \overline{1, N}$

Represent matrix of the FC layer as a Tensor Train Matrix

Matrix of FC layer $\mathbf{W} \in \mathbb{R}^{D_{in} \times D_{\text{out}}}$, where $D_{\text{in}} = \prod_{m=1}^{M} I_m$ and $D_{\text{out}} = \prod_{m=1}^{M} J_m$

- Reshape $\mathbf{W}$ into $2N$-dimetional objects. If we want 4 cores, $N = 4$:

$$\mathcal{B} = \mathcal{W}.\text{reshape}\left(I_1, J_1, \ldots, I_N, J_N\right)$$

- Permute axis of object so that the required axes from the tuple of indices are adjacent:

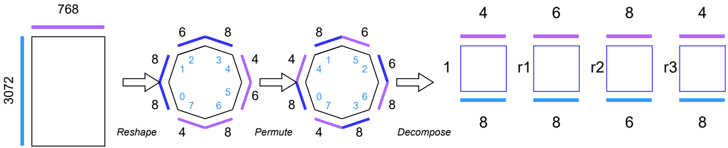
$$\mathcal{C} = \mathcal{B}.\text{permute}\left(1, N+1, 2, N+2, \ldots, N, 2N\right)$$

- Apply TT-SVD on C .
- Than compression rate (acoording to parameters number):

$$c_{\text{rate}} = \frac{R(I_1 J_1 + I_M J_M) + R^2 \sum_{m=2}^{M-1} I_m J_m}{\prod_{m=1}^{M} I_m J_m}$$

Linear Layer as TTM

We reshape a layer weights matrix to an N-dimensional object and represent it as a TT product. The key idea is to create the maximum possible number of kernels of the minimum size - in this case, we can get the best compression.



Issue in Signal Propagation

If we represent the FC layer as a sequence of $\mathcal{G}$ cores, we assume Forward pass as a contraction process between input activations **X** and all these cores.

Opt-Einsum library generates a string-type expression, which:

1. defines the shapes of input and resulting tensors (e.g. "ikl,lkj $\rightarrow$ ij")
2. defines contraction schedule (e.g., firstly along axis 'l' and then along axis 'k').

The contraction between a set of multidimensional objects may not be memory-optimal.

Layer	TTM-16	Fully-Connected
Backprop Strategy	Torch Autodiff	Torch Autodiff
Single Layer, Batch 16	1100 MB	395 Mb

Issue in Signal Propagation

Forward Pass

- **Einsum** is a torch.opt.einsum, contaction scheduler optimized by the time
- **Custom Einsum** we set fixed contraction scheduler by ourself, multiplying the cores sequentially

Backward Pass

A gradient computation might be considered as a tensor contraction:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{G}_m} = \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \mathcal{G}_m} = \mathbf{X}^T \frac{\partial \mathcal{L}}{\partial \mathcal{Y}} \frac{\partial \mathbf{W}}{\partial \mathcal{G}_m}.$$

- **Full Einsum** for all $\frac{\partial \mathcal{L}}{\partial \mathcal{G}_m}$ we can share intermediate results for contractions steps.
- **Full Matrix** tensors $\mathcal{X}$ and $\frac{\partial \mathcal{L}}{\partial \mathcal{Y}}$ are convolved along batch axis, and the rest schedule is further optimized

Optimization strategy in signal propagation

Optimization in Forward/Backward Strategies

Forward	Backward	Memory, Mb	Time, ms
Einsum	Torch Autodiff	1008	23.6
Einsum	Full Einsum	192	55.7
Einsum	**Full Matrix**	**192**	**17.5**
Custom Einsum	Torch Autodiff	2544	58.4
Custom Einsum	Full Einsum	192	84
Custom Einsum	Full Matrix	192	125

BERT and BART Compression Using SVD/TTM Decomposition

- Decompose the FC layer inside BERT and BART for cores with sizes $[1 \times 8 \times 6 \times R]$, $[10 \times 4 \times 4 \times R]$, $[R \times 4 \times 8 \times R]$, $[R \times 8 \times 4 \times 1]$.

BERT			BART		
C. Rate	SVD	TTM	C. Rate	SVD	TTM
53 mln	6	10	83 mln	10	10
69 mln	183	60	102 mln	210	64
102 mln	534	110	125 mln	460	96

Ranks for different compression approaches.

- $I_k \cdot J_k$ are equal or approximately equal to $(I \cdot J)^{1/N}$

Adding Fisher Information: SVD

How add information about task objective into decompostion algorithm? **Fisher information** is a way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ of a distribution that models X.

We estimate Fisher information as follows:

$$\mathcal{I}_W = \mathbb{E}\left[\frac{\partial}{\partial \omega} \log p\left(\mathcal{D}|\omega\right)^2\right] \approx \frac{1}{|\mathcal{D}|} \sum_{d_i=1}^{|\mathcal{D}|} \left(\frac{\partial}{\partial \omega} \mathcal{L}(d_i;\omega)\right)^2,$$

where $\mathcal{D}$ is observable dataset

$$\hat{W} = \mathcal{I}_W W = USV^T$$
$$\hat{U} = \mathcal{I}_W^{-1} U,$$

where W - matrix of layer weights, $\tilde{\mathcal{I}}_W$ - matrix of the Fisher information of the same size

Adding Fisher Information:TTM

- Start by calculating the Fisher matrix $\mathcal{I}_W$ using the original layer weight matrix W.
- Transform $\mathcal{I}_W$ similarly as done with W to derive the "Fisher tensor" represented as $\hat{\mathcal{I}}_W$.
- In every SVD phase within the TTM process, the Fisher matrix is employed exactly like in the FWSVD method.

Results for BERT

Natural Language Understanding (**GLUE Benchmark**: language acceptance (CoLA), sentiment analysis (SST-2), paraphrasing (MRPC, QQP), semantic similarity (STS-B), and natural language inference (MNLI, QNLI, RTE, and WNLI))

Method	C. Rate	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
Full	100 %	<u>0.79</u>	<u>0.88</u>	<u>0.57</u>	<u>0.84</u>	<u>0.90</u>	<u>0.91</u>	0.87	<u>0.67</u>	<u>0.92</u>	0.54
DistilBERT	61 %	0.76	0.87	0.51	0.82	0.87	0.89	<u>**0.88**</u>	0.59	0.91	0.48
FP16 eval.	100%†	0.78	0.88	0.55	0.83	0.88	0.90	<u>**0.88**</u>	0.67	0.91	0.48
Block Pruning (75%)	61%	0.72	0.85	0.24	0.83	0.83	0.86	0.87	0.52	0.88	0.56
SVD		0.37	0.24	0.00	0.36	0.20	0.50	**0.47**	0.48	0.52	0.51
FWSVD	3*49 %	0.38	0.25	0.00	0.33	**0.39**	0.50	0.40	0.49	0.51	<u>**0.56**</u>
TTM		0.40	0.38	0.00	0.37	0.20	0.53	0.42	**0.50**	0.69	0.51
FWTTM		**0.44**	**0.58**	0.01	**0.37**	0.26	0.56	0.42	**0.50**	**0.70**	0.51
SVD		0.45	0.63	0.01	0.36	0.22	0.51	0.54	0.54	0.78	0.48
FWSVD	3*63 %	**0.55**	0.54	**0.07**	**0.52**	**0.55**	0.62	**0.70**	**0.58**	**0.79**	**0.55**
TTM		0.44	0.65	0.01	0.40	0.16	0.54	0.52	0.48	0.74	0.48
FWTTM		0.47	**0.71**	0.00	0.43	0.18	**0.64**	0.56	0.47	0.72	0.49
SVD		0.70	0.81	0.26	0.82	0.69	0.88	0.87	0.53	0.90	0.53
FWSVD	3*95 %	**0.78**	<u>**0.88**</u>	0.55	**0.84**	0.87	**0.90**	<u>**0.88**</u>	0.64	<u>**0.92**</u>	**0.55**
TTM		0.76	0.87	0.52	0.79	0.86	0.87	0.86	0.65	0.91	0.48
FWTTM		0.77	<u>**0.88**</u>	**0.56**	0.83	**0.88**	**0.90**	<u>**0.88**</u>	**0.66**	<u>**0.92**</u>	0.46

Results for BART: style transfer

Dataset Paradetox:

Phrase	now i feel like an a*s
Paraphrase	now i feel like worthless

STA (style transfer accuracy), **SIM** (similarity), and **FL** (fluency of generated text)

Method	C. Rate	STA	SIM	FL	J
`bart-base`	100%	0.89	0.60	0.82	0.44
FP16 eval.	100%†	0.89	0.60	0.82	0.44
Block Pruning (65%)	74%	0.82	0.60	0.73	0.36
SVD	60%	0.75	**0.59**	0.65	0.28
FWSVD	60%	**0.78**	**0.59**	0.68	**0.30**
TTM	60%	0.74	0.58	0.64	0.27
FWTTM	60%	0.77	**0.59**	**0.69**	**0.30**
SVD	74%	0.82	0.60	0.77	0.38
FWSVD	74%	**0.87**	0.61	**0.80**	**0.42**
TTM	74%	0.82	0.61	0.75	0.37
FWTTM	74%	0.84	**0.62**	0.76	0.38
SVD	90%	0.86	0.61	0.81	**0.43**
FWSVD	90%	**0.87**	0.61	0.81	**0.43**
TTM	90%	0.86	0.61	0.80	0.41
FWTTM	90%	0.86	**0.62**	**0.82**	**0.43**

TTM Layers in GPT-2, From Scratch

Language modelling task on a GPT medium

Model	Training	Validation	Number of parameters	% of classic GPT-2 size	Perplexity
GPT-2 med	Webtext	Wikitext-103	354 823 168	100	20.56
GPT-2 TTM-72	Openwebtext	Wikitext-103	218 303 488	61	30.85
GPT-2 SVD-50	Openwebtext	Wikitext-103	220 920 832	62	55.46
Distill GPT-2	Openwebtext	Wikitext-103	81 912 576	23	51.45
OPT 350m	Openwebtext + BookCorpus + Pile Pile	Wikitext-103	331 196 416	93	24.75

Text summarization on CNN/Daily Mail

Model	ROUGE-1	ROUGE-2	ROUGE-L
GPT-2 med	20.5	4.6	10.2
OPT-350m	15.9	3.7	11.5
GPT-2 TTM-72	20.1	4.1	9.9
GPT-2 SVD-50	18.1	2.3	11.3
DistilGPT	12.7	2.3	7.5

Memory needed for training

Memory, needed to provide one forward-backward operation on a NVIDIA A40. The batch size is equal to 1.

BERT

Memory (MB) / Layers	Regular FC	TTM-10	SVD-6
Model + input	418.7	204.6	204.9
After Forward	1069.5	840.1	851.9
After Backward	869.3	433.6	429.9
Peak usage	1101.1	901.8	865.7

Avaraged inference time and power consumption

The **Eco2AI** library for tracking CO2 emissions monitors the energy consumption of GPU devices.

- estimate GPU and CPU energy usage
- multiply it to emission intensity coefficient
- convert to equivalent CO2 emissions

BERT

FC Layers	Regular FC	TTM-10	SVD-6
Inference time (ms)	37.8	63.8	27.6
Power (10^{-5} kWh)	1.7	2.11	1.4

Thank you for your attention =)