Week 4 Lab Report (Lab 3)

Youness Anouar

UM6P College of Computing
Generative AI Course

February 20, 2026

1 Introduction

The Transformer architecture, first introduced by Vaswani et al. (2017), has become the dominant paradigm in modern natural language processing. Its core mechanism, *multi-head self-attention*, enables each token in a sequence to attend directly to every other token, thereby capturing long-range dependencies far more effectively than earlier recurrent approaches.

The objective of this laboratory is threefold:

1. **Build** a multi-head Transformer Encoder entirely from scratch in PyTorch, without relying on any pre-built transformer modules.
2. **Train** the encoder on a real-world text classification task using the *20 Newsgroups* dataset (four selected categories, approximately 3 900 documents).
3. **Benchmark** five distinct hyper-parameter configurations, varying learning rate and batch size, to study their effect on convergence speed, generalisation, and final accuracy.

This report describes the architecture, the dataset preparation pipeline, the experimental setup, and provides a thorough analysis of the results obtained.

2 Model Architecture

The model follows a standard Transformer Encoder design with *pre-norm* residual connections. It is composed of the following components, stacked in sequence.

2.1 Embedding Layer

Each input token is mapped to a dense vector through a **learned token embedding** of dimension $d_{\text{model}} = 64$. Positional information is injected via a separate **learned positional embedding** of the same dimension, supporting sequences of up to 512 tokens. The two embeddings are summed element-wise and passed through a dropout layer ($p = 0.1$).

2.2 Multi-Head Self-Attention

The attention mechanism splits the d_{model}-dimensional representation into $h = 4$ heads, each of dimension $d_k = d_{\text{model}}/h = 16$. For an input $\mathbf{X} \in \mathbb{R}^{B \times T \times D}$, the scaled dot-product attention for each head is computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\,\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{1}$$

where $\mathbf{Q} = \mathbf{X}\,\mathbf{W}^Q$, $\mathbf{K} = \mathbf{X}\,\mathbf{W}^K$, and $\mathbf{V} = \mathbf{X}\,\mathbf{W}^V$ are linear projections. The outputs of all heads are concatenated and projected through a final linear layer $\mathbf{W}^O$. Dropout is applied to the attention weights.

2.3 Feed-Forward Network

Each encoder layer contains a position-wise feed-forward network consisting of two linear transformations with a GELU activation in between:

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2\,\text{GELU}(\mathbf{W}_1\,\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{2}$$

The inner dimension is $d_{\text{ff}} = 128$ (twice d_{model}). Dropout ($p = 0.1$) is applied after the activation and after the second linear layer.

2.4 Encoder Layer and Stack

Each encoder layer applies the following **pre-norm** residual pattern:

$$\mathbf{x} \leftarrow \mathbf{x} + \text{MHA}\big(\text{LayerNorm}(\mathbf{x})\big) \tag{3}$$
$$\mathbf{x} \leftarrow \mathbf{x} + \text{FFN}\big(\text{LayerNorm}(\mathbf{x})\big) \tag{4}$$

Two such layers ($n_{\text{layers}} = 2$) are stacked, followed by a final layer normalisation.

2.5 Classification Head

Rather than using a special `[CLS]` token, the model applies **mean pooling** over all non-padding token representations:

$$\mathbf{p} = \frac{\sum_{t=1}^{T} \mathbf{h}_t \cdot \mathbb{1}[x_t \neq \text{PAD}]}{\sum_{t=1}^{T} \mathbb{1}[x_t \neq \text{PAD}]} \tag{5}$$

The pooled vector $\mathbf{p} \in \mathbb{R}^{d_{\text{model}}}$ is then projected to the number of classes through a single linear layer.

2.6 Model Size

The total number of trainable parameters with the default configuration ($V = 5\,002$, $d = 64$, $L = 2$, $h = 4$) is **419 968**, making this a lightweight model suitable for CPU training.

3 Dataset and Preprocessing

3.1 20 Newsgroups (4 Categories)

The dataset is drawn from scikit-learn's `fetch_20newsgroups` utility. Four thematically distinct categories were selected to ensure a clear separation of topics while keeping the dataset small enough for rapid experimentation:

Table 1: Selected categories and their sample counts.

Category	Documents
`comp.graphics`	973
`rec.sport.hockey`	999
`sci.space`	987
`talk.politics.guns`	910
Total	**3 869**

Headers, footers, and quoted replies were removed during loading to prevent the model from exploiting non-content metadata. The class distribution is approximately balanced, with the smallest class (`talk.politics.guns`) containing 910 documents and the largest (`rec.sport.hockey`) containing 999.

3.2 Tokenisation

A simple word-level tokeniser was implemented:

1. Convert the text to lowercase.
2. Extract all alphabetic tokens using a regular expression (`[a-z]+`).
3. Build a vocabulary from the 5 000 most frequent words across the full corpus.
4. Map each word to an integer index (indices 0 and 1 are reserved for `PAD` and `UNK` tokens respectively), giving a total vocabulary size of **5 002**.

All documents are truncated or zero-padded to a fixed length of **128 tokens**.

3.3 Data Splits

The dataset was divided using stratified sampling to preserve the class distribution in each split:

Table 2: Data split sizes.

Split	Samples	Proportion
Training	2 321	60%
Validation	774	20%
Test	774	20%

4 Experimental Setup

4.1 Fixed Hyper-parameters

The following parameters were held constant across all five experiments in order to isolate the effect of learning rate and batch size:

Table 3: Fixed hyper-parameters.

Parameter	Value
Model dimension (d_{model})	64
Number of layers (n_{layers})	2
Attention heads (h)	4
Feed-forward dimension (d_{ff})	128
Dropout rate	0.1
Maximum sequence length	128
Optimiser	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
Learning rate schedule	Cosine annealing
Epochs	15
Loss function	Cross-entropy
Random seed	42

4.2 Varied Hyper-parameters

Five configurations were defined, varying only the learning rate and the training batch size:

Table 4: Experiment configurations.

Experiment	Learning Rate	Batch Size	Rationale
`baseline`	1×10^{-3}	64	Standard configuration
`low_lr`	1×10^{-4}	64	Conservative learning rate
`high_lr`	5×10^{-3}	64	Aggressive learning rate
`small_batch`	1×10^{-3}	16	More frequent gradient updates
`large_batch`	1×10^{-3}	256	Fewer, more stable gradient steps

Each experiment was trained from scratch with the same random seed to ensure fair comparison. The cosine annealing schedule gradually reduces the learning rate to near zero over the 15 epochs, which generally helps the model settle into a good minimum during the final training epochs.

5 Results

5.1 Validation Performance Summary

Table 5 presents the key validation metrics for all five experiments, sorted by best validation accuracy.

Table 5: Validation results across all experiments (sorted by best validation accuracy).

Experiment	LR	BS	Best Val Acc	Final Val Acc	Final Val F1
`high_lr`	5×10^{-3}	64	**0.8682**	**0.8656**	**0.8664**
`small_batch`	1×10^{-3}	16	0.8217	0.8152	0.8141
`baseline`	1×10^{-3}	64	0.7765	0.7739	0.7729
`large_batch`	1×10^{-3}	256	0.6047	0.6008	0.5977
`low_lr`	1×10^{-4}	64	0.4664	0.4612	0.4558

The `high_lr` experiment clearly outperforms all others, achieving a best validation accuracy of **86.82%** and a macro F1-score of **0.8664**.

5.2 Training Curves

Figure 1 displays the training loss, validation loss, and validation accuracy over 15 epochs for all five configurations.

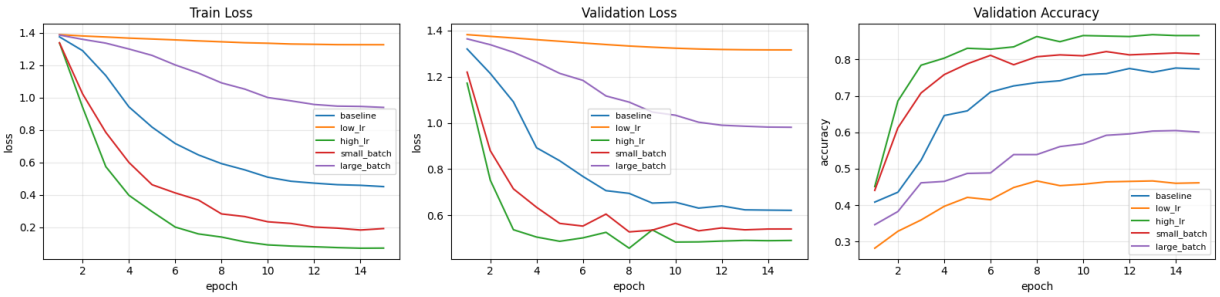

Figure 1: Training curves across all five experiments. **Left:** Training loss. **Centre:** Validation loss. **Right:** Validation accuracy. The `high_lr` configuration converges fastest and reaches the highest accuracy, while `low_lr` and `large_batch` struggle to learn effectively within 15 epochs.

Several observations can be drawn from these curves:

- `high_lr` ($\text{lr} = 5 \times 10^{-3}$) converges rapidly, reaching low training loss within the first few epochs and maintaining the highest validation accuracy throughout.
- `small_batch` ($\text{bs} = 16$) benefits from more frequent parameter updates per epoch, resulting in faster convergence than the baseline, though it exhibits slightly noisier loss curves.
- `baseline` ($\text{lr} = 10^{-3}$, $\text{bs} = 64$) shows steady but slower convergence. The model is still improving at epoch 15, suggesting that additional training epochs could further boost performance.
- `large_batch` ($\text{bs} = 256$) converges slowly because each epoch contains far fewer gradient updates (approximately 9 steps per epoch compared to 37 for the baseline). The learning rate of 10^{-3} appears insufficient for this batch size.
- `low_lr` ($\text{lr} = 10^{-4}$) barely moves from its initialisation, after 15 epochs the model has not yet broken past 47% accuracy, which is only marginally above the random baseline of 25%.

5.3 Experiment Comparison

Figure 2 provides a horizontal bar chart comparing the best validation accuracy achieved by each experiment.

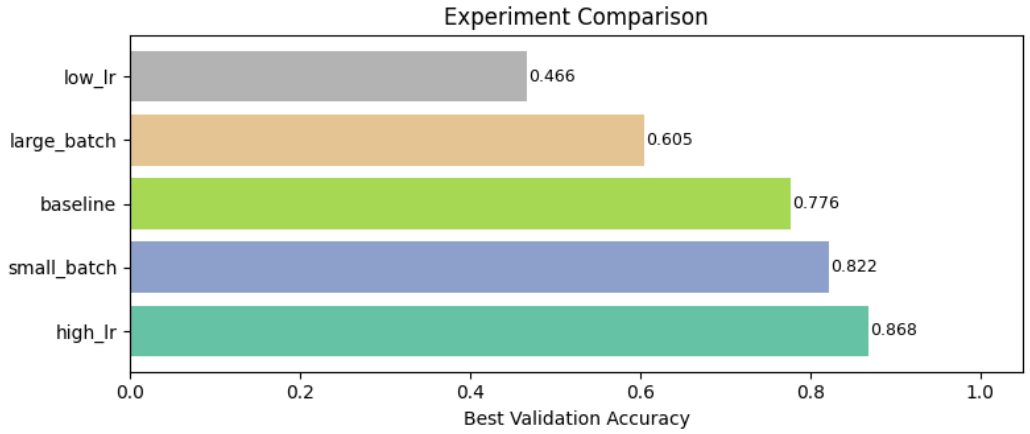

Figure 2: Best validation accuracy per experiment. The `high_lr` configuration dominates, followed by `small_batch` and `baseline`.

The gap between the top-performing `high_lr` and the worst-performing `low_lr` is over 40 percentage points, highlighting just how sensitive this small Transformer model is to the choice of learning rate.

6 Test Evaluation

The best-performing experiment (`high_lr`) was evaluated on the held-out test set of 774 documents.

6.1 Overall Test Metrics

Table 6: Test set performance of the best model (`high_lr`).

Metric	Value
Test Loss	0.7099
Test Accuracy	**0.8191** (81.91%)
Test F1 (macro)	**0.8187**

The test accuracy (81.91%) is slightly lower than the best validation accuracy (86.82%), which is expected since the validation set was indirectly used for model selection. The small gap indicates that the model generalises reasonably well.

6.2 Per-Class Performance

Table 7 provides precision, recall, and F1-score for each of the four categories.

Table 7: Per-class test metrics (best model: `high_lr`).

Category	Precision	Recall	F1-score	Support
`comp.graphics`	0.8710	0.8308	0.8504	195
`rec.sport.hockey`	0.8689	0.8950	0.8818	200
`sci.space`	0.7202	0.7970	0.7566	197
`talk.politics.guns`	0.8293	0.7473	0.7861	182
Macro average	0.8223	0.8175	0.8187	774

Key observations:

- `rec.sport.hockey` achieves the highest F1-score (0.8818), likely because sports-related vocabulary is highly distinctive and does not overlap much with the other categories.
- `sci.space` has the lowest precision (0.7202), suggesting that the model sometimes misclassifies documents from other categories as belonging to this class. This could be due to shared scientific or technical vocabulary with `comp.graphics`.
- `talk.politics.guns` shows relatively lower recall (0.7473), meaning a proportion of political documents are being misassigned to other categories.

6.3 Confusion Matrix

Figure 3 shows the confusion matrix for the test set predictions.

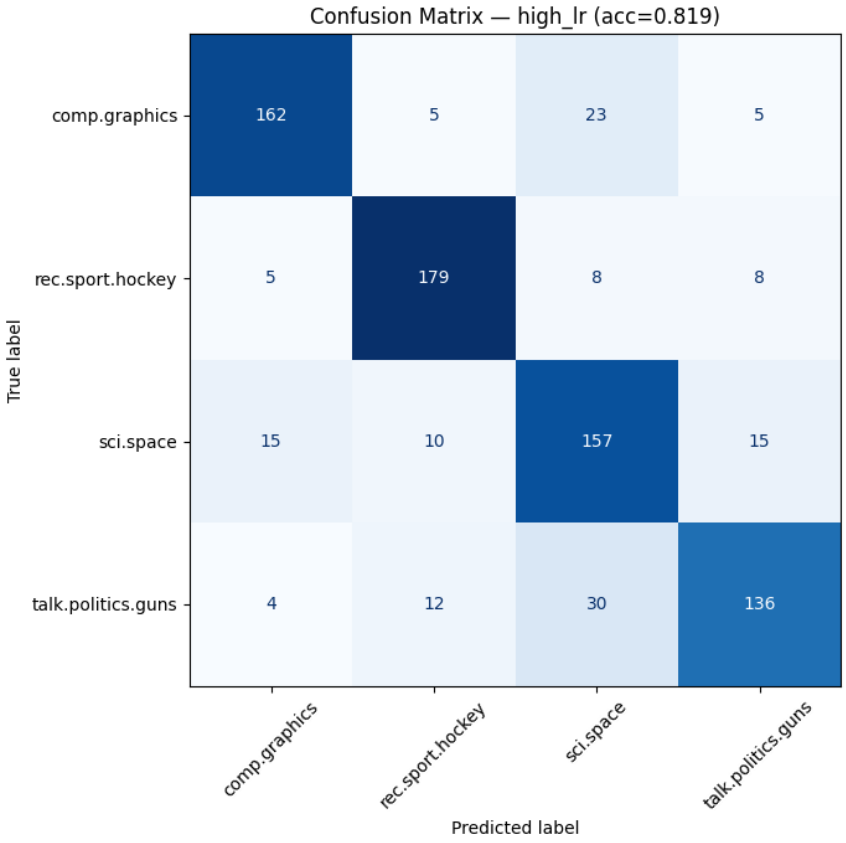

Figure 3: Confusion matrix on the test set for the `high_lr` model (accuracy = 81.9%). The strongest diagonal values are found for `rec.sport.hockey` and `comp.graphics`, confirming the per-class F1 trends.

The confusion matrix confirms that the most frequent misclassifications occur between `sci.space` and `comp.graphics`, which is understandable as both categories contain technical and scientific content.

7 Discussion

7.1 Effect of Learning Rate

The learning rate proved to be the single most influential hyper-parameter in this benchmark. With the architecture and training budget held constant, a five-fold increase from the baseline

($10^{-3} \rightarrow 5 \times 10^{-3}$) yielded a +9 percentage point improvement in validation accuracy, whereas a ten-fold decrease ($10^{-3} \rightarrow 10^{-4}$) resulted in a model that had barely begun to learn after 15 epochs.

This behaviour can be explained by the interaction between the learning rate and the cosine annealing schedule: when the initial rate is too low, the schedule reduces it even further in later epochs, effectively stalling the optimisation. Conversely, the higher learning rate allows the model to explore the loss landscape aggressively early on and then gradually settle as the rate decays.

7.2 Effect of Batch Size

Reducing the batch size from 64 to 16 increased the number of weight updates per epoch by a factor of four (from ∼37 to ∼146 steps), which accelerated convergence and improved validation accuracy by roughly 4.5 percentage points. On the other hand, increasing the batch size to 256 reduced the number of updates to approximately 9 per epoch, making the optimiser sluggish.

This observation aligns with the well-known *linear scaling rule*: when increasing the batch size, the learning rate should be scaled proportionally. The `large_batch` experiment used the same learning rate as the baseline, which was insufficient for the larger batch, resulting in poor performance.

7.3 Limitations

Several limitations should be noted:

- The word-level tokeniser is rudimentary, a sub-word tokeniser (e.g. BPE) would likely improve coverage and reduce the number of `UNK` tokens.
- The model is deliberately small (420K parameters, 2 layers). Increasing capacity could improve results but would also require a larger dataset to avoid overfitting.
- Only two hyper-parameters were varied. Other potentially impactful choices, such as the number of layers, the model dimension, dropout rate, and warm-up schedule, were not explored.
- Training was conducted on CPU, which limited the feasibility of larger-scale experiments.

8 Conclusion

This lab demonstrated the construction of a Transformer Encoder from scratch in PyTorch and applied it to a four-class text classification task on the 20 Newsgroups dataset. The benchmark of five hyper-parameter configurations revealed several practical insights:

1. A moderately aggressive learning rate (5×10^{-3}) coupled with cosine annealing was the most effective setting, reaching **86.8%** validation accuracy and **81.9%** test accuracy with a macro F1-score of **0.819**.
2. Learning rate has a far greater impact than batch size on final performance, especially within a fixed training budget.
3. Smaller batch sizes can serve as a substitute for larger learning rates by providing more frequent gradient updates, though they increase compute time per epoch.
4. The linear scaling rule is important, using a large batch without correspondingly increasing the learning rate leads to under-training.

Overall, even a compact Transformer Encoder with fewer than half a million parameters can achieve competitive results on moderately sized text classification problems, provided the hyper-parameters, particularly the learning rate, are chosen appropriately.