CSE595 Homework 1 : Logistic Regression Report

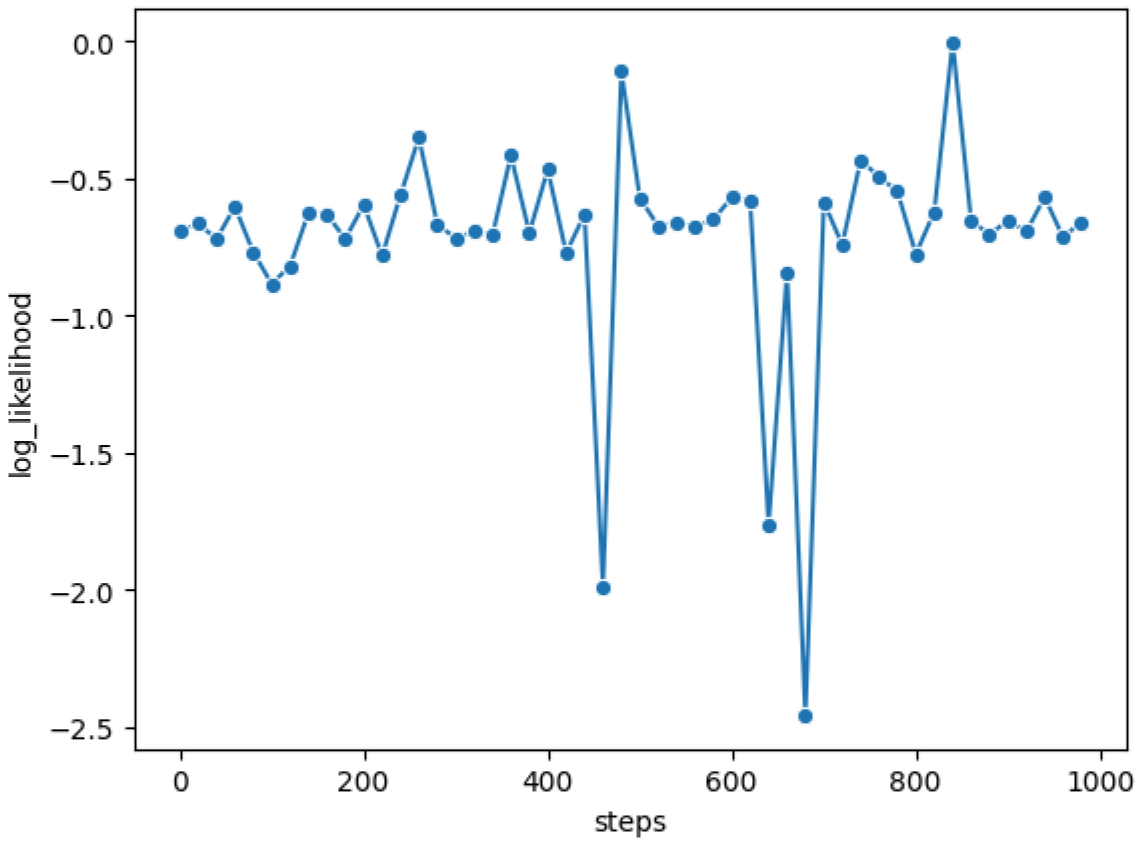
Kaggle username : nandanasheri

Problem 2 : better tokenizer design decisions

Design of better_tokenize function:

- Convert all text into lowercase - this is predominantly because of how several words are uppercase in the start of a sentence but could mean the same thing like ‘good’ and ‘Good’.
- Initially, punctuations were removed to focus predominantly on actual words though we talked about how the em dash is a very clear giveaway that text is LLM generated traditionally. So I used **regex to separate all possible punctuations into their own separate tokens** along with any sequences of words itself.
- Taking inspiration from NLTK’s tokenizer, I found a .txt file of stopwords (linked from this public GitHub repository) and removed all stopwords that were classified as tokens.
- One major error that was fixed was treating punctuations as their own tokens. When splitting documents solely by using whitespace, ‘end.’, ‘end?’, ‘end!’ would all be separate tokens which is fundamentally incorrect. Case and punctuations were handled separately in the case of the better tokenizer.

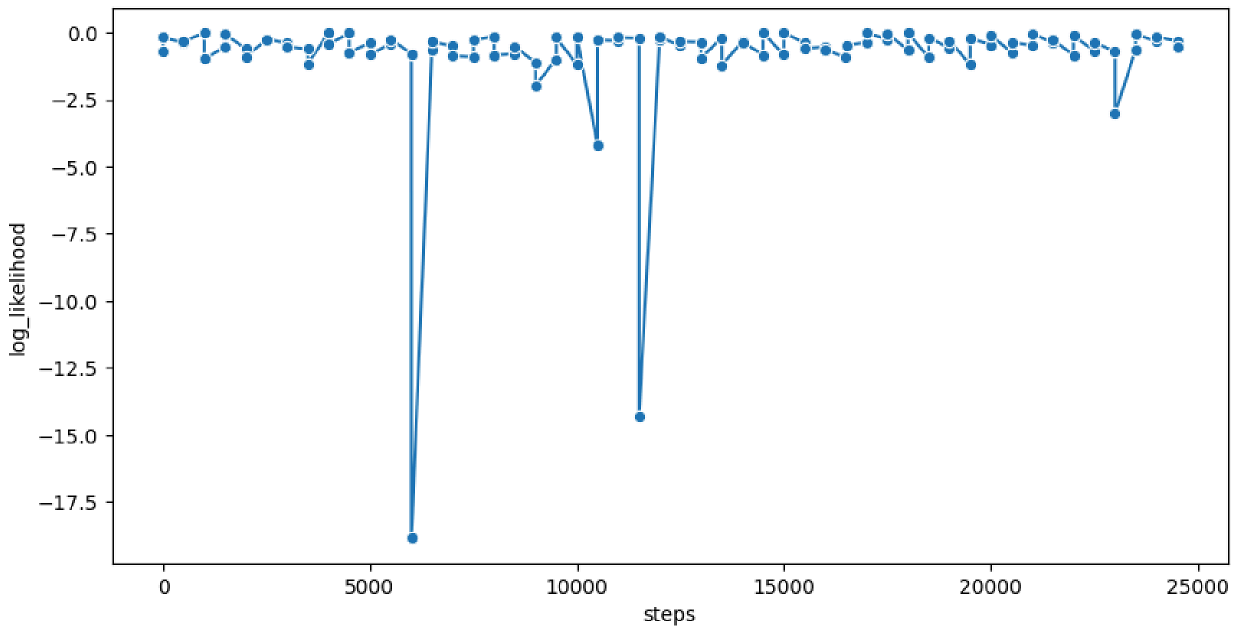
Problem 8 : Numpy model : Log likelihood plot for 1000 steps



For 1000 steps of training, the log likelihood seems to have pretty steep drops and hikes though towards the end it does seem like it gets a bit closer to convergence. Each point was plotted at every 20 steps.

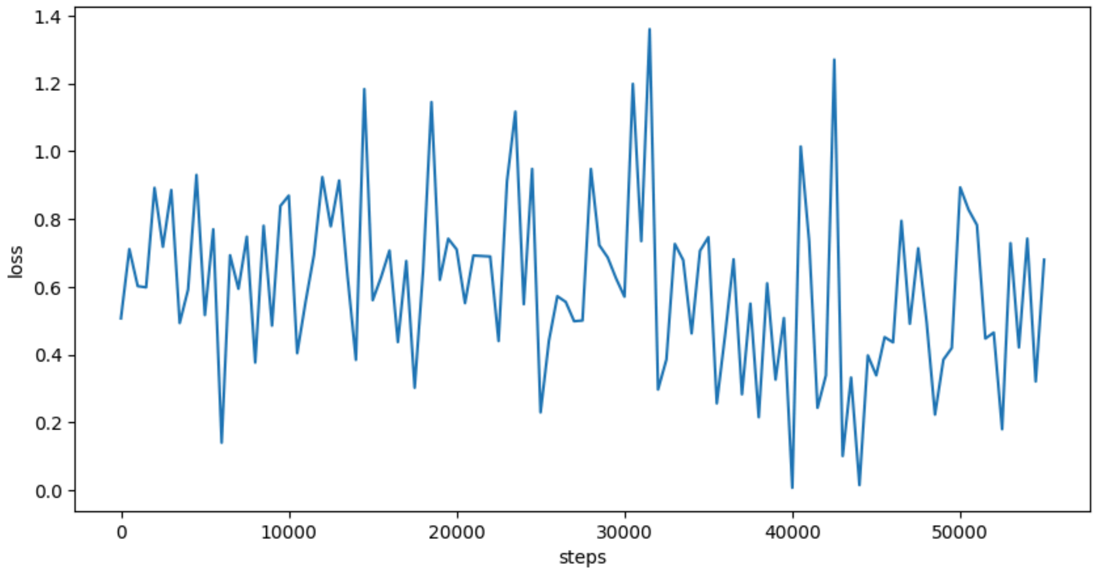
Problem 9 : Numpy model : Training until model converges - hyperparameter tuning

Plotting log likelihood for *learning rate of 5e-4 and 25,000 steps for 2 epochs.* Each point is at every 500 steps..

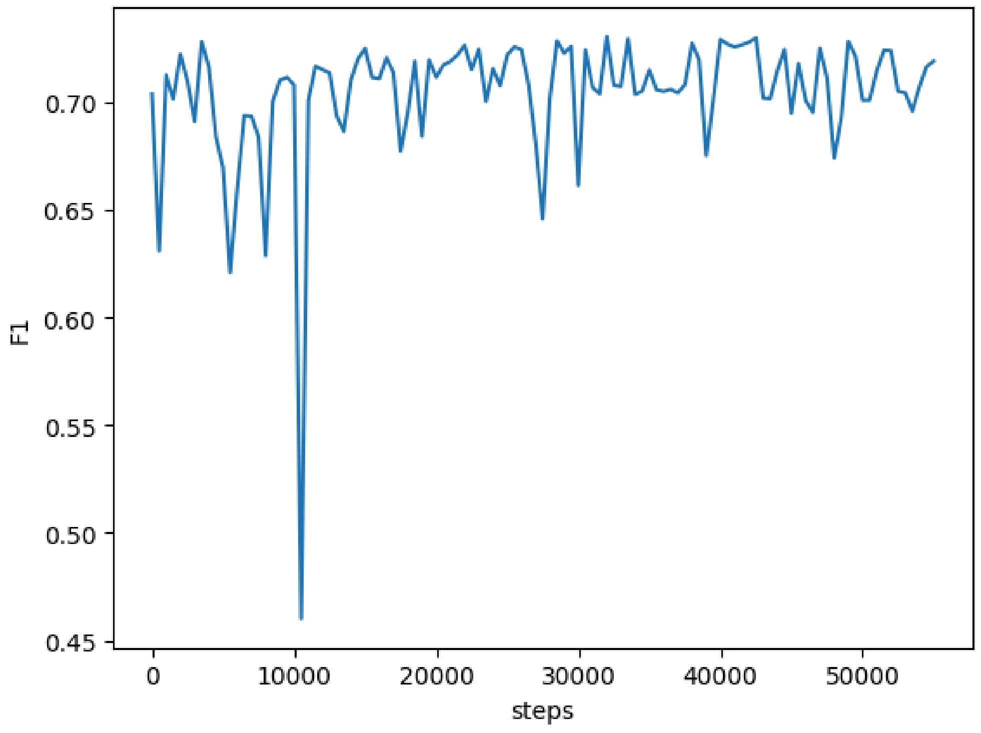


```
F1 Score:  0.6968580715059588
Accuracy:  0.7202
```

Problem 13 : PyTorch Model : Training for 1 epoch and Plot Loss and F1 Scores.

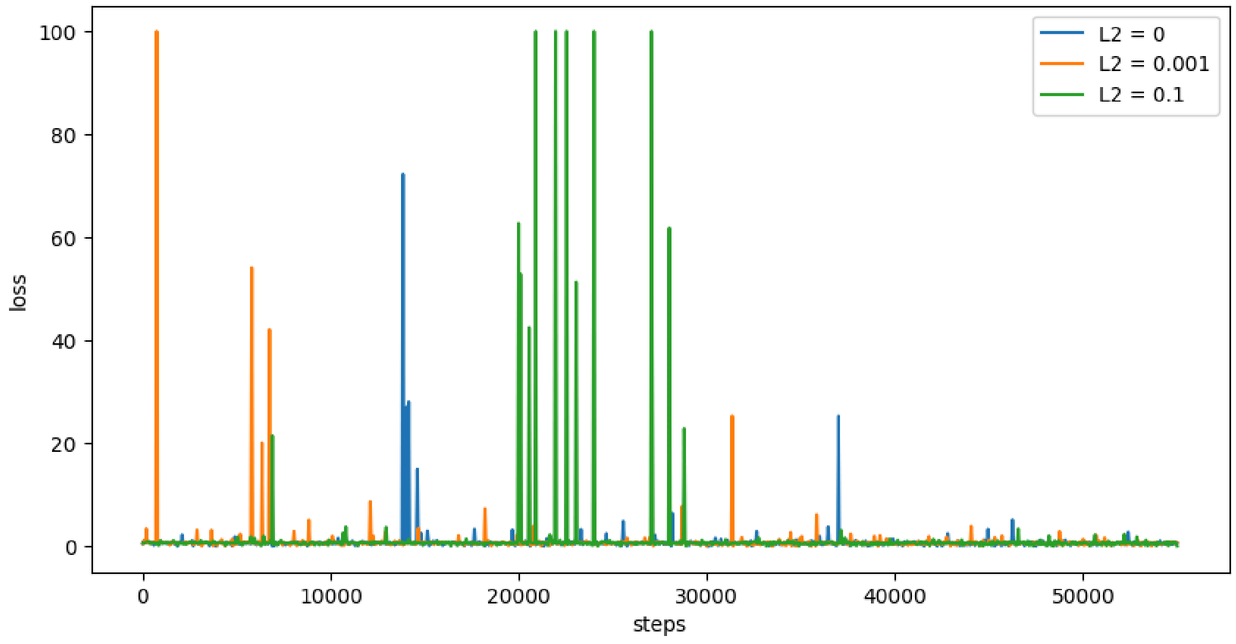


Loss other than some hikes seems to be going down overall, though there are a lot of hikes and drops in general.

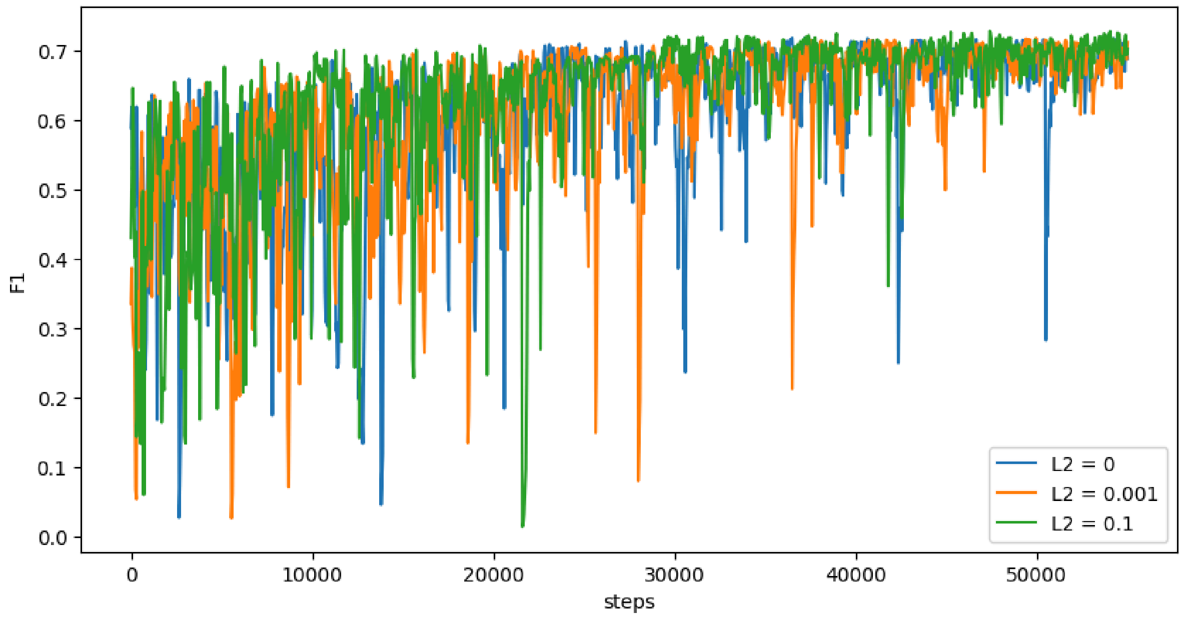


F1 scores seem to be increasing more/less consistently for 1 epoch which is the behavior we expect to see.

Problem 14 : PyTorch model - Effects of adding L2 regularization



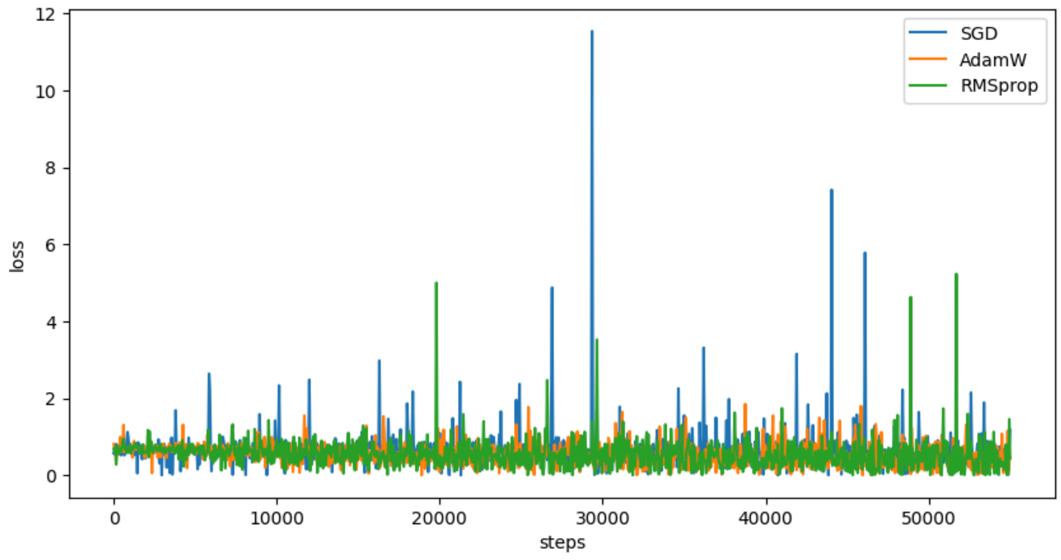
It seems like a smaller value of L2 regularization for example in our case 0.001, is much more stable in terms of minimizing loss. A large L2 value like 0.1 takes much bigger steps and in terms of loss has a lot more hikes during training though towards the end it seems like all three get very close to a similar range. When L2 is greater than zero like 0.001 but not too big like 0.1, the model seems to learn more consistently and loss is at a minimum for most of the training.



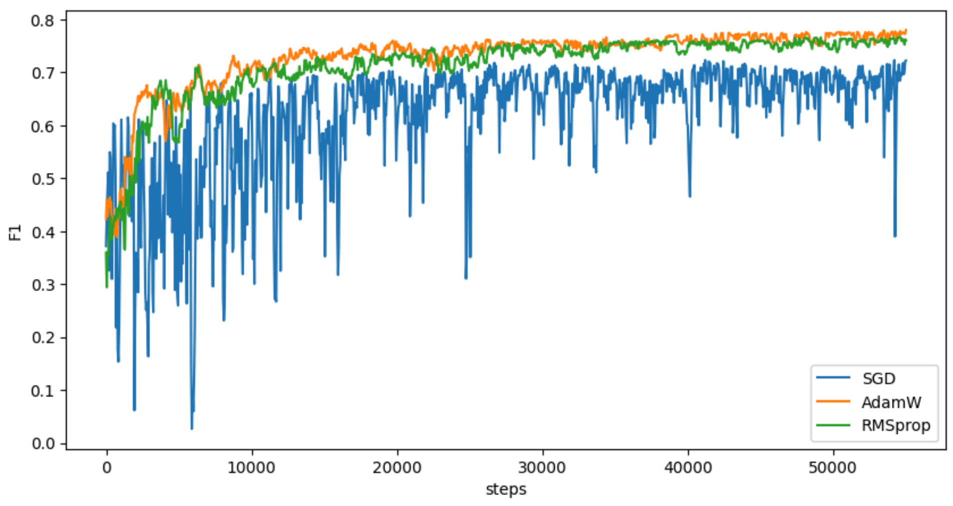
This also seems true in the case of F1 scores which seems to be consistently increasing. Once again, when L2 = 0.1 there seems to be a lot more variability within the scores especially in the beginning. It is interesting to note that default L2 clearly performs the worst in terms of F1 scores and there is a marginal difference in scores between L2=0.1 and L2=0.001.

```
Metrics for L2 = 0
F1 Score 0.6643371915265343 Accuracy 0.6926
Metrics for L2 = 0.001
F1 Score 0.7126781695423856 Accuracy 0.6936
Metrics for L2 = 0.1
F1 Score 0.7207876648708897 Accuracy 0.6994
```

Problem 15 : PyTorch model - Effects of Different Optimizers



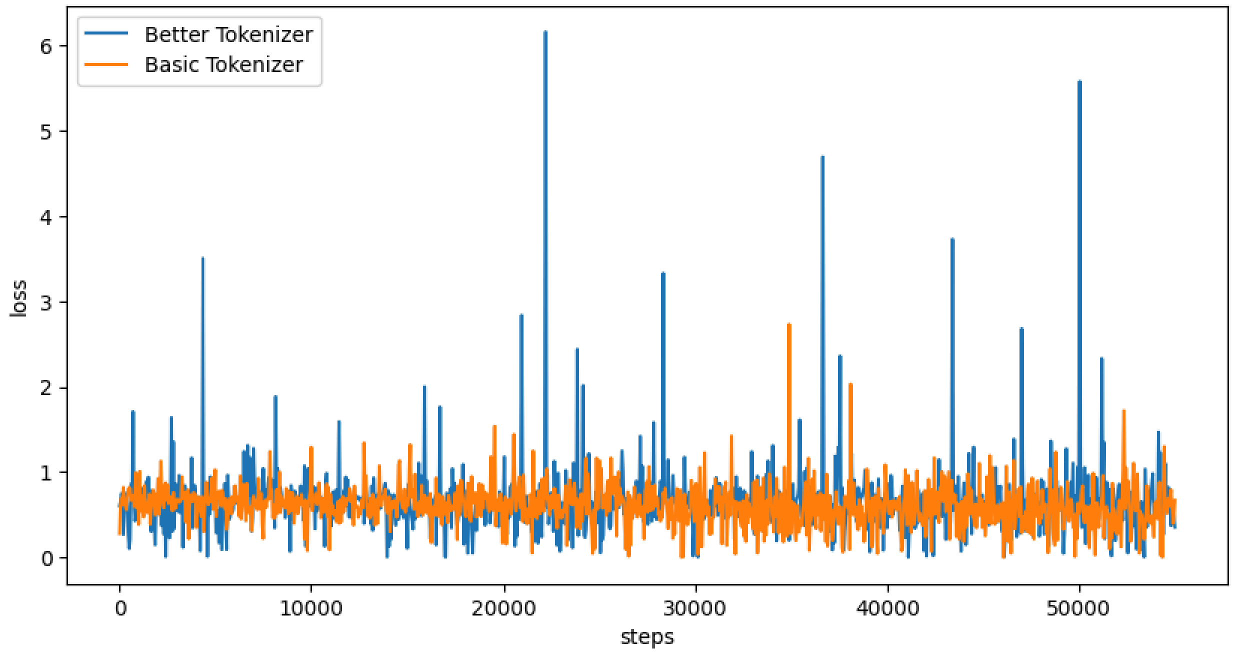
The AdamW optimizer is a clear winner in this case. Adam seems to converge much faster compared to both SGD and RMSprop and it can be clearly seen from the graph how loss is always minimal for the AdamW lineplot compared to both other plots which consistently have a lot of hikes where loss increases a lot. AdamW optimizer seems to be learning much quicker compared to both SGD and RMSprop.



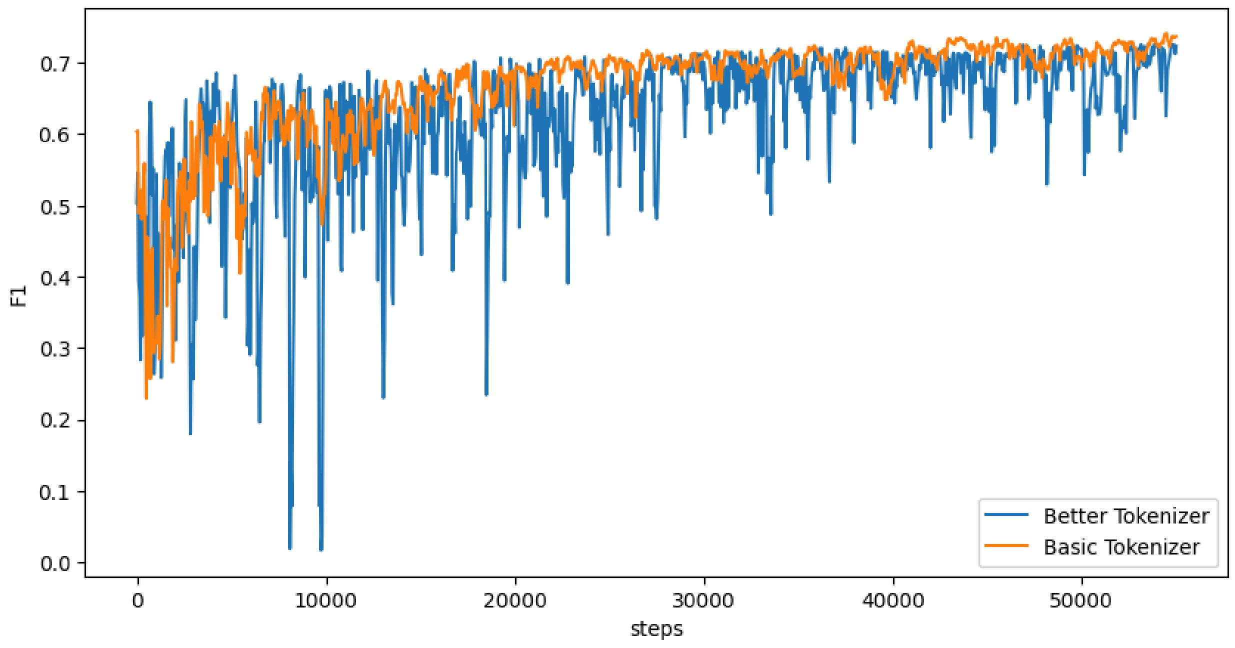
The same can be noted for the F1 scores. Though in the very beginning, SGD seems to perform better, AdamW consistently scores better than SGD and is at a neck to neck with RMSprop. It's also really important to note the amount of drops that SGD has in terms of variability which AdamW does not at all. Both accuracy and F1 scores are much higher for Adam as well.

```
Metrics for SGD
F1 Score 0.7222019686474662 Accuracy 0.6952
Metrics for AdamW
F1 Score 0.7797302819779321 Accuracy 0.7844
Metrics for RMSprop
F1 Score 0.7599591419816139 Accuracy 0.765
```

Problem 16 : PyTorch model - Effects of different Tokenizers



The basic tokenizer seems to not have any large hikes at all compared to the better tokenizer in terms of sudden upticks in loss. Towards the end both seem to have similar ranges though it's interesting to note that maybe there are certain cases where better rules for tokenization might not necessarily help. Something I thought was the effect of removing stop words in the better tokenizer which is a big difference compared to the basic function. This could potentially be an indicator towards stopwords and how they could be a feature that's important?

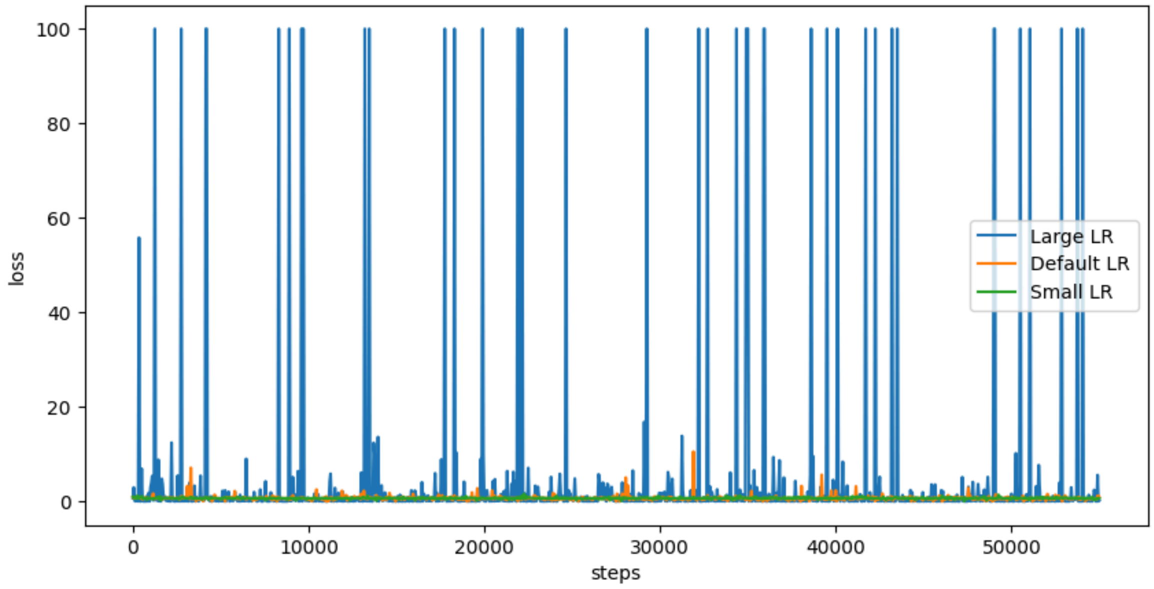


Similarly for F1 scores, overall the basic tokenizer seems to perform much better compared to the better tokenizer. The better tokenizer has a lot of clear variations especially in the beginning which decrease towards the end - though they don't completely go away. Model Performance also seems to be better in the case of the Basic Tokenizer compared to the Better Tokenizer which could possibly be an indicator that there were features that could have been included that were removed in the better tokenizer. This was a very surprising result!

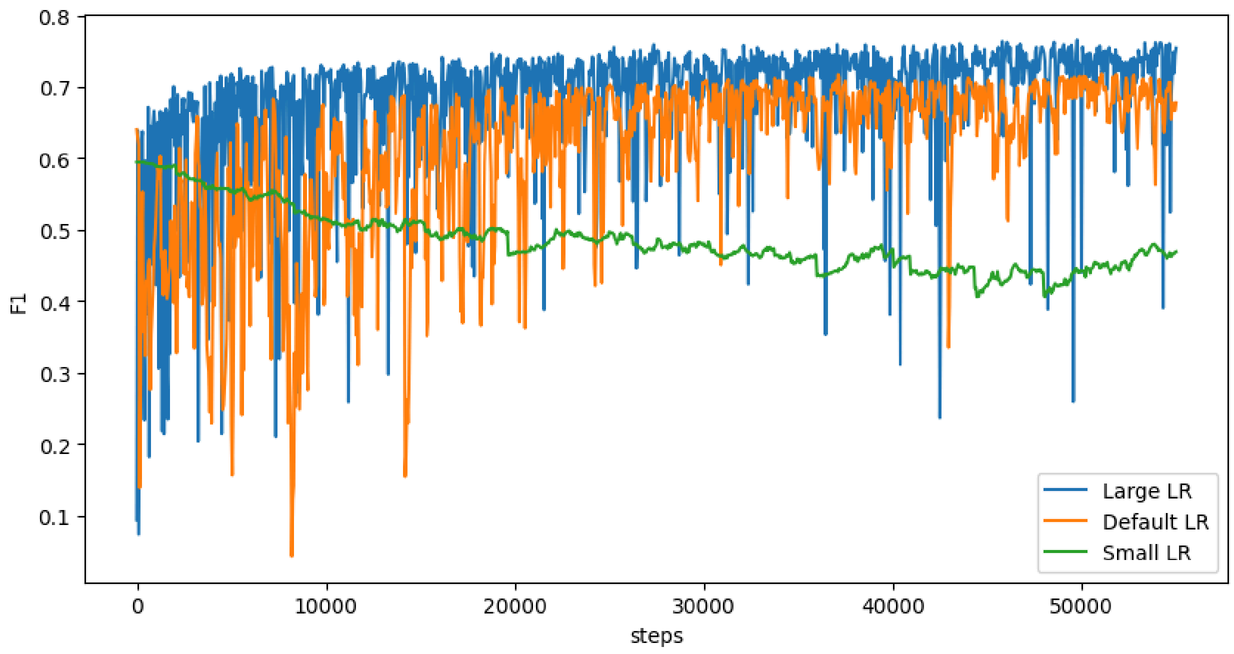
```
Metrics for Better Tokenizer
F1 Score 0.7239618788291354 Accuracy 0.6756
Metrics for Basic Tokenizer
F1 Score 0.7372408293460926 Accuracy 0.7364
```

Problem 17 : PyTorch model - Effects of different learning rates.

```
learning_rate_default = 5e-5
learning_rate_small = 5e-7
learning_rate_large = 5e-3
```



A large learning rate oversteps a lot. Loss increases constantly it's clear that it's overshooting and thus not performing well. As seen in class, it looks like it's overstepping the minimum and going past it to a situation where loss is higher. The default and small LR barely visible at the bottom don't make any big jumps. There are a few hikes in the case of the default LR which can't be seen at all in the case of the small LR which means it's learning too slow by taking too small of a step.



It's surprising to see how a large learning rate does seem to have a higher F1 score at the end compared to the default learning rate. F1 does seem to be dropping pretty heavily for both cases. What was really interesting to me in the case of a small learning rate is that F1 score seems to decrease over time! But it's also a very gradual decrease - no sharp drops or hikes at all. So it's consistently decreasing in performance. Overall the larger learning rate seems to perform the best and the smaller learning rate seems to perform the worst though it's unclear whether the loss plot where the large LR seems to overshoot is a desirable attribute or not.

```
Metrics for Default LR
F1 Score 0.6776525979485455 Accuracy 0.6166
Metrics for Small LR
F1 Score 0.46964856230031954 Accuracy 0.5352
Metrics for Large LR
F1 Score 0.7541431078182882 Accuracy 0.7478
```

Problem 16 : Best PyTorch Model

The model with the AdamW optimizer seems to have the highest F1 scores. These results have been reported to Kaggle under the username nandanasheri.

```
Metrics for AdamW - Highest F1 Scores
F1 Score 0.7797302819779321
Accuracy 0.7844
```

Problem 19 : PyTorch model - Most important features learnt

```
Top 10 Features for Model where L2 = 0:
also, :, including, novel, ingredients, film, instructions, world, a,
together

Top 10 Features for Model where L2 = 0.001:
also, :, ingredients, including, novel, instructions, significant, film,
story, named
```

(i) there is a certain amount of overlap between the two models. For example, 'also' and ':' (colon symbol) have the highest beta values for both models and though the weights change from there, words like 'novel', 'instructions' and 'including' are all still included in the both models.

(ii) A pattern that clearly stands out to me is how a lot of words that are LLM generated tend to have an underlying theme of *structure.* LLM responses are very structured whereas human generated text is usually much messier in terms of thoughts. A lot of these words like 'ingredients', 'instructions', 'including' are all usually used to lay out some structured responses which LLM's are notoriously known to do. Everything is planned and generated. Even how the ':' colon symbol shows up - that usually does convey some kind of instructions and it's one of most important features as well!

(iii) I think both `world` and `together` are a little confusing to be the heaviest weight features and could possibly be hinting at the dataset itself if those words were just commonly used in the training set. I do think ':' and 'also' make sense as features as explained in the previous answer about a lot of LLM generated responses being very articulate and structured. The colon also makes sense in that manner because I don't think a lot of human generated text would have colons extensively because we would use conjunctions instead.

(iv) I think there is so much overlap between the two models since a majority of the words do overlap it's much harder to pinpoint how the models would behave differently and which would be more likely to generalize. I believe that the model where L2=0.001 would generalize a bit more compared to the default L2 because the highest feature in default L2 contains 'a' which is an article that is used everywhere regardless of whether it's generated by an LLM. I also think that the features are a little less specific in the case of L2=0.001 but this could also be because I'm aware how in terms of F1 scores, the model performs better overall.