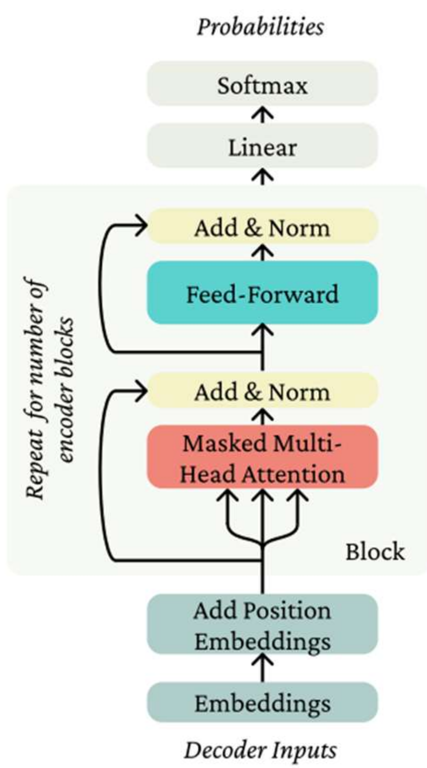
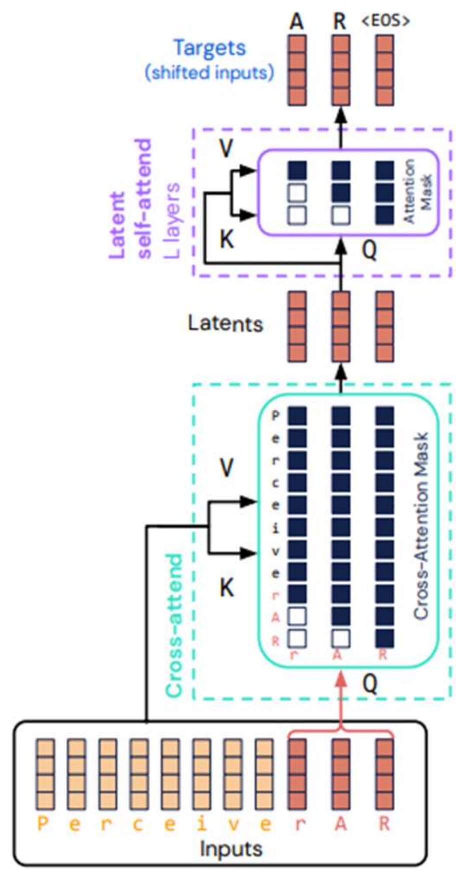
Assignment 5 Group Office Hour

XCS224N Fall 2023

MiniGPT Model & Perceiver Modification

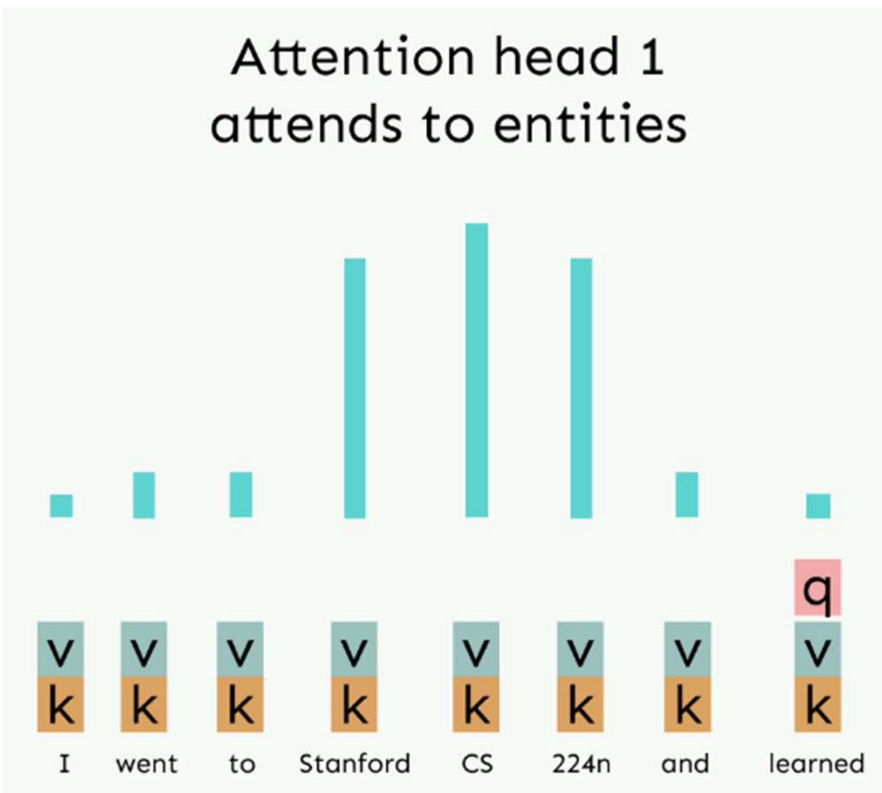


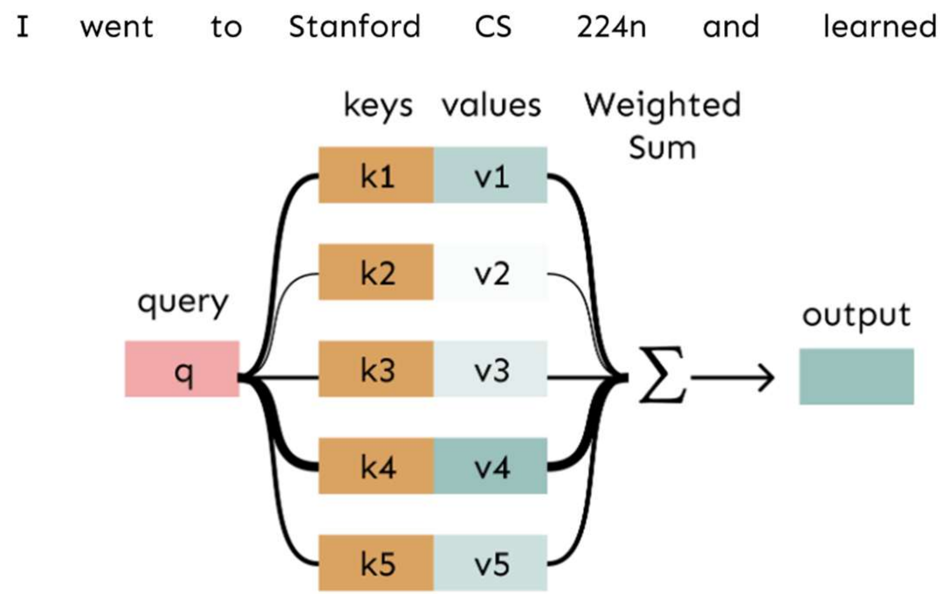
Transformer Decoder



Attention

$\boldsymbol{q}_i = Q\boldsymbol{x}_i$ (queries) $\quad$ $\boldsymbol{k}_i = K\boldsymbol{x}_i$ (keys) $\quad$ $\boldsymbol{v}_i = V\boldsymbol{x}_i$ (values)





$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})} \qquad \boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \boldsymbol{v}_i$$

1 (c) Implement Finetuning

- helper.py
- finetune() method
  - Load parameters if pretraining (can be done later in part 1 (f))
    - Use reading_params_path
    - Load parameters into model
      - torch.load
      - torch.nn.Module.load_state_dict
      - https://pytorch.org/tutorials/beginner/saving_loading_models.html
  - Make sure to specify the correct max epochs
    - Should be specific to cases: finetuning with and finetune without pretraining
  - Don’t call train inside this method
    - We have a train method for handling that

1 (d) Make Predictions

- Test locally to debug before uploading to Azure VM
  - sh run.sh vanilla_finetune_without_pretrain
- Can be trained locally with high-performance desktop GPU
  - Make sure to use the CUDA version of the XCS224N Conda Env
  - conda activate XCS224N_CUDA
- Be sure to allow training to complete
- Training output file is used for grading

1 (e) Define Span Corruption for Pretraining

- dataset.py
- Implement __getitem__ method for CharCorruptDataset class

1. Randomly truncate document
2. Break already truncated document into: [prefix][masked_content][suffix]
3. Rearrange resulting substrings:
   [prefix] MASK_CHAR [suffix] MASK_CHAR [masked_content] MASK_CHAR [pads]
4. Create input and output strings
5. Encode as Long tensor
   `torch.tensor()`
   https://pytorch.org/docs/stable/tensors.html

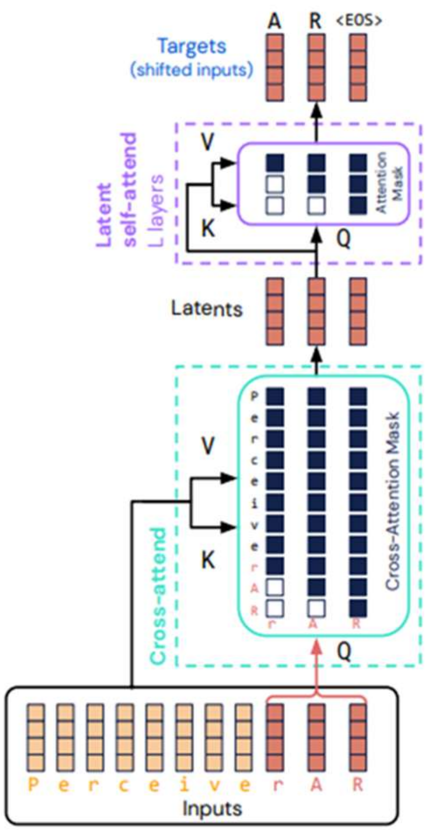
- Tips
  - Perform truncate first, use array slicing, and generate a random int (be sure to bound correctly).
  - For generating a random context lengths where the average length is ¼ . Consider ways you can generate a random int such that the mean will be ¼ the length of the document, be careful to bound the generated ints correctly.
  - To create input and output, simply strings follow the array splicing defined in the comments.

1 (f) Pretrain, Finetune, and Make Predictions

- helper.py
- Complete pretrain() method
- Update finetune() method to incorporate pretraining if not done in 1 (c)
- Test locally to debug prior to uploading to Azure VM

1 (g) More Efficient Attention

- model.py
  - UpProjectBlock and DownProjectBlock
- Down and Up Projection blocks should be very similar to standard transformer block, use that code as a starting point
- Changes
  - self.C parameter
    - Defines length of down-projected sequence
    - Learned parameter, randomly initialize with Xavier
    - The optimized parameter is a basis for the vector space the x_input is projected into
  - Cross Attention
    - Order of inputs to Cross Attention in Up vs Down, should be reversed



3 (a) Extra Credit

- Consider how the values of the dot products of the query with keys affects the softmax values
- Then consider how a distribution of attention weights might induce "copying"
- For full credit, make sure to clearly explain your reasoning and include relevant math to show the result

(a) **[2 points (Written, Extra Credit)] Copying in attention:** Recall that attention can be viewed as an operation on a query $q \in \mathbb{R}^d$, a set of value vectors $\{v_1, \ldots, v_n\}, v_i \in \mathbb{R}^d$, and a set of key vectors $\{k_1, \ldots, k_n\}, k_i \in \mathbb{R}^d$, specified as follows:

$$c = \sum_{i=1}^{n} v_i \alpha_i \tag{5}$$

$$\alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)}. \tag{6}$$

where α_i are frequently called the "attention weights", and the output $c \in \mathbb{R}^d$ is a correspondingly weighted average over the value vectors.

We'll first show that it's particularly simple for attention to "copy" a value vector to the output c. Describe (in one sentence) what properties of the inputs to the attention operation would result in the output c being approximately equal to v_j for some $j \in \{1, \ldots, n\}$. Specifically, what must be true about the query q, the values $\{v_1, \ldots, v_n\}$ and/or the keys $\{k_1, \ldots, k_n\}$?

3 (b) & (c) Extra Credit

- (b)
  - What are we trying to show?
    - Attention weights that result in an output that is essentially: ½ * v_1 + ½ v_2
  - Consider how these attention weights can be produced in a softmax
  - How can we specify a query q such that it will produce these weight?
- (c) i
  - Start reasoning from the result of 3 (b)
  - Pay attention to the covariance definition
- (c) ii
  - Now consider the effect that varying magnitudes of a just one key vector?
  - How does that affect the softmax?
  - k is sampled with a larger magnitude vs small magnitude?

(b) **[2 points (Written, Extra Credit)] An average of two:** Consider a set of key vectors $\{k_1, \ldots, k_n\}$ where all key vectors are perpendicular, that is $k_i \perp k_j$ for all $i \neq j$. Let $\|k_i\| = 1$ for all i. Let $\{v_1, \ldots, v_n\}$ be a set of arbitrary value vectors. Let $v_a, v_b \in \{v_1, \ldots, v_n\}$ be two of the value vectors. Give an expression for a query vector q such that the output c is approximately equal to the average of v_a and v_b, that is, $\frac{1}{2}(v_a + v_b)$.[6] Note that you can reference the corresponding key vector of v_a and v_b as k_a and k_b.

(c) **[3 points (Written, Extra Credit)] Drawbacks of single-headed attention:** In the previous part, we saw how it was *possible* for a single-headed attention to focus equally on two values. The same concept could easily be extended to any subset of values. In this question we'll see why it's not a *practical* solution. Consider a set of key vectors $\{k_1, \ldots, k_n\}$ that are now randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Further, assume that the means μ_i are all perpendicular; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

i. (1 point) Assume that the covariance matrices are $\Sigma_i = \alpha I$, for vanishingly small α. Design a query q in terms of the μ_i such that as before, $c \approx \frac{1}{2}(v_a + v_b)$, and provide a brief argument as to why it works.

ii. (2 point) Though single-headed attention is resistant to small perturbations in the keys, some types of larger perturbations may pose a bigger issue. Specifically, in some cases, one key vector k_a may be larger or smaller in norm than the others, while still pointing in the same direction as μ_a. As an example, let us consider a covariance for item a as $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α (as shown in figure 2). Further, let $\Sigma_i = \alpha I$ for all $i \neq a$.

When you sample $\{k_1, \ldots, k_n\}$ multiple times, and use the q vector that you defined in part i., what qualitatively do you expect the vector c will look like for different samples?

3 (d) Extra Credit

- (d) i
  - Goal: define vector q1 and q2
  - Consider how scaling affects attention weights
  - Consider how you can use your results from previous questions?

- (d) ii
  - Goal: Clearly explain what you expect the output to look like. Your description can written but should reference mathematical description of the vectors.
  - Analyze the covariance matrix and the effect of additional variance in one particular direction?
  - Consider the inner products between q1 and ka, and q2 and kb
  - Remember that problem specifies that we can ignore case where the inner product between qi and ka < 0

(d) **[3 points (Written, Extra Credit)] Benefits of multi-headed attention:** Now we'll see some of the power of multi-headed attention. We'll consider a simple version of multi-headed attention which is identical to single-headed self-attention as we've presented it in this homework, except two query vectors (q_1 and q_2) are defined, which leads to a pair of vectors (c_1 and c_2), each the output of single-headed attention given its respective query vector. The final output of the multi-headed attention is their average, $\frac{1}{2}(c_1 + c_2)$. As in question 3((c)), consider a set of key vectors $\{k_1, \ldots, k_n\}$ that are randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Also as before, assume that the means μ_i are mutually orthogonal; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

Thank You