Make It Clean

Tobias Walker, Student-ID: 990443

Master of Data Science and Economics

Knowledge Extraction and Information Retrieval, November 2023

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1. Introduction

This paper aims to develop a spelling corrector model designed to work in conjunction with an optical character recognizer (OCR) system. The mainline assumption for this OCR is that it produces only one type of error: incorrect character recognitions. It does not add or delete spaces, and therefore the character length of the text recognized by the OCR matches the character length of the correct sentence, which significantly simplifies the models' complexity and the required training, by avoiding the necessity for an encoder-decodes style of model.

The error correction process is thereby split into two stages. In the first stage, three model architectures are trained and tested to evaluate their effectiveness in error detection. Instead of directly predicting the correct character for each position in the sequence, these 3 models use binary classification to predict if each character in the sentence is correct or erroneous. By comparing 3 different models, the effects of an increased model complexity can be studied. In the second stage, a model is refined with the help of hyperparameter-tuning and cross-validation to predict the correct character for each position. This model uses not only the erroneous sentences but also the outputs of one of the error detection model's applied to the erroneous sentences. The inclusion of the error mask aims to provide the correction model with information about the position of potential errors, allowing the model to concentrate on predicting the correct character at specific positions, instead of the whole sentence. This procedure of splitting up the tasks into 2 models, allows each model to be optimized for its task. However, it also introduces a dependency: the correction model's effectiveness in correcting errors is dependent on the detection model's accuracy. Inadequate performance by the detection model could introduce extraneous noise into the correction model, potentially harming its effectiveness.

2. Data

For the dataset of correct sentences, a database of text scrapped from Wikipedia is used.[1] The erroneous sentences are artificially generated by randomly substituting characters in the correct sentences with random characters. Due to the lack of information about the error patters of the OCR system (which the system most likely has, as it is more likely for it to confuse a capital 'i' with a lower case 'l', than a capital 'i' with a lower case 'm') a complete random approach was chosen. With more information the artificial error production should be adapted to reflect these patterns, as the correction model can then learn these patterns.

The used dataset consists of 7.8 million sentences, of which a random sample of 100,000 sentences is taken to train and test the models in this project.

The preprocessing of the sentences involved several steps to standardize and prepare them for the modeling process. All characters were converted to lowercase, special characters were selectively removed, to only keep relevant ones, and apostrophes were standardized to a uniform representation. Subsequently, sentences exceeding 200 characters were removed from the dataset. Based on the remaining dataset a tokenizer was created. The tokenizer assigns a unique numeric ID to each character, enabling the conversion of text into a numerical format that can be processed by machine learning algorithms. The sentences were tokenized with the tokenizer and the padded to a consistent length of 200 characters, ensuring uniformity across the dataset. The models are therefore only able to process

[1] Data source (Kaggle): https://www.kaggle.com/datasets/mikeortman/wikipedia-sentences

sentences of a length of up to 200 characters. After pre-processing only 91,234 sentences remained in the dataset of correct sentences, of which 72,987 (80%) were declared as the training set and 18,247 (20%) as the test set.

The errors were introduced into the tokenized and padded sentences by randomly swapping characters with a 10% probability with the token representation of a random ASCII character. Importantly, these errors were not introduced into the padded sections of the sentences.

An illustrative example of the preprocessing approach is shown in Table 1.

Table 1: Hypothetical example for the preprocessing

Correct sentence	T	h	i	s		i	s		a		t	e	s	t	.			
+ Tokenized	20	8	9	19	27	9	19	27	1	27	20	5	19	20	28			
+ Padded	20	8	9	19	27	9	19	27	1	27	20	5	19	20	28	0	0	0
+ Artificial errors	20	1	9	19	27	9	8	10	1	27	20	5	19	20	28	0	0	0

A big advantage of the artificial creation of erroneous sentences is, that in each model a form of data augmentation layer can be introduced. Instead of using always the same erroneous sentences to train the models, the artificial errors are inserted in the correct sentences before each epoch newly. The context of the training sentences may stay the same, but by having changing error positions and changing errors, the models should be able to generalize much better, increasing their performance on the test set.

3. Error Detection Model

In the first stage of this paper, binary classification is used to evaluate the error probability for each character in the sentence. The model is feed the erroneous sentences and trains on the error masks, which were produced by comparing the erroneous sentences with their correct counterparts. As the errors are introduced newly before each epoch, also the error masks have to be updated before each epoch. In the error masks, a '0' denotes a correct character, while a '1' indicates an erroneous one.

To assess the impact of varying architectural complexities, three different models are trained under similar conditions: a batch size of 64, binary cross-entropy as the loss function working in conjunction with ADAM optimizer, a maximum of 20 epochs, and an early stopping condition that stops the training if the validation accuracy does not improve for three consecutive epochs.

The first detection model architecture functions as the baseline model and has the following set-up:

1. Input layer: processes the input sequence and outputs it in a dimension of 200x1.
2. Embedding layer: each character in the input sequence of erroneous sentences is embedded into a 64-dimensional space, resulting in an output dimension of 200x64. This embedding facilitates the capturing of character-level features.
3. Bidirectional long-short-term-memory (LSTM) layer: processes the embedded sequences an generates a sort of context for each position, in order to capture dependencies between positions. Compared to a normal LSTM layer, a bidirectional not only captures dependencies from the past

(previous positions) but also from the future (coming positions). The output of this layer has the dimension 200x256.

4. Dense layer with sigmoid activation: the outputs from the LSTM layer are passed through a dense layer with a sigmoid activation function. The sigmoid function, which maps values into the [0,1] range, is used to predict the probability of each character being erroneous. The dimension is thus reduced to 200x1, with each value representing the probability of a character's error status.

The second model incorporates a more complex architecture, but also uses dropout layers to decrease the risk of overfitting:

1. Input layer: processes the input sequence and outputs it in a dimension of 200x1.
2. Embedding layer: expanded to a 128-dimensional space, which provides a richer representation of each character in the sequence.
3. 1st bidirectional LSTM layer: with an increased output dimension of 200x512.
4. 1st dropout layer: this layer sets 10% of the output from the previous layer to zero, which helps to prevent overfitting and ensuring the model does not rely excessively on training specific features.
5. 2nd bidirectional LSTM layer: another bidirectional LSTM layer is added, maintaining the output dimension at 200x512. The idea of this additional LSTM layer is, to allow the model to learn more complex patterns and dependencies in the data.
6. 2nd dropout layer: applies a 10% dropout rate to the output of the 2nd bidirectional LSTM layer.
7. Dense layer with Rectified Linear Unit (ReLU) activation function: this layer consists of 128 neurons and uses the ReLU activation function. The ReLU activation enables the model to cope with non-linearity, allowing it to learn more complex relationships in the data. The idea behind this layer is to refine the features extracted by previous layers before making final predictions.
8. 3rd dropout layer: applies a 10% dropout rate to the output of the previous dense layer.
9. Dense layer with sigmoid activation: outputs the prediction for each position

The third model introduces an attention layer to the architecture:

1. Input layer: processes the input sequence and outputs it in a dimension of 200x1.
2. Embedding Layer: outputs the sequence as a 200x128 dimension.
3. 1st bidirectional LSTM Layer: outputs a dimension of 200x512.
4. 1st dropout rate: applies a 10% dropout rate.
5. Attention layer: allows the model to focus on specific parts of the sequence that are more relevant to the task at hand. Unlike LSTM layers that process sequences in a fixed order, the attention mechanism can weigh different parts of the input differently, providing a form of importance to certain characters that might be crucial for error detection.
6. Layer normalization: stabilizes the learning process by normalizing the output from the attention layer.
7. 2nd bidirectional LSTM layer: outputs a dimension of 200x512.
8. 2nd dropout rate: applies a 10% dropout rate.
9. Dense layer with ReLU activation: applies a dense layer with ReLU activation with 128 neurons across each position in the sequence.
10. 3rd dropout rate: applies a 10% dropout rate.
11. Dense layer with sigmoid activation: outputs the prediction for each position

To evaluate the performance of the three models, four metrics are used. Accuracy, the false positive rate (FPR), the false negative rate (FNR, also known as recall) and the loss on which the model were trained. Accuracy describes in binary classification tasks the ratio of correct predictions to the total predictions. In this case it gives the ratio of correctly labelled characters. As the two classes are not balanced, meaning no comparable number of observations in the two classes, it is also necessary to look at the accuracy measure for each class individually. The FPR describes the ratio between correct characters falsely labelled as erroneous to the total number of correct characters and the FNR gives the number of erroneous characters falsely labelled as correct compared to the total number of erroneous characters. At last, a look at the binary cross-entropy is important to identify how confident the models are in their predictions.

As can be seen in Table 2, the baseline model incorrectly identifies 0.7% of all correct characters and fails to detect nearly one-third of all errors. The more complex model notably reduces the FNR to 21.6% and achieves a lower FPR of 0.6%, leading to an overall accuracy improvement of approximately 0.6 percentage points (PP). The attention model makes quite big advancements in the FPR, reducing it by 0.15 PP, but it has a slightly higher FNR compared to the more complex model. Despite this, the attention model's overall accuracy, at 98.43%, is around 0.1PP higher than that of the more complex model. A significant distinction between the more complex model and the attention model is observed when looking at the binary cross-entropy losses of the two models. The attention model exhibits a notably lower binary cross-entropy loss. This reduction in loss might be attributed to an increased confidence in the correct predictions or a lower confidence in the erroneous predictions the attention model does.

Table 2: Key metric comparison between the 3 detection models

	Baseline model (1)	**More complex model (2)**	**Attention model (3)**
Binary crossentropy loss	0.0636	0.0546	0.0472
Accuracy	0.9776	0.9834	0.9843
False positive rate	0.68%	0.60%	0.45%
False negative rate	31.35%	21.58%	22.68%

When looking at the training evolution of the baseline model (Figure 1), it can be seen that the model might have some improvement potential remaining. All three metrics (loss, accuracy, and recall) do not show signs that the model was close to overfit after epoch 20, as all metrics still develop in their respective right direction. Therefore, a few more epochs might have improved the performance of this model.

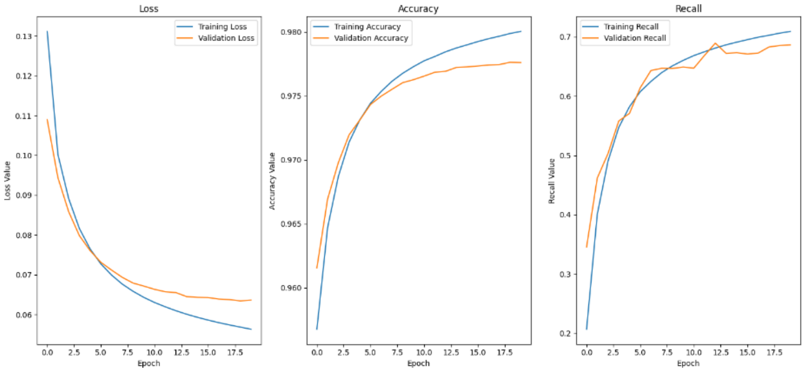

Figure 1: Training metrics of the baseline model

The more complex model shows signs of overfitting as its validation/test loss started to increase after epoch 7 (Figure 2). The training was stopped after epoch 12 by the early stop condition, therefore, the model might be better if it would be retrained only for 7-9 epochs.

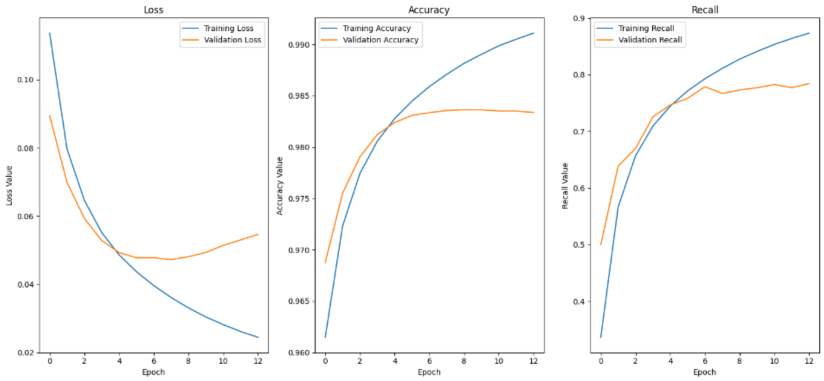

Figure 2: Training metrics of the more complex model

The attention model's training metrics can be seen in Figure 2 and show that the overfitting that can be observed in the more complex model, did not occur in this model. The loss starts to slightly increase after epoch 15, but accuracy and recall still increase after it. A training process exceeding the 20 epochs, might have shown increased signs of overfitting.

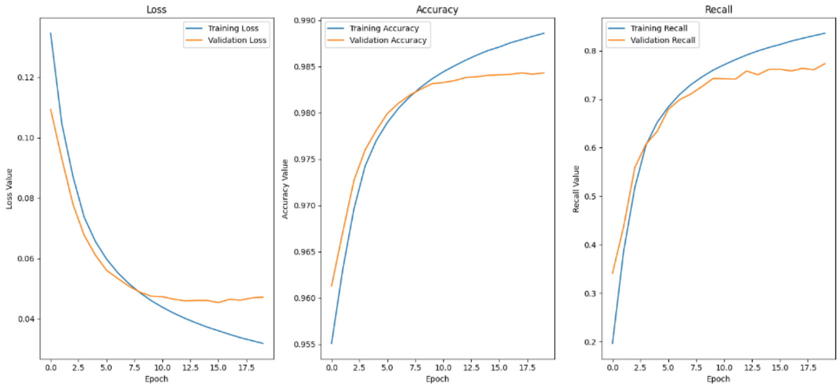

Figure 3: Training metrics of the attention model

4. Error Correction Model

In the second stage of this paper a model is created that predicts the correct character for each position in the sentence. In its core the problem of error correction is in this case a simple multiclass classification problem. The model predicts for each character the probability of being in a certain position inside the text sequence, and the character with the highest probability is chosen as the predicted character. The model is thereby not only feed the erroneous sentence, but also the error likelihoods that are generated by the attention model on the erroneous sentence. The idea of this additional input is to give the correction model a sort of attention information, so it can concentrate on relevant positions instead of on all positions equally. This model also uses a sort of data augmentation layer by introducing new errors into the training data before each epoch.

To achieve the most optimal performance of the set architecture, the hyper-parameters of this model are tuned by a random grid search in combination with 3-fold cross validation. The training set is thereby split into 3 splits and every randomly chosen hyperparameter combination is trained and validated 3 times. Each splits functions as the validation set once, while the other 2 splits function as the training set. The hyperparameters that achieve the lowest mean validation loss over the 3 trainings are considered the best ones. The model is then retrained with these best hyperparameters on the whole training set, to be then tested on the test set for an overall performance evaluation. Only 2 hyperparameter combinations are compared in the random grid search due to resource limitations, nonetheless the setup could be scaled easily to truly find the most optimal combination. The chosen hyperparameters that are tuned are the number of LSTM units (equal for both LSTM layers), the number of neurons in the dense layer with ReLu activation, and the dropout rates (equal for all dropout rates).

The general architecture of the model is the following:

1. Input layer: processes the input sequence and outputs it in a dimension of 200x2.
2. 1st bidirectional LSTM layer: outputs a dimension of 200x(U*2).
3. 1st dropout rate: applies a R% dropout rate.
4. 2nd bidirectional LSTM layer: outputs a dimension of 200x(U*2).
5. 2nd dropout rate: applies a R% dropout rate.
6. Dense layer with ReLU activation: applies a dense layer with ReLU activation with N neurons across each position in the sequence.
7. 3rd dropout rate: applies a R% dropout rate.
8. Dense layer with softmax activation: outputs the predicted likelihood for each character for each position, and therefore the dimension 200x50.

The model is compiled for hyperparameter tuning with the ADAM optimizer and trained with the sparse categorical cross-entropy loss. For each split the maximum epoch is set to 10 with an early stop condition on the validation accuracy with patience of 2.

The first tested model has 64 dense units, 64 LSTM units per direction (resulting in an output dimension of 200x128) and a dropout rate of 0.1. With these hyperparameters the model achieves after 10 epochs a mean validation accuracy of 96.55%. The second model is trained with 64 dense units again, but 256 LSTM units per direction and a dropout rate of 0.15 resulting in an average validation accuracy of 96.98% over the 3 folds. For the training of the final model therefore the hyperparameters of the second model are chosen.

The final model is trained like the tuning models, except the maximum epoch is 20 and the patience of the early stop condition is set to 3. The model achieves after epoch 20 a loss of 0.11 in the test set and an accuracy of 97.38%. As can be seen in Figure 4, which shows the plots of the development of the training metrics, both loss and accuracy develop very comparably. Still after epoch 20 the test accuracy is increasing and the test loss is decreasing, hinting on a slightly to early stop of training, therefore the model might not fully capture the potential it has.

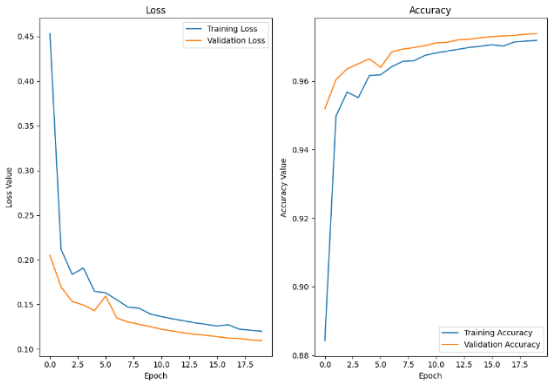

Figure 4: Training development of the correction model

In table 3 to 5 the performance of the attention detection model and the correction model is tested on three example sentences. The first sentence (Table 3) is a rather normal sentence without any bigger traps. The detection model correctly identifies 4 out of the 6 spelling mistakes. The correction model is able to predict for 2 detected mistakes the right character. The other two detected mistakes are replaced with either the original erroneous character or with a wrong character that creates an existing word, but which does not make sense in this context. Interestingly the correction model identifies an erroneous character that is not captured by the detection model (predicts for this position 0.37, therefore it is not very confident in its prediction), but it is not able to predict the right character (the character 'p' is predicted with likelihood 0.3, directly followed by what would be the correct letter 't' with likelihood 0.25). No correct letter is detected or corrected erroneously.

Table 3: First example sentence

Erroneous sentence	tjis is a test senzence with wjich l wanp to test the midel.
Detection model	tjis is a test senzence with wjich l wanp to test the midel.
Correction model	this is a test sentence with wjich a wand to test the midel.

The second sentence (table 4) challenges the model on identifying errors in words it might not have seen in training. The performance of the detection model is rather weak, only detecting one mistake, which is also the only one not in an uncommon word. This mistake is then correctly corrected by the correction model.

Table 4: Second example sentence

Erroneous sentence	the hyderparaneters of this model arc not tuted very well.
Detection model	the hyderparaneters of this model arc not tuted very well.
Correction model	the hyderparaneters of this model are not tuted very well.

The third sentence (table 5) checks the model's performance on how good it is in recognizing errors that create existing words like 'lives' and 'lifes' and on how good it is in coping with differences in British and American English, using the example 'neighborhood' (AE) and 'neighbourhood' (BE). The detection model identifies correctly the error in 'neighborhood', but the correction model is not able to predict the correct character. The other two errors are missed by both models.

Table 5: Third example sentence

Erroneous sentence	he lifes in this verk nice neighburhood.
Detection model	he lifes in this verk nice neighburhood.
Correction model	he lifes in this verk nice neighburhood.

In summary, looking at the example sentences, it can be stated that although the correction model achieves an accuracy of over 97% a lot of errors are missed or incorrectly corrected. At least the model is performing well in identifying correct characters as correct ones. False positive mistakes are only rarely occurring, nonetheless there is still quite an improvement necessary in correcting erroneous characters, for the model to be really usable. This can be achieved by more training data and more complex model architectures for both the detection and the correction model.

Sources:

Rian Dolphin (21.10.2020): https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 (last accessed at 11:10 10.12.2023)

Jason Brownlee (27.9.2022): https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/ (last accessed at 13:56 10.12.2023)

Anish Nama (18.05.2023) https://medium.com/@anishnama20/understanding-bidirectional-lstm-for-sequential-data-processing-b83d6283befc (last accessed at 14:06 10.12.2023)