Project 2: Transformer Language Modeling
CS 388 Project 2

Juan Diego Rodriguez
The University of Texas at Austin

1 Implementation details

For Part 1, I implemented a Transformer (Vaswani et al., 2017) with one head. I reused most of this code for Part 2, except that I used the pytorch[1] *nn.TransformerEncoderLayer* and *nn.TransformerEncoder* for the transformer layers. For Part 1, I found that trained positional encodings were crucial for good performance (a roughly 30 point drop when not using them). For Part 2, the language model struggled when using 480-dimensional feed forward layers; 2048 dimensional feed forward layers worked much better. I also found that I needed more than one head; 3 heads was sufficient to get a perplexity of 5.7.

2 Exploration 1: Attention Maps

Figure 1 shows an attention map of a single-head Transformer with one transformer layer for one of the character sequences in the dev set[2].

As expected, characters can be seen to (mostly) attend entirely to those characters occurring before them (thanks to the positional encoding and nature of the task, since masking wasn't used here). Note that every space character attends to all the previous space characters (and not others), while *a* in "and" attends to the previous *a* in "rank", the final *e* in "file" attends to the the previous *e* in "ed" (and similarly, for *n* and *d* in "and").

The attention maps look very different for a four-layer Transformer[3]. Here, the first layer's attention map was no longer lower-triangular (characters were attending to characters both before and after), and there was no discernible pattern in which character pairs had high attention.

[1]version 1.10.1, on CPU

[2]I also observed the same patterns in some of the other samples from the dev set.

[3]To get a four-layer Transformer to work well, I had to lower the learning rate from 1e-3 to 1e-4 (resulting in a .98 accuracy on the dev set).

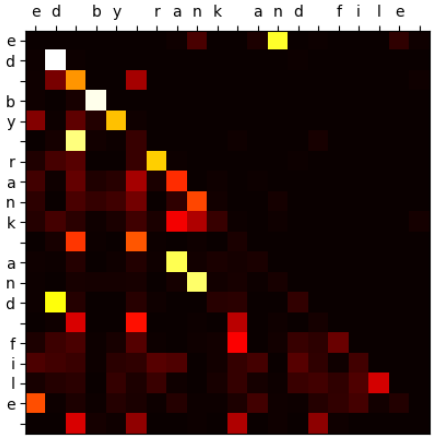

Figure 1: Attention map for a single-layer Transformer shows that characters are attending to those characters *before* them which match them.

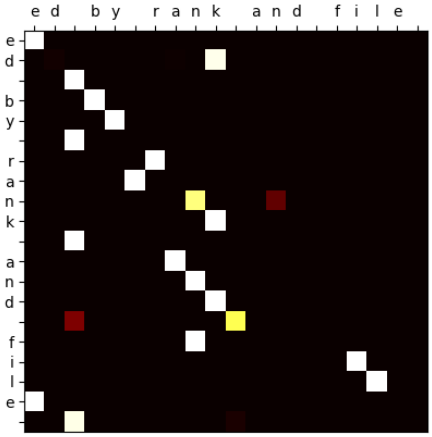

Figure 2: Attention in layer 2 of a 4-layer Transformer.

The attention maps for the second and third layers are shown in Figures 2 and 3 (the fourth attention map was similar to the third so I omit it).

Although there are a couple exceptions, for all attention maps in the 4-layer Transformer, most characters are only attending to *one* previous character (unlike for the 1-layer Transformer). Do characters attend to the right things? Not every layer does so perfectly (e.g., the second layer's second *d* incorrectly attends to *k*, and not to the previous *d*; and the fourth layer's second *a*, *d* and *e* do not attend

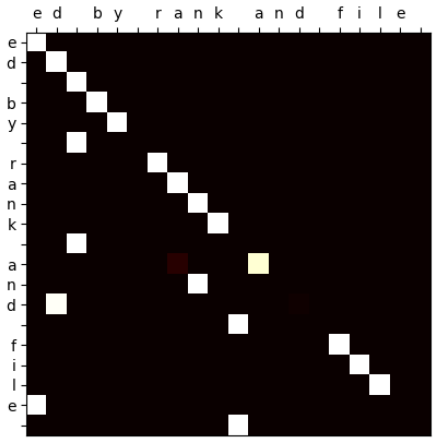

Figure 3: Attention in layer 3 of a 4-layer Transformer

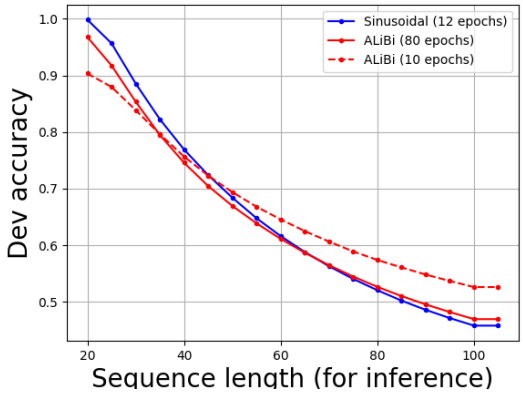

Figure 4: ALiBi and a Transformer with sinusoidal positional embeddings, evaluated on character sequences longer than they were trained on.

to their previous respective characters). However, when the layers are taken together (union), each character is correctly attending to the previous characters which match it. For example, the third space should attend to both previous spaces; the second layer's attention attends to the first space, while the fourth layer's attention attends to the second space.

3 Exploration 2: ALiBi variant

Here I implemented Attention with Linear Biases (ALiBi) from (Press et al., 2021). This model uses the same Transformer architecture, except that positional embeddings are not used, and a bias term is added to the attention scores for each head. This bias is a linear function of the distance (number of characters) between the key and the query. Intuitively, this forces the model to pay less attention to characters which are further apart. The magnitude of this effect depends on the slope parameter, which is different for each head[4].

I compared the accuracy of ALiBi against the Transformer without positional encodings[5] over character sequences with lengths ranging from 20 to 100 characters[6].

The results are shown in Figure 4. Both methods suffer significantly when evaluated on sequence lengths much longer than those they were trained on. Unfortunately, ALiBi (when trained to convergence for 80 epochs) did not fare much better than the Transformer. This is probably not surprising – after all, this task is very different from language modeling; here, the network needs to pay equal attention to distant characters in order to count them.

However, I noticed that an earlier checkpoint of ALiBi (trained for only 10 epochs) has a lower accuracy for 20-character sequences (.9 vs .97), but higher accuracy for character sequences over 35. Past 50 characters, the under-trained ALiBi starts to outperform the Transformer, and it continues to degrade much more gracefully (with a nearly 7 point improvement on 100-character sequences).

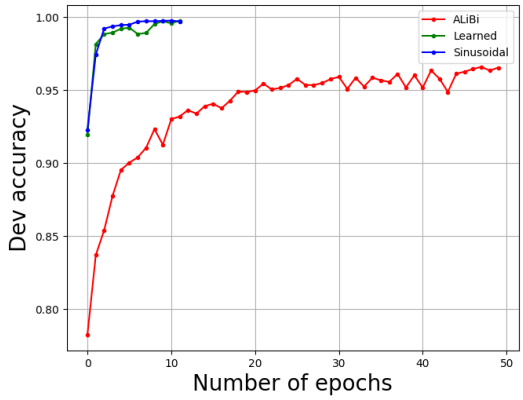

Figure 5: ALiBi takes longer to converge for the "previous character counting" task

Figure 5 shows that ALiBi also took longer to converge (the dev accuracy did not increase even after training for 80 epochs). On the other hand, the Transformers with learned and sinusoidal positional encodings converged in 10 epochs.

[4]The slope hyperparameter from (Press et al., 2021) (starting at $2^{-8/n}$, for n heads and decreasing geometrically) caused the model to have nearly identical accuracy as the Transformer with no positional encoding. Experimenting with $2^{-8/n}$, $2^{-16/n}$ and $2^{-32/n}$ showed that $2^{-32/n}$ yielded the best results. Other hyperarameters are listed in Appendix A.

[5]Unfortunately there is no obvious way to extrapolate with learned positional encodings (which has .9957 accuracy when evaluated with sequence lengths of 20).

[6]For these experiments, I created a new set of dev sets. First I formed a set where each sequence had length 100; for the dev sets with small sequences I just truncated each sample.

References

Ofir Press, Noah A Smith, and Mike Lewis. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

A Hyperparameters for Exploration 2

In order to ensure the slope hyperparameters of ALiBi work effectively, I decided to implement multihead attention, and used 8 heads for all my experiments (both for ALiBi and for the learned and sinusoidal positional embeddings). I used a learning rate of 1e-3 for the Transformer with sinusoidal embeddings (it overfit with 1e-4) and a learning rate of 1e-4 for ALiBi. Other hyperparameters are shown in Table 1

Hyperparameter	Value
Layers	1
Heads	8
Feedforward dimension	512
d_model	128
d_value	128
d_internal	16

Table 1: Hyperparameter values for ALiBi experiments