Chunking Up The Problem With Query Integrated Memory Interfacing Attention

Zack Dugue

February 2023

1 Introduction

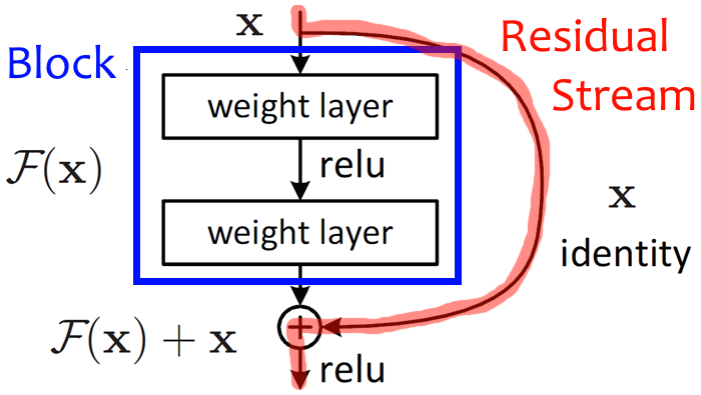
1.1 Background

Deep Learning is a field of machine learning which uses Deep Neural Networks to learn from data how to do certain tasks. Despite the name, for a long time in Deep Learning research deeper didn't mean better. In fact, at this time, adding more layers often degraded performance[4].

The solution to this problem was the Residual Neural Network, AKA "Resnet" [3]. Rather than having the input of the next block be the output of the last block, Resnets use something called a "residual stream" to control the flow of information in the network. Every block's output is added to this residual stream (via something called a "skip connection") as , and every block's input is the value of the residual stream at that block (rather than simply the output of the prior block) (fig 1). At the end of the network some final block processes the value of this residual stream and then generates the output. Resnet architectures tend to smooth the optimization space of the neural network, allowing efficient learning of deeper models. Virtually all networks deeper than 3 hidden layers use a Resnet architecture.

A drawback to this approach is that the residual stream must contain all the information relevant to the latter blocks of the neural network. For iterative tasks, this is not much of an issue. For example, imagine the steps necessary for multiplying out a factorial. You only need to remember what the running product is, and the number remaining left to multiply. But for tasks that involve parallel steps, like integration by parts, this means that you have to remember information from prior completed steps that are irrelevant to the current step, but necessary for a future step. In the case of a human brain, this might clutter up the short term memory, and in the case of a Resnet, this will clutter up the finite amount of information which can be held in the residual stream.

Figure 1: A diagram representing the structure of a Residual Neural Network from [3], modified for clarity.



A solution to this problem would allow would allow the neural network to effectively "chunk up the problem", by analyzing relevant information while ignoring information produced by other, irrelevant, parallel steps.

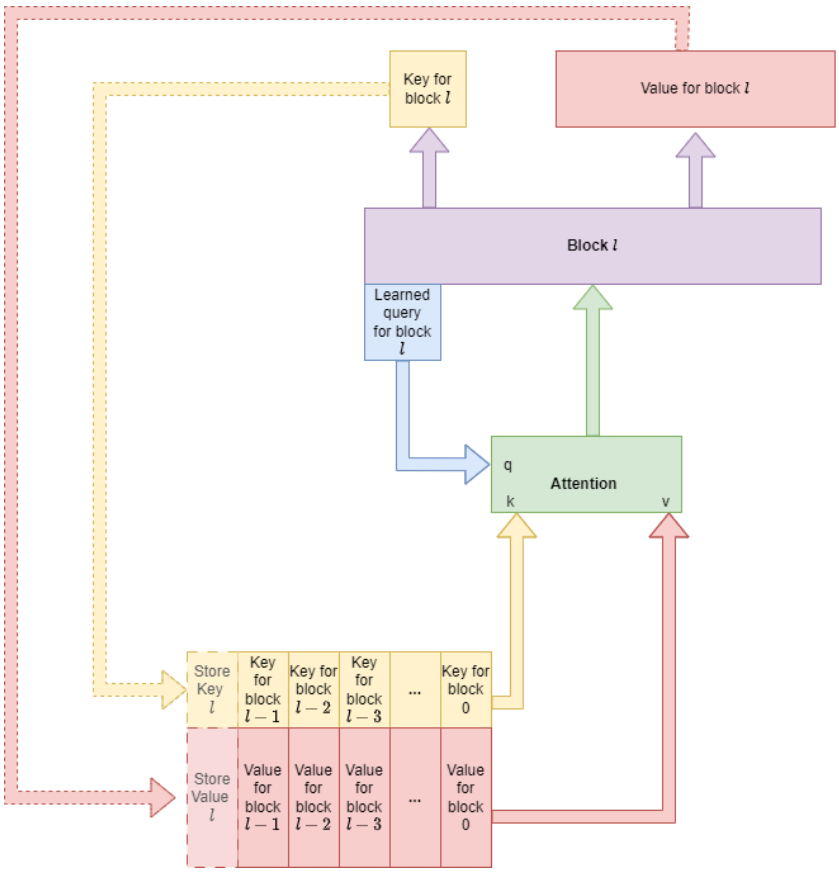
1.2 Query Integrated Memory Interfacing Attention

The solution that we propose is an architecture that allows any block to "look up" the outputs of prior blocks using attention. This solves our problem by creating an accessible "long term memory", such that current block only needs to take as input the relevant outputs of prior blocks, thus reducing the information clutter from irrelevant prior steps.
The most simplified way to explain attention is a bit like google search. Queries are like your search query, keys are like a URL, and values are like the results on each web page. Attention (using a single query) lets our neural network pull relevant information from a set of tokens and condense it into one. Using a more mathematically rigorous explanation, The query and keys are vectors (or at least some mathematical object with an operation analogous to the dot product) [6]. The attention mechanism uses a *softmax* over the dot product between the query and each key to compute a convex combination of the values.

$$A_l = \sigma(\mathbf{q}\mathbf{K}^T) * \mathbf{V} = \frac{\sum_{i=0}^{l-1} \exp(\mathbf{q} \cdot \mathbf{k}_i)\mathbf{v}_i}{\sum_{i=0}^{l-1} \exp(\mathbf{q} \cdot \mathbf{k}_i)} \tag{1}$$

Equation 1: *This equation shows represents the input to a block via the attention mechanism at layer l. Where σ is the softmax operation, q is the learned query at block l, k_i is the output key for block i and v_i is the output value for block i.*

Figure 2: A diagram of block l performing Query Integrated Memory Interfacing Attention, processing that input into a key and value, and then storing that key value pair in the set of key value pairs that serves as "memory".



In our implementation, every block outputs a key and a value, and every block learns a query. This query is then used to do attention on the outputs of prior blocks. This allows each block to signal in their output key what they think their output value will be useful for. Each block also learns a query to search for the most relevant outputs of prior blocks. The queries are directly learned, and the keys and values of prior blocks acts as a sort of "memory", hence the name.

There are variations on this idea which we intend to explore in the course of our research. One of these variations is called "component wise" QIMIA. Instead of outputting one key to represent the output of the entire block, the block could output a key for each component of the value (a component being,

a pixel in a CNN, a token in a Transformer, etc.). This way the attention can select for which components seem relevant, rather than looking up the entire value output of the block.

2 Objectives

In our project we aim to investigate the usefulness of the Query Integrated Memory Interfacing Attention (QIMIA) architecture. In theory QIMIA can replace any architecture that uses skip connections, and such architectures are ubiquitous in deep learning. So, we intend to evaluate QIMIA implementations across a variety of neural network architectures, including Convolutional Networks for tasks like image classification and Transformers for tasks like NLP. There are many well established benchmark tasks in these areas that we will evaluate on. Rather than striving to beat the state of the art, we will simply focus on comparing Resnet implementations, with QIMIA implementations using similar block structures (adjusted so that they have similar parameter counts). The result of a useful project will hopefully be a new architecture, which outperforms the standard Resnet formulation in at least some applications.

3 Approach

To accomplish this objective, we will implement a PyTorch Module for the QIMIA architecture. This Pytorch Module will store a list of blocks, (which can be attention and feed forward layers for a transformer, convolution blocks from a Resnet, etc.), a list of key projection networks, and a list of learned queries. The container's forward pass will perform QIMIA between the blocks, and then will output the whole list of values and keys for all the blocks as a final output. Such a container will allow us to implement QIMIA for a variety of different block architectures, by simply "plugging in" the desired block. This flexibility is necessary given that we will be testing QIMIA on many different architectures with many different input types.

We will then evaluate the QIMIA architecture against other Resnet based architectures in different applications of Deep Neural Networks. To do this we will be using the Pytorch Lightning framework and the Hydra config manager, in order to make developing and running the training pipeline for all of these different architectures / tasks as seamless as possible. In adapting the Resnet architectures to QIMIA, we will maintain as much of the structure from the blocks in the original architectures as possible. IE rather than making some brand-new architecture with novel block structure, I want to take the skip connection architectures and "QIMIA-ify" them. That way the comparison is as direct as possible.
We will test this with:

- Resnet-50 on ImageNet [3]
- Transformer (of some small size) on the WikiText dataset [6]
- UNET for denoising on Imagenet [2]

We intend to explore different hyper-parameters and architectural variations on the QIMIA architecture. For example, the number of heads in the attention mechanism, using layer wise vs component wise attention, the dimensionality of the keys, etc. We also won't shy away from using our analysis of the architectures performance in order to make improvements to it.
There are also a few specific applications that we intend to explore, due to our belief that the QIMIA architecture would have a unique advantage in these areas. We would like to try a "weight sharing" implementation of QIMIA. Weight sharing is the practice of repeating the same weights in all the blocks of a neural network[1] [5]. We believe a modified version of QIMIA, could overcome common problems with weight sharing and outperform current models implemented in this way. We also want to try transfer learning with QIMIA. In standard transfer learning approaches, you have a larger network pretrained on some task, cut off at some intermediate layer, and then concatenated with a smaller network. The base model simply outputs a set of features to be learned on, and the fine tuned model simply learns to adapt those features to the correct output (with a slower rate of learning occuring in the base neural network) [7]. However, in this case, all of the intermediate outputs of the blocks could also be accessed via QIMIA. This would seem to allow whatever fine tuned neural network, to access more of the information extracted by the base neural network, which will hopefully help transfer learning be more efficient.

This project will likely require significant compute resources to compare across all of these applications. For this we will use the Caltech HPC, and potential Amazon Web Services resources as well. We believe the most difficult steps will be building the training pipeline, and adapting the Resnet architectures to work with QIMIA.

4 Work Plan

Our plan for each week is

1. implement architecture / create pipeline
2. implement architecture / create pipeline
3. evaluate QIMIA with transformer blocks on NLP task
4. evaluate QIMIA with convolutional blocks on Image Classification
5. evaluate QIMIA with convolutional blocks on DeNoising

6. evaluate QIMIA's performance in transfer learning (preferably in a task it previously performed well in).

7. evaluate QIMIA weight sharing implementation (possibly on PG19 long text dataset)

8. Analyze Results

9. Overflow Week

5 Works Cited

References

[1] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. *CoRR*, abs/1807.03819, 2018.

[2] Javier Gurrola-Ramos, Oscar Dalmau, and Teresa E. Alarcón. A residual dense u-net neural network for image denoising. *IEEE Access*, 9:31742–31754, 2021.

[3] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. *CoRR*, abs/1412.1710, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W. Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition, 2018.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[7] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019.