NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

VISUAL PROGRAMMING REPORT

Document Similarity Checker of Pages Using Line Segmentation and Word Embeddings

Submitted By

CT-17047 – Shaheer Ghani Imam

CT-17055 – Shaheer Akram

CT-17064 – Abdul Qadir Shaikh

Submitted To

Mr. Waseemullah

Date

23/01/2020

Table of Contents

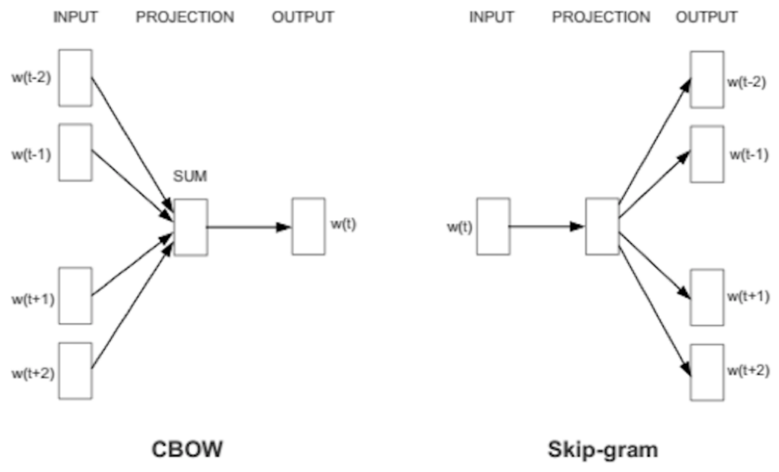
1. INTRODUCTION TO PROJECT

The problem of predicting similarity between a pair of handwritten document images written by potentially different individuals. This has applications related to matching and mining in image collections containing handwritten content. A similarity score is computed by detecting patterns of text re-usages between document images irrespective of the minor variations in word morphology, word ordering, layout and paraphrasing of the content. Our method does not depend on an accurate segmentation of words and lines. We formulate the document matching problem as a structured comparison of the word distributions across two document images. To match two-word images, we propose a convolutional neural network (CNN) based feature descriptor. Performance of this representation surpasses the state-of-the-art on handwritten word spotting. Finally, we demonstrate the applicability of our method on a practical problem of matching handwritten assignments.

2. LIBRARIES

2.1 GENSIM

Gensim is an open source python library for natural language processing and it was developed and is maintained by the Czech natural language processing researcher. Gensim library will enable us to develop word embeddings by training our own word2vec models on a custom corpus either with CBOW of skip-grams algorithms. There are two main training algorithms for word2vec, one is the continuous bag of words (CBOW), another is called skip-gram. The major difference between these two methods is that CBOW is using context to predict a target word while skip-gram is using a word to predict a target context. Generally, the skip-gram method can have a better performance compared with CBOW method, for it can capture two semantics for a single word



2.2 MXNET

MXNet is a powerful open-source deep learning framework instrument. In the last few years, the impact of deep learning has been widespread from healthcare to transportation to manufacturing and more. Deep learning is sought by companies to solve hard problems like speech recognition, object recognition and machine translation.
MXNet is used to define, train and deploy deep neural networks. It is lean, flexible and ultra-scalable i.e. it allows fast model-training and supports a flexible programming model and multiple languages.

2.3 NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars

2.4 SKIMAGE

scikit-image is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more.

3. HOW TEXT EXTRACTION WORKS

3.1 PAGE SEGMENTATION

Despite living in the 21st century, a large portion of everyday documents are still handwritten. Many school notes, doctor notes, and historical documents are handwritten. Archiving the handwritten documents is essential but it is usually limited to storing high resolution images of the documents. As the textual information in the documents is difficult to recognize, people resort to storing manually transcribed text of the handwriting There are currently considerable efforts to develop automated methods to transcribe handwritten text. Usually, the data flow consists of performing page segmentation to identify regions of texts within a document then running handwritten text recognition Page segmentation has been widely studied in historical documents where documents are segmented into decoration, background, text block, and periphery. Traditionally, handcrafted features are used for segmentation but recently, Chen et al. demonstrated that convolutional neural networks (CNNs) in an encoder-decoder architecture can automatically learn high-level feature representations of historic documents. The feature representations are fed into an SVM classifier to learn the segmentations of the document.

Here we focus on the segmentation of handwritten texts from the IAM dataset. Documents from the IAM dataset contain a printed portion and handwritten portion. The goal of this algorithm is to fit a bounding box around the handwritten portion of the document.

3.2 METHODS FOR BOUNDING BOX

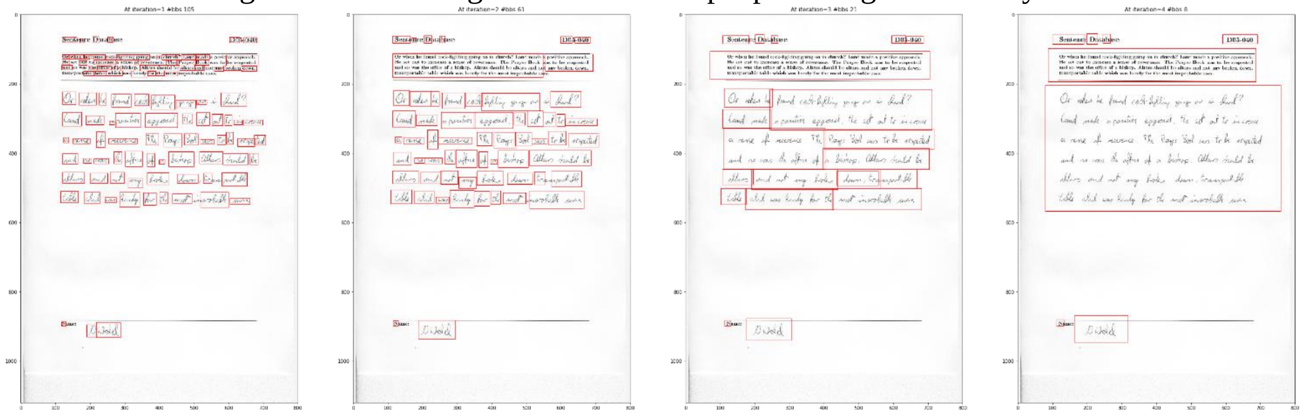
Two methods of obtaining the bounding box were explored: handcrafted features using the Maximally Stable Extremal Regions (MSERs) algorithm and using a deep CNN approach.

3.2.1 MSERS ALGORITHM APPROACH

The MSERs algorithm was used to detect "blobs" on the image which correspond to text on the images. The detected regions are post-processed to identify continuous regions of text with the following algorithm:

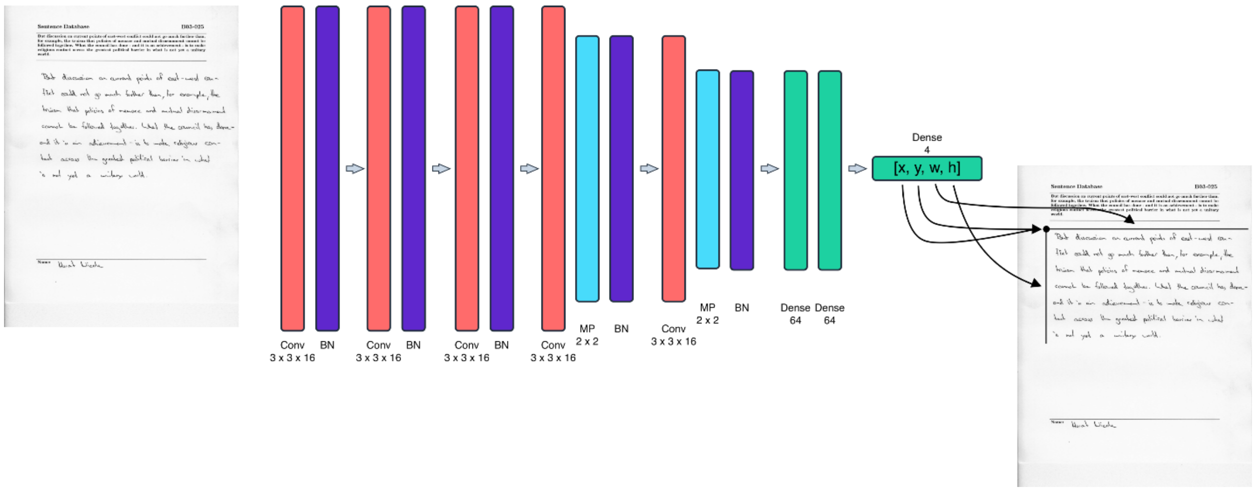
For *i* in iteration:

1. Expand the bounding boxes in all directions by a fraction denoted by *Δ*
2. Merge all the bounding boxes that overlap a percentage denoted by *I*



The MSERs algorithm was previously successful for detecting blocks of printed text. However, when the parameters (Δ and I) were not carefully tuned for the specific document, the algorithm fails to detect the passage. When the same image was used with different parameters, the algorithm fails to detect continuous regions of text. Handwritten text is more diverse compared to printed text where different people can have different writing styles, distance between letters, etc. After our experimentation, we found that it's difficult to obtain parameters that can generalise between different individuals. We therefore decided to implement a CNN approach to paragraph segmentation.

3.2.2 DEEP CNN APPROACH



The deep CNN was written using Apache MXNet and takes the IAM document as an input and predicts the bounding box of the handwritten passage. The network was initially trained to minimise the mean squared error of the predicted and actual bounding boxes. The bounding boxes of eight images are shown as training progresses using MXBoard (the MXNet logger to TensorBoard). As the weights of the network were randomly initialised, we can observe that the predicted bounding boxes initially tended towards the bottom right corner of the image. We can observe that as number of iterations increased, the predicted bounding box drifted towards the correct area. During the last few epochs presented (240 & 280), the size of the bounding boxes fluctuated and the network most likely overfitted to the training data.

The mean squared error was initially used as loss function because the intersection over union (IOU) loss function requires overlap between the predicted and actual bounding boxes (otherwise the values will be undefined). Therefore, after reasonable bounding boxes were generated with the mean squared area, the network was fine-tuned to minimise the IOU.

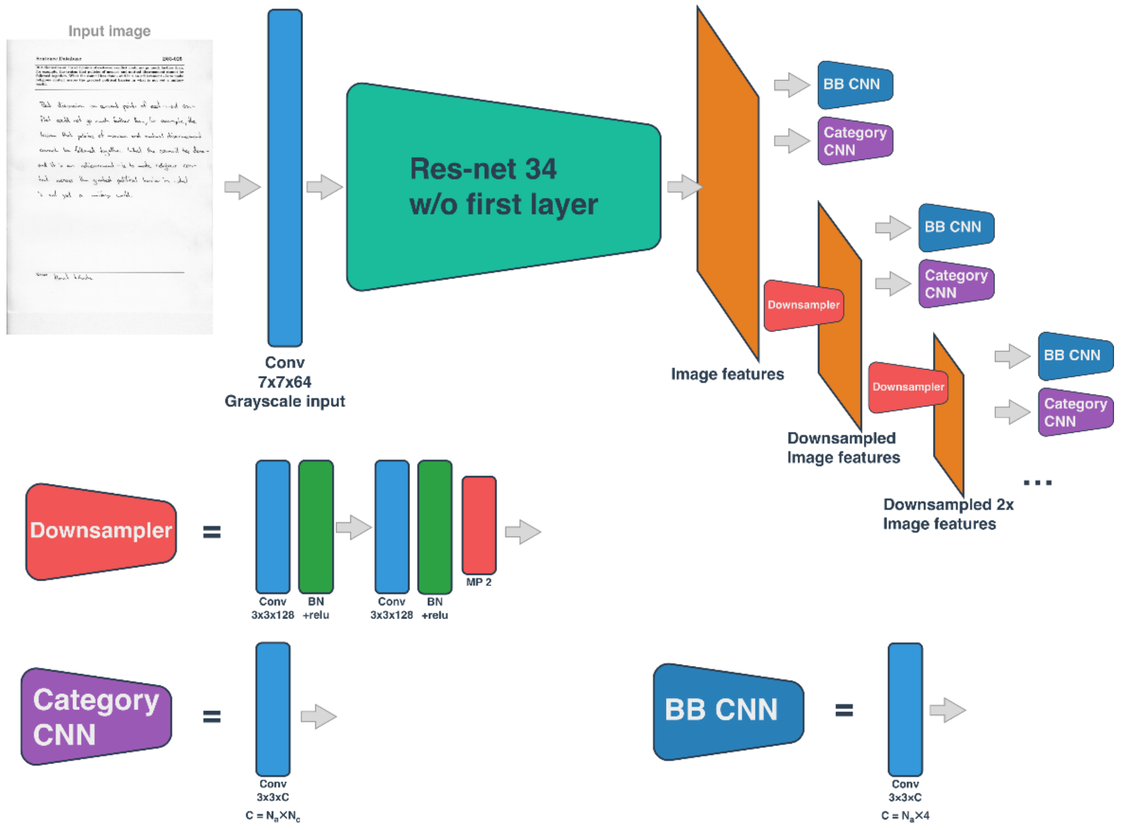
3.3 METHODS FOR IMAGE EXTRACTION

The input to the model is an image that only contains handwritten text. The outputs are bounding boxes that correspond to each line of the text. The problem is similar to the object detection problem in computer vision. In this method, we utilized the single shot multibox detector (SSD) architecture to detect the positions of each line of the passage. The SSD architecture essentially takes image features and repeatedly down samples the features (to account for different scaling factors). At each down sample step, the features are fed into two CNNs: one to estimate the locations of bounding boxes relative to anchor points, and one to estimate the probability of the bounding box encompassing an object. SSD was implemented using MXNet and was optimized to this application by altering:

- Network architecture
- Anchor points
- Data augmentation

- Non-maximum suppression

3.3.1 NETWORK ARCHITECTURE



The first convolutional layer of a pre-trained Resnet 34 (RGB) was replaced with a 1-channel convolutional layer (grayscale) by averaging the weights of the RGB channels.

3.3.2 ANCHOR POINTS

The bounding boxes encompassing lines of handwritten text are mostly restricted to horizontal rectangles. On the other hand, the bounding boxes required for general object detection dramatically vary in size. Therefore, rectangles with aspect ratios >1 were chosen for the current application. We also utilized two more anchor points compared to the blog post.

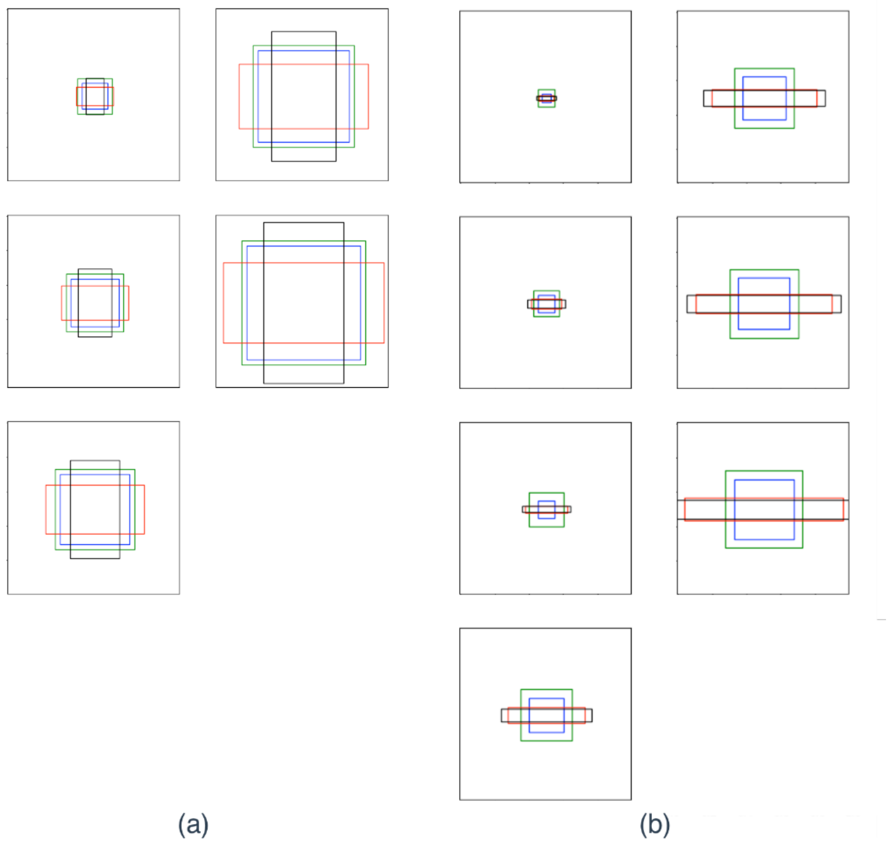
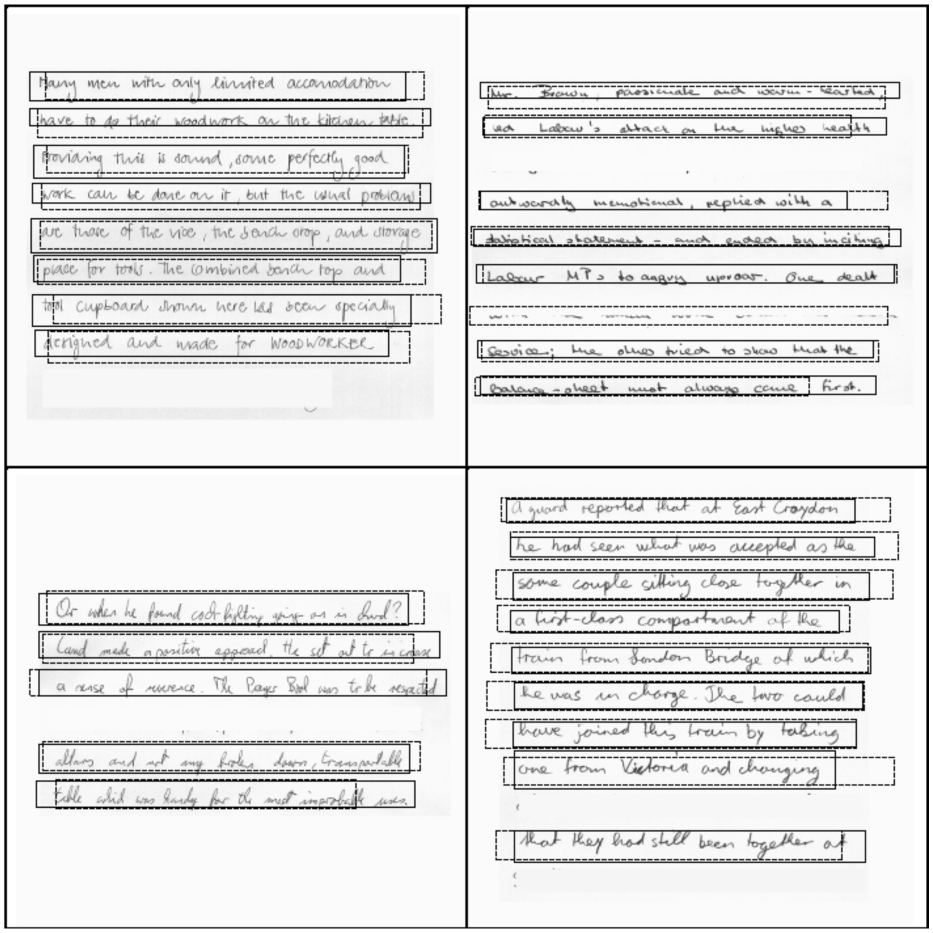


Figure: Network Architecture

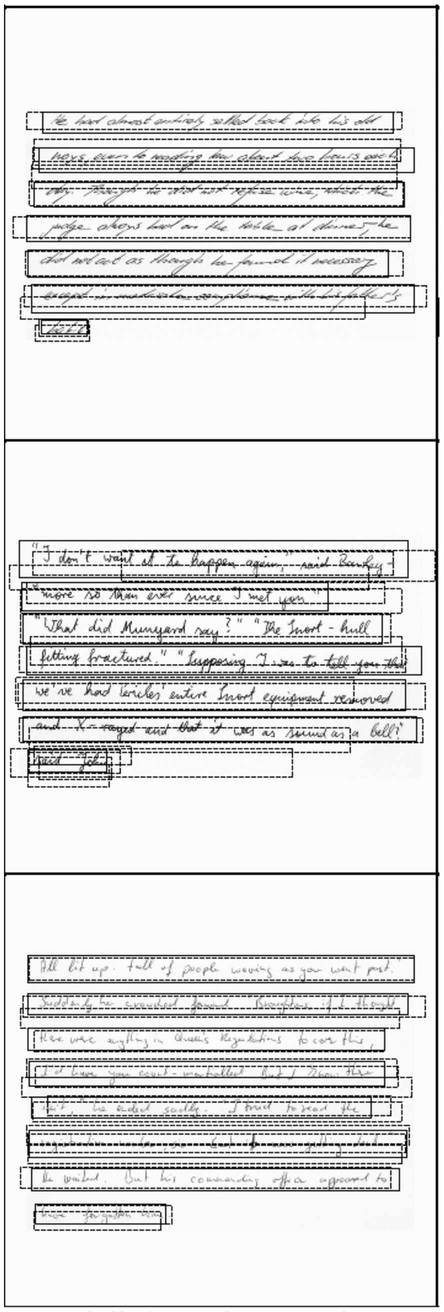
3.3.3 DATA AUGMENTATION

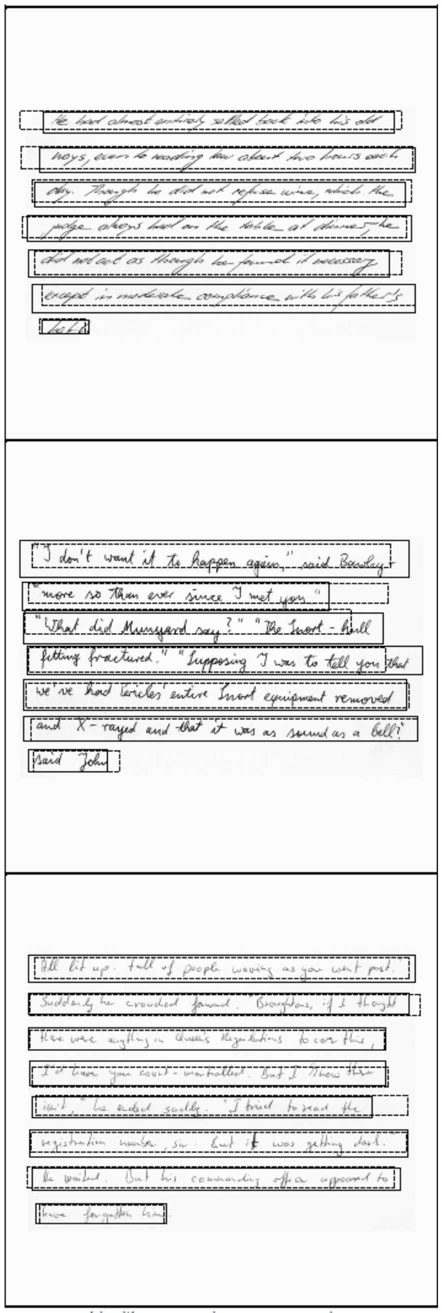
Many papers emphasize the importance of data augmentation when training the SSD model. In this application, random cropping and flipping is not appropriate as cropping will compromise the continuity of the text and flipping will reverse the writing direction. In this work, we similarly used random translations and we also introduced a method that randomly removed lines (that may be similar to random cropping). Specifically, each line was removed with a probability of *p (p=0.15* in this work*)*. The image was filled with the colour of the background where the bounding box is located and then the bounding box was removed



3.3.4 NON-MAXIMUM SUPPRESSION

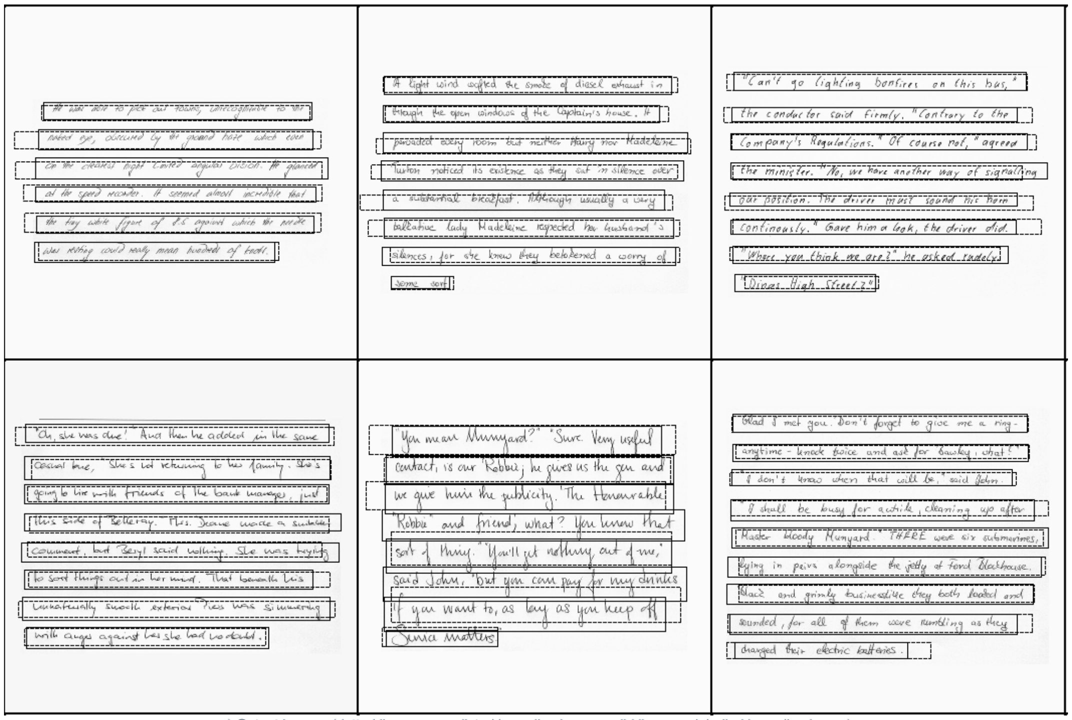
The network predicts numerous overlapping and redundant bounding boxes. To obtain more meaningful results, the box_nms (box non-maximum suppression) function was applied on the output of the network. Three parameters: overlap threshold (overlap_thres), top k boxes (topk), and minimum threshold (valid_thres) were varied and tuned. The parameters overlap_thres=0.1, topk=150, and valid_thres=0.01 were selected and the results are shown in Figure

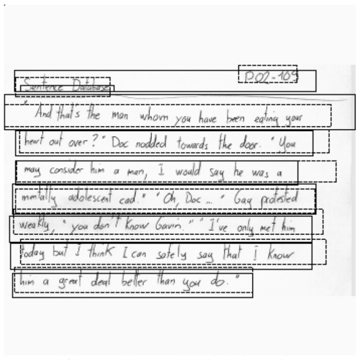

a) without non-maximum suppression

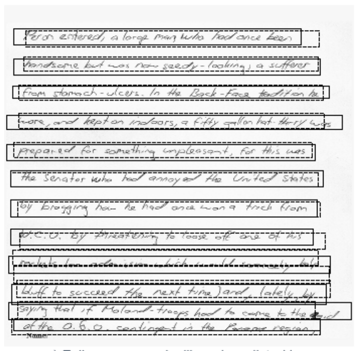

b) with non-maximum suppression

3.4 **RESULTS**

The final results are shown in Figure. As shown in Figure, we can see that the predicted bounding boxes have large overlaps with the labelled bounding boxes (train IOU = 0.593, test IOU = 0.573). We also observed that the network can learn the bounding boxes for handwriting with distinct lines substantially easier than large handwriting which overlaps.

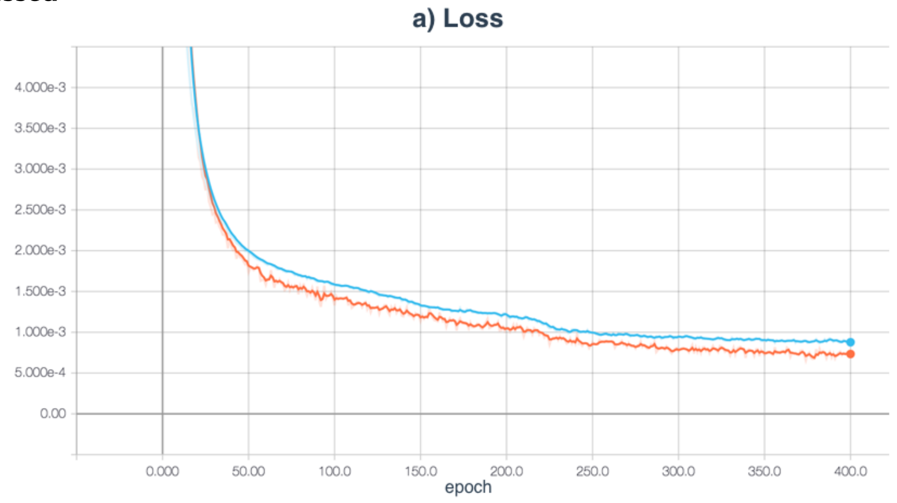

a) Output images (dotted lines are predicted bounding boxes, solid lines are labelled bounding boxes)

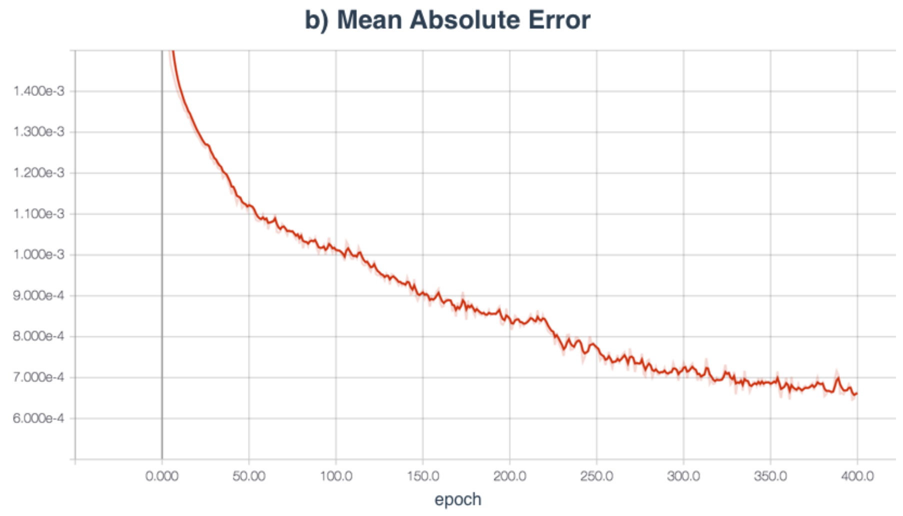

b) Ignored incorrectly labelled smudge
(second line from the top)


c) Failure cases: missalligned predicted box
(forth line from the bottom)

3.5 TRAINING PROGRESSION

In Figure, the training and testing loss, and the mean absolute error (L1 loss) is presented as a function of the epochs. The predicted bounding boxes are also predicted as the epochs progressed





4. CHECKING SIMILARITY

4.1 INTRODUCTION

Word2Vec is the foundation of NLP (Natural Language Processing). Tomas Mikolov and the team of researchers developed the technique in 2013 at Google. Their approach first published in the paper ‘Efficient Estimation of Word Representations in Vector Space’. They refined their models to improve the quality of representation and speed of computation by using techniques like sub-sampling of frequent words and adopting negative sampling. This work was published in the paper ‘Distributed Representations of Words and Phrases and their Compositionality’.

Word2Vec is an efficient and effective way of representing words as vectors. The whole body of the text is encapsulated in some space of much lower dimension. In this space, all vectors have certain orientation and it is possible to explicitly define their relationship with each other. The distributed representation of words as embedding vectors opens up many possibilities to find the words in ways that suits many applications in NLP

Consider the following similar sentences: *Have a good day* and *Have a great day*. They hardly have different meaning. If we construct an exhaustive vocabulary (let's call it V), it would have V = {Have, a, good, great, day}.

Now, let us create a one-hot encoded vector for each of these words in V. Length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better.

Have = [1,0,0,0,0]`; a=[0,1,0,0,0]` ; good=[0,0,1,0,0]` ; great=[0,0,0,1,0]` ; day=[0,0,0,0,1]` (` represents transpose)

If we try to visualize these encodings, we can think of a 5 dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means 'good' and 'great' are as different as 'day' and 'have', which is not true.

Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.

4.2 IMPLEMENTATION

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW). We are going to implement the Bag of Words

4.2.1 CBOW MODEL

This method takes the context of each word as the input and tries to predict the word corresponding to the context. Consider our example: *Have a great day.*

Let the input to the Neural Network be the word, *great.* Notice that here we are trying to predict a target word (*d*ay*)* using a single context input word *great.* More specifically, we use the one hot encoding of the input word and measure the output error compared to one hot encoding of the target word (*d*ay). In the process of predicting the target word, we learn the vector representation of the target word.

Let us look deeper into the actual architecture.

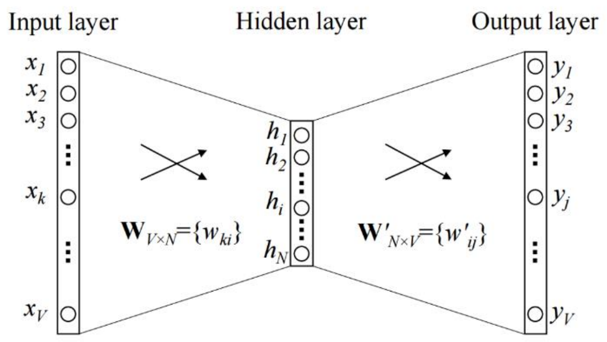


Figure 1: A simple CBOW model with only one word in the context

The input or the context word is a one hot encoded vector of size V. The hidden layer contains N neurons and the output is again a V length vector with the elements being the softmax values.

Let’s get the terms in the picture right:
- *Wvn is the weight matrix that maps the input x to the hidden layer (V*N dimensional matrix)*
-*W`nv is the weight matrix that maps the hidden layer outputs to the final output layer (N*V dimensional matrix)*

The hidden layer neurons just copy the weighted sum of inputs to the next layer. There is no activation like sigmoid, tanh or ReLU. The only non-linearity is the softmax calculations in the output layer.

But, the above model used a single context word to predict the target. We can use multiple context words to do the same.

4.3 RESULTS

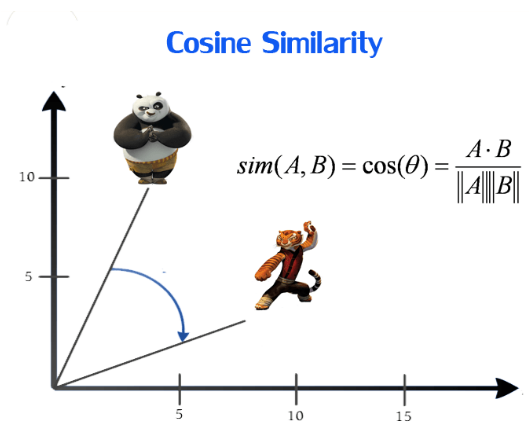
Here we will review a series of matrices. It is much easier to interpret pictures/patterns which give us a clear idea of about the numbers spread throughout the matrix. The content of each heat map below is discussed and interpreted in the respective captions. Each row in Score matrix is the product of the embedding vector for word as input and word as the target. The products are seen as probabilities in the probability matrix. Neighbouring characters within a window of 2 characters have higher values and dense probabilities. The distance matrix is the cosine distances from each embedding vector for input word to all the vectors embedding vectors for words as input including itself. Hot diagonal values are the product with itself and have distances of 1. Values close to diagonal are warm indicating they are close neighbours to each diagonal element. This pattern is much enhanced when the matrix is plotted as 1, positive and negative distances from each other as in smoothed distances.

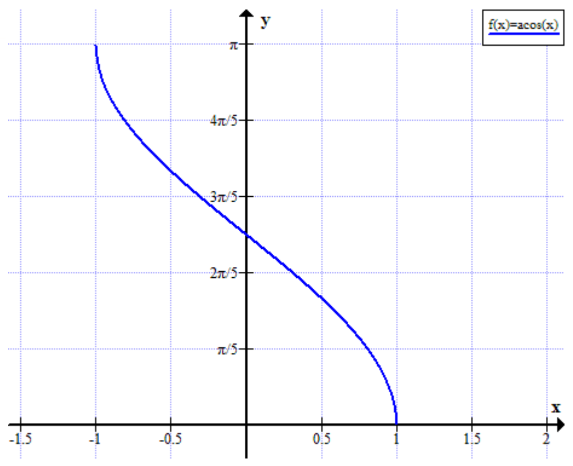
4.4 EMBEDDING VECTORS

First, let's see the embedding for 'a' in w1 and 'b' in w2 representing input/surrounding and target/middle words respectively. The vector product between 'a' — as input and 'b' — as a target is 18.217, a relatively large positive number whereas for all other characters it is small or negative (except for 'c'). Which implies 'b' (and 'c') will have a much higher probability. On the other hand, taking both vectors from w1, the cosine similarity between vector 'a' -as input and 'b' — as input is 0.4496 or about 63 degrees implying some similarity.

4.5 COSINE SIMILARITY

Among different distance metrics, cosine similarity is more intuitive and most used in word2vec. It is normalized dot product of 2 vectors and this ratio defines the angle between them. Two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.





Once the words are represented by vectors, the task of finding similar or dissimilar words becomes easier. Any combination vectors result in a new vector and the cosine distances or other similarity measures can be used as before. These operations underlie the famous equation 'king -man + woman = queen'.

4.6 FINDING NEIGHBOURS

Once the words are represented by vectors, the task of finding similar or dissimilar words becomes easier. Any combination vectors result in a new vector and the cosine distances or other similarity measures can be used as before. These operations underlie the famous equation 'king -man + woman = queen'.

4.7 CONCLUSION

The key to the implementation of word2vec is the construction of 2 complimentary weight matrices to represent words as input and as context or targets. Embedding dimension could be any arbitrary dimension depending upon vocabulary size. The output or prediction of the model is different from the similarity or distances. While the model learns to map from the input to output the embedding capture their relative position with each other. Softmax works well for smaller corpus but it becomes inefficient as the size of vocabulary grows due to the fact that it involves summing scores across the whole length of vocabulary. Skip gram with negative sampling is the preferred algorithm.
Finally, Gensim is a popular and freely available library which we can employ readily for word embedding and text analysis. But knowing the underlying principles makes it much clear and easier to use these tools.