Hidden State Cache Compression

Daniel Xu

Abstract

We propose and evaluate methods for compressing the KV cache memory footprint. We first explore storing the hidden states instead of the the keys and values and propose an adapted attention function that can reduce the runtime of the attention function in this scenario. We then explore methods to compress the hidden state cache memory footprint by storing them sparsely at the layer level. We also propose and evaluate a method to downsample the hidden state into lower dimensionality and evalute its performance. We find our methods can lead to moderately degraded model performance with no finetuning involved. Further work could explore whether finetuning can restore model performance. The code is available at `https://github.com/danielxu9393/hiddenstate_cache.git`

1 Introduction

The transformer architecture [6] has transformed the landscape for language modelling. The ability to process tokens in parallel has lead to order of magnitudes improvement in training efficiency. However, this ability to process tokens in parallel, along with the large increase in model parameter count, has lead to the computation of many language modelling workloads to be memory bound. State of the art language models contain trillions of parameters, much too large to fit on a single GPU. Additionally, the attention mechanism requires memory that scales quadratically with the sequence length. To address this problem, a number of techniques have been proposed including Flash Attention [3] which enables the attention function to be computed without having to load the entire attention map into memory at any time, which reduces the memory footprint to linear with sequence length. Another technique, the KV cache takes advantage of the fact that many language modelling workloads are generation tasks and at each step only one token is appended to the sequence. Thus, the keys and values of all past tokens can be cached and used in the computation of the following token. However, in transformer models that utilize multi head attention, the memory footprint of the keys and values scale with the model dimension. To address this issue, multi query attention [5] and group query attention [1] propose using one or a smaller number of key value heads while preserving the number of query heads which greatly reduces the KV cache footprint. In [1], they were able to finetune the T5 model to group query attention with only 5% of the original training cost. The Falcon-7B model [2] is able to achieve similar performance to state of the art models of similar size with only a single key and value head, which decreased the KV cache size by 71 times.

A lot of models have already been trained with multi head attention. While [1] has shown the ability to finetune models to use group query attention, it still takes a significant training cost, spending 600 GPU days to finetune an 11B parameter model. In this work, we investigate whether it is possible reduce the KV cache usage of multi head attention models with no finetuning. We propose and evaluate two methods for compressing the KV cache for multi head attention transformers. Both leverage storing the hidden states instead of the keys and values in the cache.

The first method we propose is the Hidden State Cache. Instead of storing the keys and values, we store the hidden states in the cache. This alone will reduce the memory usage of the cache by 50%. While naively using the hidden states increases the attention function computation to be $O(ND^2)$ where N, D are the sequence length and model dimension respectively, we propose a simple adjustment to the attention function that reduces this cost to $O(NHD)$ where H is the

Preprint. Under review.

number of heads. Additionally we explore evicting certain layers for older tokens in the cache. Our motivation comes from [4] where they found that the hidden states between consecutive layers have cosine similarity of over 99% in OPT-175B. In our evaluation, we found that we can reduce memory footprint of the KV cache by 68% while still maintaining slightly degraded performance with no finetuning.

The second method investigates whether we can store a downsampled transformation of the hidden states to further decrease the memory usage. We do this by computing an singular value decomposition on the matrices involved in the attention mechanism to extract the important directions to the attention computation. This allows us to dynamically adjust the resolution at which to store the hidden states.

2 Methods

In many multi head attention architectures, the head dimension multiplied by the number of heads is the model dimension. Thus, the total memory footprint of the keys and values are twice that of the hidden state. Because of this, we propose storing the hidden states instead of the keys and values.

2.1 Computational Efficiency

We analyze the computation complexity of the attention function. All of these computations scale linearly with the batch size, so we will assume batch size 1. In the case where there is 1 query token and N past tokens, the attention mechanism with the traditional KV cache in multihead attention takes $3D^2$ MACS to calculate the queries, keys, values of the new token, ND operations to calculate the attention scores, and ND operations to multiply the attention scores by the values.

One of the problems with simply storing the hidden states in the cache is that in order to decode the hidden states into Keys and Values, every previous token must be multiplied with two DxD matrices where D is the hidden dimension. Thus, recovering K, V takes $2ND^2$ extra MACs.

$$Q = X_q \times W_q^T$$
$$K = X_k \times W_k^T$$
$$\texttt{attention weights} = QK^T$$

We provide an alternative attention computation that takes advantage of the fact that for sequential generation, there is only a single query token. In the attention computation, we can fuse the W_q, W_k multiplications. We denote X_q, X_k to be the hidden states for the query and kv tokens, and W_q, W_k, W_v to be the query, key, and value projections. Because there is only a single query token, multiplying X_q, W_q^T, W_k together is cheap, requiring $2D^2$ macs.

$$(X_q \times W_q^T \times W_k) \times X_k^T$$

The $X_q \times W_q^T \times W_k$ computation takes $2D^2$ MACs, and multiplying this resulting $1xHxD$ tensor with X_k takes NHD MACs where H is the number of heads. For very large sequence length N, the NHD term dominates in terms of number of MACs. However, this is still much faster than ND^2. In my implementation with batch size 1 on MPT-7B, where the computation is not memory bound, with sequence length up to 2000, the model latency is about 10% slower with the hidden state cache compared to the traditional KV cache. A future direction could be testing this implementation on a memory bound workload and comparing the performance.

2.2 Layer Sparsity

The above sections propose a memory-computation tradeoff with no performance degradation. In [4], they empirically observed that the cosine similarity between adjacent layers of the OPT-175B model was over 0.99 on average. This suggests that there could be memory savings at the layer level. We modify the hidden state cache so that for the most recent tokens, all hidden layers are stored. For older tokens, only a subset of layers are kept in the cache. For the missing layers, the next available layer is substituted in place of it. Figure 1 shows an example where the odd index layers are stored.

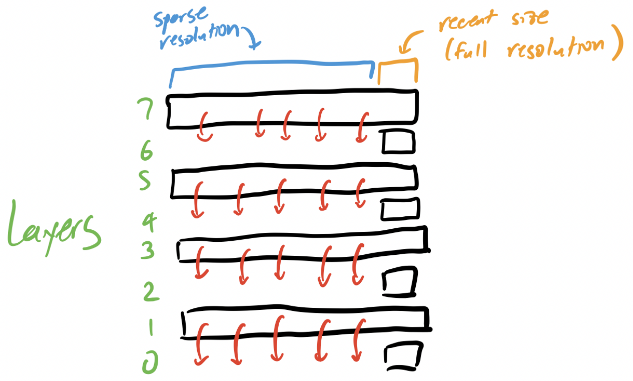

Figure 1: Example Sparse Hidden State Cache.

2.3 Low Rank Approximation

We wish to learn an encoder that can compress the hidden state into a lower dimensionality. In the MPT-7B architecture, the layer norm appears before the attention function, giving the following computation:

$$\texttt{attention weights} = norm(x_q) * scale * W_q^T * W_k * scale * norm(x_k)^T$$

$$\texttt{attention output} = softmax(\texttt{attention weights}) * norm(x_k) * scale * W_v^T * W_o^T$$

We separate the scaling factor from the layer norm and fuse them into W_q and W_k. Notice that after the norm function, the hidden state becomes unit length. This motivates us to consider the singular value decomposition of the matrices $scale * W_q^T * W_k * scale$, $W_o * W_v * scale$ in order to extract the most important directions of $norm(x)$.

For each head, we concatenate the matrices $scale * W_q^T * W_k * scale$ and $W_o * W_v * scale$ along dimension 0 and compute the stacked SVD USV^T. We use V to build our low rank approximation encoder.

If we wish to store the hidden states at resolution D_{lr} dimensions, then our encoder V_{lr} is the first D_{lr} columns of V. We store the encoded hidden state $norm(x) * V_{lr}$ in the cache. Our new attention computation becomes

$$\texttt{attention weights} = norm(x_q) * scale * W_q^T * W_k * scale * V_{lr} * V_{lr}^T * norm(x_k)^T$$

$$\texttt{attention output} = softmax(\texttt{attention weights}) * norm(x_k) * V_{lr} * V_{lr}^T * scale * W_v^T * W_o^T$$

Where $norm(x) * V_{lr}$ is stored in the cache. We can further speed up this computation by fusing the matrices $W_k * scale * V_{lr}$ and $V_{lr}^T * scale * W_v^T$.

3 Results

We implement and test our results on the MPT-7B model.

3.1 Layer Sparse Cache

I found that if I removed some of the first 8 layers, this would lead to a big degradation in model performance. This is probably due to the fact that the hidden state changes the most during the earlier layers. The layer sparsity policies I evaluated were keeping every 2, 3, 4 layers along with all of the first 8 layers. These policies contained 20, 16, 14 total layers respectively.

We evaluate the hidden state cache on three multiple choice datasets, 5-shot prompting, and kept only the 5 most recent tokens at full resolution. This is to simulate most of the relevant information being stored in the sparse resolution context. I also evaluated both substituting for the evicted hidden states, and the baseline of not substituting in the missing hidden states. (i.e. for those layers, those tokens would not be a part of the attention computation)

Cache Policy		lambada openai		piqa		boolq	
# Layers	Memory Reduction	Substitute	Baseline	Substitute	Baseline	Substitute	Baseline
full(32)	50%	0.65	NA	0.81	NA	0.75	NA
20	68%	0.60	0.33	0.81	0.78	0.66	0.55
16	75%	0.46	0.19	0.80	0.76	0.58	0.51
14	78%	0.35	0.12	0.79	0.77	0.47	0.45

The results show moderate degradation in the lambada openai and boolq datasets. While there is very little degradation in the piqa dataset, I believe this is because the prompt lengths in the piqa dataset were very short, and so most of the information could be found in the 5 most recent full resolution tokens. Interestingly, for lambada openai and boolq, the substitution policy vastly outperformed the baseline of no substitution, indicating that the substitution did indeed contribute to the performance.

3.2 Low Rank Cache

We test the low rank cache on the same 3 datasets with 5 shot prompting. We find that even slightly reducing the rank can lead to severe model performance degradation. This method probably would require finetuning or a more sophisticated method to train the encoder.

Rank	lambada openai	piqa	boolq
4096 (full)	0.65	0.81	0.75
3750	0.53	0.79	0.73
3500	0.38	0.77	0.67
3000	0.30	0.68	0.43

3.3 Latency

I measured the model latency of MPT-7B using NVIDIA V100 GPUS wiith 32GB of memory from the satori compute cluster. For batch size 1, I found that before any changes, the model had a latency of 36ms. With the hidden state cache, the model had a latency of 40ms.

4 Conclusion and Future Directions

In this work, we proposed and implemented strategies for compressing the KV cache for multi head attention transformers. Although our methods lead to degradations in model performance, our methods require no finetuning of the model. An interesting future direction, is whether a lightweight finetuning procedure like LoRA could decrease this performance gap. In DejaVu [4], they were able to train a small MLP to be able to predict which attention heads or FFNs to prune, showing that a simple model can partially learn the dynamics of the attention mechanism. An interesting future direction is replacing the singular value decomposition of the QKV projection with a linear projection that is trained to preserve the attention values given the hidden states as training data. Perhaps the space of hidden states covers smaller dimensionality than the model dimension, and thus could lead to more efficient compression.

References

[1] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

[2] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo. The falcon series of open language models, 2023.

[3] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[4] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re, and B. Chen. Deja vu: Contextual sparsity for efficient llms at inference time, 2023.

[5] N. Shazeer. Fast transformer decoding: One write-head is all you need, 2019.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.