UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Christina Mitsiou*
sdi: 1115202000129

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Spring Semester 2025*

Contents

1. Abstract

Following HW1, this project's goal is to develop a deep-neural-network model for tweet sentiment analysis. Specifically, we

1. Create and fine-tune a custom Word2Vec embedding layer in PyTorch,
2. Fine tune various hyperparameters
3. Examine training dynamics using learning curves

With this approach, our best model achieved a final test accuracy of **76.71%**. These results show that embeddings and neural networks work well for tweet sentiment classification.

Notebook: https://www.kaggle.com/code/christinamitsiou/sdi2000129-hw2/edit

2. Data processing and analysis

2.1. Pre-processing

The first step to successful tweet sentiment analysis is data cleaning (pre-processing). As in HW1, we implemented a `preprocess_text` function with several cleaning steps. To identify which operations helped—and which actually hurt performance—we ran each step in isolation (keeping all other steps fixed), trained for three epochs, and recorded validation loss and accuracy.

Since some techniques (e.g. stop-word removal) require tokenization first, we applied them in the correct sequence. We also tested removing empty tweets and empty tokens both separately and together.

By comparing each method's results, we grouped the preprocessing steps into three categories:

- **Hurt performance**: lower accuracy or higher loss than the baseline.
- **Neutral**: similar accuracy and loss to the baseline.
- **Improved performance**: higher accuracy or lower loss than the baseline.

First, we recorded the baseline performance without pre-processing:

Preprocessing Step	E1 Loss	E2 Loss	E3 Loss	Val Acc (%)
None	0.686 07	0.679 87	0.678 55	56.26

Table 1: Baseline (no data cleaning.)

Then we tested each preprocessing step separately:

Preprocessing Step	**E1 Loss**	**E2 Loss**	**E3 Loss**	**Val Acc (%)**
Converting emojis to words	0.685 72	0.679 95	0.678 11	54.91
Remove # but keep the word	0.685 39	0.679 57	0.678 19	54.70
Remove punctuation	0.688 63	0.684 18	0.682 45	54.01
Remove non-English characters	0.685 29	0.679 87	0.678 02	54.62

Table 2: Preprocessing steps that *hurt* performance (baseline = 56.26 %).

Preprocessing Step	**E1 Loss**	**E2 Loss**	**E3 Loss**	**Val Acc (%)**
Removing empty tweets	0.685 59	0.679 82	0.678 33	55.31
Making the text lowercase	0.687 63	0.682 49	0.681 61	55.76
Replace every @username with "@user"	0.684 44	0.678 38	0.676 43	55.97
Normalize elongated words	0.685 26	0.679 71	0.678 36	56.55

Table 3: Preprocessing steps that performed *about the same* as the baseline (56.26 %).

Preprocessing Step	**E1 Loss**	**E2 Loss**	**E3 Loss**	**Val Acc (%)**
Expand contractions	0.685 80	0.679 98	0.678 36	57.62
Remove HTML entities	0.685 52	0.679 90	0.678 78	57.45
Replace URLs with "`<URL>`"	0.684 63	0.678 29	0.677 06	59.74
Replace emails with "`<EMAIL>`"	0.685 86	0.679 10	0.678 38	57.51
Replace phone numbers with "`<PHONE>`"	0.685 71	0.679 46	0.678 26	58.77
Removing empty tokens	0.627 72	0.620 64	0.618 41	75.14
Removing empty tweets and tokens	0.627 49	0.620 48	0.617 97	73.43
Tokenize	0.627 75	0.620 65	0.618 17	74.25
Tokenize + Correct spelling	0.627 52	0.620 86	0.618 72	73.95
Tokenize + Lemmatize	0.627 87	0.620 36	0.618 37	74.83
Tokenize + Remove stopwords	0.631 82	0.625 28	0.622 66	73.05
Remove non-English words	0.628 30	0.622 05	0.619 54	73.60

Table 4: Preprocessing steps that *improved* performance over the baseline (56.26 %).

After that, we combined all the "winning" steps from Table 4 into a single preprocess function. But, contrary to expectation, the combined pipeline actually performed worse than several of its components on their own. On the first run we got:

Preprocessing Step	**E1 Loss**	**E2 Loss**	**E3 Loss**	**Val Acc (%)**
All steps from Table 4	0.635 88	0.629 51	0.626 87	71.95

Table 5: Combination of all steps from the 3rd category.

This performance drop occurs because isolated gains do not always stack when methods interact. To find which steps were clashing, we took the full pipeline and removed one (or a few) at a time, tracking validation accuracy:

Removed Steps	**E1 Loss**	**E2 Loss**	**E3 Loss**	**Val Acc (%)**
Remove nothing	0.635 88	0.629 51	0.626 87	71.95
Expand contractions	0.632 01	0.626 06	0.623 68	72.87
HTML unescape	0.635 98	0.630 13	0.627 15	71.63
Expand contr. + HTML unescape	0.632 41	0.626 56	0.623 67	73.01
Expand contr. + HTML unescape + URL	0.632 49	0.626 62	0.624 06	71.95
Expand contr. + URL	0.632 09	0.626 56	0.624 02	72.95
Expand contr. + Email	0.632 44	0.626 23	0.623 67	73.08
Expand contr. + HTML unescape + Email	0.632 58	0.626 45	0.623 88	70.75
Expand contr. + Phone	0.632 14	0.626 40	0.623 84	71.57
Expand contr. + keep empty tokens	0.632 14	0.626 40	0.623 84	71.57
Expand contr. + Spell-correction	0.632 09	0.626 20	0.623 54	71.36
Expand contr. + Lemmatization	0.632 58	0.626 07	0.623 29	73.40
Expand contr. + Lemmat. + Stopwords	0.627 22	0.620 59	0.618 00	72.91
Lemmatization + Stopwords	0.627 03	0.619 60	0.617 40	74.06
Lemmatization only	0.635 89	0.629 79	0.626 55	71.57
Lemmat. + Stopwords + English filter	0.626 66	0.619 79	0.617 72	74.78
Lowercase	0.635 60	0.629 25	0.626 08	70.68
No empty tokens	0.626 44	0.619 51	0.617 43	75.17

Table 6: Combinations results: impact of removing combinations of preprocessing steps on validation accuracy.

From these results we came to the following conclusions:

1. **Isolated steps vs Combinations:** Steps that help in isolation—like spell-correction or URL anonymization—can harm performance when they are used with other filters.

2. **Aggressive filtering:** The combination of lemmatization, stop-word removal, and English-word filtering removed too much of the tweet vocabulary, making the the model miss useful features.

3. **Lowercasing is essential:** Even though it appeared neutral in Table 3, without it the accuracy dipped fell to 70.68 %.

2.2. Greedy Selection Results

We also ran a greedy selection over our preprocessing steps, starting from the minimal pipeline `[drop_empty, lowercase]`. At each iteration we added one step and kept it only if it improved validation accuracy. Table 7 shows the steps this method highlighted as best.

Pipeline Steps	**Val Acc (%)**
drop_empty, lowercase	53.86
+ expand_contractions	57.26
+ anonymize_email	58.22
+ tokenize	74.70
+ lemmatize	75.31

Table 7: Greedy selection: each step is added only if it raises validation accuracy.

2.3. Analysis of Pre-processing Pipeline Performance

The experiments in Tables 2–7 highlight a few clear points:

1. **Isolated successful steps don't always add up.** Steps that helped on their own—like URL anonymization or spell-correction—can clash when combined, over-pruning tokens and losing sentiment cues.
2. **Tokenization is a extremely valuable.** Just splitting text into words gives the biggest boost (from about 58 % to 75 %). That is logical since embeddings only work if you feed them real word units.
3. **Order matters.** Lowercasing looks neutral by itself, but it is needed before expanding contractions or lemmatizing—otherwise filters miss things and accuracy drops.
4. **Watch the filter strength.** Lemmatization alone helps, but pairing it with stop-word removal and an English-only filter is too harsh. This combination ends up removing low-frequency words that carry sentiment.

Putting all findings from Table 6 and 7 together and experimenting with the order of the steps, we end up with this pipeline

```
drop empty tweets → lowercase → expand contractions → anonymize
        email → tokenize → drop empty tokens → lemmatize
```

This achieves the following results: We will use this as our base going forward.

Preprocessing Step	**E1 Loss**	**E2 Loss**	**E3 Loss**	**Val Acc (%)**
All in final pipeline	0.627 02	0.619 66	0.617 21	75.60

Table 8: Final Pipeline

2.4. Additional Refinement Trials

We tested four targeted refinements on top of our previus pipeline to see if any would push accuracy beyond the 75.6 % baseline. Table 9 shows the *average* validation accuracy over multiple runs for each step.

Refinement Step	**Avg Val Acc (%)**
URL anonymization	75.25
Normalize repeated exclamation marks	73.51
Normalize repeated question marks	74.10
OOV → `<UNK>` mapping	75.63

Table 9: Average validation accuracy for additional preprocessing refinements.

- **URL anonymization:** Results were around the baseline (75.25 %), but still lower, so we skip it.
- **Normalize repeated exclamation marks:** Consistently below baseline, so we skip it.
- **Normalize repeated question marks:** High variance with no reliable gain, so we skip it.
- **OOV mapping:** Gave a consistent ∼0.3 % lift, so we keep it.

Putting all of our findings together, the final, best-performing preprocessing pipeline is:

```
drop empty tweets → lowercase → expand contractions → anonymize
email → tokenize → drop empty tokens → lemmatize  →  OOV mapping
```

Our final validation score after perfecting the preprocess function is **75.63 %**

2.5. Vectorization

After cleaning and tokenizing each tweet, we need to turn it into a fixed-size vector so our model can use it. To achieve this, we did the following:

1. For every token in a tweet, look up its 300-dimensional Word2Vec vector (if it exists in our model).
2. If no tokens match use a zero vector of length 300.
3. Otherwise, take the average of all those vectors.

That gives us one 300-dimensional vector per tweet.
We run this on all tweets in train/val/test to get `train_vectors`, `val_vectors`, and `test_vectors`.
This approach captures the overall "meaning" of a tweet.

3. Algorithms and Experiments

3.1. Model Architecture

We built a simple two-layer feed-forward network in PyTorch that takes a tweet's 300-d average Word2Vec vector and gives us a single logit for positive vs. negative sentiment. Specifically:

- **Input layer:** 300-dimensional vector.
- **Hidden layer 1:** Fully connected, $\text{Linear}(300, H)$, followed by ReLU and dropout p.
- **Hidden layer 2:** Fully connected, $\text{Linear}(H, H)$, followed by ReLU and dropout p.
- **Output layer:** Fully connected, $\text{Linear}(H, 1)$, producing raw logits.
- **Loss:** BCEWithLogitsLoss.
- **Optimizer:** Adam with learning rate η.

3.2. Baseline Hyperparameter Configuration and Performance

Before beginning fine-tuning, we note our baseline configuration and examine it's behavior over 30 epochs.

Hyperparameter	**Value**
Epochs	30
Hidden size	128
Batch size	64
Learning rate	0.001
Dropout	0.20
Peak Val Acc (%)	**77.62**

Table 10: Baseline model hyperparameters and peak validation accuracy (epoch 2).

Statistics

- **Epoch 1:** Train Loss = 0.3955, Val Loss = 0.4956, Val Acc = 77.34 %.
- **Epoch 2:** Train Loss = 0.3947, Val Loss = 0.4953, Val Acc = **77.62 %**.
- **Epoch 3:** Train Loss = 0.3928, Val Loss = 0.4966, Val Acc = 77.55 %.
- **Epoch 4:** Train Loss = 0.3917, Val Loss = 0.4943 (minimum), Val Acc = 77.41 %.
- **Epoch 5:** Train Loss = 0.3911, Val Loss = 0.5040, Val Acc = 77.28 %.
- **Epoch 30:** Train Loss = 0.3621, Val Loss = 0.5359, Val Acc = 77.14 %.

Analysis

- **Quick early gains:** Validation accuracy jumps to 77.62 % by epoch 2, and validation loss bottoms out at epoch 4.
- **Flattening after:** Despite a steady decline in training loss (0.3955 → 0.3621), validation accuracy remains between 77.1–77.6 % from epochs 2–30, indicating an issue.
- **Onset of overfitting:** After epoch 4, validation loss drifts upward to 0.5359 by epoch 30, even as training loss continues to fall. This gap signals mild overfitting.

These observations guide our fine-tuning strategy: we will sequentially tune learning rate, hidden size, dropout and batch size to get further improvements.

3.3. Baseline Learning Curves

Figures 1 and 2 show the evolution of training vs. validation loss and validation accuracy over 30 epochs under the baseline hyperparameters (Table 10).

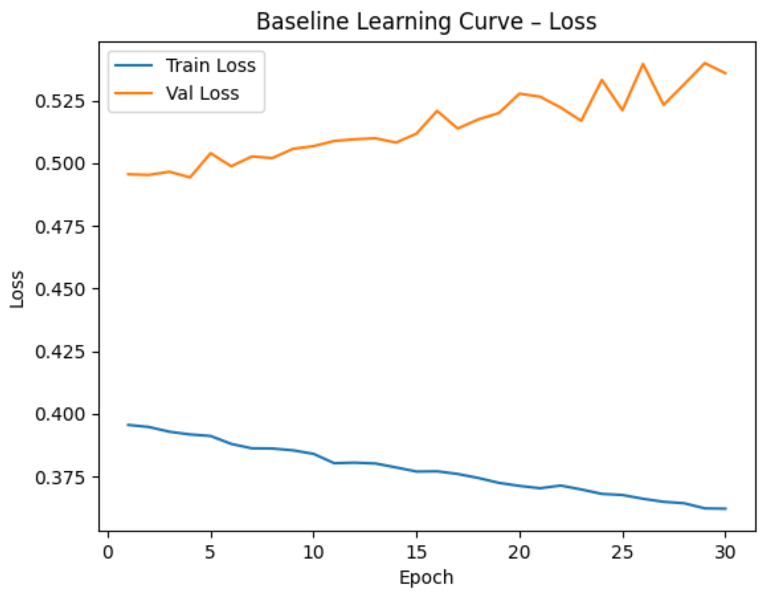

Figure 1: Training and validation loss over 30 epochs.

Loss Curve Analysis

- **Training loss** falls steadily from 0.3955 (epoch 1) to 0.3621 (epoch 30), indicating the model continues to fit training data throughout.
- **Validation loss** hits its minimum (0.4943) at epoch 4, then goes up to 0.5359 by epoch 30, showing mild overfitting once training loss continues to decline.

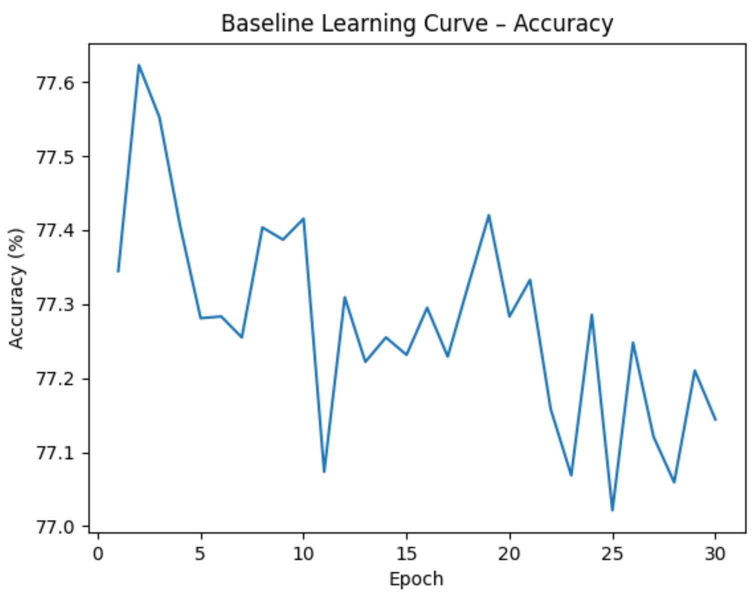

Figure 2: Validation accuracy over 30 epochs.

Accuracy Curve Analysis

- **Validation accuracy** climbs quickly to a peak of 77.62 % at epoch 2, then oscillates narrowly between 77.0–77.6 % thereafter.
- The early flattening—despite further decreases in training loss—confirms that little or no generalization gain is achieved after epoch 5.

3.4. Hyperparameter Fine-Tuning Strategy

To systematically improve our model's performance, we use a *One-at-a-time (Sequential) Tuning* practice:

3.4.1. One-at-a-time (Sequential) Tuning.

1. **Fix:** keep all hyperparameters at their baseline values except one.
2. **Sweep:** vary the selected hyperparameter over a predefined set of candidate values.
3. **Select:** record the value that gave best results.
4. **Freeze:** lock in that optimal value and proceed to the next hyperparameter.

3.5. Learning Rate Tuning

We first started with learning rate fine-tuning.

To evaluate the effect of different learning rates, we trained the same model configuration for **30 epochs** with three different learning rates: **1e-04**, **5e-04**, and **1e-03**. This setup allows for a direct comparison with the baseline configuration.

The partial training logs (including training loss, validation loss, and validation accuracy) are summarized in the table below:

Epoch	Learning Rate	Val Loss	Val Accuracy (%)
10	1e-04	0.4753	77.21
20	1e-04	0.4661	77.85
30	1e-04	**0.4661**	**77.98**
10	5e-04	0.4664	77.65
20	5e-04	0.4696	77.83
30	5e-04	0.4762	77.53
10	1e-03	0.4672	77.72
20	1e-03	0.4743	77.64
30	1e-03	0.4897	77.08

The learning curves for loss and accuracy are shown below:

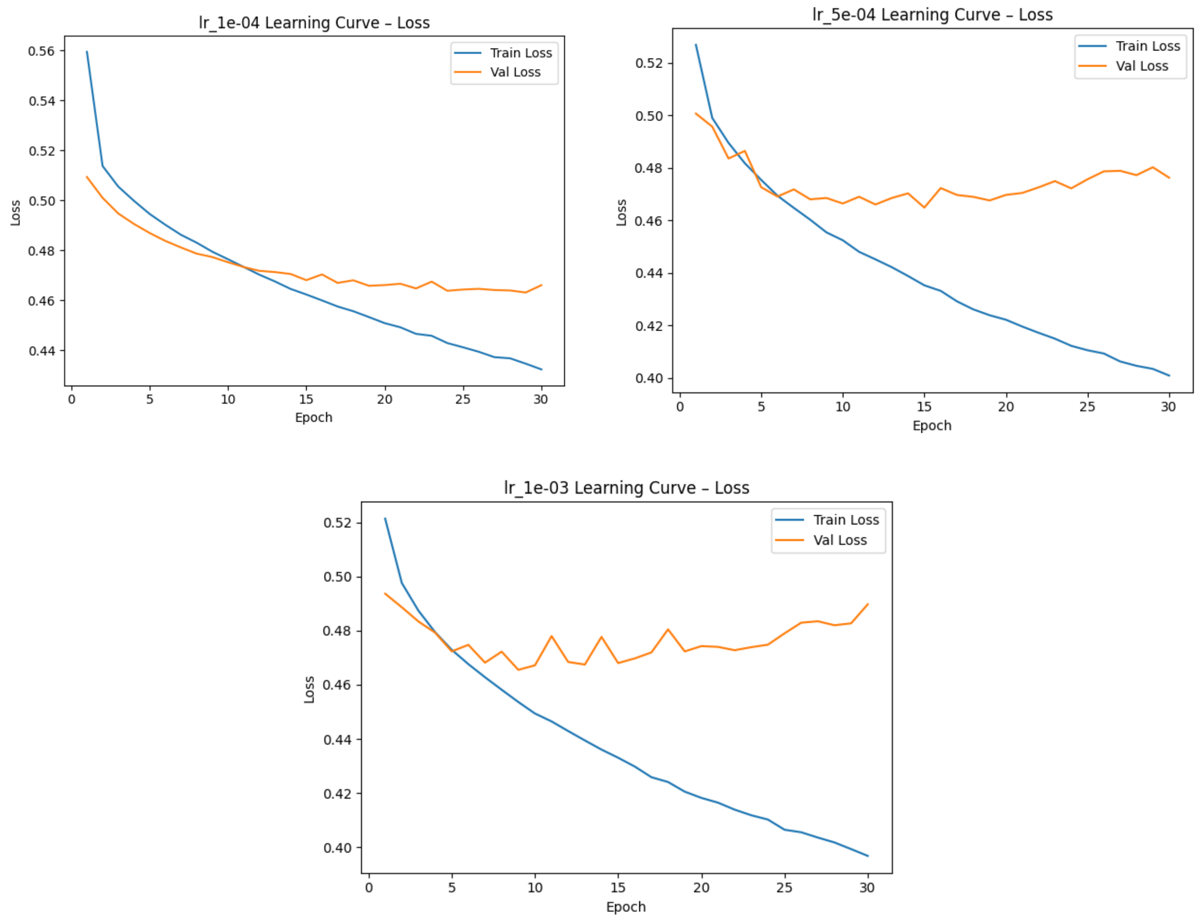

Figure 3: Loss curves for learning rates 1e-04, 5e-04, and 1e-03

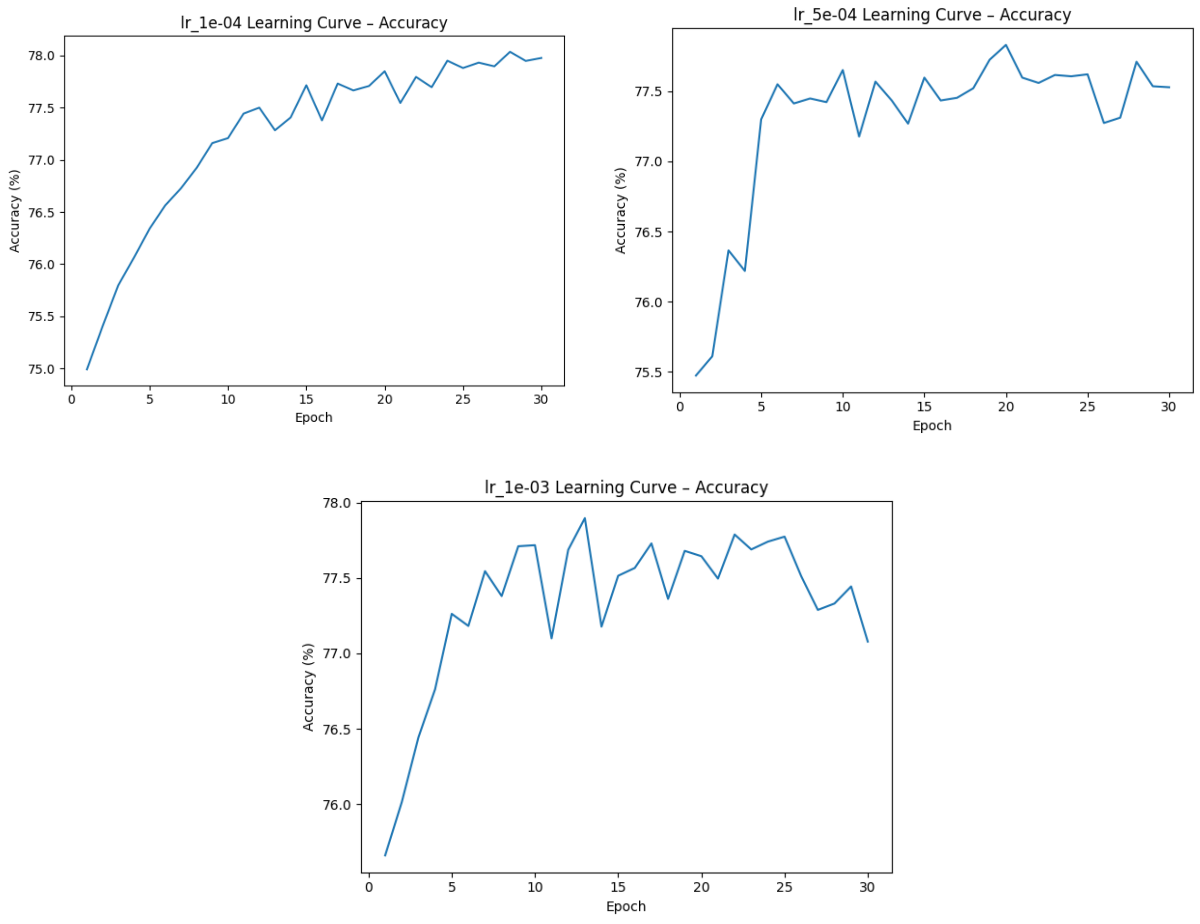

Figure 4: Accuracy curves for learning rates 1e-04, 5e-04, and 1e-03

By carefully observing the learning curves, we can note the following:

1. **1e-04** displays the most desirable training behavior. The validation loss decreases steadily and flattens smoothly, while the validation accuracy increases consistently across all 30 epochs. There are no abrupt fluctuations. Compared to the baseline, this learning rate gives a more stable and generalizable training process, with better final performance.

2. **5e-04** initially performs similarly to 1e-04, with a sharp drop in loss and a quick gain in accuracy during the early epochs. However, from around epoch 10 onward, the validation loss flattens, while the validation accuracy curve becomes noisy. This indicates that the model starts to overfit slightly or fails to make consistent progress, which can be seen clearly in the loss curve flattening and bouncing upward in later epochs.

3. **1e-03** shows a more erratic behavior. Although it achieves competitive accuracy in the first half of training, its validation loss curve becomes increasingly unstable after epoch 10. The accuracy curve reflects this with significant fluctuations. These signs indicate that the learning rate is likely too high. Compared to both 1e-04 and the baseline, 1e-03 results in less reliable convergence and poorer generalization by the final epochs.

4. When comparing the validation loss curves side by side, we see that 1e-04 exhibits a smooth and downward-sloping curve that consistently improves until the end of training. In contrast, both 5e-04 and 1e-03 begin to flatten or rise slightly near the end — a classic indicator that learning is no longer effective or has become unstable. Similarly, the accuracy curves further support this interpretation: only 1e-04 continues to climb gently and remains smooth throughout.

5. Lastly, compared to the baseline (not shown here), which was trained using a default setting without tuning, all three tuned models offer better or comparable performance. However, only 1e-04 improves steadily without trade-offs in stability or overfitting, making it the most balanced choice.

Therefore, the best choice of learning rate is **1e-04**, which we lock in for the remaining fine-tuning experiments. It not only delivers the highest final validation accuracy (77.98%), but also demonstrates smooth convergence, robust generalization, and no signs of instability or overfitting in the learning curves.

3.6. Hidden Size Tuning

We next fine-tuned the hidden layer size of our model. This parameter controls the model's capacity to learn complex patterns. We explored the following four configurations for 30 epochs using the previously chosen learning rate of 1×10^{-4}:

```
hidden_sizes = [64, 128, 256, 512]
```

The goal was to balance representational power and generalization. The table below summarizes the final validation accuracies:

Hidden Size	Final Validation Accuracy (%)
64	77.78
128	77.98
256	**78.05**
512	77.30

Table 11: Hidden size sweep results

The training and validation learning curves for loss and accuracy are shown below:

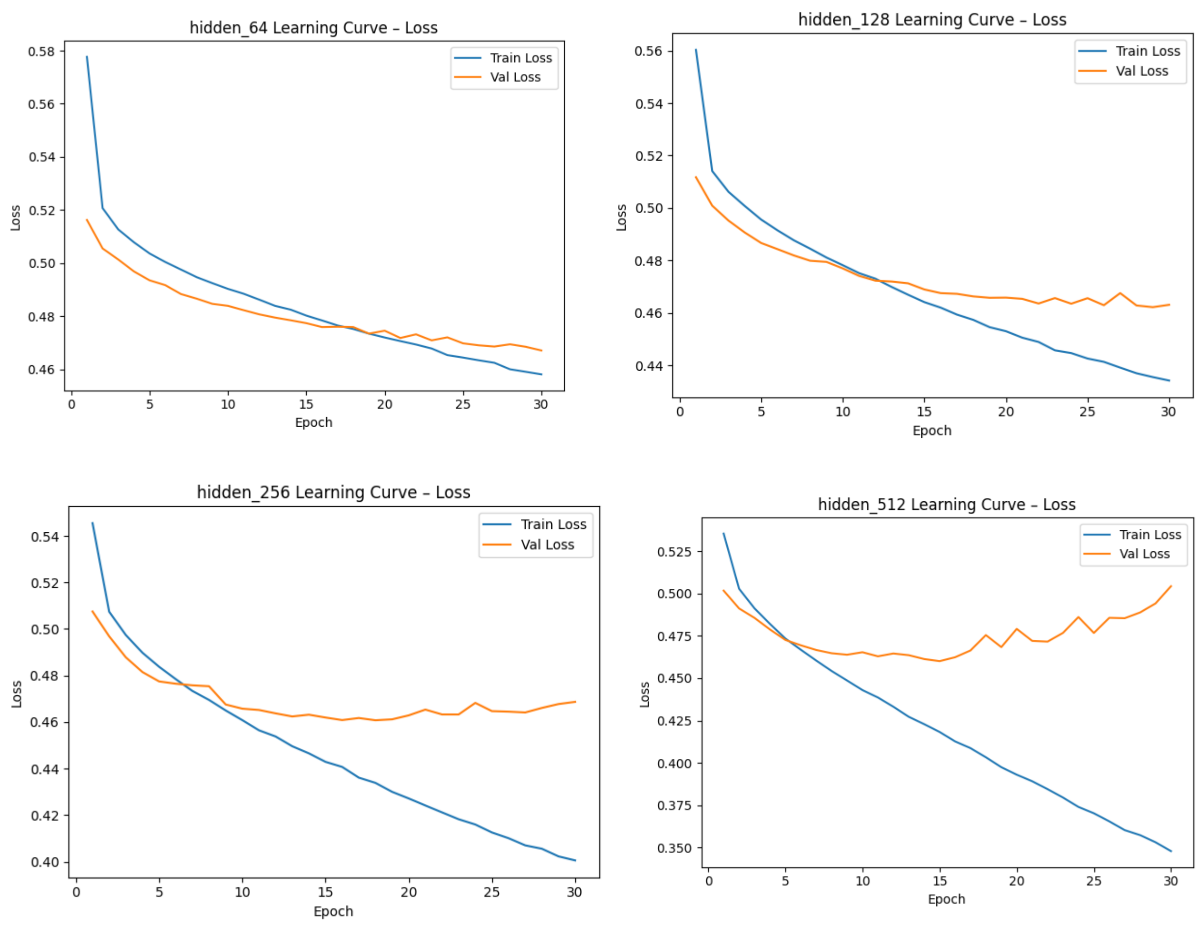

Figure 5: Loss curves for hidden sizes 64, 128, 256, and 512

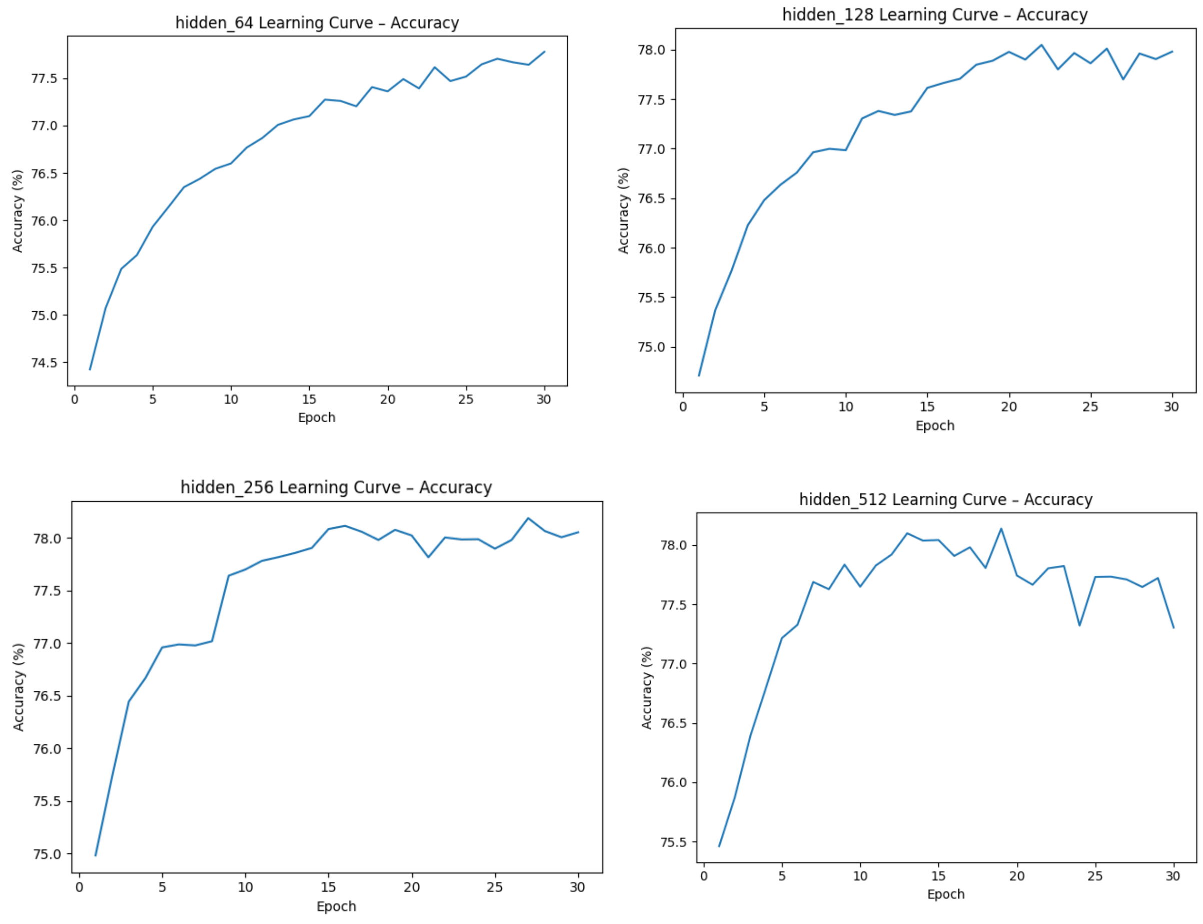

Figure 6: Accuracy curves for hidden sizes 64, 128, 256, and 512

By analyzing these plots, we observe the following:

1. **Hidden = 64**: Shows the most stable and consistent training behavior. Both loss and accuracy curves improve smoothly throughout training, with no signs of overfitting. Although the final accuracy is slightly lower, the model generalizes well and offers clean learning dynamics.

2. **Hidden = 128**: Maintains stable and well-behaved curves overall, but introduces minor fluctuations in validation loss after epoch 20. The accuracy gains taper slightly, suggesting the onset of mild overfitting.

3. **Hidden = 256**: Achieves the highest final validation accuracy (78.05%) but exhibits increased noise in both loss and accuracy plots during the later epochs. The performance gains come at the cost of less consistent generalization.

4. **Hidden = 512**: Demonstrates clear overfitting after epoch 15. Validation loss worsens while accuracy stagnates, even as training loss continues to decrease.

Despite the slightly higher peak accuracy of larger hidden sizes, **Hidden Size = 64** offers the most robust and stable training profile. Its validation loss consistently declines and accuracy steadily improves without variance or instability.

We therefore select **Hidden Size = 64** as the optimal setting, prioritizing smooth convergence and strong generalization over marginal gains in final accuracy.

3.7. Dropout Rate Tuning

Having selected our optimal learning rate and hidden size, we proceeded to fine-tune the **dropout rate**, which helps prevent overfitting by randomly deactivating a proportion of hidden neurons during training. We evaluated four values: 0.0, 0.2, 0.4, and 0.5, keeping all other hyperparameters fixed. The results are summarized below:

Dropout Rate	Final Val Accuracy	Lowest Val Loss
0.0	76.91%	0.4790
0.2	**77.07%**	**0.4772**
0.4	77.05%	0.4803
0.5	76.64%	0.4856

Table 12: Validation accuracy and lowest loss per dropout setting (with hidden size = 64)

The learning curves for loss and accuracy are shown below:

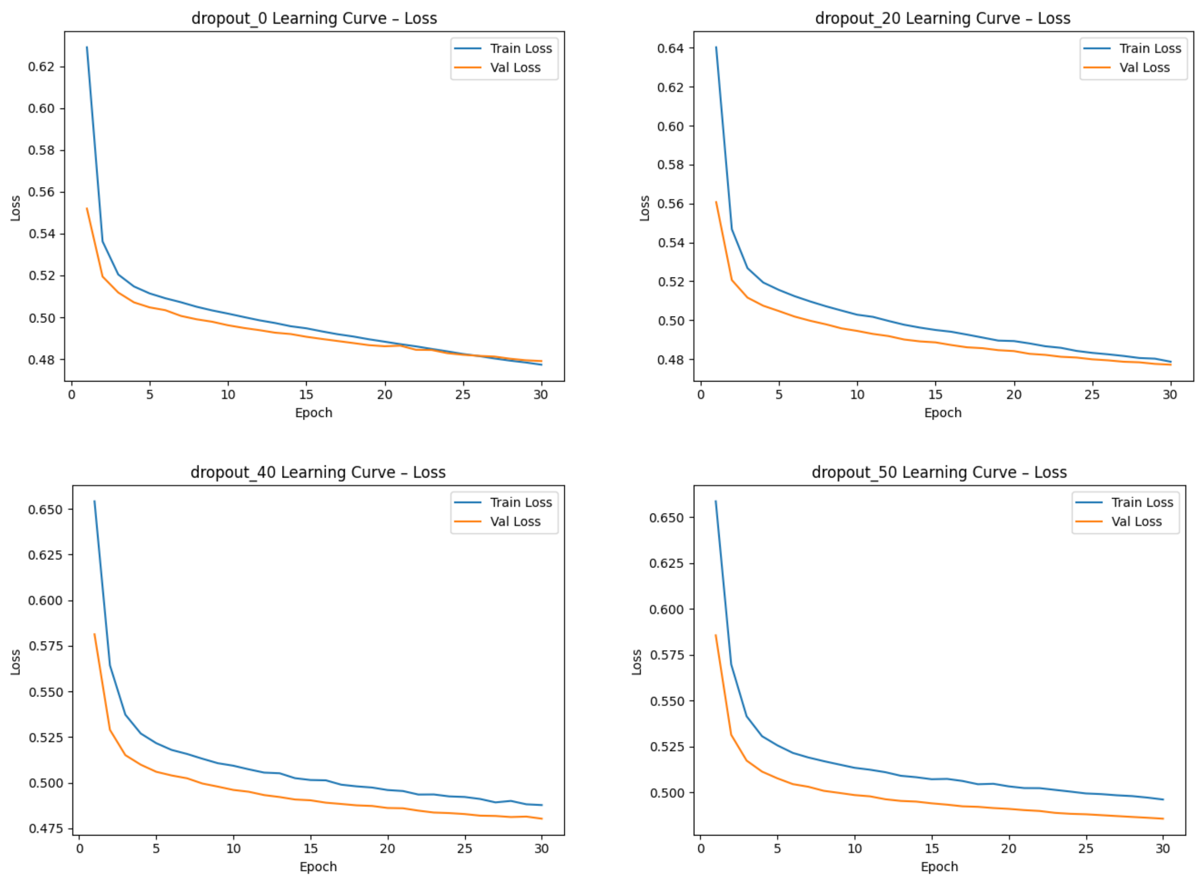

Figure 7: Loss curves for dropout rates 0.0, 0.2, 0.4, and 0.5

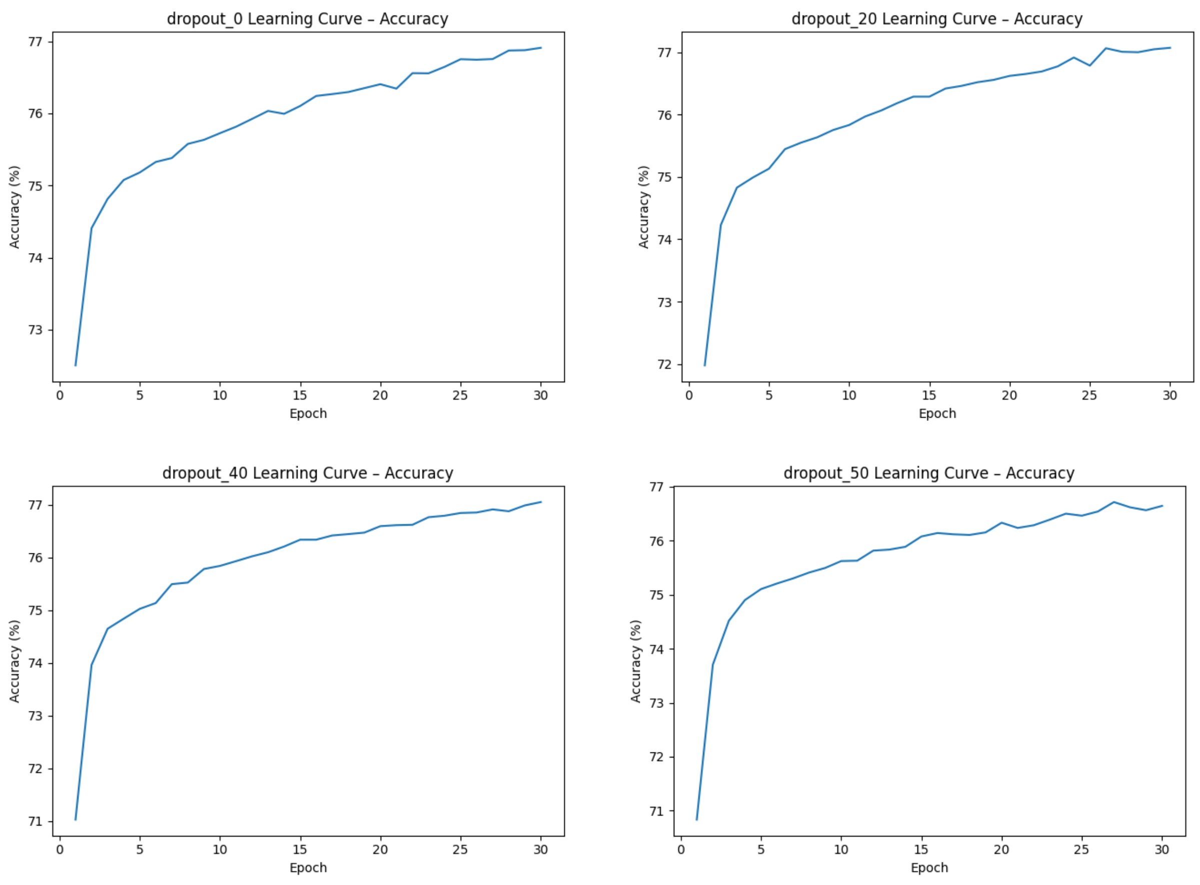

Figure 8: Accuracy curves for dropout rates 0.0, 0.2, 0.4, and 0.5

From these results, we can make the following observations:

1. **Dropout = 0.0**: Performs well overall, but shows signs of mild overfitting in the loss curves after epoch 25.
2. **Dropout = 0.2**: Achieves the best validation accuracy and lowest loss, with clean and stable curves that steadily improve through training.
3. **Dropout = 0.4**: Closely trails dropout = 0.2, with slightly less smooth convergence in accuracy and loss near the end.
4. **Dropout = 0.5**: Curves flatten early and learning progresses more slowly, indicating mild underfitting.

Therefore, we select **dropout = 0.2** as the optimal value for the current model. It offers the best generalization performance and training stability when used with a hidden size of 64.

3.8. Batch Size Tuning

After optimizing the learning rate, hidden size, and dropout, we turned to fine-tuning the **batch size**, which governs how many samples are used to compute each weight update. We evaluated four values: 32, 64, 128, and 256, while holding all other hyperparameters fixed. The results are summarized below:

Batch Size	Final Val Accuracy	Lowest Val Loss
32	**77.82%**	0.4655
64	77.57%	0.4683
128	77.40%	0.4720
256	77.15%	**0.4770**

Table 13: Validation accuracy and lowest loss per batch size

The corresponding learning curves for loss and accuracy are shown below:

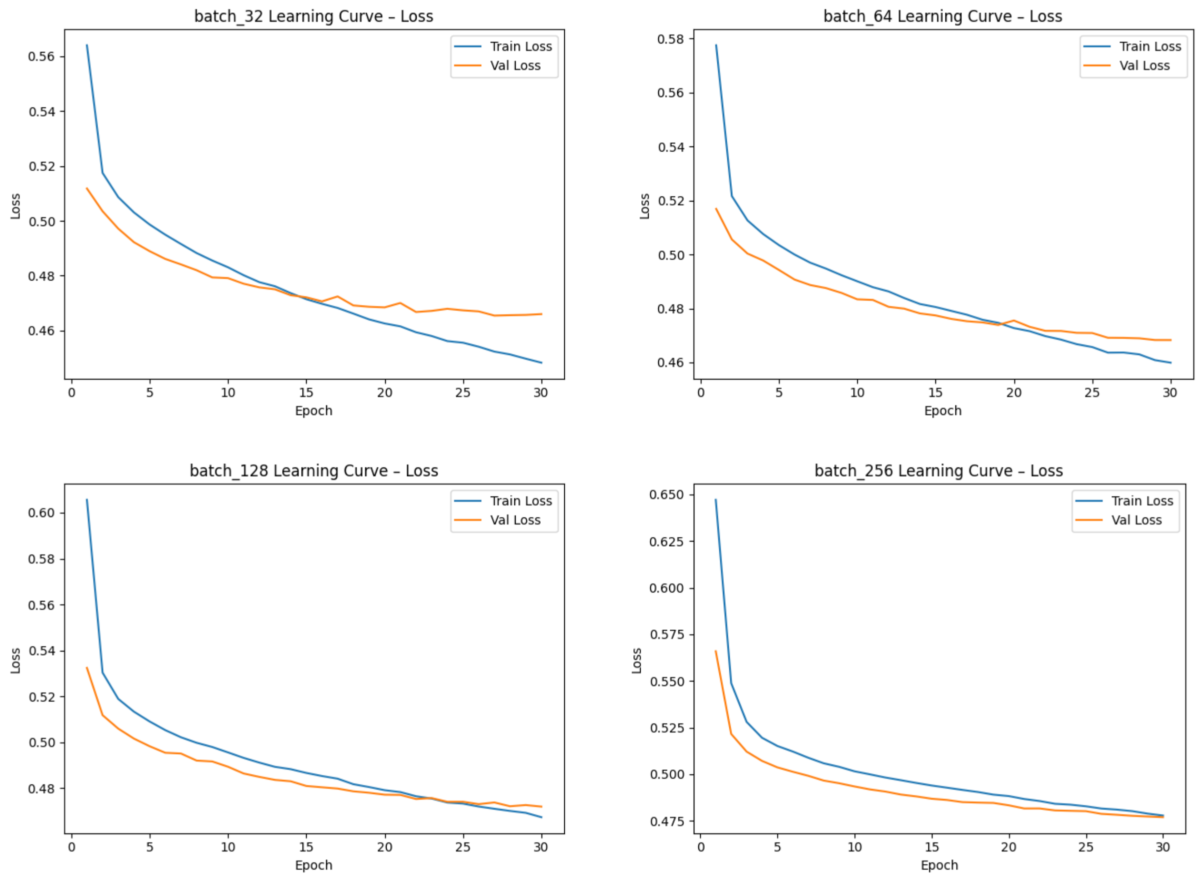

Figure 9: Loss curves for batch sizes 32, 64, 128, and 256

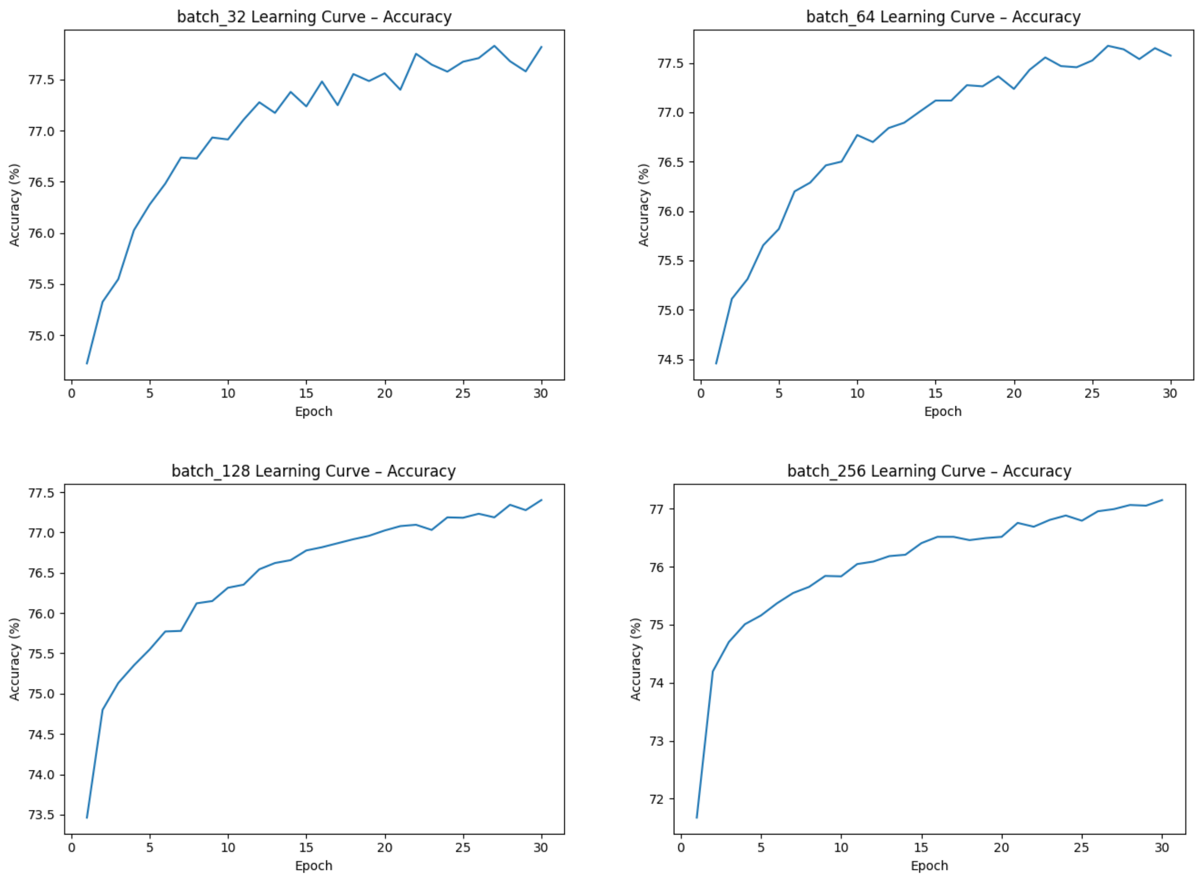

Figure 10: Accuracy curves for batch sizes 32, 64, 128, and 256

From these results and plots, we make the following observations:

1. **Batch = 32** reaches the highest final accuracy but exhibits noticeable variance in both accuracy and loss during the second half of training.

2. **Batch = 64** performs well but suffers from mild noise and inconsistency in validation loss.

3. **Batch = 128** offers stable training but flattens early and achieves lower accuracy overall.

4. **Batch = 256** gives the most stable and consistent loss and accuracy curves, with minimal variance and clear, steady improvement throughout training.

While batch size 32 performs best in terms of final accuracy, its learning curves suggest higher variance and less consistent generalization. In contrast, **batch size = 256** offers smoother training dynamics and more reliable behavior across epochs, making it a better choice for robust model performance. We therefore select it as our final batch size moving forward.

4. Conclusion

In this homework, we moved from a TF–IDF Logistic Regression baseline to a lightweight deep-neural-network with Word2Vec embeddings. By carefully testing and refining our preprocessing steps, we found that a minimal pipeline—dropping empty tweets, lowercasing, expanding contractions, anonymizing emails, tokenizing, dropping empty tokens, lemmatizing, and mapping OOV words—gave us a solid 75.6 % validation accuracy.

Then, through one-at-a-time tuning, we settled on a learning rate of 1e-4, hidden size of 64, dropout of 0.2, and batch size of 256. These choices pushed our model up to 78.05 % on validation and 76.71 % on the test set.

Overall, our results show that a straightforward neural network with well-tuned Word2Vec embeddings can beat traditional TF–IDF–based models at classifying tweet sentiment.