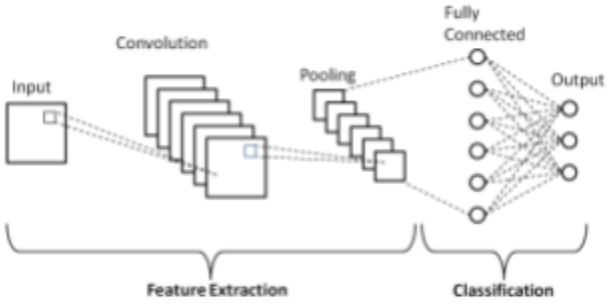
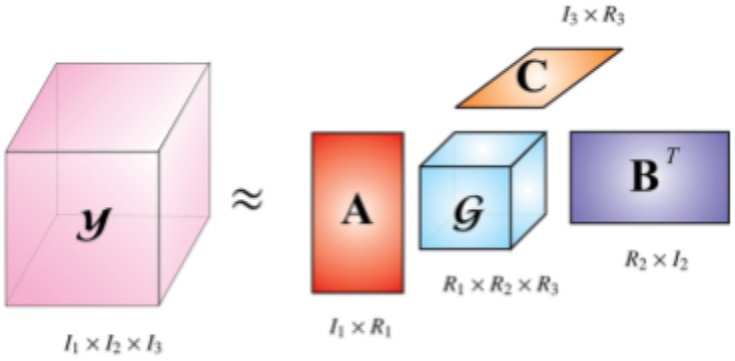
Literature Review of Tensor Decomposition Layers in Convolutional Neural Networks and Large Language Models

Fully Connected Layers use a lot of parameters as each input node must have a weight connecting to each output node. Because of this, they consume a lot of time during forward propagation calculation, and they require a lot of memory to store these parameters. There have been many papers proposing replacing Fully Connected Layers with Tensor Decomposition Layers in machine learning applications that deal with higher dimensional data. Convolutional Neural Networks are a great example of this as they take an input tensor (an image) and perform convolutions on that image to generate another tensor which is then sent through a maxpool layer which generates another tensor, and this process is done repeatedly until the final layer which is a Dense Layer. This Dense layers are comprised of multiple Fully Connected Layers where the number of input nodes is the number of elements in the input tensor, the number of output nodes is determined by the different number of objects which it could be classified, and the number of hidden layers and hidden nodes is up to the choice of the designer. A diagram of a CNN is shown below and there exists 1 Fully Connected Layer at the end of the model.



Phung, & Rhee (2019)

After a review of the literature, there are two prominent interpretations of a Tensor Decomposition Layer. The first was proposed by Giuseppe et. al. published in an article titled *Compression and Interpretability of Deep Neural Networks via Tucker Tensor Layer: From First Principles to Tensor Valued Back-Propagation*. In this piece the author explains that a Fully Connected Layer can be abstracted into a weight tensor which is then multiplied by the input tensor in order to result in the output layer. This weight tensor can be decomposed into a core tensor and a collection of matrices in memory. Whenever the weight tensor is needed, the core tensor is then multiplied by the matrices using the N mode product in order to result in a much larger tensor that is then multiplied by the input tensor. The idea is that the core tensor and matrices will learn dimensional trends during back propagation which will compensate for the extreme decrease in parameters. (Giuseppe, 2020) The second interpretation was proposed by Kossaifi in an article



Billiet, Lieven & Swinnen, Thijs & de Vlam, Kurt & Westhovens, Rene & Huffel, Sabine. (2018)

titled *Tensor contraction layers for parsimonious deep nets*. This piece describes the process of taking the input tensor and decomposing it into a core tensor using N mode product with a collection of matrices, which would also learn dimensional trends during backpropagation, and then sending the smaller core tensor into a smaller fully connected layer. (Kossaifi, 2017) For this project, we will opt for this interpretation.

The first reason being that the back propagation for the first interpretation is very expensive. The back propagation process requires calculating multiple matrices that are the size of the product of all the dimensions of the weight tensor squared. Using MNIST dataset of 28 by 28 black and white images of handwritten digits, ff we were to implement a single convolution layer of with 32 3 by 3 kernels, and a single 2 by 2 max pooling layer, the weight tensor would contain 13 * 13 * 32 * 10 = 54080 elements. Equation 30 in the paper shown below, describes how to find the permutation matrix P which would have a number of rows and columns equal to 54080. That would be a total of 54080 * 54080 = 2924646400 elements. Even if each element in memory was a single byte, that would still result in a total memory usage of around 3TB for this single variable in this simple case.

$$\text{vec}(\mathcal{W}_{(n)}) = \mathbf{P}_n \text{vec}(\mathcal{W}_{(N+1)})$$

(Giuseppe, 2020)

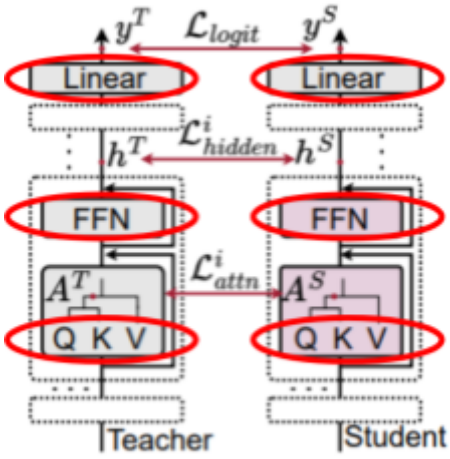
Given that C++ can only allocate memory up to a few GB this design could not be implemented on our technology. These computations would also make back propagation so slow that it would take so long that it would not make up for the decrease in memory usage. In conclusion, it's not worth implementing. The second interpretation does not suffer from this problem as the biggest tensor would be the input tensor that would contain 13 * 13 * 32 = 5408 elements which are then decomposed into a smaller tensor whose dimensions would be determined by the designer based on the desired accuracy and memory size. This smaller tensor would then be sent through a much smaller fully connected layer which is where the savings in time and space come into play. This idea would have much cheaper back propagation as the biggest tensor computed in back propagation would be the derivative of the core tensor with respect to the input tensor. That is a matrix with the number of rows equal to the number of elements in the input tensor and the number of columns would be equal to the number of elements in the core tensor. If the core tensor is of size 10 by 10 by 20, the number of elements in that tensor would be 5408 * 2000 = 10816000 which would be in the range of MegaBytes.

Relative speedup can be calculated by comparing the number of weights in the Fully Connected Layer before adding a Tensor Decomposition Layer, and comparing that to the number of weights in the Fully Connected Layer after adding a Tensor Decomposition Layer plus the number of weights in each matrix used in the N mode products with the input tensor. Because multiplication operations are 3 times slower than addition operations, (Fog, 2022) it is only important to compare the number of multiplications in each case to get an accurate idea of the expected speedup. Given input tensor of 13 by 13 by 32 having 5408 elements which would be connected to 10 output nodes, there will be a total of **54080** multiplication operations done in the forward pass of the fully connected layer with no tensor decomposition layer. Given a tensor decomposition layer prior to the fully connected layer with a size of 10 by 10 by 20 which we will use in our model, there will be a total of **20000** multiplication operations in the fully connected

layer. Performing the N mode product requires folding the input tensor along the Nth dimension and multiplying with a matrix. The 0th dimension folded input tensor would be of size 13 by 416 which is then multiplied by a 10 by 13 matrix which would be 10 * 13 * 416 = **54080** multiplications. The 1st dimension folded tensor would now be of size 13 by 320 which is then multiplied by a matrix of size 10 by 13 for an additional 10 * 13 * 320 = **41600** multiplications. Finally, the 2nd dimension folded tensor would be of size 32 by 100 multiplied by a matrix of size 32 by 20 for an additional 20 * 32 * 100 = **64000** multiplications. So as you can see, the total number of multiplications required in the tensor decomposition layer case would be **179680**. That is approximately 3.32 times slower than a simple fully connected layer. The number of parameters used in this case would be 20900 which is 39% of the initial number of parameters. If a core tensor of 5 by 5 by 10 was used, the number of multiplications would be **45690** which is a 16% reduction in the computation time and **3580** total parameters which is 6.6% of the initial space. The accuracy of both of these models will be determined in this work but the idea is that this method will not save time but it will reduce the number of parameters. In the entire CNN, given that we will be using 32 3 by 3 kernels in addition to the Tensor Decomposition Layer and Fully Connected Layer, there will be a 59.4% reduction in the number of parameters used with a core tensor of 10 by 10 by 20 and a 98.1% reduction in total parameters if a 5 by 5 by 10 core tensor .

The paper shows results on AlexNet using the CIFAR100 database of images. This model has 2 fully connected layers which each have 4096 hidden units. Their findings show that there is a peak accuracy when they add a tensor decomposition layer at the beginning of these layers which is a core tensor of size 128 by 3 by 3, followed by 2 fully connected layers of size 2048. This results in an accuracy increase from 65.41% to 66.57%. The case with the most space savings is where both fully connected layers are replaced with tensor contraction layers with a core tensor of size 192 by 3 by 3 and 144 by 3 by 3, which experiences a space savings of 99.22% compared to the standard while still maintaining an accuracy of 62.06%.

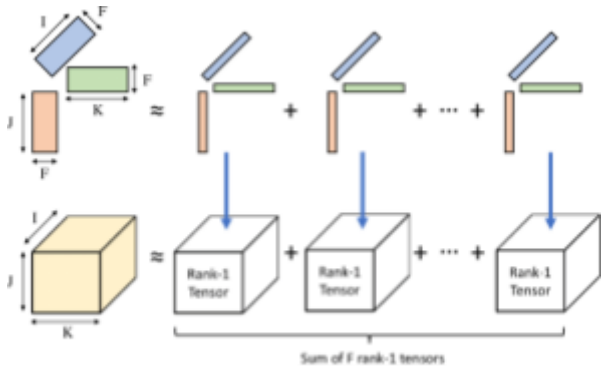
Large Language Models use transformers to learn aspects of language input data. A diagram of one can be seen below which is composed of N blocks. Each of these N blocks requires calculating vectors Q, K and V which each require a Fully Connected Layer. The end of each block also contains a feed forward network which is a set of M fully connected layers and at the end of all of the blocks there is a linear layer which is analogous to a Fully Connected Layer. In total there are a total of (3 + M) * N fully connected layers in each transformer.



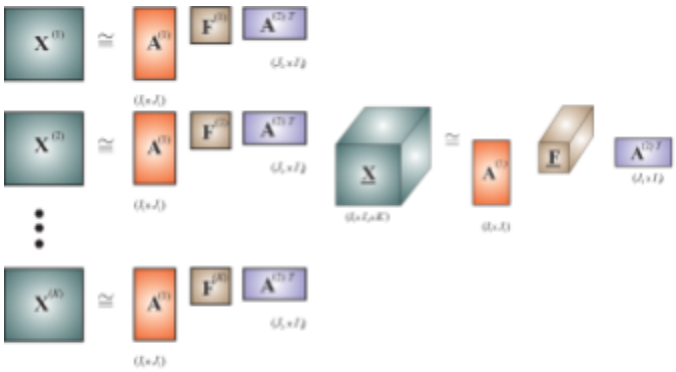
Gu. J (2022)

When it comes to Large Language Models, there is even less in the literature focusing on Tensor Decomposition Layers. Jiaqi et. al. wrote a paper titled *Heat: hardware-efficient automatic tensor decomposition for transformer compression*, they compared many different methods of tensor decomposition and their relative performance. The first method is Tucker Tensor Decomposition which is the same as the 2 methods mentioned above where the N mode

product of a core tensor with a collection of matrices along each dimension is taken in order to reach a tensor of a different size. The second method is called CANDECOMP/PARAFAC Tensor Decomposition which is the process of splitting a tensor into a sum of outer products of vectors with length equal to the dimensions of the input tensor. The final method is called Tensor-Train Matrix decomposition which, similar to the first but for matrices instead of vectors, describes the process of taking the summation of the dot product of a set of matrices.



Srivastava, Nitish. (2020).



Phan, A., & Cichocki, A. (2010)

The methods were compared to the base BERT model which used 109.5M parameters and had an F1 score of 88.16. F1 score is a measurement used in machine learning to gauge precision. Using Tensor-Train Matrix Decomposition they were able to use 41.60M parameters with an F1 score of 86.01. Using Tucker Tensor Decomposition they were able to use 42.14M parameters with an F1 score of 83.41. Using CANDECOMP/PARAFAC Tensor Decomposition they were able to use 48.88M parameters with an F1 score of 87.11. This shows that the number of parameters can be cut in half while still maintaining accuracy.

References:

Billiet, Lieven & Swinnen, Thijs & de Vlam, Kurt & Westhovens, Rene & Huffel, Sabine. (2018). Recognition of Physical Activities from a Single Arm-Worn Accelerometer: A Multiway Approach. Informatics. 5. 20. 10.3390/informatics5020020.

Fog, A. (2022, November 4). Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs. https://www.cs.utexas.edu/~hunt/class/2018-spring/cs340d/documents/Agner-Fog/microarchitecture.pdf

Giuseppe G. Calvi, Ahmad Moniri, Mahmoud Mahfouz, Qibin Zhao, and Danilo P. Mandic, “Compression and Interpretability of Deep Neural Networks via Tucker Tensor Layer: From First Principles to Tensor Valued Back-Propagation,” ArXiv, Jan 2020.

Gu, J., Keller, B., Kossaifi, J., Anandkumar, A., Khailany, B., & Pan, D. Z. (2022, November 30). *Heat: Hardware-efficient automatic tensor decomposition for transformer compression*. arXiv.org. https://arxiv.org/abs/2211.16749

Jiaqi Gu, Ben Keller, Jean Kossaifi, Anima Anandkumar, Brucek Khailany, David Z. Pan, “Heat: hardware-efficient automatic tensor decomposition for transformer compression” ArXiv, Nov 2022.

Kossaifi, J., Khanna, A., Lipton, Z., Furlanello, T., & Anandkumar, A. (2017). Tensor contraction layers for parsimonious deep nets. 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). https://doi.org/10.1109/cvprw.2017.243

Phan, A., & Cichocki, A. (2010). Tensor decompositions for feature extraction and classification of high dimensional datasets. Nonlinear Theory and Its Applications, IEICE, 1, 37-68.

Phung, & Rhee,. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Applied Sciences. 9. 4500. 10.3390/app9214500.

Srivastava, Nitish. (2020). DESIGN AND GENERATION OF EFFICIENT HARDWARE ACCELERATORS FOR SPARSE AND DENSE TENSOR COMPUTATIONS.