Training and Tokenization Set Up

This document describes the common training and tokenization set up used for all models in this project.

Tokenization

- All training sequences start with a BOS token and end with one or several END tokens that mark the positions where the label will be predicted
- Special tokens like BOS and END are indexed last in the embedding matrix. If the task contains tokens that represent numbers (either in the input or output vocabulary), those numeric tokens are indexed by the number they represent
    - e.g. token ‘2’ would correspond to W_E[2, :] or W_U[:, 2]
- For binary classification tasks the unembed vector indexed by zero corresponds to the label ‘False’, while the vector indexed by one is the label ‘True’
- All tokens defined in the embedding and unembedding matrix are used at some point during training

Training

For each algorithmic task I search over different architectures and pick the model with the lowest number of parameters that passes a task-specific validation accuracy threshold. The search space for the architecture is determined by:

Model Hyperparameter	**Values**
Number of layers	1, 2, 3, …
Number of heads	1, 2, 4, 8, …
Model dimension	16, 32, 64, …
Attention only	True, False

Additionally, the following specifications are common across all models and tasks

Model Hyperparameter	**Value**
MLP dimension	4 * Model dimension
Activation function	ReLU
Normalization type	LayerNorm[1]
Attention mask	Causal (autoregressive)

[1] I use the TransformerLens default where LayerNorm is applied separately to the input of attention heads and MLPs at each layer and only once to the residual stream right before the unembedding.

Initialization seed	42

Training Hyperparameter	**Value**
Learning rate	Linearly scaling from 1e-3 to 3.33e-4 at the last training step
Weight decay	0.0
Batch size	512
Train set size	100.000
Validation set size	10.000
Starting seed	42

For each task I chose the number of epochs to train all architectures on by manually looking at which point the training loss and validation accuracy settled down into desirable values. Each epoch the model was trained on an independently drawn sample of size ‘Train set size’ from the training distribution using a seed that increases by one each epoch. The validation set is drawn from the same distribution using a seed different from the one used at any training step[2].

[2] In the GitHub repository it’s specified how to use the starting seed to determine the seeds for each training and validation epoch. When running on my local computer, running twice on the same seed produced the exact same loss trajectory.