Group 20: DSA4212 Project

November 17, 2025

Group Composition:

1. Hoo Kai Sng, A0251727A, e0957315@u.nus.edu

1 Introduction

It is concerning that many AI projects fail today. One of the reasons is the lack of clear goals, and mismatch between the tool used and the goal needed to achieve. In this assignment, we would follow a step-by-step approach to define the possibilities and limitations of what we are able to achieve given the resources at hand. We would use the resources provided in the starter-kit, and expand on it wherever possible.

1.1 Problem

Given a text corpus $D = (x_1, x_2, ..., x_T)$, where each x_t is a character in the text8 dataset, our task is to train a model f_θ that learns the conditional distribution:

$$\max_L(f_\theta) = P_\theta(x_{t=1}|x_{t-L+1}, ..., x_t), \tag{1}$$

for a context window of length L. The model should predict the most likely next character given the previous L characters.

1.2 Dataset

We would also use the text8 dataset provided in starter-kit. The dataset is divided into 90 million characters for training and 5 million characters for testing. The dataset comprises 26 unique alphabets + space, which makes for a total of 27 distinct alphabets.
A preliminary examination of the dataset shows that, while many sequences form valid words, others are grammatically incoherent or semantically nonsensical. This limits the degree of linguistic structure available to the model. Nevertheless, we aim to approximate the underlying generative process to the extent that such a function exists.

1.3 Goal

1. Implement and train a small transformer-based model for next-character prediction.
2. Evaluate the model's performance on a held-out test set
3. Report the accuracy (percentage of correct predictions) when predicting the next character given the previous L characters.
4. Optimize the value of L for the best performance

2 Transformer

In this assignment, we would focus on decoder-only transformer provided. In Equation (1), we predict the next character given the previous L characters. The token here is the characters. For a token x_i, we can encode it by giving it:

1. Word embedding in a d_{model} dimensional space.
2. position encoding i

2.1 Casual self-attention

For each token x_i, we compute its query, key and value vectors:

$$\mathbf{Q}_i = W_q\text{Embed}(i), \quad \mathbf{K}_i = W_k\text{Embed}(i), \quad \mathbf{V}_i = W_v\text{Embed}(i), \tag{2}$$

where the projection matrices W_q, W_k, W_v are shared across all positions.
In this scenario, a decoder-only transformer applies casual masking, so each token can only attend to itself and earlier positions. Stacking all token representations into matrices Q, K, V the self-attention output is.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}} + M_{\text{casual}})V \tag{3}$$

where M_{casual} sets all positions $j > i$ to $-\infty$.
Because Q, K, V contain all the' representations of all the tokens in a single batch, the attention outputs of all all tokens are computed in parallel, which is a key advantage of the transformers.

$$\text{Embeddings} \to \underbrace{[\text{LN} \to \text{MHA} \to + \to \text{LN} \to \text{FFN} \to +]^N}_{N \text{ Decoder Layers}} \to \text{Final LN} \to \text{Linear } (d_{\text{model}} \times V) \to \text{Logits}$$

In this assignment, we would focus mainly on the impact of n_{heads} (number of attention heads) and n_{layers} (number of decoder blocks/layers).

3 Experiments and Tuning

In this assignment, we would explore the following strategies in order to find a balance between accuracy and run-times.

1. Hyperparameter tuning
2. Loss function
3. Schedulers

3.1 Hyperparameter tuning

Motivation in using Bayesian Optimization.

- Expensive: Number of parameters start from the millions, each evaluation of F_θ is costly.
- No gradients: $\nabla f(x)$ is not available.
- Small budget

We use python's scikit-optimize library to execute BO. The most sensitive hyperparameters were L, n_{layers}, n_{heads} and $\ell_{\text{learning\_rate}}$.

Algorithm 1: Stable Bayesian Hyperparameter Optimization

Input: Maximum BO iterations n_{iter}

Output: Stable hyperparameter configuration $\mathbf{h}^\star$

flag $\leftarrow$ True
$n_{\text{iter}} \leftarrow 50$
while flag **do**
  $H \leftarrow$ empty list
  for $i = 1, \ldots, 10$ **do**
    seed $\leftarrow$ RandomSeed()
    $\mathbf{h}_i \leftarrow$ BayesOpt(seed, niter)
    append $\mathbf{h}_i$ to H
  end for
  $\mathbf{h}^\star \leftarrow$ MajorityVote(H)
  if $\mathbf{h}^\star$ is undefined **then**
    niter $\leftarrow$ niter + 10
  else
    flag $\leftarrow$ False
  end if
end while
return $\mathbf{h}^\star$

We run this algorithm with low n_{iter} to reduce unnecessary computation, while a sufficiently large n_{calls} preserves the ability of BO to identify the optimum with high confidence.

Hyperparameter	**Majority (or Tie)**
T	32
d_{model}	152, 256 (tie)
n_{layers}	4
n_{heads}	8
loss_signal	Weighted
learning_rate	0.0001
Lowest Validation Loss	2.4494

Table 1: Majority hyperparameters (including ties) across 10 Bayesian optimization runs.

3.2 Loss function

Changing the way we compute the loss function can shift the model's learning focus. For batch size B, we compute cross-entropy loss as follows.

$$\sigma_i = \log\left(\frac{\exp(x_{iy_i})}{\sum_j \exp(x_{ij})}\right) \qquad \forall i \in B \tag{4}$$

In this assignment, we would experiment with 3 different loss functions and observe if there is any performance improvement. Note that we would use the hyperparameters obtained earlier.

$$L_{\text{mean}} = \frac{1}{B}\sum_{i=1}^{B}\sigma_i$$

$$L_{\text{last}} = \sigma_B$$

$$L_{\text{weighted}} = \frac{1}{B^2}\sum_{i=1}^{B} i\,\sigma_i$$

We were able to halve the training error by just shifting the loss function from σ_{mean} to $\sigma_{weighted}$. However, testing errors remain relatively similar between the different loss functions. Given reduced training error while test error remains essentially constant, we hypothesize that for training data

$$D = (x_1, x_2, \ldots, x_T),$$

the learned model f_θ converges toward an underlying *training-specific* function f^*_{train}, which differs from the true test-time function f^*_{test}. Formally,

$$f_\theta \xrightarrow{\text{train}} f^*_{\text{train}} \qquad \text{with} \qquad f^*_{\text{train}} \neq f^*_{\text{test}}.$$

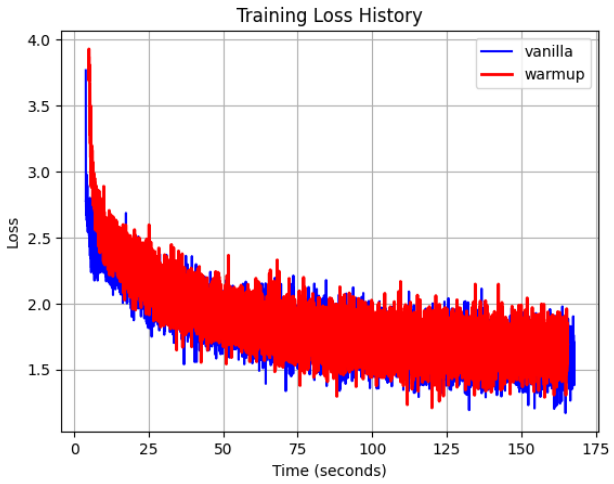

Figure 1: Training Loss: With vs Without Warmup

3.3 Learning Rate Scheduling

Transformer models are highly sensitive to the choice of learning rate schedule. As demonstrated in [VSP+17], performance improves substantially when the learning rate (i) begins at a small value, (ii) increases linearly during a short warmup phase, and (iii) subsequently decays smoothly.

In preliminary experiments using a constant learning rate, we observed large oscillations in the training loss and approximately 2,000 iterations were required before convergence behavior appeared. To address this instability, we evaluate two modifications commonly used in modern transformer training pipelines:

1. **Warmup scheduling:** a linear increase in the learning rate over an initial warmup period, intended to stabilize early optimization dynamics.

2. **Cosine decay scheduling:** a smooth annealing of the learning rate, which prevents premature convergence and encourages better long-term optimization behavior.

We compare these schedules against the baseline constant learning rate to determine whether warmup and decay improve training stability and final validation performance. Results are shown in Figure 1

4 Conclusion

Our initial round of hyperparameter tuning provided valuable insight into an appropriate model architecture, preventing us from prematurely adopting an overly complex design. Several parameters converged to boundary values, which in turn informed our subsequent methodological choices. We also examined how different loss formulations alter the model's optimization objective, offering a possible perspective on the underlying functions driving train–test behavior.

Finally, we explored warmup steps and learning–rate schedulers; although our results were inconclusive, they suggest that the effects of warmup may become more pronounced when applied to larger or more expressive models. Future work should therefore investigate these scheduling strategies under higher–capacity settings.

References

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.