Name: Saadullah Khan

# Sentiment Classification with Neural Networks

## Objective:

This project's objective is to apply a convolutional neural network to the Sentiment140 dataset and quantify the effects of varying the model's architecture and hyperparameters on training time, parameter count, and accuracy. This project aims to find configurations that strike a good balance between predictive accuracy and efficiency by methodically modifying architectural components and training settings. It evaluates how different design choices impact model performance and computational cost.

## Hyperparameters Experiments:

- Learning rates: 0.01, 0.1
- Optimizers: SGD, Adam
- Hidden layers: add two (three total)
- Kernel sizes: 5, 3
- Embedding dimensions: 100, 500
- Input length: 50, 250
- Train up to 20 epochs with early stopping (patience = 3)

## Model Summary Table:

| MODEL # | LEARNING RATE | OPTIMIZER | HIDDEN LAYERS | KERNEL SIZE | EMBEDDING DIM | EPOCH | INPUT LENGTH |
|---|---|---|---|---|---|---|---|
| **BASELINE** | 0.001 | Adam | 1 | 5 | 100 | 10 | 50 |
| **1** | 0.001 | Adam | 2 | 5 | 100 | 10 | 50 |
| **2** | 0.1 | Adam | 3 | 3 | 500 | 20 | 50 |
| **3** | 0.1 | SGD | 2 | 5 | 100 | 15 | 50 |
| **4** | 0.01 | SGD | 2 | 3 | 100 | 18 | 50 |
| **5** | 0.01 | Adam | 2 | 3 | 500 | 18 | 50 |
| **6** | 0.1 | SGD | 3 | 3 | 500 | 20 | 50 |
| **7** | 0.01 | SGD | 2 | 5 | 100 | 20 | 50 |
| **8** | 0.01 | SGD | 2 | 5 | 500 | 20 | 250 |
| **9** | 0.01 | SGD | 2 | 3 | 500 | 20 | 250 |
| **10** | 0.1 | Adam | 3 | 5 | 100 | 20 | 50 |
| **11** | 0.01 | Adam | 3 | 5 | 500 | 20 | 50 |
| **12** | 0.01 | Adam | 3 | 3 | 100 | 20 | 50 |
| **13** | 0.1 | Adam | 2 | 5 | 500 | 20 | 250 |
| **14** | 0.1 | Adam | 2 | 3 | 100 | 20 | 250 |

| MODEL # | TRAIN ERROR | VAL ERROR | TEST ERROR | PARAMETER COUNT | TIME | EARLY-STOPPING EPOCH |
|---|---|---|---|---|---|---|
| BASELINE | 8.86% | 20.80% | 20.93% | 1,064,357 | 92.90s | 10 |
| 1 | 8.84% | 20.39% | 20.29% | 1,146,405 | 150.98s | 10 |
| 2 | 50.01% | 49.91% | 50.02% | 5,291,317 | 426.68s | 4 |
| 3 | 17.07% | 20.67% | 20.66% | 1,228,453 | 192.56s | 8 |
| 4 | 15.66% | 19.51% | 19.36% | 5,242,037 | 233.91s | 9 |
| 5 | 19.73% | 20.88% | 21.00% | 5,242,037 | 343.05s | 7 |
| 6 | 17.32% | 19.53% | 19.59% | 5,242,037 | 239.11s | 6 |
| 7 | 19.83% | 21.93% | 22.14% | 1,146,405 | 135.18s | 7 |
| 8 | 16.51% | 20.01% | 20.23% | 5,402,805 | 1278.38s | 7 |
| 9 | 17.66% | 19.70% | 19.76% | 5,242,037 | 1002.33s | 6 |
| 10 | 50.01% | 49.91% | 50.02% | 1,228,453 | 237.73s | 4 |
| 11 | 49.99% | 50.09% | 49.98% | 5,484,853 | 491.04s | 4 |
| 12 | 18.46% | 19.88% | 19.83% | 1,137,317 | 181.55s | 5 |
| 13 | 49.99% | 50.09% | 49.98% | 5,402,805 | 1347.73s | 4 |
| 14 | 49.99% | 50.09% | 49.98% | 1,088,037 | 407.70s | 4 |

**Analysis Questions:**

- How did learning rate affect the number of epochs needed to converge?

  **Ans:** It was obvious that the learning rate had an effect on how quickly models came together. Early stopping meant that models trained with a higher learning rate (0.1) usually stopped much sooner, usually after 4–6 epochs. But this faster convergence often meant that the optimization wasn't very good, and many models didn't learn useful patterns, making mistakes almost half the time. On the other hand, lower learning rates (0.001–0.01) needed more epochs, usually between 7 and 10, but they made training more stable and the model worked better on new data.

- Did larger models (extra Conv1d layers or extra hidden layers) improve accuracy?

  **Ans:** Adding more hidden layers to a model didn't always make it better. Some models that were a little deeper did get better, but a lot of the biggest ones didn't get better at all or even got worse, sometimes dropping to random-guess accuracy. This shows that just making things more complicated or deeper didn't always help with sentiment classification on this dataset.

- Was the added model complexity justified by the accuracy gain?

  **Ans:** The number of parameters went up a lot as the models got more complicated. It usually went from about one million to more than five million. But the accuracy didn't get much better. In a lot of cases, the accuracy improvements were less, and some big models did worse than the baseline. Overall, the small or nonexistent gains in accuracy did not make the model's added complexity worth it.

- How much additional computation time did deeper models require?

  **Ans:** It took a lot longer to compute models that were deeper and wider. The basic model learned in less than two minutes. It took hundreds of seconds, or even more than twenty minutes in the worst cases, for larger models with more layers, higher embedding dimensions, or longer input sequences. This shows that adding more depth and width costs a lot of computing power and only slightly improves performance.

- Did longer input sequences (50 vs. 250) help or hurt performance?

  **Ans:** When we used longer input sequences, we sometimes saw small changes in how well the tests or validations worked. This means that having more context can be helpful. But these improvements were small and not always the same, and the time it took to train went up a lot, sometimes by ten times. Longer input sequences usually make things less efficient and only slightly more accurate.

- How did kernel size influence convergence and accuracy?

  **Ans:** The size of the kernel affected both how accurate and how stable the convergence was. Models that used smaller kernels generally had lower validation and test errors and converged more consistently, especially when used with SGD. Larger kernels didn't always help, and sometimes they made the training process less stable.

- How did embedding dimension (100 vs. 500) affect training time and accuracy?

  **Ans:** When the embedding dimension increased from 100 to 500, the quantity of parameters and the duration required for training significantly escalated. Even with this extra ability to represent things, the changes in accuracy were small and not the same in all experiments. In some cases, larger embeddings didn't work any better than smaller ones. This means that adding more dimensions to an embedding doesn't always help.

- Which configuration gave the best trade-off between accuracy and cost?

  **Ans:** Model 4 was the best choice because it was the most accurate and the least expensive. It had the fewest test errors while keeping the number of parameters and training time reasonable. This setup showed that a well-tuned, medium-sized model can work better than bigger, more expensive ones, making it the best and most practical choice.

## Conclusion:

The experiments show that deeper architecture, larger embeddings, and longer input sequences increase training time and parameter count but do not improve accuracy. Moderate learning rates and simpler architectures improved convergence and generalization. On Sentiment140, a well-tuned, moderately sized CNN had the best accuracy-computational efficiency trade-off.

## Appendix:

https://colab.research.google.com/drive/1E-g0UbDEe60YdvCsGVfehErjsK-buOUO?usp=sharing