Advanced Deep Learning *ECS 289L, Spring 2024*

# HW1: LLMs

We will use an existing implementation of LLMs and play with the architecture to better learn how it works.

1. Download nanoGPT code from: https://github.com/karpathy/nanoGPT

   Run the code out of the box to train an LLM on "shakespeare_char" dataset. You may use a single GPU from Google Colab.

2. The code uses "n_embd/n_head" for all key, value, query vectors. To speed up the attention module, one can reduce the dimensionality of query and key vectors. Please modify the code to input the size of key and query vectors as an argument. You may see the standard numbers at the config file. The size of key and value vectors are 384/6=64. Please try smaller numbers 32 and 8 to see the effect on the final loss after 5000 iterations.

3. Change the attention module to perform sliding window attention. This means any new token can attend to only "wind" most recent tokens instead of attending to any previous token. Please modify the code by inputting "wind" in the input as an argument and see the effect of reducing the window size. Please try the following window sizes: 100, 10, 3. Please report the result. You may read more about this method in the following paper:
   https://arxiv.org/abs/2309.17453

4. Change the MLP layer from a simple 2-layer MLP to another version that is used in LLaMA models. The standard MLP is: fc2(ReLu(fc1(x))). We can change it fc3(ReLU(fc1(x)) .* fc2(x))) where ".*" is element-wise product between two vectors. Please run the original model (Section 1) with this version of MLP and report the loss. This is described in the following paper:
   https://arxiv.org/pdf/2002.05202

5. You may add a few more tokens (called register tokens) that are not influenced by the input. These tokens can act as a form of memory that the model may use to store information to use later. The value in the first layer for these tokens can be learned at the training time. Please modify the code by implementing "n_regist" number of register tokens. n_regist is an argument in the input. Then, try n_regist=1 and n_regist=5 and compare the loss with the original model with no register token.

6. Combine the last two steps to come up with a model that has both extra register tokens and small sliding window attention. The register tokens are always included in the attention, so the attention will run over wind+n_regist number of tokens. Try wind=3 and n_regist=1 and compare the results with previous methods where n_regist=1 or wind=3.

7. (Extra credit) Softmax appears to be expensive due to the exp(.) operation. Let's replace exp(.) operation with simple abs(.) operation in both numerator and denominator of the Softmax function. Please do this in the original model (section 1) and report the loss. A version of this method for ViT is described in this paper:
   https://arxiv.org/abs/2206.08898