Deep Learning (ST456)
April 2022

Where did it change? Exploring the limits of Siamese Neural Networks for Style Detection

Group 4

Team Members' Candidate ID: 37424, 31241, 39889
Statement of contribution: All candidates contributed equally, taking part in the coding and modelling as well as the report writing process

Department of Statistics
London School of Economics and Political Science

ABSTRACT

Plagiarism detection is a persistent challenge that provides opportunity for the use of deep learning methods. A growing stand of literature approaches the problem as a matter of detecting an author's intrinsic style fingerprint. In this project, we seek to improve on a Siamese neural network architecture using bidirectional LSTM layers, GloVe embeddings, and a negative cosine similarity function proposed by Nath (2021) for the 2021 PAN style change detection task. After experimentation with various architectures, including reformulation of the architecture for triplet loss, we improve on the baseline model. In this paper we explore the justifications behind our modelling decisions and discuss our best performing model - a much faster model that uses, amongst other changes, more hidden units in the LSTM layer, an increased batch size, and a lower learning rate than the baseline model.

CONTENT

1. INTRODUCTION

Plagiarism detection is a persistent challenge that requires the use of computationally efficient methods. Current popular systems, such as Turnitin, compare submitted documents against an archive of internet documents and media. Approaching the problem as a style change detection task allows for plagiarism detection when no comparison text is available. The goal of style change detection is to capture a "style fingerprint" that renders an author personally identifiable. By focusing on an author's intrinsic writing style, it becomes possible to identify distinct changes of style in a text.

PAN, a series of challenges and conferences focused on digital forensics, has called for submissions to tackle the style change detection problem since 2017. The PAN 2022 challenge sets out three tasks. The first task is to locate the position of a style change in a text written by two authors. The second task is to assign texts to their respective authors, and the third task is to locate the position of a style change at the sentence-level. In this project, we tackle task 1 but use the task 2 dataset, given that it is several times larger. We therefore seek to find the location of all style changes in a document written by multiple authors.

Given that authors are not universally consistent through the dataset— i.e. Author 1 in one file is not necessarily equal to Author 1 in another file— we do not structure the problem as a multi-class classification problem. Instead, following the lead of Nath (2021), we structure our solution as a 'one-shot' binary classification problem using a Siamese Neural Network (SNN). Our replication of the model in Nath (2021) represents our baseline model and we attempt to construct a Siamese architecture that improves on the baseline performance.

After testing various architectures, we were able to improve on this baseline. Similar to the baseline model, our best performing model uses bidirectional LSTM layers, GloVe embeddings and a negative cosine similarity function. The model's improvements on the baseline are due to larger batch size, decreased vocabulary size and maximum paragraph length, lower learning rate, and increased hidden units in the LSTM layer. An additional model, a Siamese recurrent network with triplet loss, yields an improvement in test F1, but other performance metrics are lower.

The rest of this paper is as follows: section 2 discusses approaches from previous PAN challenges, section 3 discusses the theory of our architecture, section 4 presents the data and pre-processing, section 5 details our modelling experiments and results. We conclude with discussion of our findings and avenues of further research in section 6.

2. LITERATURE REVIEW

Style change detection relies on intrinsic stylometric analysis of text, meaning that external documents are not required. To create these stylistic profiles, previous approaches in the literature focus on lexical (character n-grams, word frequencies, average word/sentence lengths), structural (indentation usage), and syntactic features (part-of-speech tagging for frequency/structure analysis). (Bevendorff et al. 2020).

A recent and growing strand of literature has found the use of word embeddings in deep learning methods to be effective for authorship detection (e.g. Nath 2021). Neural network architectures have also been introduced as solutions to the PAN style change detection task. As a submission to PAN 2018, when the task was to discover if a document was authored by multiple individuals, Schaetti (2018) uses a character-based convolutional neural network (CNN) to learn features extracted from grammar structures, while Hosseinia et al. (2018) uses two parallel recurrent neural networks (RNN). In PAN 2019, Zuo et al. uses a feedforward classifier to predict the number of authors in a document. However, the task in both PAN 2018 and 2019 was to determine if a document was authored by one or more authors and the number of authors. Our project tackles the task included in PAN 2020, 2021 and 2022: find the location of authorship changes.

Neural networks have also featured in solutions to the location task. Iyer and Vosoughi use Google's BERT language model to extract sentence-level features before using a random-forest classifier in PAN 2020. In PAN 2021, the highest scoring approach, with a F1 score of 0.751, was Zhang et al.'s feed forward neural network with pre-trained BERT embeddings. Strøm also used BERT for sentence-level embeddings for binary classification via a stacking ensemble, and received a F1 score of 0.707. Deibel and Lofflad used a two-layered bidirectional LSTM (F1 = 0.669) and Nath utilizes Siamese neural networks to compute paragraph similarities (F1 = 0.647). (Zangerle et al. 2021).

3. THEORY

Our solution to the style change detection task uses a Siamese neural network architecture. Although the Siamese neural network was not the highest performing model from PAN 2021, the architecture is an intuitive choice to measure textual similarity due to its twin subnetworks. The implementation of SNNs from previous years also exhibited considerable room for improvement, and we looked to maximize use of the Siamese architecture in our project, including with the use of triplet loss.

Within this Siamese architecture, we experimented with various activation functions, optimizers, loss functions, and RNN layers to improve performance and reduce overfitting. This section discusses the theory and justification behind the models we experimented with.

3.1 NEURAL NETWORKS

Loosely modelled on the brain, a neural network consists of layers of deeply interconnected nodes. Each node is analogous to an individual neuron, and neural networks are organized as layers of these nodes. Nodes are linked to other nodes with connections whose strength is determined by weights. In feedforward neural networks, data moves through them in one direction. (Dasaradh 2020; Hardesty 2017; Goldberg 2016; Woodford 2021).

The simplest neural network, a perceptron, has a single layer and can be expressed mathematically in vector notation as in equation (1), where w are the weights, x are the inputs, h is the threshold function, and Φ is the activation function. Activation functions introduce non-linearity into a network, without which it would be a linear function. (Rosenblatt 1958; Hardesty 2017).

$$y(x) = h\,(\boldsymbol{\phi}(\mathbf{x})^{T}\mathbf{w}) \tag{1}$$

Weights of a neural network are learned, and the learning algorithm consists of backpropagation and optimization. Backpropagation is the algorithm that computes the gradient of the loss function with respect to weights. A loss function repeatedly estimates the loss of the model so weights can be updated to reduce loss in the next iteration. Optimization selects the best weights and bias for the network to minimize the loss, and the learning rate determines how much these parameters are changed. (Agrawal 2017; Brownlee 2020; Chauhan 2020).

After testing various optimizers, this paper presents models with the best performing optimizer, Adam. Proposed by Kingma and Ba (2014), Adam is an extension of stochastic gradient descent. Classical stochastic gradient descent involves calculating the gradient with a single example, and mini-batch stochastic gradient descent involves a mini-batch of samples. Adam, the equation for which is provided in (2), adapts parameter learning rates based on the average first and second moments of the gradients. Adam is more efficient than other optimizers, and its bias-correction also allows for better performance. (Kingma and Ba 2014; Brownlee 2017; Yang 2021).

$$\begin{gathered} s_{n} = (1-\rho_1^{n})^{-1} + (1-\rho_{1}{}_{n-1}^{s} + (1-\rho_1\ )g_n) \\ r_n = (1-\rho_2^{n})\text{-}1(1-\rho_{2}{}_{n-1}^{r} + (1-\rho\ \ )g_n \odot g_n) \\ \theta_n = \theta_{n-1} - \frac{\eta_{n-1}}{\sigma+\sqrt{r_n}} \odot s_n \end{gathered} \tag{2}$$

3.2 SIAMESE NEURAL NETWORKS (SNN)

Proposed by Bromley et al. (1993), a Siamese neural network trains a similarity metric from data. The underlying idea is to learn a function that maps inputs into the target space, where the L1 norm in the target space approximates the semantic distance of the inputs (Chopra, Hadsell, and LeCun 2005). More formally, the network is composed of two identical subnetworks that are joined by their outputs (Bromley et al. 1993). During training, the subnetworks are constrained to have identical weights and parameter updating is mirrored across both subnetworks (Chicco 2021).

The subnetworks extract features from input during training, and the outputs from these twin sub-networks are then fed into a distance function, which evaluates the similarity of the two feature vectors (Bromley et al. 1993). The weights of the twin subnetworks must be identical to calculate a distance that is not contingent on the order of inputs. Unlike traditional neural networks that predict multiple classes, the SNN learns a similarity function where the goal is to minimize distance between similar inputs and maximize distance between dissimilar inputs (Koch et al. 2015).

The SNN architecture is conducive to the style change detection task. Mueller and Thyagarajan (2016) and Ichida, Meneguzzi, and Ruiz (2018) use recurrent Siamese networks with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) structures to learn semantic similarity between sentences. Convolutional neural networks have also been used within the Siamese architecture to identify the author of a text (Saedi and Dras 2021). The length of texts used in these studies, however, are longer than texts provided in the PAN challenge. Given that smaller texts contain less information from which to draw a "style fingerprint," our challenge is not only to develop an architecture for similarity learning, but an architecture that can also learn from limited context. Nath (2021) used a SNN with bidirectional LSTM to learn similarity with limited text in PAN 2021, and we seek to improve upon these results in this paper.

3.3 PROBLEM FORMULATION

To use the Siamese architecture, we formulate the style change detection task as a one shot learning problem. In one shot learning, the model must learn information about each class from a single training example (Koch et al. 2015). The field of computer vision offers a classic example of one shot learning, where two images are identified as similar or not. Within the context of the style change detection task, one shot learning can determine if two texts share a similar style (Nath 2021).

To transform the style change detection problem into one shot learning, we follow Nath (2021) and pair two consecutive paragraphs as a single data point. We then label the data based on whether the two paragraphs were written by the same author or multiple authors. This data is fed into the SNN, where the two inputs are labelled with the same output label indicating whether they are the same class.

3.4 RECURRENT NEURAL NETWORKS

Recurrent neural networks are well-suited for natural language processing tasks. Unlike traditional neural networks, RNN's are able to work with input and output sequences of varied length, a particularly useful characteristic for text processing. RNNs can handle these variable length sequences due to recursive processing units with hidden states derived from previous timesteps. In other words, the RNN's internal hidden state is updated every time the RNN reads a new input; this internal hidden state is then fed back into the model. Every timestep of the computation shares the same weight matrix. (Stanford 2017; Yanhui 2021; Mikolov et al. 2010).

The challenge with RNNs is a vanishing or exploding gradient. Backpropagation occurs through time in RNNs, as output is produced at every timestep of the sequence. To calculate the gradient of the loss with respect to the initial hidden state, the network backpropagates through every timestep, and the final gradient is many factors of the weight matrix. When the largest singular value of the weight matrix is greater than one, the gradient explodes during backward pass, and when the largest singular value of the weight matrix is less than one, the gradient vanishes exponentially. (Stanford 2017).

LSTMs and GRUs are popular approaches to the vanishing and exploding gradient problem. Both of these approaches design a gated architecture for improved gradient flow properties. LSTMs extend the memory of an RNN to recall long-term dependencies. An LSTM unit contains three gates that control the flow of information used for prediction of the output: Input, Output and Forget gates. Intuitively, the input gate controls how much to input into the cell, the forget gate controls how much cell memory to forget from the previous time step, and the output gate controls how much of the output to reveal. These three gates use a sigmoid nonlinearity, meaning that their values are between 0 and 1, and can represent the percentage of information to input, output, or forget. This architecture means that backpropagation through the cell state involves element-wise multiplication by the forget gate, which can vary by timestep, and avoids vanishing/exploding gradients. The three gates also allow the LSTM to remember both long and short term information. (Colah 2015; Donges 2021; Nigam 2021; Yanhui 2021; Zaremba, Sutskever, Vinyals 2014).

GRUs operate in a similar fashion to the LSTM's and also aim to extend the long-term memory of RNN models. Instead of three gates, the GRU has a reset gate and an update gate. The update gate combines the forget and input gates from the LSTM model and decides how much information should be updated. The reset gate allows for information from the previous state to be forgotten. GRUs also only contain the hidden state, while LSTMs make use of a cell state and hidden state. GRUs are an attractive choice in RNNs as their simpler design allows for faster training times. However, the complexity of the LSTM may offer better performance for sufficiently large datasets. Without a clearly superior option, we test the Siamese architecture with both LSTM and GRU. (Saxena 2021; Yang, Yu, Zhou 2020; Nigam 2021).

Beyond considering recurrent structures, we also assess unidirectional and bidirectional versions of these units. Unidirectional structures operate in positive time direction, meaning that input is fed in forward direction. With this structure, the network can learn pre-word context, but is unable to consider post-word context. Bidirectional LSTMs approach this problem by stacking two LSTM layers. The first LSTM layer operates as standard by taking input in a forward direction, while the second LSTM layer operates in negative time direction, such that the input sequence flows backwards in the layer. The outputs of the two LSTM layers are then combined. By feeding input in both directions, the network receives more information, and can learn both pre-word and post-word contexts. (Huang 2015; Verma 2021; Mungalpara 2021; Zvornicanin 2022).

3.5 LOSS FUNCTIONS IN OUR SIAMESE NEURAL NETWORK

Different loss functions can be used for the SNN architecture. Our project uses binary cross entropy and triplet loss functions. Since Siamese networks provide binary classification by classifying if inputs are similar or not, binary cross entropy is an intuitive choice for a loss function. The binary cross entropy loss function estimates performance of the classifier with an output probability between 0 to 1 (Saxena 2021; Wang and Liu 2021). Equation (3) presents the loss function, where y is the classification label, and p is the prediction probability. The loss value increases when the predicted probability deviates from the true label (Saxena 2021; Wang and Liu 2021).

$$L = -y \log p + (1-y) \log (1-p) \tag{3}$$

A more appropriate choice of loss function for a Siamese network architecture, however, are pairwise loss functions, such as contrastive or triplet loss (Wang and Liu 2021; Sarigoz 2022; Martino 2020). The objective of a Siamese network is not to classify pairs of inputs, but to differentiate between them — to ensure that the distance between a pair of inputs from the same category is low, and the distance between a pair of inputs from different categories is large

(Shorfuzzaman and Hossain 2021). A classification loss, such as binary cross entropy, is less suited for evaluating how well the network is distinguishing pairs of inputs (Rashad 2020; Martino 2020; Hermans, Beyer, Leibe 2017).

Contrastive loss aims to penalize the network differently according to the classes of the inputs. The loss function requires pairs of input samples. If the two inputs are from the same class, the loss function encourages the network to produce more similar feature embeddings, and conversely, less similar feature embeddings if the two inputs are from different classes. Formally, the contrastive loss function is provided in equation (4), where y is the true label (0 when inputs are similar, and 1 if they are dissimilar), d_w is the distance between feature embeddings of the two inputs, and m is the margin. When y is 0, meaning the two inputs are from the same class, the amount of loss contributed by similar pairs is simplified to the first term in the equation, where the distance, d_w, is minimized. When y is 1, the loss is simplified to the second term in the equation, and d_w is maximized to the margin, m . This term means that when pairs of inputs are dissimilar, loss is only incurred when their distance is greater than the margin. (Wang and Liu 2021; Shorfuzzaman and Hossain 2021).

$$L = (1-y)\frac{1}{2}(d_w)^2 + (y)\frac{1}{2}\{\max(0, m - d_w)\}^2 \tag{4}$$

Triplet loss, another pairwise loss function, operates with a different network structure than discussed thus far. A triplet network takes three inputs: an anchor, positive, and negative. Anchors and positives are from the same class, while anchors and negatives are from different classes. The network is fed these three inputs, but returns as output two values: the L2 distance between the anchor and positive, and the L2 distance between the anchor and negative. The underlying idea is that the distance between anchor and negative is maximized, while the distance between the anchor and positive is minimized. The triplet loss function is provided in equation (5), where loss is incurred if the distance between similar and dissimilar pairs plus a margin term is minimal. In other words, the distance between the anchor and positive inputs should be smaller than the distance between the anchor and negative inputs. Triplet loss encourages the network to create similar feature embeddings for inputs in the same class, and less similar feature embeddings if the two inputs are from different classes. (Benhur 2021; Polela and Suprapto 2021; Sarigoz 2021; Hermans, Beyer, Leibe 2017).

$$L(A, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m, 0) \tag{5}$$

Triplet loss is widely considered theoretically stronger than contrastive loss (Polela and Suprapto 2021; Sarigoz 2021). Contrastive loss encodes anchors and positives as the same point in the vector space. The implication of this encoding is that contrastive loss forces the distance between anchors and positives to close to 0, whereas triplet loss allows intra-class variance. The tolerance of triplet loss can include outliers while maintaining a margin between positive pairs and negative pairs. Another key difference between the two loss functions is the role that margin plays; triplet loss seeks to maintain a margin between distances of positive pairs and distances of negative pairs, while contrastive loss only considers the margin when comparing dissimilar pairs. For this reason, triplet loss may encourage the network to continue organizing the vector space, while contrastive loss reaches a local minimum. (Rashad 2020; Sarigoz 2021).

Balancing time constraints for this project, and given the theoretical advantages of triplet loss, we do not experiment with contrastive loss. This decision is supported by Polela and Suprapto (2020), who test both loss functions in a text matching similarity task. They find that the SNN with triplet loss is better able to measure similarity between two texts than contrastive loss.

3.6 MODEL ARCHITECTURE

Our model was built from three main components: the twin network, similarity function and output layer.

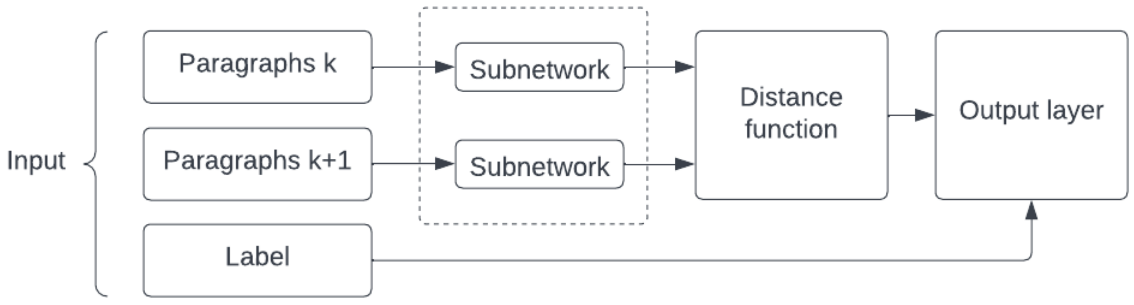

Figure 1: Outline of Siamese Neural Network for Style Detection.

Twin subnetworks: The twin subnetworks serve to extract features from the text. As stated earlier, the two subnetworks are identical, sharing the same weights. Two inputs – paragraphs of text – are fed into the Siamese architecture. Each subnetwork consists of an embedding and bidirectional LSTM layer. The two outputs from the subnetworks represent features learned from the inputs.

Similarity function: The distance between the two outputs is calculated. We tested models with Euclidean distance before deciding to use negative cosine similarity for its theoretical and technical improvement. Euclidean distance measures the magnitude of distance between texts, while cosine similarity measures the cosine of the angle between two texts (Prabhu 2019; Prabhakaran 2021). A smaller angle implies higher similarity between texts. Models presented in our results section use negative cosine similarity.

Output layer: The last layer is comprised of a single node connecting nodes from the previous layer that calculates distance. This output layer calculates a similarity score between 0 and 1 indicating whether texts are written by the same author. A similarity score close to 1 suggests that the two inputs are similar, while a similarity score close to 0 implies they are dissimilar. We use the sigmoid function presented in equation (6) to compute this probability, where h_j^1 and h_j^2 are the j-th neurons of each of the two subnetworks, and j is the weight between the j-th neuron and the output neuron. (Polela and Suprapto 2020).

$$p = \sigma(\Sigma_j \sigma_j \, |h_j^1 - h_j^2|) \tag{6}$$

3.7 PERFORMANCE METRICS

We measure the performance of our models by assessing accuracy and F1 score. Accuracy allows us to easily understand how many observations in our model are being correctly classified, as mathematically formulated below. The accuracy metric is not sufficient, however, as our task requires us balance the recall and precision of a model. Given that we are building a plagiarism detection tool, it is important that we are not accusing of plagiarism in instances when it has not occurred. To balance the model's ability to sufficiently detect plagiarism when it is present with the need for it to not detect it when it is not present, we use an F1 metric.

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + True\ Negative + False\ Positive} \tag{7}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

4. DATA

4.1 DATA STRUCTURE AND KEY STATISTICS

The data is provided as part of the PAN22 Authorship Analysis: Style Change Detection competition (PAN 2022). The dataset contains ‘problem’ .txt files, which are comprised of posts and comments collected from the Stack Overflow forum. These posts and comments are split into multiple paragraphs which can be written by multiple authors. Accompanying each ‘problem’ file is a ‘truth’ file which indicates which author wrote which paragraphs. Crucially, in each truth document, the authors are only labelled in the context of that document as ‘1’, ‘2’, ‘3’ or ‘4’. It is therefore not possible to identify which actual author on Stack Overflow wrote which paragraphs across the whole set of problem.txt documents, preventing us from formulating the problem as a multiclass classification task. This data format contrasts to the PAN21 data where a unique author id, e.g. 1221, was provided. To clarify, this means author 1 in one ‘problem’ document is not the same as author 1 in another ‘problem’ document.

Given this data format, we follow the lead of authors from previous competitions (e.g. Nath, 2021; Zhang, 2021) in constructing a one shot binary classification problem where each set of consecutive paragraphs in each problem.txt file is classified as written by the same author or not. The original text files are transformed into a Pandas dataframe, where each consecutive paragraph pair is a row. This dataset then becomes suitable for our one-shot classification problem. The only instance when this structure differs is when we use a 'triplet' rather than 'twin' model, but the structure is equivalent – with each triplet formed from the same problem file.

Our research project focuses on PAN22 task 1 – 'finding the point of author change.' However, we use the data from task 2, which contains multiple points of style change, rather than just one. We focused on Task 2 data after realising that this would provide us with substantially more data. Task 1 has 1400 problems files of multiple paragraphs with one style change, whilst task 2 has 7000 with multiple authors and style changes. Given that we wanted enough data to train our neural networks with, we used the larger dataset.

Although the PAN22 competition provides a validation dataset and test dataset, the truth files for the test dataset are not provided, as this is meant to be used as a final 'test' for the judges. Therefore, to get good estimates of test error, we follow previous authors in PAN competitions and split the provided training set into a training set (80%) and validation set (20%)[1], and keep the provided validation set as the test set.

From the table of descriptive statistics shown below, two key points stand out. Firstly, we can see that the training, validation, and testing sets are all appropriately comparative. Secondly, we can see that the number of words in each comment is skewed by very large outlier comments.

Statistic	**Training**	**Validation**	**Testing**
Mean no. of authors in problem document	3.00	3.00	3.00
Number of paragraph pairs	36578	9145	9573
% of pairs in which author changes	61.61%	61.01%	62.45%
Median number of words in paragraph	43.79	43.92	43.69
Mean number of words in paragraph	37.0	37.0	37.0
Min. words in paragraph	1	1	1
Max. words in paragraph	819	251	228

Table 1: Summary statistics for training, validation, and test sets.

[1] The data comes pre-shuffled on a document basis, so we do not do a random split at the paragraph level.

4.2 NUMERICAL REPRESENTATIONS OF TEXT

Following the removal of special characters and transforming all characters into lower case, we follow Nath (2021) in tokenizing our text at the word, rather than the character, level. Following tokenization, we transform the single digits to multi-dimensional GloVe word embeddings (Pennington et. al, 2014). GloVe word embeddings were trained on a diverse 6-billion-word corpus of texts. In essence, each word's likelihood of co-occurrence with another is represented in an n-dimensional vector. GloVe embeddings are a more sophisticated alternative to older 'bag-of-words' models in that they can account for context, grammar, and word order. Whilst we specify a maximum length of words from each paragraph to be tokenized, which we change during the modelling process, it should be noted that padding is added to each word vector so that each vector is of the maximum length. The advantage of this is that vectors of variable lengths can be compared.

The following diagram shows the overall word manipulation process that is carried out prior to modelling. A 'DataGenerator' class is used to ensure efficient tokenization, label generation and batch preparation. However, the generator is not used in the triplet loss model due to the one-off need to process paragraphs in 'triplets' as well as 'pairs'.

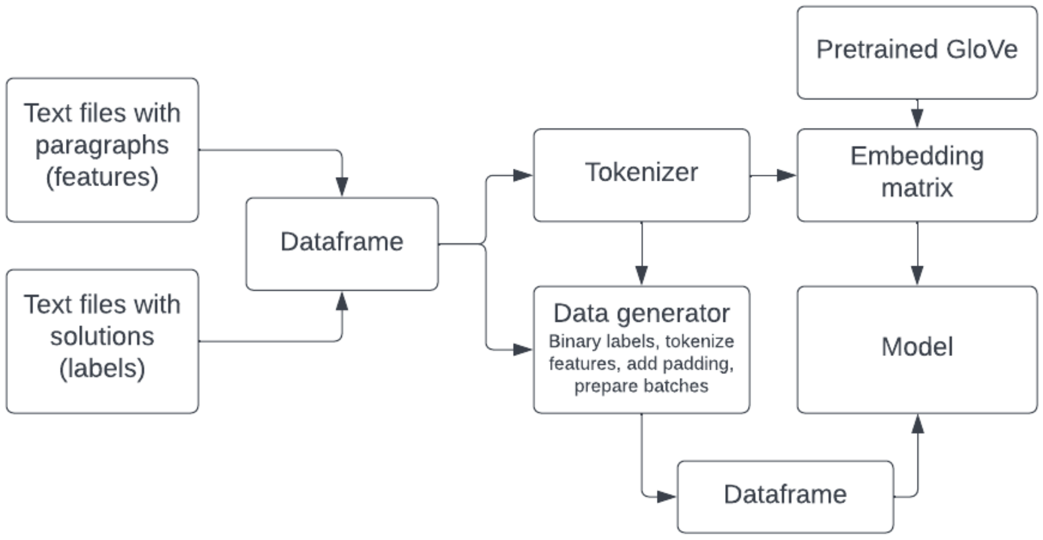
Text files with paragraphs (features)
Text files with solutions (labels)
Dataframe
Tokenizer
Data generator
Binary labels, tokenize features, add padding, prepare batches
Pretrained GloVe
Embedding matrix
Model
Dataframe

Figure 2: Simplistic flow chart of our data import, pre-processing, text, and modelling process

5. MODELLING

5.1 TECHNICAL SETUP & CODE ORGANISATION

We implemented the neural networks in Python 3.9 using the Tensorflow Keras 2.4.1 API. To speed up processing time and enable GPU-support for our models we used a Jupyter Server on Saturn Cloud (saturncloud.io) with 4 cores, 16 GB RAM and 1 GPU. To avoid a conflict between Numpy 1.22.3 and Tensorflow version 2.4.1 on this server, we downgraded Numpy to version 1.18.5. We set Numpy and Tensorflow seeds to ensure that our models are reproducible on several machines with various configurations, but we do note that a minor non-configurable random component is introduced by using a GPU.

We used Github to collaborate and set up a file structure that could help overcome the shortcomings of Nath (2021) — discussed in the next section — that made it very difficult to replicate its results. We run our models in several files depending on the task at hand, setting up a functional framework to aid experimentation and to avoid human errors. This framework allows us to easily change model architectures, optimizers, tokenization/embedding settings as parameters in functions, and autosave model history, plots, and model checkpoints.

When we reformulate the loss function to use triplet loss, we need to substantially alter the inner working of these functions. To make these changes clear, we let this model formulation stand on its own in its own file. As it is not clear that the triplet loss improves the model fundamentally, we do not set up a full data generator batch control framework for this model, and for that reason the run time of the model is slightly longer. Table 2 presents a guide for the files in our GitHub repository.

	File	Content
Technical setup	.gitignore config.py	Setup for running models in several environments, keeping track of paths and dependencies.
Data	/data/ /processed/	Raw data Processed data as .csv
Functions	functions_auto.py	Collection of our custom functions
Preprocessing	preprocessing.ipynb	From text files to data frames
Replication	modelling_replication*.ipynb	Our replication of Nath (2021)
Experiments	modelling_experiments.ipynb	Selected experiments on architecture
Final models	modelling_modelABC.ipynb modelling_modelD.ipynb	Selected models based on Nath (2021) Triplet loss model
Output	/checkpoints/ /history/ /plots/	Checkpoints, model weights, epoch history, loss, and accuracy graphs

Table 2: Code and file organization.

5.2 CREATING A BASELINE

Our first step in creating a baseline was to replicate the model used by Nath (2021) and to regenerate their results on data for PAN22. We did this by using the author's publicly available implementation on Github. It turned out that the code was not well organised, the results were not directly producible, and that the code had a bug in the construction of the embedding matrices.

While the code looked structured at first glance, it was not well-documented. It included several functions with the same names—but different implementations—and had messy Jupyter Notebooks with a mix of local and global variables. Furthermore, it was clear that the code was borrowed from other partially non-referenced sources, including previous year's PAN solutions. It was, however, outside the scope of this assignment to thoroughly track the code sources. We have organized the functions and the code for replicating the Nath results on 2021 data in a separate file in our Github repository.

The running replication model turned out to have several additional problems. The model had severe overfitting issues and Nath (2021) doesn't set a seed in any of the provided files. This means, that we couldn't replicate the results exactly, but after systematic experiments we found a seed that could almost reproduce the results and give a similar curve and accuracy metrics on 2021 data. We used the same seed when using the model on 2022 data. To replicate the results, we enabled shuffling of the data for every training epoch (Nath, 2021 runs 'only' 100 steps per epoch, but doesn't shuffle the data). Table 3 provides an overview of the model architecture of this baseline model.

Due to the lack of documentation, a thorough investigation of the code highlighted that, while Nath (2021) specifies a vocabulary size of 10,000 words, the embedding matrix created is still of size approximately of 45,000 words. This appeared due to an evident misinterpretation of how the inbuilt Tensorflow tokenizer handles the parameter *num_words*. We propose a minor fix to this issue in accordance with the Keras documentation (Github/keras-team, 2017) but otherwise, keep most of Nath's (2021) implementation intact. Fortunately, this error turned out not to seriously affect the model performance on 2021 data.

Network	**Specification**
Tokenizer	Word-level, special characters removed
Vocabulary size	45360 (max)
Sentence length (input size)	300 (padding enabled)
Embedding type	GloVe 50D, pre-trained
Trainable weights	True
Shared RNN	Bidirectional LSTM layer
Hidden units	Hidden units: 50, Dropout: 0.2, Recurrent dropout: 0.2
Similarity function	Negative Cosine Distance
Classifier (output) type	Dense layer, with 1 unit
Activation function	Sigmoid
Loss function	Binary Crossentropy
Optimizer	Adam (Learning rate: 0.001, Clipnorm: 1.5)
Batch size	64
Steps per epoch	100 (shuffling enabled*)

Table 3: Summary of network specification
* Nath (2021) is not using shuffling.

Figure 3 displays the learning performance for our baseline model. The model achieves a baseline validation accuracy of 64.3%, while validation loss reaches 0.63. After 20 epochs, however, the model begins to suffer from overfitting.

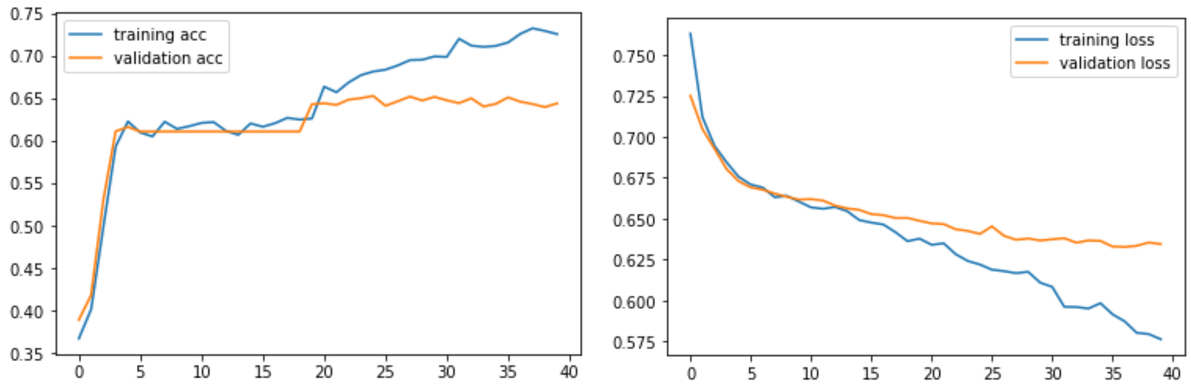

Figure 3: Training curves.

Our project seeks to build an architecture that avoids the overfitting evident in the baseline model. To this end, our experiments aimed to address three key areas of improvement: decrease run time of the model, ensure the model does not get stuck in a local minimum, and reduce the overfitting.

5.3 MODELS AND ARCHITECTURES

In this section we provide an overview of our systematic approach and outline our experiments, before presenting four exemplary models. While these models show some improvement in performance over the baseline, they are all much more computationally efficient than the baseline.

5.3.1 APPROACH AND CONCLUSIONS FROM EXPERIMENTS

To improve upon our baseline, we test a variety of Siamese network architectures, beyond what we had room to present in this paper. Our experiments aimed to address three key areas of improvement based on the baseline model's shortcomings identified above – model runtime, convergence to local minima, and overfitting. To meet these key improvements, we experiment with training methods, optimizers, and model settings. We also make fundamental architectural changes within the Siamese network architecture, such as modifying or replacing recurrent structures and reformulating the loss function, to make the model better learn features by using architectures less prone to overfitting and memorization.

Many of our experiments, not highlighted in the paper, attempted to introduce more regulatory measures to reduce the severe overfitting that plagued the baseline model. In other words, our concerns relate to the *bias-variance trade-off*: striking a balance between architectural complexity that can solve the problem most accurately, without introducing too close a fit to the training data that hinders generalizability. To this end, we tested a higher proportion of dropout, L1 or L2 penalties, and batch normalization. We also attempted to ensure the model was not getting stuck in a local minima by shuffling data. With all of the above measures, we were only able to delay the epoch at which overfitting occurred, but we also received lower accuracy metrics. We also tested different optimization methods (Adam, SGD, RMSProp), learning rates, weight decay, and clipnorm.

Next, we focused on the Siamese architecture. To lower network complexity, we experimented with reduced LSTM hidden units and unidirectional LSTM layers. However, accuracy in these models converged at lower levels than before. Conversely, we tested results from models with increased network complexity. For this purpose, we added more hidden units to the LSTM and introduced an additional LSTM layer. We also added a 1D convolutional layer before the recurrent unit for better extraction of sentence features. However, network complexity tended to result in worse performance metrics. Even more substantial changes to the network architecture, such as the use of Time Convolutional Network (TCN) architectures that are reported to exhibit longer memories than RNNs, didn't improve performance (Remy, 2020). Like Nath (2021), we also found that GRU layers performed worse than LSTM.

The above paragraphs summarized models that we do not include in this paper. Some experimentation can be found in our GitHub repository. Within this paper, we highlight four exemplar models of our experiments. Models A-C finetune the baseline model. While perhaps not the most dramatic changes, they aim to improve runtime, reduce overfitting, and ensure the model does not get stuck in the local minimum. Model D reformulates the SNN architecture to use triplet loss.

5.3.1 MODEL A: BATCH, PARAGRAPH LENGTH, AND EARLY STOPPING

In Model A, we manage to attain similar performance metrics as the baseline model, but also make use of GPU. By not including recurrent dropout in the model we can reap the benefit of GPU-support (which recurrent dropout prohibits). We find that by regularizing long paragraphs, increased batch size and reduced vocabulary size, we can achieve similar accuracy to our baseline model — the justifications for these changes are provided in Table 4. By using GPU, we radically improve training time per epoch, even with a larger batch size. In addition, the use of early stopping halts model learning at the 17th epoch, half of the epochs of the baseline.

Change	**Motivation**
Remove recurrent dropout	Recurrent dropout prevents the model from accessing GPU, so to improve running time we remove it from the model.
Increased batch sized	Given that LSTM layers work by 'remembering' past states, the less frequently the layer weights are updated the more effectively the layers are able to learn. Given that the weights are reset at the beginning of each batch, a larger batch size can help improve the accuracy of the model.
Reduce the maximum length of each paragraph to 100 words, from 300	As many of the paragraphs are considerably shorter than 300 words, longer paragraphs might be considered overly 'dissimilar' due to the relative lack of sparsity. By reducing the maximum length to 100, we might mitigate this problem.
Introduce early stopping into the model	Given that the model is overfitting, it is likely that stopping the model earlier (once validation loss has stopped improving) could lead to a better model than if it was left running.
Decrease vocabulary size	After fixing the bug of Nath's (2021) construction of the embedding matrix, we reduce the vocabulary to 10,000 words, as Nath had originally intended to run.

Table 4: Summary of key changes for Model A and their rationale

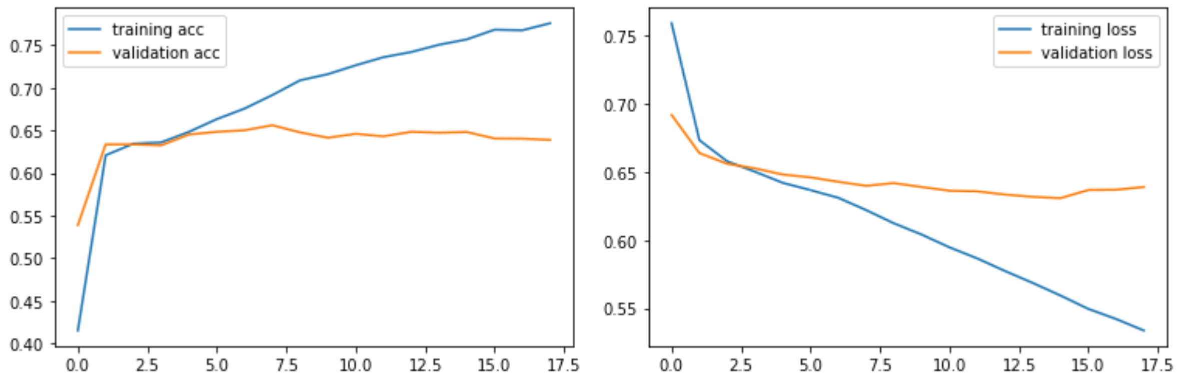

Figure 4: Training curves for Model A

While validation loss and validation accuracy converge quickly for Model A, the model evidently still suffers from overfitting, so we need to make further changes.

5.3.2 MODEL B: VOCABULARY SIZE, LSTM HIDDEN UNITS, LEARNING RATE

Model B seeks to further improve results of Model A. We decrease the total vocabulary size used in the model, double the number of hidden LSTM units, and decrease the learning rate by a factor of 10. These changes, which aim to ensure the model escapes any local minima and can effectively balance the flexibility-generalizability trade-off, are successful. Model evaluation metrics are higher for Model B than Model A.

Change	**Motivation**
Decrease the vocabulary size	It could be that very rarely used, niche words are overly influencing the results. Therefore, by reducing the number of recognised words in the model we might reduce the overfitting problem.
Double the number of hidden LSTM units	More hidden units might help the model to learn more effectively and improve the accuracy.
Decrease the learning rate by a factor of 10	To ensure a greater likelihood of finding a global minimum, we decrease the learning rate.

Table 5: Summary of key changes for Model B and their rationale

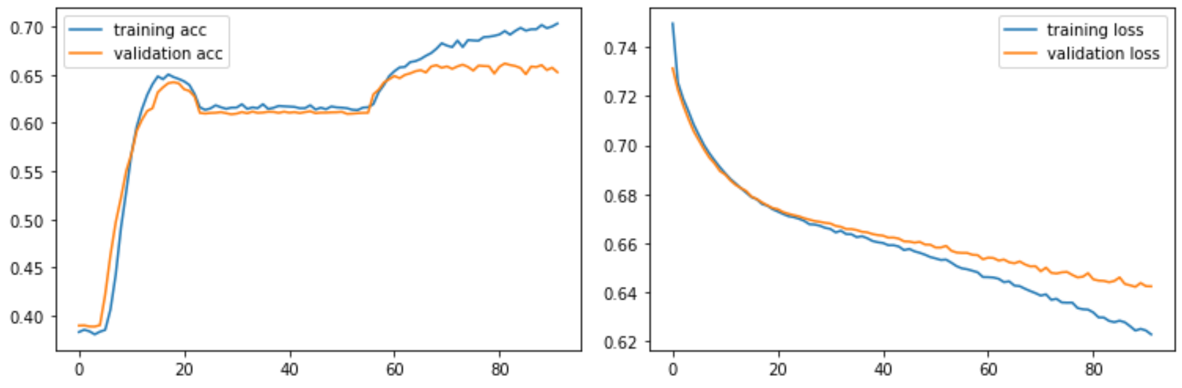

Figure 5: Training curves for model B

Our results from figure 5 show a much smoother loss curve, caused by the lower learning rate of the optimizer, but this comes at the cost of total run time. Ultimately the curve shows that we might be stuck in local minima between ~20-60 epochs but the model manages to escape it. We see evidence of overfitting after the 60th epoch and it is evident that validation accuracy does not improve after this point.

5.3.3 MODEL C: DENSE LAYER, VOCABULARY SIZE

In model C, we further decreased the number of words in the vocabulary, given our earlier success with this, and introduced an additional fully connected layer towards the end of the model. We add the additional dense layer because other experiments, not featured in this paper, yielded better results with the addition. However, the results from this model were not broadly an improvement to model B. The overfitting problem was not solved and accuracy and F1 scores deteriorated.

Change	**Motivation**
Adding a further dense layer	Currently the network feeds the similarity measure calculation directly into a one MLP dense layer. Many commonly used classic networks (eg. LeCun et. al (1998)) include a number of fully connected layers towards the latter stages of their architectures, so this might be beneficial for our model. It could allow the model to better understand the relationship between distances and the dependent variable.
Decrease the number of words in the vocabulary	We thought that the vocabulary still might be too large, and niche words still might be driving the overfitting so we tried to reduce it further.

Table 6: Summary of key changes for Model C and their rationale

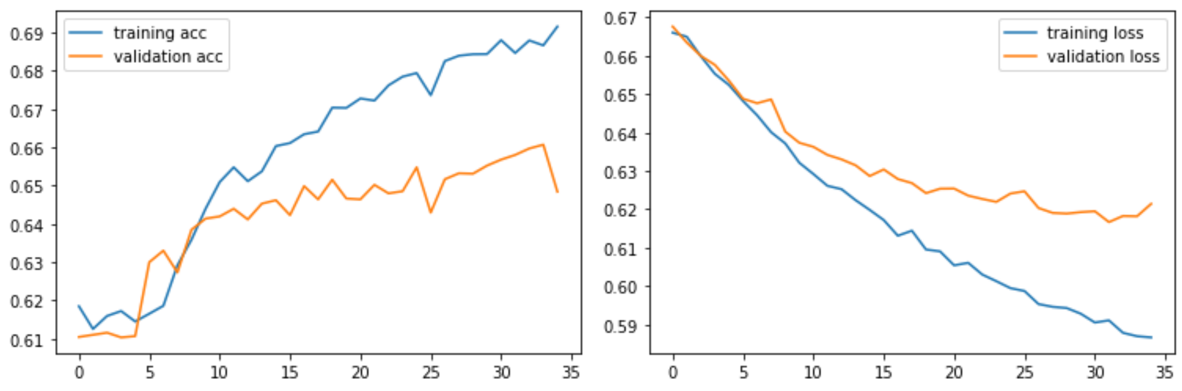

Figure 6: Training curves for Model C

The curves in figure 6 indicate that overfitting is still a significant problem for our model. The results of Model C broadly exemplified some of the circularity we experienced during our modelling process; we found that changes that led to improvements in accuracy metrics in one model did not result in total improved performance when combined with architectural improvements from other models. We therefore found ourselves in a loop of regression and progression when finetuning the baseline model, and so we looked to see if we could make fundamental changes to the architecture.

5.3.4 MODEL D: TRIPLET LOSS

Polela and Suprapto (2020) find that triplet loss is advantageous for Siamese architectures. Given that other modelling changes were not helping us move beyond an upper bound of validation accuracy of around 65%, we decided to change our loss function.

The core idea in a Siamese network with triplet loss is that the weights are instead optimized to minimise the distance between the positive and anchor pair and to maximise the distance between the positive and negative pair. Model D is therefore an adaptation of Model B using triplet loss.

It should be noted that, to evaluate our training data, which is still comprised of pairs of paragraphs, the modelling process with triplet loss needed to be adjusted. The new process consists of:

1) Data is formed into triplet groups for training
2) A similar modelling process is used as previously: a base 'triplet network' layer feeds into a similarity calculation which then feeds into a final output classifier.
3) Once this model has been trained we run the triplet pairs through the trained model to generate word embeddings that have been trained to minimise the distance between the positive and anchor paragraphs and to maximise the distance between the positive and negative paragraph pairs.
4) These word embeddings and their labels are then used to train a logistic classifier.
5) The testing data paragraph pairs are then run through the shared base layer, and these distances are fed into the logistic classifier.

Although it was expected that introducing a more appropriate loss function would improve the model, in fact the results were mixed. As we discuss in more detail in the following conclusion section, although the model training appeared to indicate very low loss values, the model performed poorly on the validation set and the results were mixed on the test set.

Change	Motivation
Introducing triplet loss	Theory suggests that triplet loss might be more effective in training the model. Introducing triplet loss, however, meant changing the data structure into triplets of paragraphs, rather than pairs.

Table 7: Summary of key changes from Model B to Model D and their rationale.

6. DISCUSSION & CONCLUSION

6.1 DISCUSSION

In the table below, we summarize the results from the baseline model and the four additional models that provided the most promising results. We can see from the results that, generally, the most successful model appears to be model B. This model represents an improvement in validation accuracy and validation F1, as well as test accuracy when compared to our baseline, and big improvements in run time.

Model D, which uses triplet loss rather than binary cross entropy loss, performed particularly poorly on the validation data but achieved the best test F1 result of all models (0.76). Notable about Model D is the divergence between its validation performance— which is poorer than our other models— and the training and test performance— which is higher than our other models in terms of F1. The divergence could be a function of the triplet structure, where our validation data may contain more linguistically-similar anchor-positive pairs than the training model is capable of differentiating between. Therefore, further improvement on the model could include training the data on “harder” data; such as texts that are semantically similar but are written by different authors, or texts that are semantically different but are written by the same author. By training the model on a larger volume of challenging samples, we could perhaps improve validation metrics.

	Validation Loss	**Validation Accuracy**	**Validation F1**	**Test Accuracy**	**Test F1**	**Avg. training time per epoch**
Baseline*	0.6344	0.6439	0.7156	0.6464	0.7225	34s **
A	0.6389	0.6388	0.7056	0.6478	0.7109	4s 39ms
B	0.6423	0.6525	0.7403	0.6588	0.7383	5s
C	0.6214	0.6484	0.7090	0.6528	0.7046	5s
D	0.6842 ***	0.5275	0.4953	0.6213	0.7565	3s 78ms ****

Table 8: Model comparisons. Highlighted performance metrics for top performing models.
* Nath (2021) model on 2022 data
** Without GPU-support
*** Different loss function
**** Without efficient batch generation, average training time for both

6.2 CONCLUSION

In this project, our aim was to build a neural network architecture that can detect locations of style change in a document. Our modelling process involved, first, replicating the Siamese network architecture created by Nath (2021). Despite challenges with erroneous code (e.g. erroneous tokenization), we were able to replicate the model on data from this year’s PAN challenge, and obtain similar results with shuffling. This model, our baseline, suffered from overfitting and convergence to a local minima rather than a global minimum. The baseline was also unable to be run on GPU and therefore was very slow to run. Our aim throughout subsequent modelling was twofold: to improve model performance and speed.

To this end, we experimented with different architectures, from finetuning training settings, to overhauling the recurrent architecture. We were able to improve upon the baseline model. Models B and D are the best performing models. Model B finetuned the baseline by decreasing vocabulary size, sequence length, and learning rate, while increasing batch size and the number of LSTM hidden units. Model D restructured the Siamese architecture to use triplet loss. Although we were hopeful that triplet loss would substantially improve the model evaluation metrics, the results were mixed, with only our Model D F1 test accuracy metric representing an improvement from model B. Training our architectures with "harder" samples, as discussed above, may yield clearer conclusions on model architectures.

Although we were able to improve on the baseline model, there is still substantial room for improvement. Generally, test accuracy metrics of approximately 65% and test F1 metrics of 73% are still some way off from a reliable classifier that could be used to classify plagiarized documents in a real-life setting. Further research may focus on deep learning methods optimized for limited context. Literature that we based our SNN architecture on – and that exhibited much better results than our own – were trained on longer texts than the PAN dataset. The median length of a paragraph from the PAN challenge was 37 words, and our architecture might have been more effective with more context than was provided through the provided StackOverflow Q&A data. An important consideration for future research is therefore not only to consider effective architectures, but also methods that can extract meaningful information with limited text.

7. REFERENCES

Agrawal, Apoorva. “Loss Functions and Optimization Algorithms. Demystified.” *Medium*, Data Science Group, IITR, 29 Sept. 2017, https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c.

Benhur, Sean. “A Friendly Introduction to Siamese Networks.” *Medium*, Towards Data Science, 29 Jan. 2021, https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942.

Bevendorff, J., Chulvi, B., Peña Sarracén, G. L. D. L., Kestemont, M., Manjavacas, E., Markov, I., ... & Zangerle, E. (2021, September). Overview of PAN 2021: Authorship Verification, Profiling Hate Speech Spreaders on Twitter, and Style Change Detection. In International Conference of the Cross-Language Evaluation Forum for European Languages (pp. 419-431). Springer, Cham.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., & Shah, R. (1993). Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, *6*.

Brownlee, Jason. “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.” *Machine Learning Mastery*, 12 Jan. 2021, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

Brownlee, Jason. “How to Choose Loss Functions When Training Deep Learning Neural Networks.” *Machine Learning Mastery*, 25 Aug. 2020, https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

Chicco, D. (2021). Siamese neural networks: An overview. *Artificial Neural Networks*, 73-94.

Chopra, S., Hadsell, R., & LeCun, Y. (2005, June). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 1, pp. 539-546). IEEE.

Colah. (2015). *“Understanding LSTM Networks.” Understanding LSTM Networks -- Colah's Blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.*

Dasaradh S., K. (2020) “A Gentle Introduction to Math behind Neural Networks.” *Medium*, Towards Data Science, 30 Oct. 2020, https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.

Donges, Niklas. (2021) *"A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks." Built In, https://builtin.com/data-science/recurrent-neural-networks-and-lstm.*

Github/keras-team (2017). Using Tokenizer with num_words #8092. https://github.com/keras-team/keras/issues/8092. Accessed: 29 April 2022.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, *57*, 345-420.

Larry Hardesty | MIT News Office. "Explained: Neural Networks." *MIT News | Massachusetts Institute of Technology*, https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

Hermans, A., Beyer, L., & Leibe, B. (2017). In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737.*

Marjan Hosseinia and Arjun Mukherjee. A Parallel Hierarchical Attention Network for Style Change Detection—Notebook for PAN at CLEF 2018. *CLEF 2018 Evaluation Labs and Workshop – Working Notes Papers, 10-14 September*, September 2018. CEUR-WS.org.

Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint arXiv:1508.01991.

Ichida, A. Y., Meneguzzi, F., & Ruiz, D. D. (2018, July). Measuring semantic similarity between sentences using a siamese neural network. In *2018 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-7). IEEE.

Aarish Iyer and Soroush Vosoughi. Style Change Detection Using BERT—Notebook for PAN at CLEF 2020. *CLEF 2020 Labs and Workshops, Notebook Papers*, September 2020. CEUR-WS.org.

KDnuggets (2020). "Optimization Algorithms in Neural Networks." *KDnuggets*, https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html.

KerasTCN (2020). Temporal Convolutional Networks for Keras (Philippe Remy). https://github.com/philipperemy/keras-tcn. Accessed: 15 April 2022.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Koch, G., Zemel, R., & Salakhutdinov, R. (2015, July). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop* (Vol. 2, p. 0).

LeCun, Y., L. Bottou, Y. Bengio and P. Haffner. (1998). *Gradient-based learning applied to document recognition*, Proc. of the IEEE, 1998

Martino, Thomas Di. "How to Choose Your Loss When Designing a Siamese Neural Net : Contrastive, Triplet or Quadruplet ?" *Medium*, Towards Data Science, 5 Nov. 2020, https://towardsdatascience.com/how-to-choose-your-loss-when-designing-a-siamese-neural-net-contrastive-triplet-or-quadruplet-ecba11944ec.

Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. (2010). "Recurrent neural network based language model." In *Interspeech*, vol. 2, no. 3, pp. 1045-1048.

Mueller, J., & Thyagarajan, A. (2016). Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

Mungalpara, Jaimin. (2021). "What Does It Mean by Bidirectional LSTM?" *Medium, Analytics Vidhya, 2 Mar. 2021,* https://medium.com/analytics-vidhya/what-does-it-mean-by-bidirectional-lstm-63d6838e34d9.

Nath, Sukanya (2021). *Style change detection using Siamese neural networks*, Notebook for PAN at CLEF 2021. Accessed 22 April 2022: http://ceur-ws.org/Vol-2936/paper-183.pdf

Nigam, Vibhor. (2021) "Natural Language Processing: From Basics, to Using RNN and LSTM." *Medium, Analytics Vidhya, 4 Jan. 2021, https://medium.com/analytics-vidhya/natural-language-processing-from-basics-to-using-rnn-and-lstm-ef6779e4ae66.*

PAN Competition (2022). https://pan.webis.de/clef22/pan22-web/style-change-detection.html

Prabhakaran, Selva. (2022). "Cosine Similarity - Understanding the Math and How It Works? (with Python)." *Machine Learning Plus*, 20 Apr. 2022, https://www.machinelearningplus.com/nlp/cosine-similarity/.

Prabhu. (2019). "Understanding NLP Word Embeddings-Text Vectorization." *Medium*, Towards Data Science, 21 Nov. 2019, https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223.

Pennington, Jeffrey; Richard Socher, and Christopher D. Manning. (2014). GloVe: Global Vectors for Word Representation. Accessed at:

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Saedi, C., & Dras, M. (2021). Siamese networks for large-scale author identification. *Computer Speech & Language*, *70*, 101241.

Sarigoz, Yusuf. (2022). "Triplet Loss-Advanced Intro." *Medium*, Towards Data Science, 25 Mar. 2022, https://towardsdatascience.com/triplet-loss-advanced-intro-49a07b7d8905.

Saxena, Shipra. (2021). *"Gated Recurrent Unit: Introduction to Gated Recurrent Unit(GRU)." Analytics Vidhya, 18 Mar. 2021, https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/.*

Schaetti, N. (2018). UniNE at CLEF 2018: Echo State Network-based Reservoir Computing for Cross-domain Authorship Attribution. In: Cappellato, L., Ferro, N., Nie, J.Y., Soulier, L. (eds.) *Working Notes of CLEF 2018 - Conference and Labs of the Evaluation Forum*. CEUR Workshop Proceedings, CLEF and CEUR-WS.org.

Shorfuzzaman, M., & Hossain, M. S. (2021). MetaCOVID: A Siamese neural network framework with contrastive loss for n-shot diagnosis of COVID-19 patients. *Pattern recognition*, *113*, 107700.

Stanford University. (2017). *"Lecture 10 | Recurrent Neural Networks*." YouTube, YouTube, 11 Aug. 2017, Lecture 10 | Recurrent Neural Networks

Strøm, Eivind. (2021) Multi-label Style Change Detection by Solving a Binary Classification Problem—Notebook for PAN at CLEF 2021. In Guglielmo Faggioli et al., editors, *CLEF 2021 Labs and Workshops, Notebook Papers*, September 2021. CEUR-WS.org.

Verma, Yugesh. *"Complete Guide to Bidirectional LSTM (with Python Codes)." Analytics India Magazine, 20 Nov. 2021, https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/.*

Wang, F., & Liu, H. (2021). Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2495-2504).

Woodford, Chris. (2021) "How Neural Networks Work - a Simple Introduction." *Explain That Stuff*, 30 Aug. 2021, https://www.explainthatstuff.com/introduction-to-neural-networks.html.

Yang, Parley. (2022). "Extra Mathematical Notes for Lectures and Classes.". Class notes for ST456.

Yang, S., Yu, X., & Zhou, Y. (2020). Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. In *2020 International workshop on electronic communication and artificial intelligence (IWECAI)* (pp. 98-101). IEEE.

Yanhui, Chen. (2021) “A Battle against Amnesia: A Brief History and Introduction of Recurrent Neural Networks.” *Medium, Towards Data Science, 8 Mar. 2021, https://towardsdatascience.com/a-battle-against-amnesia-a-brief-history-and-introduction-of-recurrent-neural-networks-50496aae6740*

Zangerle, E., Mayerl, M., Specht, G., Potthast, M., & Stein, B. (2020). Overview of the Style Change Detection Task at PAN 2020. In CLEF (Working Notes).

Zaremba, W., Sutskever, I., & Vinyals, O. *(2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.*

Zhijie Zhang, Zhongyuan Han, Leilei Kong, Xiaogang Miao, Zeyang Peng, Jieming Zeng, Haojie Cao, Jinxi Zhang, Ziwei Xiao, and Xuemei Peng. (2021). Style Change Detection Based On Writing Style Similarity—Notebook for PAN at CLEF 2021. In Guglielmo Faggioli et al., editors, *CLEF 2021 Labs and Workshops, Notebook Papers.* CEUR-WS.org.

Zuo, Chaoyuan; Yu Zhao, and Ritwik Banerjee. (2019). Style Change Detection with Feed-forward Neural Networks. *CLEF 2019 Labs and Workshops, Notebook Papers*, September 2019. CEUR-WS.org.

Zvornicanin, Enes. (2022). *“Differences between Bidirectional and Unidirectional LSTM.” Baeldung on Computer Science, 5 Feb. 2022, https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm*.