Mainrun - Generating Hacker News Headlines

Songyan Teng

1. Starting Point

The original code I was given achieved a validation loss of around **1.754** on the Hacker News dataset. It was a decent starting point, but I felt the code was somewhat limited in certain aspects relating to the problem, such as using SGD as the optimiser, sequential batching, and lacking support for modern GPU optimisations. My first step was to contextualise the problem of working with Hacker News headlines, which, upon inspection, mostly consisted of just a few words but were highly heterogeneous. I wanted to begin by considering the model and training process itself, as these fundamentals would have the greatest impact. Fine-tuning the hyperparameters would come later, once the model's structure had been established.

2. The Model

The Transformer blocks were already in place, but the attention mechanism used a manually written masked softmax with a triangular mask. While that approach works, PyTorch now provides the built-in `scaled_dot_product_attention` method which is faster, numerically safer, and automatically leverages FlashAttention if available. Replacing the old attention code made sense, as it reduced boilerplate, lowered the risk of mistakes, and delivered a more efficient implementation.

I also left some aspects unchanged. I kept the overall model structure (stacked Transformer blocks with residuals and MLPs) not only because it is a proven design (Lin et al., 2022), but also because holding it constant provides a stable baseline for comparison. This meant that any improvements could be attributed to training procedures or implementation refinements, rather than confounding changes to the core architecture. The embeddings were already tied between input and output, which is considered best practice, so I left that unchanged as well (Press & Wolf, 2016). My focus was therefore on refining the implementation details and training dynamics without straying from the core design.

3. Data Pipeline

The dataset itself was fixed, but I wanted to refine how it was sampled. The baseline loop always iterated through the data sequentially. My concern was that this introduced a kind of curriculum, where the model would always encounter the same early titles at the start of training and only reach others later. That's acceptable if the goal is curriculum learning, but in this case, I felt it risked overfitting to ordering effects (Soviany et al., 2021).

To address this, I rewrote the batching process to use random contiguous crops. Each batch is now drawn from a random position in the dataset, forcing the model to generalise across the full distribution. Since titles about technology news are short, noisy, and often appear in bursts (e.g. multiple posts about the same new technology), random sampling helped prevent the model from being biased by temporal or topical clusters.

4. Training

This is where I made the most significant changes. The baseline used SGD with cosine annealing, but SGD didn't seem like the best fit for this task. Hacker News titles are short, sparse, and noisy; as a result, I considered adaptive methods such as AdamW to be better suited. Given the small and heterogeneous dataset, I expected gradients to be noisy. AdamW, with its adaptive step sizes and decoupled regularisation, should handle this volatility more effectively than vanilla SGD (Zhang et al., 2019). I also added decoupled weight decay, since it is now standard practice for Transformers.

I then introduced a warmup schedule. With small snippets of text, early updates can easily destabilise training, so a warmup phase helps prevent the optimiser from overshooting. After warmup, I retained the cosine schedule, which remains a good fit for decaying over a fixed number of epochs.

Next, I incorporated gradient accumulation. Instead of training with a batch size of 64, I accumulated gradients over four steps, resulting in an effective batch size of 256. This allowed me to keep GPU memory usage manageable while still benefiting from the stability of larger batches.

Finally, I enabled mixed-precision training. This provided faster training, reduced memory consumption, and leveraged the maturity of PyTorch's Automatic Mixed Precision. Since Hacker News titles are relatively short, I expected the precision trade-offs to have little impact on convergence.

5. Experimentation and Logging

Changing these fundamentals allowed me to reduce the validation loss to around **1.493**, a significant improvement over the 1.754 baseline, achieved before any hyperparameter tuning.

The original code lacked systematic logging. Results were written to a single file, and running multiple experiments would overwrite previous outputs. I refactored this so that each run generates a unique log, and I added a sweep runner that automatically records results in a CSV. To me, this was not just about convenience but also about making experiments reproducible and comparable. With this infrastructure in place, I could then run proper sweeps over dropout, weight decay, vocabulary size, and model dimensions.

6. Hyperparameter Tuning

Once the core implementation had been updated, I ran a series of controlled sweeps to tune the hyperparameters. The process was staged so that each step reduced uncertainty before narrowing the search space, which I believe to be an approach that suited the resource and time constraints of training.

6.1. Initial: Broad Sweep

I began with a broad exploration over dropout, weight decay, learning rate, and model capacity (depth × width), sampling 80 random configurations from a large candidate grid. The goal was to build a general picture of the hyperparameter landscape without exhaustively evaluating every combination, which would have been computationally infeasible under the given time and resource constraints.

This stage revealed that the most significant driver of improvement was the optimiser's learning rate, followed by modest gains from using slightly smaller model configurations. Across all runs, the 16k vocabulary size consistently produced the best results, while varying the warmup ratio (0.05 vs 0.1) had negligible impact.

6.2. Tier 1: Optimiser Parameters

Holding model size (6 layers, 640 hidden), vocabulary size (16k), and warmup ratio (0.1) constant, I swept dropout (0-0.1) and weight decay (0-0.1) across learning rates from $(2e{-}4)$ to $(1e{-}3)$. This showed that AdamW was highly robust, with most optimal combinations clustering around **1.32** loss. The experiments again showed that learning rate had the greatest impact on validation loss, with higher rates proving more effective. This makes sense in the present context: the dataset is relatively small, the sequences (Hacker News titles) are short and noisy, and training is capped at only seven epochs. A higher learning rate allows the optimiser to make faster progress in this limited budget, quickly capturing broad patterns across heterogeneous data. With AdamW, warmup, gradient clipping, and cosine decay providing stability, the model can safely exploit a larger step size without diverging, leading to better convergence within the time and resource constraints. Varying the weight decay and dropout rates had little effect on performance, suggesting that regularisation was not a major bottleneck in this setting. With a relatively small dataset and a modestly sized model, the dominant factor remained the learning rate rather than additional forms of regularisation. The best performance was achieved near lr = $(1e{-}3)$, dropout = 0.05, weight_decay = 0.02.

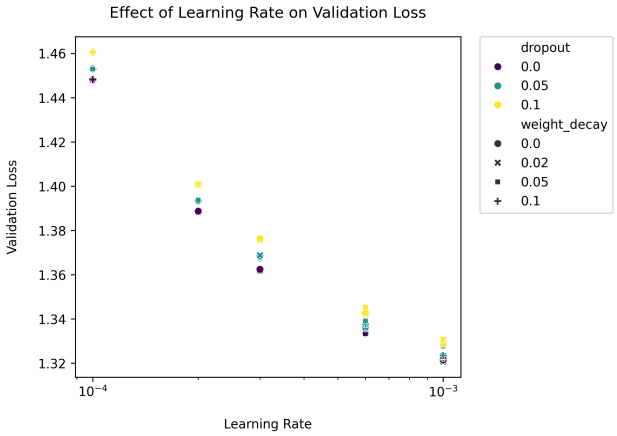

Figure 1. Effect of learning rate, dropout, and weight decay on validation loss. Higher learning rates generally reduced validation loss, whereas regularisation (via dropout and weight decay) had less impact on performance.

6.3. Tier 2: Model Capacity

Next, I varied both depth (5-7 layers) and width (hidden size 512-768) while holding optimiser settings fixed at their Tier-1 best. The sweep indicated that moderate-capacity configurations achieved the best generalisation, with performance improving when depth was reduced and width moderately increased. The strongest result came from a 5-layer model with a hidden size of 704, which achieved the lowest validation loss (1.305). Increasing capacity further, such as to 7 layers with hidden size 768, provided no benefit and, in some cases, slightly degraded performance, likely due to overfitting on the relatively small dataset. At the other end, reducing the hidden size too much, such as to 5 layers with hidden size 512, also reduced performance, suggesting the model lacked sufficient expressiveness. This points to an optimal balance where the network is large enough to capture heterogeneous patterns in the data, but not so over-parameterised that it memorises them.

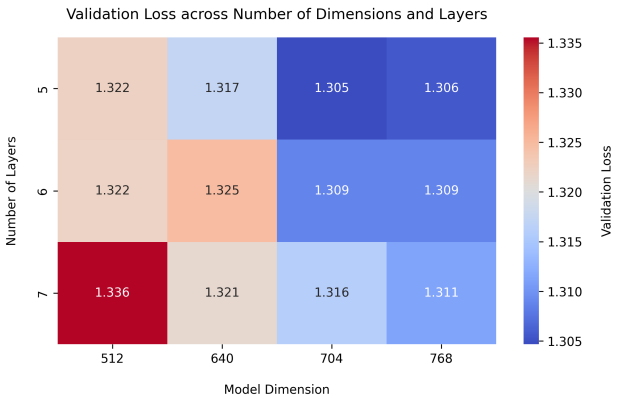

Figure 2. Validation loss across combinations of model dimension and number of layers. Models with higher dimensions (704-768) consistently achieve lower validation losses, with the best performance observed at 704 dimensions and 5 layers.

6.4. Tier 3: Fine Learning Rate Tuning

Finally, I zoomed in on the learning rate. A confirmatory sweep in the $(7.5\text{e}{-4})$ to $(3\text{e}{-3})$ range pinpointed an optimum at $(1.25\text{e}{-3})$, yielding the best observed validation loss of **1.301**. Lower rates tended to underfit slightly, while very high rates gave mild instability, making the mid-range a safe and effective choice.

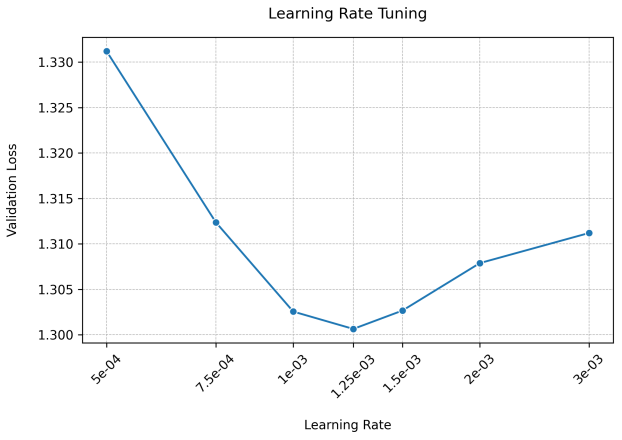

Figure 3. Validation loss across different learning rates. The lowest validation loss is achieved around a learning rate of $1.25\text{e}{-3}$, after which performance begins to degrade, indicating an optimal range for training stability and convergence.

6.5. Final Results

After performing these steps, we arrive at this final hyperparameter configuration:

Hyperparameter	Value
vocab_size	16000
n_layer	5
d_model	704
dropout	0.05
weight_decay	0.02
lr	1.25e-3
batch_size	64
accumulation_steps	4
warmup_ratio	0.1

7. Sample Outputs

Out of curiosity, I implemented a simple `generate()` function and used the updated model to create Hacker News-style headlines. Below are the first ten generations produced:

1. Ask HN: How to keep an open source web-source GitHub site?
2. Show HN: A static framework for how to learn to use your business.
3. Show HN: A simple way to read your team-in-driven design
4. The First-Techest Guide to the Brain
5. The Power of Life Of The Story of American Education
6. The Case of Life
7. A new database for the Mac's Guide to Software
8. Ask HN: How do you use your site?
9. A New Design to Make the New Yorker
10. The Future of All

References

Lin, T., Wang, Y., Liu, X., and Qiu, X. A survey of transformers. *AI Open*, 3:111–132, 2022. ISSN 2666-6510. doi: 10.1016/j.aiopen.2022.10.001. URL http://dx.doi.org/10.1016/j.aiopen.2022.10.001.

Press, O. and Wolf, L. Using the output embedding to improve language models, 2016. URL https://arxiv.org/abs/1608.05859.

Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. Curriculum learning: A survey, 2021. URL https://arxiv.org/abs/2101.10382.

Zhang, J., Karimireddy, S. P., Veit, A., Kim, S., Reddi, S. J., Kumar, S., and Sra, S. Why are adaptive methods good for attention models?, 2019. URL https://arxiv.org/abs/1912.03194.