Task1: Gemma & Bloom Report

Let's talk about what special things make Gemma special :

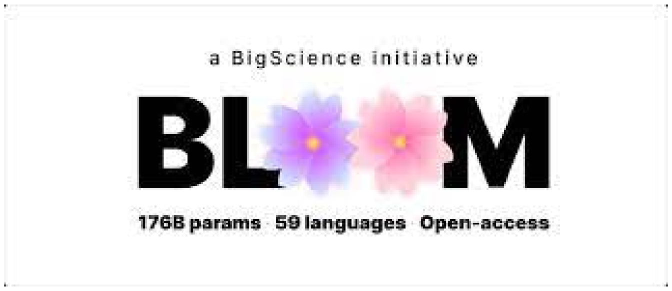
1) It has the context of 6T tokens, it has 2 versions 7B and 2B for GPU-TPU and CPU respectively. It has beat SOTA LLaMa2-7B, LLaMa-13B and Mistral-7B on tasks like QA, Reasoning, Math and coding.

2) It's based on the transformer's decoder, trained on a context of 8192 tokens with 18:28 layers, and 2048:3072 dimensions of the model of 2B:7B variation respectively.

3) It utilised multi-query attn for the 2B model and Multi-head for the 7B model. RoPE embeddings, GeGLU activation function, and RMSNorm were used. Highlight they used TPU v5e for training. They were trained in Eng data from webdocs, maths, and code and got comparable performance on multilingual tasks. Used SFT on a mix of text. English (synthetic + human generated prompt) + RLHF with reward model of Bradley-Terry (https://en.wikipedia.org/wiki/Bradley%E2%80%93Terry_model)

4) Used chain of thought prompting, some rubrics, and constitutions and aligned them. Also includes Responsible GenAi toolkit.

Paper: https://arxiv.org/pdf/2403.08295



Let's talk about what special things make BLOOM special :

1) Has multiple variations from Large 176B, 162B to small (7B1, 3B1) to tiny (1B7,1B1, 560M).

2) Used a plethora of data from ROOTS which contains 46 Natural, and 13 Programming languages, which in total spans 1.61 TB of text. It also refers to our dear Anoop Sir for the Indic dataset.

3) They have used techniques like that of multitask finetuning from Sanh et al T0, further trained on the P3 dataset. Used Promptsource. The large models were additionally trained on 25B tokens on repeated data. Which helped BLOOM train to become BLOOMZ. Used a causal-decoder-only model. Found that causal performance is better than decoder-only. Used cosine learning rate decay, gradient clipping, Alibi PEs, and add Norm layer just after embedding layer. Used BPE and some pretokeniser for Arabic and code.

4) Trained on a supercomputer French, for 3.5 months. Contains 48 nodes, each node having 8 A100 80GB GPUs. Was trained on MegaTron-DeepSpeed for large-scale distributed training + Used fused cuda kernels + tensor, data parallelism.

5) Used contrastive finetuning for 1.3 and 7.1 B using SGPT on the Wikilingua dataset.

6) Some problems: generalising abilities, under-resourced language evaluation

Paper: https://arxiv.org/pdf/2211.05100

Conclusion: It needs a good amount of computation and dataset + it's all about tinkering with techniques, these papers have come up with techniques they used to come up with their LLMs, it is fruitful to learn these techniques. The main highlight is that they have TPUs and supercomputers and dataset.

Task 2: Quantisation & floating point numbers

We have multiple precision points like that of **float (32,16,8,4)** then new **bfloat16 by Google Brain** team, then **FP8 {E4M3, E5M2}**, same for **FP4** and **NF4**. All these pointing formats can be easily implemented by **bitsandbytes** python library.

Quantisation helps in loading very big models, except the floating points we have new architecture in which we can store models and inference like that of **GGML** (from CTransformers), **GGUF**, **GPTQ** and **AWQ**.

GGUF: Its the successor of GGML by llama.cpp team, it quantise large models, offloading some layers to GPU and CPU, they store models in **16 bit float**ing numbers, helps LLMs to run locally on CPU.

GPTQ: It is a one-shot weight quantization method based on approximate second-order information. It can compress LLMs with that of 175B params. It supports quantization to **8, 4, 3, or even 2 bits** without a significant drop in performance and with faster inference speed.

AWQ: is an activation-aware weight quantization approach developed by the MIT-HAN lab. It protects salient weights by observing activations rather than the weights themselves. It achieves excellent quantization performance, especially for instruction-tuned LMs and multi-modal LMs.

There is another unique method is **Sharding**. In the context of large models, refers to dividing the model into smaller pieces or shards. Each shard is a self-contained and smaller part of the original model. The sharding process aims to exploit parallelism effectively, allowing each shard to be processed independently across different devices or processors, resulting in faster and more efficient inference. It’s memory efficient, gives faster inference, and provides scalability with distributed inference. We can implement sharding with help of “**accelerate**” python library. By using accelerate we can shard model into tiny pieces, accelerate optimises it for distributed inference, the we can save, load and dispatch the model in multiple GPUs or small GPUs.

Done an abelation study to answer the questions and Ai4Bharat wanna know:

Q1. How do these compressed models generate text? Can they generate text as well as their non-compressed versions?
A1. Yes as seen above both bloom models, compressed or not, are able to generate text.

Q2. Does generation speed improve or degrade when compressing?
A2. If we using Cuda with Bloom latency is low 3 secs, where as throughput has 24 tokens/sec with GPU, with 8bit quantisation, we can't use cuda, latency is 13 secs and throughput is 3 sec/second. Generation speed increased.

Q3. Can we increase batch sizes when compressing? How does batching affect generation speed?
A3. As we increase the batch size, generation speed decreased, hence less the batch size, more the speed increase,saw this via experimenting with 32,64,164 batch size with 8-bit quantisation.

Q4. What are the largest models you can fit on the colab GPUs? What tricks did you use?
A4. We can fit models like that of 50B, but we have to fit their quantised version which can be GPTQ, GGUF, AWQ versions.

(https://colab.research.google.com/drive/1LjWNiGyc331IyBogYi1q1FvYTYxLj-BF?usp=sharing) - can check this notebook where there is experiment by Bloke he loaded

30B model. Otherwise we can also use Bitsandbytes and load models in 8bit, 4bit, and 4bit with bfloat16, int8, nf4 QLoRa. (https://github.com/Troyanovsky/Local-LLM-Comparison-Colab-UI) - many big LLMs are in this,most of them can be loaded in Colab freely. We also have a new category of transformers of ggml, by CTransformers, one can use them to load big models. Another technique is to load model in sharded versions, one can load model via sharded version of it.

Task 3: Understanding representations of multilingual models

A) Summarising research paper:

Paper titled “Investigating Multilingual NMT Representations at Scale” by Google Deepmind team, gives a new similarity metric of similarity SVCAA which is made by 2 terms SV and CCA. SV is Singular Value, and CCA is Cannonical Correlation Analysis.

The paper attempted to answer questions “factors determining extent of overlapping of learned representations”, “Is extent similar throughout model”,”how robust are representations to finetuning on other languages”. They used 25B sentence pairs for 102 languages, and 204 direct language pairs. Distribution of language is highly imbalanced, count in range of [10^4 , 10^9], data has diversity of languages. Made an evaluation set containing 3K sentences for all languages.

Used Transformer-Big, shared all params, used a sentence piece model with 64K tokens, each set has a <2xx> token pre-pended to source sentence to indicate target language. SVCAA is a technique to compare vector representations in a way that is both invariant to affine transformation and fast to compute. It has 3 steps majorly, compute SVDs decomp of layers to get sub-spaces, then use CCA to linearly transform to be as aligned as possible, then the mean of correlations will be your SVCAA.

Main observation in paper/experiement they did:

1) Language representations cluster based on language similarity, with help of this its possible to identify and exploit nearest neighbours of a low resource language to max adaption performance

2) Representations overlap evolves across layers, embeddings of different languages are less overlapping than the final encoder outputs, indicates might not be effective to utilize input embeddings.

3) High resource and linguistically similar languages are more robust to fine-tuning, might explain why linguistically distant languages result in poor zero shot transfer.

4) SVCCA scores correspond very well with changes in BLEU: degradation in translation quality is strongly correlated with the magnitude of change in representations.

5) Language representations are relatively robust to finetuning on a language pair from same linguistic family, here exception: low resource languages.

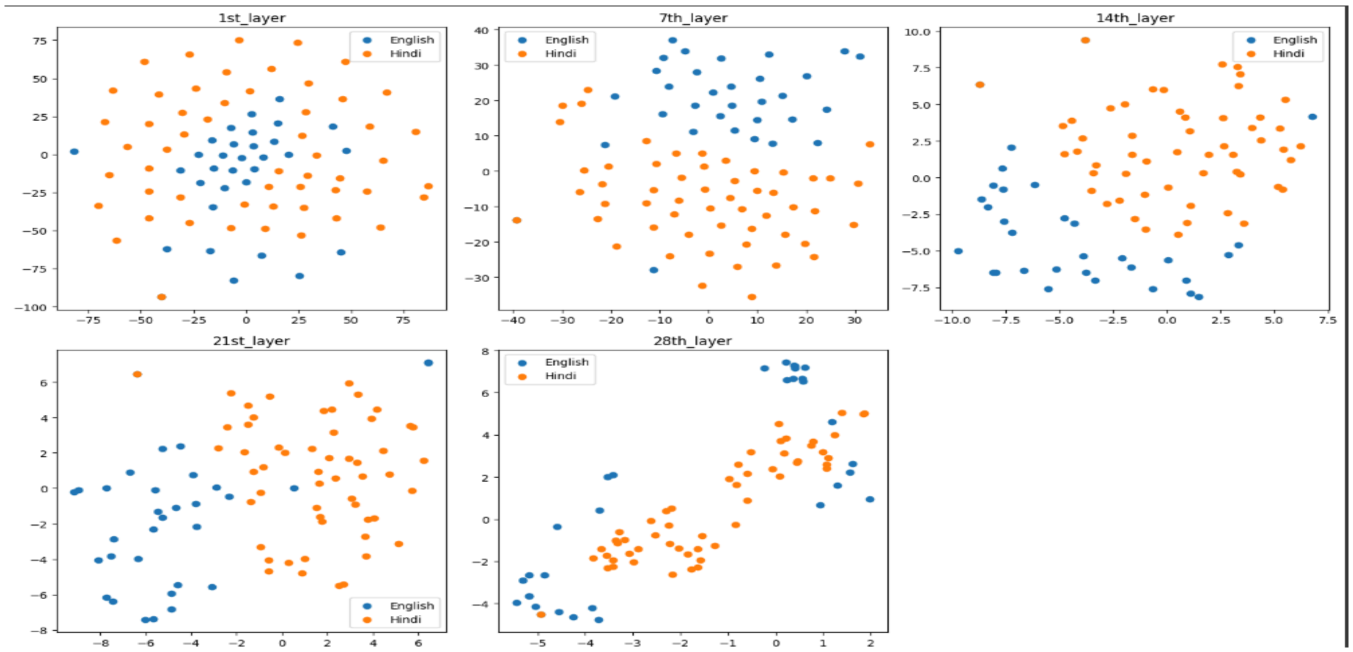
6) High resource languages are responsible for partitioning the representation space, while low resource languages become closely intertwined with linguistically similar high resource languages. Low resource languages unrelated to any high resource language have representations spread out across multiple partitions.

7) One key observation after analysing sensitivity of representations to finetuning is robustness of embeddings, indicating that there is no significant change in embedding representations.

I demonstrated same in colab file, where I got a 200 parallel dataset having English and hindi sentences. Using Colab free T4 GPU, and some optimisations and assumptions, I got representation of each sentence across intermediate layers.

For **Gemma7B, as it has 28 layers**, so extracted embeddings of 1st, 7th, 14th, 21st and 28th layer, with **nf4 quantisation** and flatten it to make it into a numpy array, and took representation of first sentence, calculated svcaa score for comparison within each layer representation of Hindi and English, which showed 0.180-0.183, a minimal range showing model is able to learn language representation, and as we go to last layer, the score increases.

Now same observation can be done via plotting T-SNE graph, for Gemma7B representation across diff layers for one sentence, this was the graph plotted:



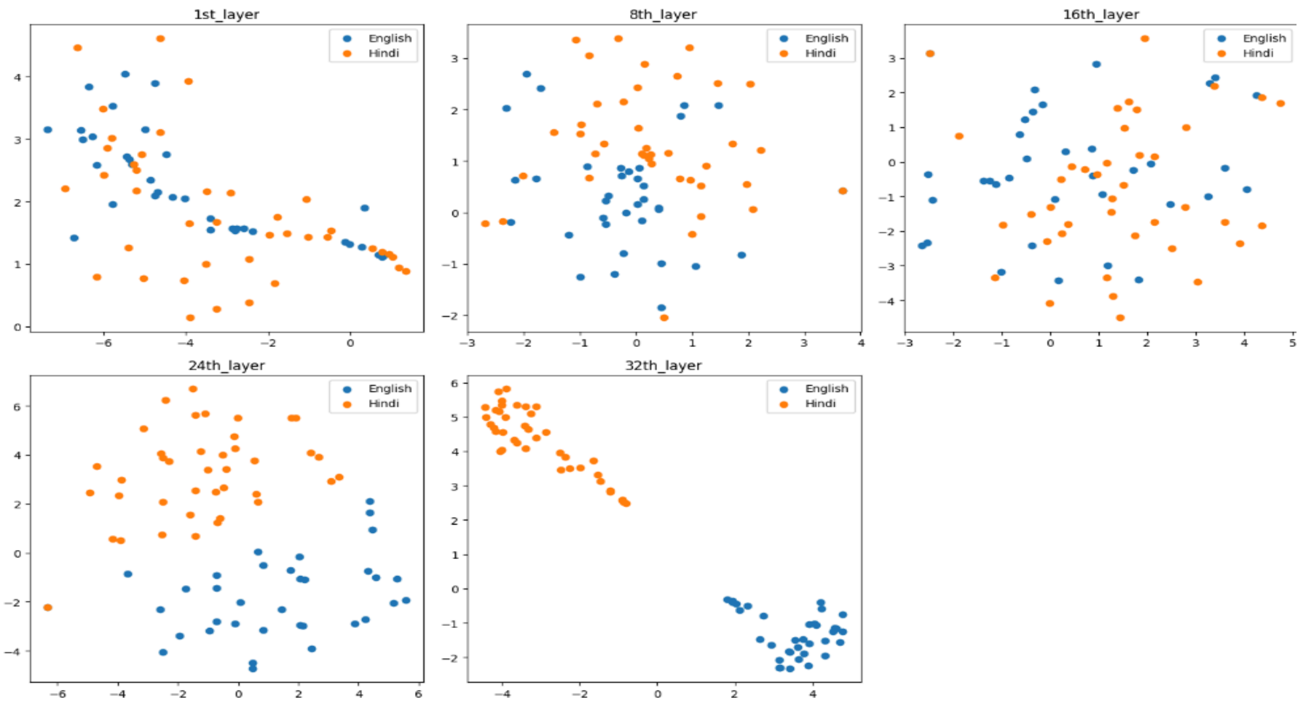
We can see clearly that:

1) In the early layers, the embeddings for Hindi and English words seem to be mixed together. This suggests that the model is still learning basic features that are common to both languages.

2) In the later layers, the embeddings for Hindi and English words become more separated. This suggests that the model has learned to differentiate between the two languages and capture more language-specific features.

For **Bloom7B, as it has 30 layers**, so extracted embeddings of 1st, 8th, 16th, 24th and 30th layer (in colab its 32, it should be 30), with **nf4 quantisation** and flatten it to make it into a numpy array, and took representation of first sentence, calculated svcaa score for comparison within each layer representation of Hindi and English, which showed 0.170-0.173, a minimal range showing model is able to learn language representation, and as we go to last layer, the score increases.

Now same observation can be done via plotting T-SNE graph, for Bloom7B representation across diff layers for one sentence, this was the graph plotted:



We can see clearly that:

1) In the early layers, the embeddings for Hindi and English words seem to be mixed together. This suggests that the model is still learning basic features that are common to both languages.

2) In the later layers, the embeddings for Hindi and English words become more separated. This suggests that the model has learned to differentiate between the two languages and capture more language-specific features.

3) Bloom7B 30 th layer is able to differentiate more in comparison to Gemma’s last 28 th layer representation.

THANK YOU