# ToyGPT: A Miniature GPT Model for Arithmetic Addition

Vansh Kapoor

## 1 Training the Transformer

(a) The device used for training is CPU and the total training time is 42.75 *sec*

(b) Final training loss: 0.17523 , Final training Accuracy: 0.9982, Final Test Accuracy: 0.998.

(c) For the following examples the model gives an incorrect output. Observe that the model the model is making an error in predicting the $2^{nd}$ digit (last but one digit) but is able to correctly predict the rest of the digits.

| Expression | GPT Claim | Ground Truth |
|---|---|---|
| 42 + 47 | 99 | 89 |
| 77 + 99 | 166 | 176 |
| 17 + 3 | 10 | 20 |

Table 1: Comparison of GPT Claims and Ground Truth

## 2 Exploring Positional Encoding

(a) For the `CausalSelfAttention` class,
In the `forward` function:

```
if self.use_rotary_positional_encoding:
    # TODO: implement rotary positional encoding
    q = self.rotate(q)
    k = self.rotate(k)
```

In the `rotate` function

```
x1, x2 = x[..., ::2], x[..., 1::2]
x_rot = torch.zeros(x.size())
x1_rot = x1*cos_theta - x2*sin_theta
x2_rot = x1*sin_theta + x2*cos_theta
x_rot[..., ::2] = x1_rot
x_rot[..., 1::2] = x2_rot
```

(b) The training loss curve for the three positional encodings for Pre-LN is shown below

(i) **Learnable Absolute Positional Encoding: APE**
- Model Size (number of parameters): 0.8 Million
- Final Training accuracy: 99.82%
- Final Test accuracy: 99.80%

(ii) **No Positional Encoding (NoPE)**

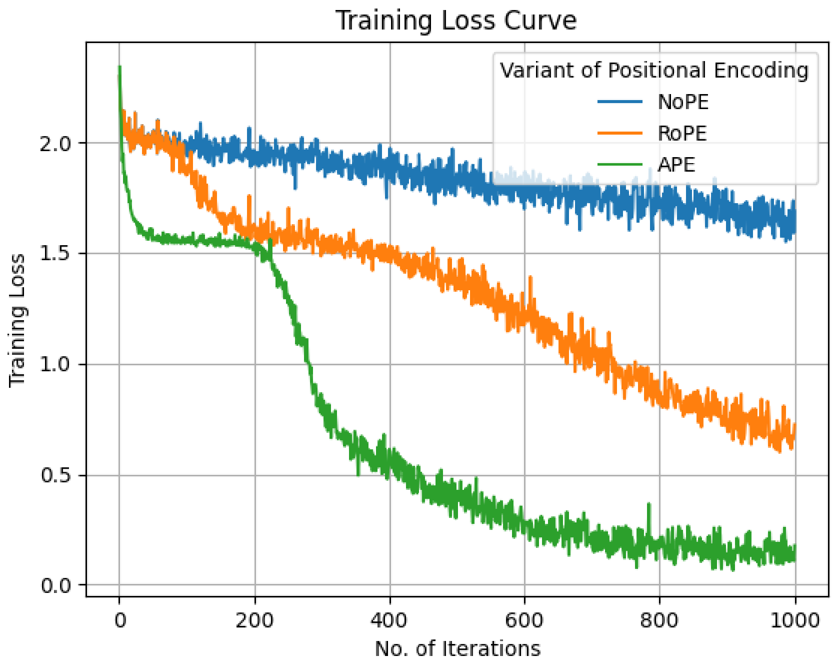

Figure 1: Training loss curve for different variants of PE for Pre-LN

- Model Size (number of parameters): 0.79 Million
- Final Training accuracy: 4.12%
- Final Test accuracy: 3.4%

(iii) **Rotary positional encoding (RoPE)**

- Model Size (number of parameters): 0.79 Million
- Final Training accuracy: 48.51
- Final Test accuracy: 45.60%

(c) We observe that both the final training and test accuracy follow the order APE > RoPE > NoPE. This is intuitive, as the positional information of each digit is crucial for capturing the output of addition, and a learnable encoding performs better than a fixed positional encoding; however, the number of parameters for the learnable APE encoding is higher than the other two due to the learnable parameters.

## 3 Layer Normalization

(a) For the `forward` function in `Block` class

```
def forward(self, x):
    if self.layer_norm_placement == 'Pre-LN':
        x = x + self.attn(self.ln_1(x))
        x = x + self.mlpf(self.ln_2(x))
    elif self.layer_norm_placement == 'Post-LN':
        x = self.ln_1(x + self.attn(x))
        x = self.ln_2(x + self.mlpf(x))
        # TODO: implement post layer normalization
    elif self.layer_norm_placement == 'No-LN':
        x = x + self.attn(x)
```

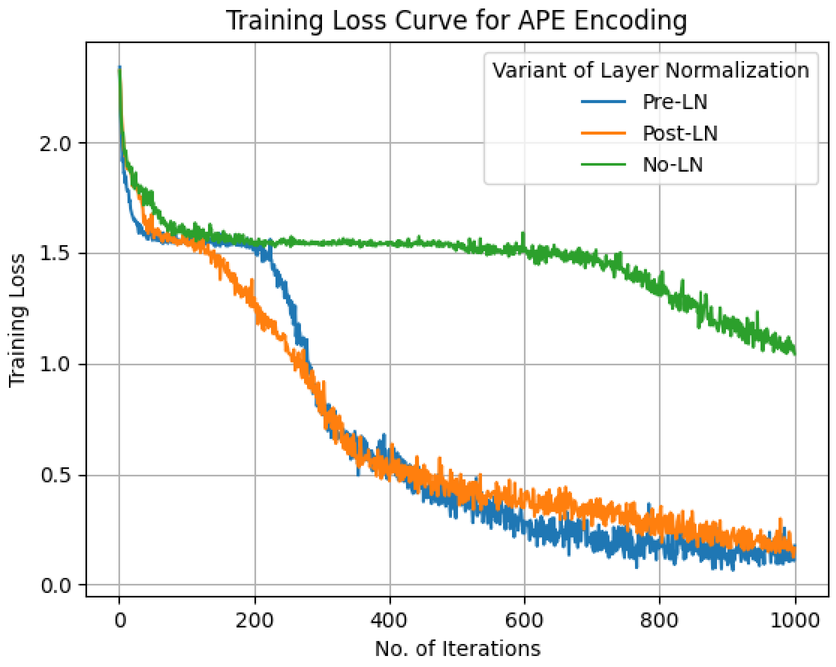

Figure 2: Training loss curve for different choices of Layer Normalization

```
        x = x + self.mlpf(x)
        # TODO: implement no layer normalization
    else:
        raise NotImplemented
    return x
```

(b) For APE (learnable absolute positional encoding ) positional Encoding, training loss curves with different variants of Layer Normalization are shown below

(a) **Pre-LN**
- Final Training accuracy: 99.82%
- Final Test accuracy: 99.80%

(b) **Post-LN**
- Final Training accuracy: 90.25%
- Final Test accuracy: 89.60%

(c) **No-LN**
- Final Training accuracy: 9.15%
- Final Test accuracy: 7.60%

(c) The final training loss for both Pre-LN and Post-LN is very similar, with Pre-LN performing better overall, including in final training and test accuracies. In contrast, without layer normalization, the training loss decreases slowly over iterations, and the final training loss is significantly higher, leading to lower final training and test accuracies.

## 4 Visualizing Attention Maps

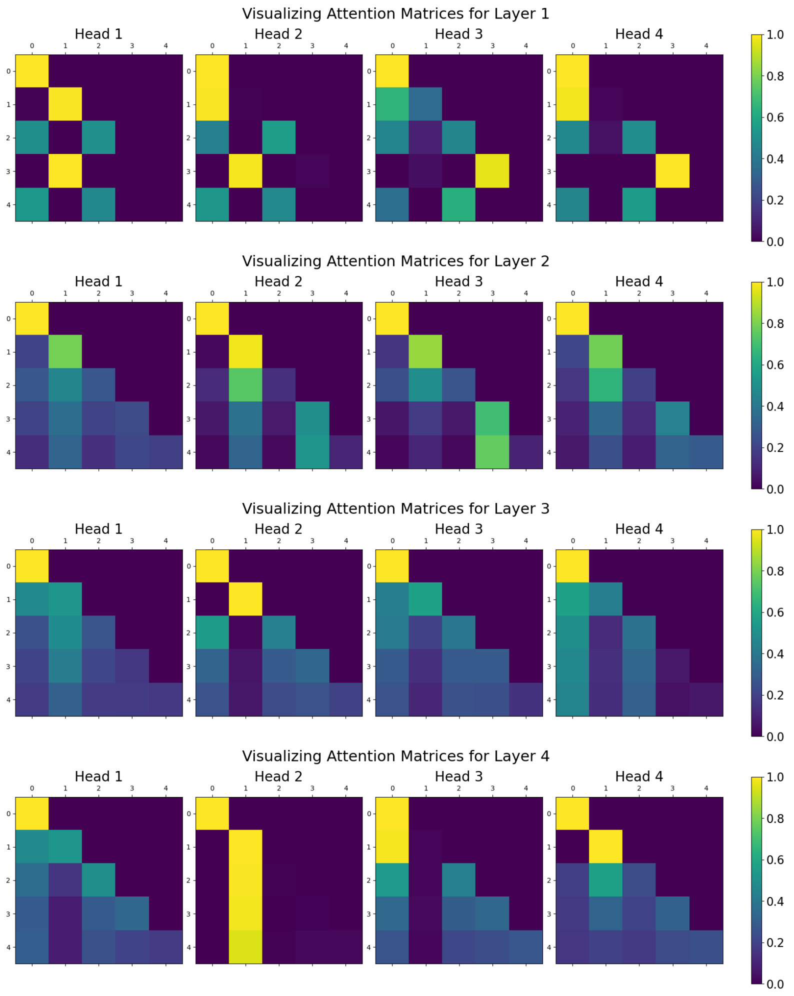

Figure 3: Visualizing attention matrices for each Head and Layer

(a)

(b) Observe that the attention matrices are lower-triangular matrices because of the mask that ensures that attention is only applied to the digits that appear to the left of the query digit (to maintain causality).

(c) We observe that roughly, as we go deeper, the attention matrices remain somewhat similar but become less sparse (particularly from layer 1 to layer 2) as they start attending to more of the previous inputs. However, they consistently remain lower triangular matrices. Additionally, within a given layer, the attention matrices are sort of similar — for instance, between Head 3 and Head 4 for Layer 2.