**Name: Dhruv Sinha**

**Collaboration:** I discussed this assignment with Cole Frank and Chris Kang. They helped me understand various parts of each model that I have used.

**Q. Understand and distinguish these concepts**

References

1. <https://stats.stackexchange.com/questions/560845/what-is-the-difference-between-network-sparsification-and-model-pruning>
2. <https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b>
3. <https://deepai.org/publication/moefication-conditional-computation-of-transformer-models-for-efficient-inference>
4. <https://en.wikipedia.org/wiki/Knowledge_distillation>
5. https://neptune.ai/blog/knowledge-distillation

Pruning and Sparsification- In pruning, we remove weight networks in a connection to increase the inference speed and decrease the model storage size. In general, we believe that there are some weights that are not very important in determining the final objective of the task. We can do away with these weights and yet achieve the same inference accuracy. There are several methods to prune but essentially, we just want to store the most important determinants. A pruned network becomes sparse and is also known as a sparse network. These two concepts are almost identical.

Quantization- Quantization is a method to enable efficient high performance deep learning computation on small devices. As we will learn in this assignment, large models (of scale in billions) are important to achieve high accuracy for natural language tasks. Quantization is used to bring the neural network to a reasonable memory size without compromising on the accuracy.  Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers.

Distillation- this is a process of transferring the knowledge from a large model to a smaller model that can be used with less compute. Usually, large models are not always operating at their capacity and using them for inference can be very expensive. If we can transfer the knowledge to smaller models, they can be used with less compute without the loss of validity.

MoEfication- Transformer based pre trained model achieve superior performance because of their sizes. However, they also incur a very high computation cost. Zhang et al found that most inputs only activate a tiny ratio of neurons during inference. Hence, they want to accelerate the large-model inference by conditional computation based on the sparse activation. This means they select the neurons that need to be activated based on some condition.

**Q. Choose your models. Pick 3 models, 1 from each category. Each pick should be of more than 1B parameters before pruning.**

The 3 models that I chose are:

1. GPT2 XL: This model has 1.5 billion parameters and does NLP tasks like question answering, machine translation, reading comprehension and summarization.
2. Facebook M2M 100: This encoder decoder model has 1.2 billion parameters and can directly translate between the 9,900 directions of 100 languages.
3. Deberta v2 XXlarge: This encoder only model has 1.5 billion parameters and is used for sentence and label generation/prediction

**Q. Devise approaches to assess sparsity structure in your choice of models**

To access the sparsity structure, I traversed through each layer of the model and flattened all the tensors in that layer. I concatenated all the tensors in that layer into a list. To get a sense of sparsity, I devised two approaches.

**Approach 1->** In the first approach, I created ranges between which weights are falling. These ranges are:

W<-.08, -.08<W<-.05, -.05<W<-.01, -.01<W<.01, .01<W<.05, .05<W<.08, W>.08

Then I calculated the percentage of weights belonging to each of these ranges in each layer and plotted them as bar histogram

**Approach 2->** According to me, this is a more intuitive way to assess the sparsity structure of the model. I plotted the density of weights for each layer. This way, we can see how and where the flattening of curve is changing across layers.

**GP2:** I have plotted the percentage graphs and the densities for all the 48 layers in GPT. Here, I am only showing a few of them but please feel free to refer to the notebook for GPT2 for more examples.

A picture containing histogram

Description automatically generatedChart

Description automatically generated with medium confidenceChart

Description automatically generated with low confidence

The percentage graphs I have plotted are for layers 0, 16, and 40. We can see that between layers 0 and 40, the percentage of weights concentrated near the center have reduced and the percentage of weights concentrated on the edges of the graph has increased. In layer 0, the percentage of weight around 0 (between -0.03 and 0.03) was close to 30% (for each band). In layer 40, this percentage has reduced and percentage around the edges (greater than .03 and less than -.03) has reached close to 30%. Moreover, the height of bar for ‘0’ is also decreasing. This transition is even more visible in the density plots below.

Logo, company name

Description automatically generatedLogo, company name

Description automatically generatedLogo, company name

Description automatically generated

In the density plots, we can see that curve is flattening and tails are getting thicker as layers increase. The peak in layer 0 for mean=0 is at freq 14 whereas the peak in layer 40 is between 8 and 10.

Thus, we can conclude that GP2 XL, through the course of training becomes less sparse.

**Facebook M2M 100:** I have plotted the percentage graph for 24 layers of encoder and decoder of M2M 100.

A picture containing chart

Description automatically generated Chart

Description automatically generated with medium confidence Chart

Description automatically generated

A picture containing chart

Description automatically generated Chart

Description automatically generated with medium confidence Chart

Description automatically generated

In the figures above, I have plotted the percentage of weights in each layer and where they usually lie. As with GPT, in the initial layers, the weights are concentrated around 0. However, as we transition through layers, the percentage of weights around 0 reduce and percentage of weights on the edge of the graph increase. If we take a closer look at the layers of the encoder, we can see that percentage of weights between -.01 and .01 in layer 0 is approx 30% whereas the same percentage reduces to 25% in layer 10 and 23% in layer 23. The same trend can be see in the decoder layers also.

We will now plot the density curve for M2M.

Logo, company name

Description automatically generated Logo, histogram

Description automatically generated Chart, histogram

Description automatically generated

Logo, company name

Description automatically generated Logo, histogram

Description automatically generated Chart, histogram

Description automatically generated

In the density graphs abvoe, for both encoder and deconders, we can see that the curve is flattening as we trainsition through layers. Morever, the height of the curve (frequency) is also reducing which depicts that the the model is becoming less sparse.

**DEBERTA:** I have plotted the grpah for 48 layers of Microsoft Deberta

Chart

Description automatically generated with medium confidence Chart

Description automatically generated with medium confidence Chart

Description automatically generated with low confidence

In the graphs above, the percentage of weights around zero is increasing as we increase the layers. In layer zero, the percentage of weights between (0.01 and -0,01) is approx 20% whereas this percentage increases to ~26% in layer 40. I am not exactly sure why we see this trend. The same trend can be seen in the density plots below. We can see that unlike the two models we discussed above, the density is not flattening. If only, the height of the curves around 0 has increased (frequency has increased)

Logo, company name

Description automatically generated Logo, company name

Description automatically generated Logo, company name

Description automatically generated

**Q. Produce sparsified versions of your models at 10%, 50%, 90%, 95%, 99%, by either coding your methods or using existing tools provided below Explain the nature of your methods, regardless of whether you code it yourselves.**

To produce pruned versions, we are using the method provided by pytorch. For each model, we are iterating through each layer and applying l1 unstructured pruning to that layer. L1 unstructured pruning means that we are removing the specified amount of units with lowest L1-norm. How we are doing this can be seen in the code snippet below-

Text

Description automatically generated

In the case of GPT2, I am specifying which components to prune in each layer. This needs to be specified manually as named\_modules also contain objects that cannot pruned (for example activation function).

For M2M and deberta as well, I have specified the components to prune in each layer. This can be seen in their respective scripts.

**Q. Find 2 common benchmarks used by your models**

Currently working on it (will upload the updated version on github with permission)

**Q. Compare size of models and runtime for sparsified models**

Currently working on it (will upload the updated version on github with permission)

**Q. Explain the challenges of sparsification on LLMs.**

On of the biggest challenges is not figure out which components to prune within each layer. In Exercise 4, I pruned x% of weights for all the prunable components in each layer. Maybe we will get more optimum results if we only prune MLP components in each layer. I think this is done mainly through trial and error which takes a lot of time. Also, specifying layers to prune for each model is very cumbersome if someone is employing multiple models.