**CSE598 – Intro to Deep Learning**
**Fareed Osman**

# Project Report – SMS Spam Filter

## 1. Project Summary

Over the years, spam messages have grown to take up a large share of SMS traffic. Most cellphones and email service providers will deploy a spam filter to help solve this problem. The job of a spam filter is to automatically filter out any suspected spam messages from a user's inbox, which reduces inbox cluttering and protects the user from falling victim to scams.

My project proposal is to build and train a spam filter through machine learning, using a data set that contains the texts of several thousand SMS messages.

## 2. Problem

The model should take the text of an SMS message, and either classify it as a spam message or decide that it is legitimate. To develop the spam filter, it will be necessary to find a way to represent the text of an SMS message as a data type and to define its attributes. The attributes should be engineered such that they can help distinguish normal messages from spam.

## 3. Dataset

The dataset I used for this project can be viewed here:
https://www.kaggle.com/uciml/sms-spam-collection-dataset

The data is a selection of 5,574 SMS messages, each tagged as either "spam" or "ham," contained in a csv file. The tag is in the first column of each row, and the SMS text itself is in the second column. As one might expect, the classes/labels for the data are identical to the tag, with "spam" representing a spam message and "ham" representing non-spam messages.

To divide the dataset, 1,000 SMS messages were randomly extracted from the data and set aside for evaluating the model. 870 of those messages were non-spam and the remaining 130 were spam, so as to make sure that the rate of spam in the validation data is the same for the rest of the data (approximately 13% for the entire dataset).

Due to issues encountered while training the model, another dataset was created out of the 4,574 messages in the training data, that consisted of equal numbers of spam and non-spam messages (617 messages each). This was done to solve the problem of the model quickly converging to a solution where it assumes all messages are non-spam.

## 4. Evaluation Metrics

Accuracy will be calculated using a weighted $\mathrm{F}_\beta$ score, with $\beta = 0.5$, since I would place a higher value on precision in spam detection. (We generally would prefer to deal with spam messages than to potentially miss an important message that got filtered by mistake)

Therefore: $\mathrm{F}_\beta = (1 + 0.25) * \frac{\text{precision} * \text{recall}}{0.25 * \text{precision} + \text{recall}}$

The baseline model is one that compares the SMS message against a list of keywords, and if the message contains an instance of one of keywords, it will flag the message as spam. For the purposes of testing, the keyword list is a set of 4 frequently occurring words in spam messages: [“CLAIM”, “FREE”, “PRIZE”, “AWARD”].

The following is a snippet of the code used in the forward pass for this baseline:

```
if [i for i in self.keywords if i in words]:

      return -1

else:

      return 1
```

Where `keywords` is the list of common spam terms, `words` is the list of words in the input message, and outputs of -1 or 1 represent the highest confidence value for a spam and non-spam prediction respectively.

Using this evaluation metric on the baseline described above, we get the following results:

$\text{Accuracy} = 90.40\%$

$\text{Precision} = 0.8269$

$\text{Recall} = 0.3308$

$\mathrm{F}_\beta = 0.6361$

## 5. Experiments

The model I developed is a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM). The architecture of my model was based on course material for a deep-learning course at the Technical University of Denmark (DTU), titled *How to build RNNs and LSTMs from scratch with NumPy*, by Nicklas Hansen, Peter E. Christensen and Alexander R. Johansen. The full material can be found here: https://github.com/nicklashansen/rnn_lstm_from_scratch

Since the model described in the course is a many-to-many RNN which outputs strings of characters, my model was modified into one that gives a binary classification as its output.

### - Vocabulary

To encode the SMS message input, I created a set of 1,000 vocabulary terms, which consist of the most commonly occurring words in both spam and non-spam messages (excluding a number of words that frequently occurred in both to avoid redundancy). The vocabulary was created from "simplified" instances of words that occurred in the training data. This was done by first converting the SMS messages into lists of words (separated by whitespace), turning all lowercase letters into uppercase, replacing all punctuation symbols with the character '!', replacing all numerical digits with a '5', and replacing all other remaining characters with a '@'. This was done to minimize the issue of duplicate terms in the vocabulary.

For example, the following SMS message:
```
Kit Strip - you have been billed 150p. Netcollex Ltd. PO Box
1013 IG11 OJA
```

Would be represented as:
```
["KIT", "STRIP", "!", "YOU", "HAVE", "BEEN", "BILLED", "555P!",
"NETCOLLEX", "LTD!",  "PO", "BOX", "5555", "IG55", "OJA"]
```

Using the list of vocabulary terms, inputs into the model are separated into words, simplified to avoid duplicates, and then each word is one-hot encoded into a (1000, 1) NumPy array, based on if they match one of the terms in the vocabulary. Words in the input that do not correspond to any of the words in the vocabulary are simply ignored.

For example, given the original SMS message input:
```
do u want 2 meet up 2morro
```

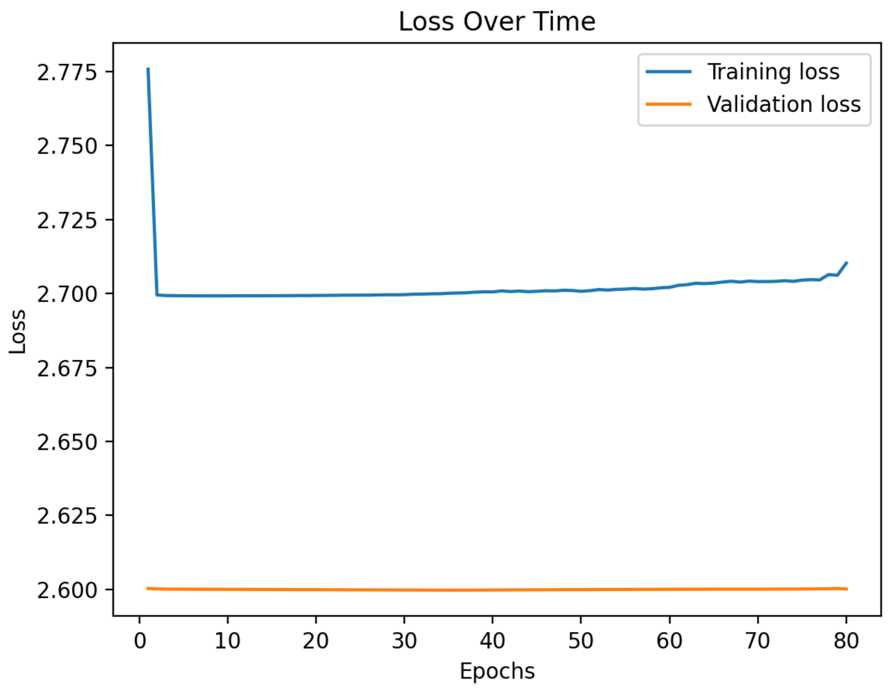
The input is separated into a list of words and simplified into the following:
```
["DO", "U", "WANT", "5", "MEET", "UP", "5MORRO"]
```

Finally, the list is then one-hot encoded, after removing words that aren't in the vocabulary:
```
[DO], [MEET], [UP]
```

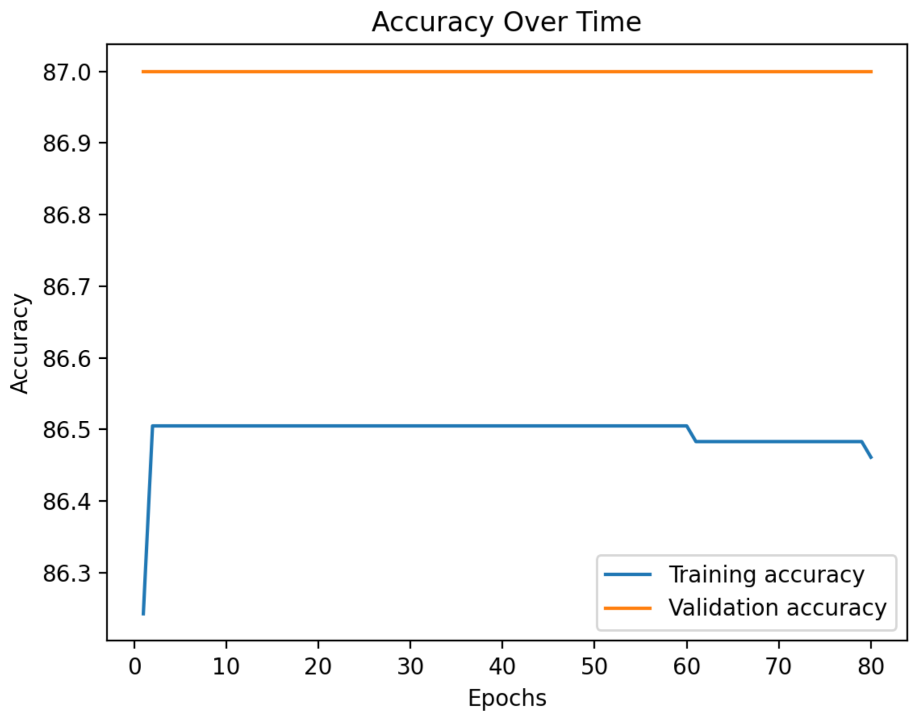
The encoded input is fed into the LSTM in the same order the words appear in the original message. The output is a value between -1 and 1 representing the model's prediction.
(A unique entry in the vocabulary is reserved for message that have no vocabulary words)

**- Training**

While first attempting to train my model, I frequently encountered a problem where the model would rapidly converge into predicting all messages to be non-spam. This is because the training dataset, like the full dataset, consists mostly of non-spam messages.

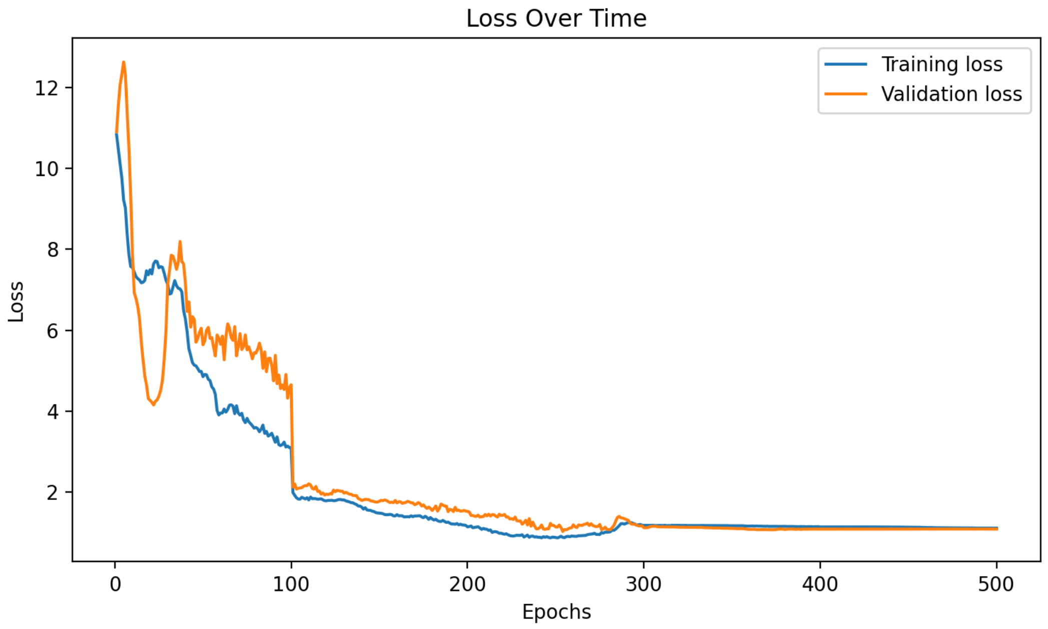


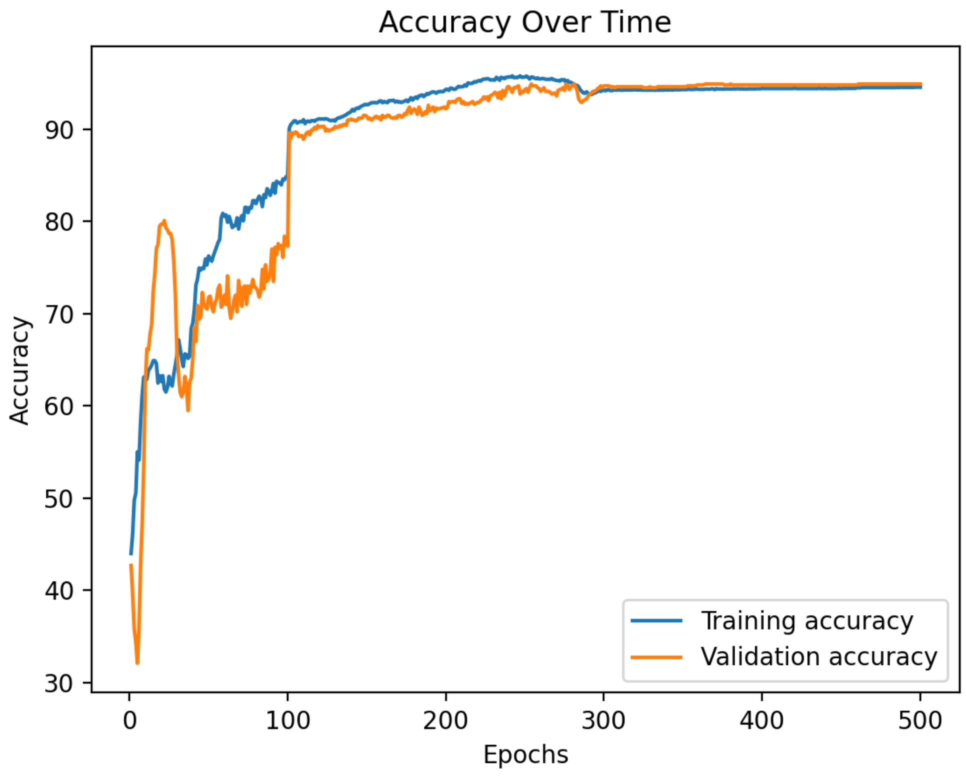
Loss and accuracy over time while training with unbalanced data



To remedy this, I instead created a new dataset that consisted of all the spam messages in the training dataset, plus an equal number of non-spam messages (i.e. a subset of the training dataset). I then trained the model using the smaller balanced dataset for 100 epochs, and then switched back to the full training dataset for the remainder of the training.

Using the 1,000-word vocabulary to encode inputs, and a hidden layer size of 20, my model showed the following training result:





The learning rate applied was 0.05 for epochs 1 through 100, using the balanced dataset. Starting from epoch 101, the full training dataset was used instead, at which point the model saw a sharp decline in the loss. After epoch 300, the learning rate was decreased to 0.005. For the last 100 epochs (401 to 500), the learning rate was decreased again to 0.001.

**- Evaluation**

Evaluating the trained model against the validation dataset, we get the following results:

$\text{Accuracy} = 94.90\%$ (baseline $90.40\%$)

$\text{Precision} = 0.9072$ (baseline $0.8269$)

$\text{Recall} = 0.6769$ (baseline $0.3308$)

$\text{F}_{\beta} = 0.8494$ (baseline $0.6361$)

After 500 epochs, the trained LSTM model, with a hidden size of 20 and an input vocabulary size of 1,000, outperforms the baseline in all metrics, with the largest improvement over the baseline being in the recall value.

However, since I weighted the F-score so that precision would be emphasized more than recall, the F-score improvement over the baseline is somewhat modest. Considering that the overall dataset (as well as the training and validation datasets) overwhelmingly consists of non-spam messages, any model that is conservative in flagging messages as spam should put up good results for both accuracy and precision. Therefore, I believe it is still meaningful that the LSTM model can achieve significantly improved recall values, for the same high accuracy and precision as the baseline.

**- Areas for Improvement**

One of the areas where I believe my model could be potentially improved is in scouting for common misspellings and variations of certain words. The vocabulary I used avoids the issue of duplicates to some extent by ignoring case and punctuation, but it will not register misspelled or abbreviated words, something that is particularly a problem considering the dataset is of SMS messages. The fact that numbers are ignored also means the model won't be able to interpret words that use numbers in place of certain syllables.

To improve the model, I could gather data on common spelling variations for words on social media, and then have all such instances be represented in a standardized form in the vocabulary. For example, the words "UR" and "YOUR" would have the same one-hot encoding, as would "4" and "FOR", etc.

## - Bonus Experiment: Spam Emails

As a bonus experiment, I wanted to see how well my model would perform in classifying emails, rather than SMS messages. A similarly sized email dataset can be found here:
https://www.kaggle.com/venky73/spam-mails-dataset
The dataset contains 5,170 emails with a spam frequency of approximately 29%. Like the SMS dataset, the emails are labelled as being either ‘spam’ or ‘ham’.

Using a 1,000 email sample from this dataset and evaluating the baseline and SMS trained model against it, I got the following results:

$\text{Accuracy} = 39.80\% \text{ (baseline } 71.20\%)$

$\text{Precision} = 0.1767 \text{ (baseline } 0.5205)$

$\text{Recall} = 0.6538 \text{ (baseline } 0.2923)$

$\text{F}_{\beta} = 0.2069 \text{ (baseline } 0.4502)$

Surprisingly, here the baseline significantly outperforms the trained model in all metrics except for recall. The model’s precision takes an especially bad hit, dropping from 0.9072 with the SMS validation set all the way down to 0.1767. This implies that the parameters the model developed to detect spam in SMS messages were not abstract enough to translate well over to email messages.

One potential reason for this may be the length of the messages. After skimming over the SMS message dataset, I noticed that spam messages were, on average, longer than non-spam messages. Since emails are typically much longer than SMS messages, the model may have been inclined to label most of the emails as being spam due to the length of the input it received. This led to the model having very poor accuracy but reasonably high recall.

By contrast, the “naïve” baseline, which only looks for keywords and ignores other factors such as the message length, would be less inclined to misclassify longer emails as spam compared to the LSTM model.