SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.021 Artificial Intelligence

AY 2021 Project Report

COVID-19 Retweet Prediction

https://github.com/chenyan1998/Term8-Artificial-Intelligent-Project

Team members:

Chen Yan	1003620
Dong Ke	1003713
Gladys Chua Shi Ying	1003585
Pang Luying	1003631
Tan Jun Yong	1003057

Content Page

1. Introduction

With the ongoing Covid-19 pandemic still rampant in today’s society, our source of news is not primarily limited to offline sources, but also online sources. Through global digitalisation, data sources are not limited to the popular forum Reddit, social media (Facebook, Twitter, Instagram), where consumers obtain their plethora of information. To better understand how consumers spread their information from various news outlets during the Covid-19 pandemic. Hence the CIKM2020 AnalytiCup Covid-19 Retweet Prediction Challenge was chosen for 50.021 Artificial Intelligence (AI) course project.

For this challenge, the TweetsCOV19 dataset, a dataset available to the public with more than 8 million Covid-19 related tweets from the duration of October 2019 to April 2020. However, for the sake of this challenge, only the May 2020 dataset is used. Based on each tweet, the dataset provides metadata and preprocessed features such as entities and sentiment scores.

The Covid-19 Retweet Prediction Challenge requires predicting the number of retweets from the post made. On twitter, users can post tweets which are short text posts, and retweet refers to the sharing of another user’s tweet (post) and as a means to share information, hence amplifying the spread of information through the network. This is inline with the AI course project to implement an innovation solution for an AI related problem.

2. Data Preprocessing

2.1 Datasets

The dataset is from CIKM202 AnalyticCup COVID-19 Retweet Prediction Challenge where a public dataset of more than 8 million tweets spanning the period Oct 2019 to Dec 2020 is available. Dataset spanning May 2020 was used for this paper together with the keyword list at the same time. The dataset contains the following information (see Table 2-1).

Feature Name	Description
Tweet Id	Long. An unique ID for each tweet.
Username	String. Unique encrypted string for each user.
Timestamp	String with format “EEE MMM dd HH:mm:ss Z yyyy”. The timestamp that the tweet is posted.
Number of Followers	Integer. Number of followers the user has.
Number of Friends	Integer. Number of friends the user has.
Number of Retweets	Integer. How many times the tweet has been retweeted.
Number of Favorites	Integer. The number of favorites the tweet has.
Entities	String. For each entity, we aggregated the original text, the annotated entity and the produced score from the FEL library. Each entity is separated from another by char “;” and in the format of “original_text:annotated_entity:score”. If FEL did not find any entities, “null” is stored.
Sentiment	String. SentiStrength produces a score for positive (1 to 5) and negative (-1 to -5) sentiment. Stored in the format of “positive_score negative_score”.
Mentions	String. If the tweet contains mentions, we remove the char "@" and concatenate the mentions with whitespace char " ". If no mentions appear, "null;" is stored.
Hashtags	String. If the tweet contains hashtags, we remove the char "#" and concatenate the hashtags with whitespace char " ". If no hashtags appear, we have stored "null;".
URLs	String: If the tweet contains URLs, we concatenate the URLs using ":-: ". If no URLs appear, we have stored "null;"

Table 2-1

2.2 Features

As the features in the original dataset includes numerical values and words, we have to implement encoding on the non-numerical data and extract the necessary features as the input to the neural network model.

Tweet Metrics Number of retweets is affected by the number of followers and friends of the user who post the tweet and the number of favorites of the tweet. The more followers and friends and followers the user has, the more likely the tweet will be retweeted. Large number of favorites means the tweet is popular and the tweet is more likely to be retweeted. Therefore, other than number of followers, friends and favorites, we have other features such as friends_favorites which is multiplication of number of friends and number of favorites.

Tweet Metrics Z Transformation The values of the features in the tweet metrics mentioned above are large integer numbers, especially the features that are the result of multiplication. We normalize each value X with mean μ and standard deviation $\sigma$of the whole training dataset to get a normalized value XZ.

$$XZ = \frac{X-\mu}{\mu}$$

Positive Sentiment Score & Negative Sentiment Score Sentiment scores calculated by SentiStrength [LINK] are stored as non-numerical values in the original dataset. We separated the positive and negative sentiment scores and stored them as numerical values so that they can be used for calculation.

Weekday, Hour, Day, Week of Month The time when a tweet is posted will affect the number of retweets. For example, if the tweet is posted on the weekend, the number of retweets might be larger as most people do not work on weekends and they might have more time to read and retweet all the tweets. Therefore, the non-numerical value of timestamp in the original dataset is split into numerical values - Weekday, Hour, Day, Week of Month. Weekday is labeled from 0-6 to represent Monday to Sunday. Hour is labeled from 0-23 to represent a particular hour in a day and Day is labeled from 1-31 to represent a particular day in a month. Week of Month *i* is ranging from 1-5 to indicating *i*th week in a month.

Entities Entities in a tweet reflect the content of the tweet which also affects the number of the retweets. The score produced by Faster Entity Linker is extracted to represent the entity in the tweet. If there are multiple entities, the value is the sum of all FEL scores of all entities.

Hashtag The more hashtags in a tweet, the more likely the tweet will be viewed by people because hashtags link the tweet to particular hot topics at the moment of posting. Therefore, we count the number of hashtags in a tweet.

Mentions Mentions direct the tweet to other users which increases the chances of the tweet being viewed and also increases the chances of retweet. We count the number of mentions in a tweet as a feature to input into the model.

URLs URLs link the tweet to other resources which enrich the content of a tweet and thus may make the tweet to be more interesting and attract more users to read the tweet and retweet. Therefore, we use the number of URLs in a tweet as a feature as well.

Components of URLs As URLs can link to different resources such as a video on YouTube or a News in Strait Times, we have to extract the information that indicates where the URLs direct to. An URL to video is easier to view than an URL to News with the same content because videos convey the information in a more straightforward way which reduces the workload of getting information compared to News. Therefore, we extract hosts (e.g., youtube) of URLs and calculate the frequency of specific hosts that appear in the dataset. If the host is more likely to be viewed by users, there is a higher chance the tweet will be retweeted.

Keyword Count Words appear in entities, hashtags, mentions and URLs might be in the keyword list. The keyword list is an extension of the seed list of Chen et al [LINK]. It contains 268 COVID-19 related words which indicate the broad view on the societal discourse on COVID-19 on Twitter. More keywords appear in the tweet, the more likely the tweet is related to hot topics. Therefore, We count the number of keywords that appear in entities, hashtags, mentions and URLs.

Count Encoding This feature counts the observations of a particular category appearing in the dataset. Count encoding is applied to positive sentiment, negative sentiment, weekday,hour,day and week of month.

Target Encoding The value for this feature is obtained by applying target encoding [LINK] to positive sentiment, negative sentiment, week, hour, day and week of month.

3. Model & Loss, Hyperparameters Optimisation

Using the extracted features, we trained the different neural networks with the newest dataset that have features selection. We trained 910145(1300207 * 0.7) samples to our neural network models. We use MSE loss as our loss function. Adam optimizer as our optimizer. The basic structure is a neural network. There are four different parameters that we tune in our neural networks:

1. Number of hidden layers of neural networks
2. Dropout Rate
3. Batch Normalization
4. Last Activation Layer

In order to better understand the sensitivity of the various parameters, we ran multiple experiments with parameters that are kept constant while others are altered. Considering the training time, I did 5 epochs for each model.

Before tuning the model, we use kaiming initialization. We did not want to initialise the weights to 0 because the learning rate would be the same for all the weights of the neural network, because random initialization could cause an exploding or vanishing gradient. Kaiming initialisation solves this problem by multiplying random initialization with a factor $\sqrt{\frac{2}{size^{[l-1]} + size^{[l]}}}$. This avoids slow convergence, ensuring that we do not keep oscillating off the minima. Kaiming initialization is also more accurate than Xavier initialization, especially if the activation function does not have a derivative of 1 at 0, like the ReLU function which we are using for the project.

In the first experiment, we modified the number of hidden layers for neural networks, keeping other parameters fixed(Neurons of each layer, Loss Function, Optimizer, Dropout, Last Activation Layer and Batch Normalization). By increasing the number of hidden layers, it could potentially improve the accuracy of the model, as it depends on the complexity of the problem. The results are shown in Table 3-1. We observe that the three hidden layers neural network gives the best performance, followed by the two hidden layers neural network.

Model	Number of hidden layers of neural networks	Each layer	Dropout rate	Training loss	Validation loss
Net0	2	nn.Linear(input,2048), nn.Linear(2048,128),	0.5	0.6874393123101358	0.42668487353844 46
Net1	3	nn.Linear(input,2048),	0.5	0.3512702	0.257726

		nn.Linear(2048,128), nn.Linear(128,64)		9478670796	3164165532
Net2	4	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,256), nn.Linear(256,128),	0.5	0.7900380790708533	0.6451338381705567

Table 3-1

We observed that our validation loss has a bit of oscillation for the above models when training for multiple epochs, which might be caused by overfitting. Batch normalization solves a major problem called internal covariate shift. It helps by making the data flow between intermediate layers of the neural network look, this means you can use a higher learning rate. Additionally, it is one of the solutions to solve overfitting.

In our second experiment, we add batch normalization to the best model that we have selected(The number of hidden layers for neural networks equals to three). We found the model with batch normalization decreased the training loss by a significant amount. The results are as shown in Table 3-2.

Model	Each layer	Training loss	Validation loss	Batch Normalization
Net1	nn.Linear(input,2048), nn.Linear(2048,128), nn.Linear(128,64)	0.35127029478670796	0.2577263164165532	No
Net5	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0.15754297450064259	0.26738035046574044	Yes

Table 3-2

In the third experiment, we keep the number of layers equal to three and other parameters the same. We used the dropout rate of 0, 0.3, 0.5 for each neural network. The main advantage of the dropout layer is that it prevents all neurons in a layer from synchronously optimizing their weights. Therefore, we tuned the model with various dropouts. The results are shown in Table 3-3.

Model	Last Activation Layer	Each layer	Dropout rate	Training loss	Validation loss	Batch Normalization
Net5	ReLu	nn.Linear(input,2048), nn.Linear(2048,512),	0	0.1575429745006	0.267380350465	Yes

		nn.Linear(512,128)		4259	74044	
Net7	ReLu	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0.3	0.3334901887205 7795	0.22623114627545982	Yes
Net4	Sigmoid	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0	0.4179223859998 988	0.21745016016721658	Yes
Net6	Sigmoid	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0.25	0.5503472220268 596	0.29493665312271 81	Yes

Table 3-3

After the third experiment, we observed that the training loss and validation loss have a reasonable result. However, we continue to try to tune some other parameters to see if they are better than our previous model. Activation functions are a key part of neural network design. The modern default activation function for hidden layers is the ReLU function. The activation function for output layers depends on the type of prediction problem. Therefore, in our fourth experiment, we change the last activation layer after a fully connected layer with ReLu and Sigmoid. The results are shown in Table 3-4.

Model	Last Activation Layer	Each layer	Dropout rate	Training loss	Validation loss	Batch Normalization
Net5	ReLu	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0	0.15754297450064259	0.2673803504657 4044	Yes
Net4	Sigmoid	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0	0.4179223859998988	0.21745016016721658	Yes

Table 3-4

The full results can be seen as in Table 3-5.

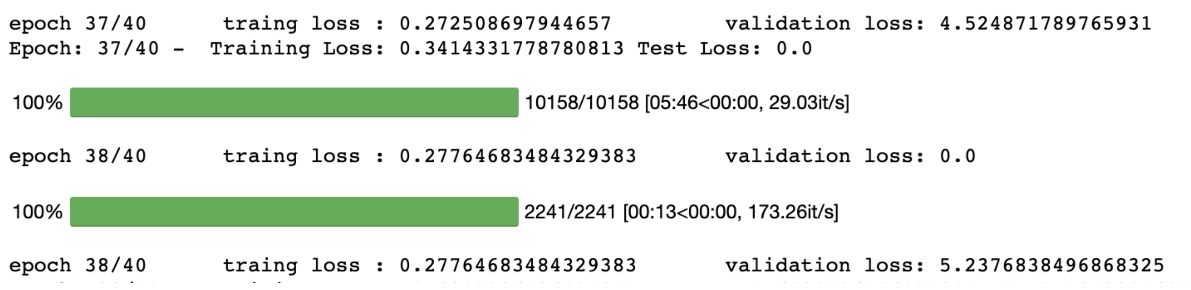
Model	Each layer	Dropout rate	Training loss	Validation loss	Batch Normalization
Net0	nn.Linear(input,2048),	0.5	0.6874393123101358	0.4266848735384446	NO

	nn.Linear(2048,128),				
Net1	nn.Linear(input,2048), nn.Linear(2048,128), nn.Linear(128,64)	0.5	0.35127029478670796	0.2577263164165532	NO
Net2	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,256), nn.Linear(256,128),	0.5	0.7900380790708533	0.6451338381705567	NO
Net3	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0	0.20690901891491786	0.4849992956962681	No
Net4	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0	0.4179223859998988	0.21745016016721658	Yes
Net5	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0	0.15754297450064259	0.26738035046574044	Yes
Net6	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0.25	0.5503472220268596	0.2949366531227181	Yes
Net7	nn.Linear(input,2048), nn.Linear(2048,512), nn.Linear(512,128)	0.3	0.33349018872057795	0.22623114627545982	Yes

Table 3-5

4. Model Evaluation & Performance, Results

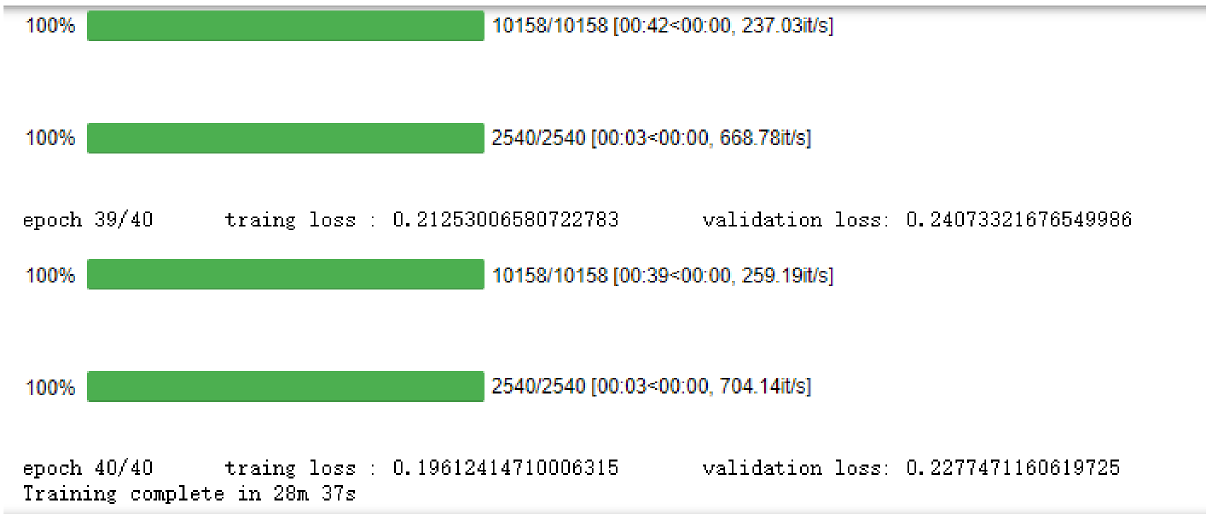
After exploring models and parameters, the best performing model is Net4 and Net7. Therefore, we decided to compare the two models using more epochs. We found out that the validation loss for Net4 diverges probably due to overfitting since it does not have any dropout layer. The model is not generalized. Therefore we used Net7 as our final model.



```
epoch 37/40      traing loss : 0.272508697944657        validation loss: 4.524871789765931
Epoch: 37/40 -  Training Loss: 0.3414331778780813 Test Loss: 0.0

100% 10158/10158 [05:46<00:00, 29.03it/s]

epoch 38/40      traing loss : 0.27764683484329383      validation loss: 0.0

100% 2241/2241 [00:13<00:00, 173.26it/s]

epoch 38/40      traing loss : 0.27764683484329383      validation loss: 5.2376838496868325
```

Fig 4-1. Diverging validation loss due to overfitting

We achieved the lowest validation loss of 0.22167146 using 40 epochs. We used Adam optimizer with learning rate 0.001, MSE loss function and StandardScaler to normalize the input. The lowest training loss reaches 0.196124147. Although there is oscillation in the validation loss, and it successfully converged, we also overcame the issue of oscillation by recording down the model with the lowest validation loss instead of saving the last model weights.



```
100% 10158/10158 [00:42<00:00, 237.03it/s]


100% 2540/2540 [00:03<00:00, 668.78it/s]

epoch 39/40      traing loss : 0.21253006580722783      validation loss: 0.24073321676549986
100% 10158/10158 [00:39<00:00, 259.19it/s]


100% 2540/2540 [00:03<00:00, 704.14it/s]

epoch 40/40      traing loss : 0.19612414710006315      validation loss: 0.2277471160619725
Training complete in 28m 37s
```

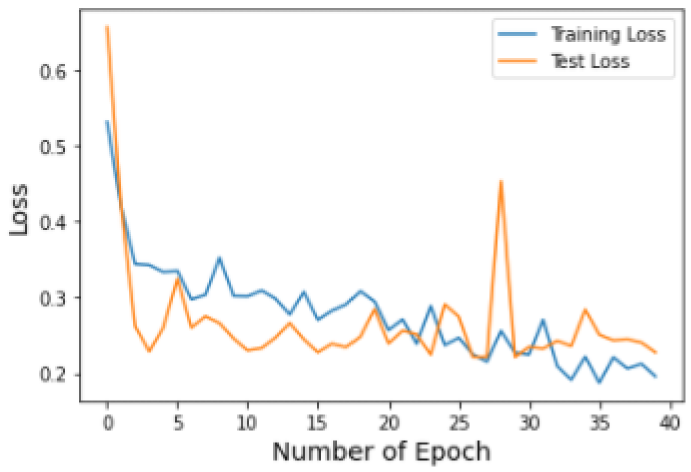
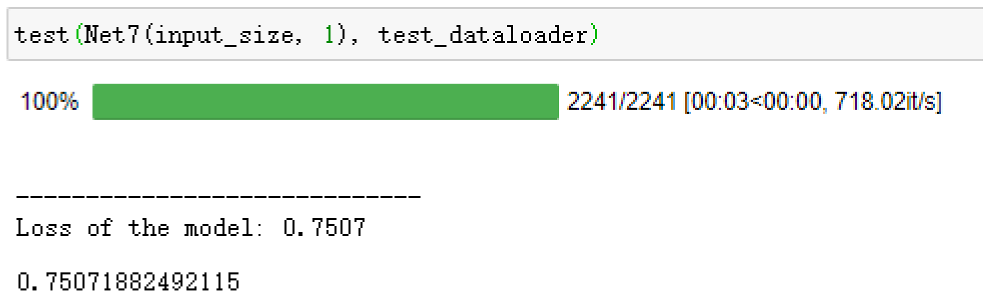


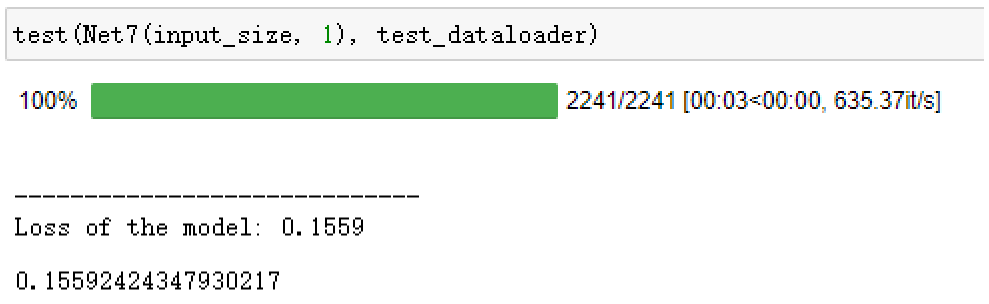
Fig 5-2. Training and validation loss of Net7

With the saved model for best performance, we tested the test dataset. The final test loss we obtained is 0.75 for MSE and 0.156 for MSLE.



```
test(Net7(input_size, 1), test_dataloader)
```

```
100%  2241/2241 [00:03<00:00, 718.02it/s]


------------------------------
Loss of the model: 0.7507

0.75071882492115
```



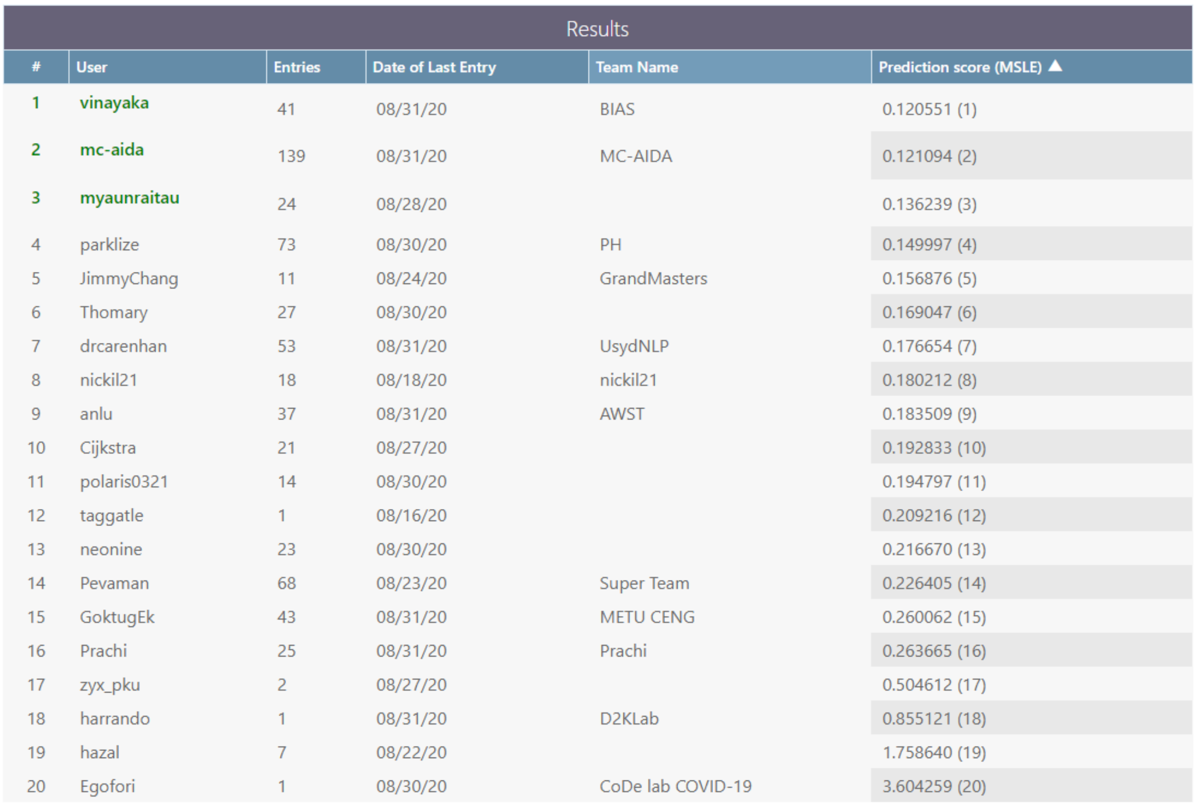
```
test(Net7(input_size, 1), test_dataloader)
```

```
100%  2241/2241 [00:03<00:00, 635.37it/s]


------------------------------
Loss of the model: 0.1559

0.15592424347930217
```

Fig 5-3. Testing loss of Net7 [Top: MSE; Bottom: MSLE]

Comparing our model with the state of the art model with those who were in the leaderboard of the competition, in the training we used MSE instead of MSLE. The reason for the choice is that MSLE only cares about the relative difference between the true and the predicted value, or in other words, it only cares about the percentual difference between them, however, we wanted to take into consideration the real difference between the prediction and actual output. Additionally, MSLE penalizes underestimates more than overestimates, thus it is not as fair as MSE. Therefore, we used MSE during training.

However, in the final test, we recorded our MSLE score for comparison. The MSLE score was better than the 6th place.



Results

#	User	Entries	Date of Last Entry	Team Name	Prediction score (MSLE) ▲
1	vinayaka	41	08/31/20	BIAS	0.120551 (1)
2	mc-aida	139	08/31/20	MC-AIDA	0.121094 (2)
3	myaunraitau	24	08/28/20		0.136239 (3)
4	parklize	73	08/30/20	PH	0.149997 (4)
5	JimmyChang	11	08/24/20	GrandMasters	0.156876 (5)
6	Thomary	27	08/30/20		0.169047 (6)
7	drcarenhan	53	08/31/20	UsydNLP	0.176654 (7)
8	nickil21	18	08/18/20	nickil21	0.180212 (8)
9	anlu	37	08/31/20	AWST	0.183509 (9)
10	Cijkstra	21	08/27/20		0.192833 (10)
11	polaris0321	14	08/30/20		0.194797 (11)
12	taggatle	1	08/16/20		0.209216 (12)
13	neonine	23	08/30/20		0.216670 (13)
14	Pevaman	68	08/23/20	Super Team	0.226405 (14)
15	GoktugEk	43	08/31/20	METU CENG	0.260062 (15)
16	Prachi	25	08/31/20	Prachi	0.263665 (16)
17	zyx_pku	2	08/27/20		0.504612 (17)
18	harrando	1	08/31/20	D2KLab	0.855121 (18)
19	hazal	7	08/22/20		1.758640 (19)
20	Egofori	1	08/30/20	CoDe lab COVID-19	3.604259 (20)

Fig 5-4. State-of-the-art MSLE Loss

5. Graphic User Interface (GUI)

• A description how to set up your code in order to be able to run the GUI.
• GUI demonstration.

Our GUI is created through the Streamlit library. Streamlit is an open-source web framework primarily used for Machine Learning and Data Science as they provide a simple environment to create interfaces with the removal of many frontend and design barriers. The GUI contains data visualisation of the dataset and to predict the number of times a tweet will be retweeted (#retweets). It can load the saved models and allow users to easily try the model on the front-end.

5.1 Setup of Code for GUI

The following libraries are needed to run the GUI:

1. numpy == 1.19.0
2. scipy == 1.5.2
3. streamlit == 0.81.0
4. plotly == 4.12.0
5. torch == 1.8.0
6. pandas == 1.0.5
7. matplotlib == 3.2.2
8. scikit_learn == 0.24.2
9. python == 3.7

From the command line with the current path set to our project directory, run the command: **py -3 -m streamlit run gui.py**. The GUI can be accessed through the web browser at **http://localhost:8501.** The first run would take longer as the GUI script is reading the data from the large dataset (over one million rows of data). The subsequent runs will be faster as Streamlit has provided the cache option to allow our GUI to stay performant. The video can be accessed in the Github README.md file.

6. Conclusion

This project presents our solution for the AI Project through the CIKM2020 AnalytiCup Covid-19 Retweet Prediction Challenge. The solution proposed is a Net7, a 3-layered NN model using 30% dropout rate and batch normalisation. To further improve the performance, additional features were extracted (numerical feature transformation and introducing count and target) to be fed into Net7. The final solution for the AI project and Retweet Prediction Challenge, the MSE Loss achieved is 0.1559, which is in the top 5 of the challenge. Future improvements of the model include increasing the number of layers and further data processing through binning or continuous and numerical variables into categorical features.

7. References

Mean squared logarithmic error (MSLE): Peltarion Platform. (n.d.). Retrieved from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle)

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter, 3*(1), 27-32. doi:10.1145/507533.507538

Yahoo. (n.d.). Yahoo/FEL: Fast Entity Linker Toolkit for training models to link entities to KnowledgeBase (Wikipedia) in documents and queries. Retrieved from https://github.com/yahoo/FEL

Echen102. (n.d.). COVID-19-TweetIDs/keywords.txt at master · echen102/COVID-19-TweetIDs. Retrieved from https://github.com/echen102/COVID-19-TweetIDs/blob/master/keywords.txt