Deep Recurrent Models for Authorship Attribution

Sanat Sharma
Dept. of Computer Science
University of Texas at Austin
sanatsharma@utexas.edu

Dhruv Rajan
Dept. of Computer Science
University of Texas at Austin
dhruv@krishnaprem.com

ABSTRACT

In this work, we look at various ways of tackling the problem of authorship detection. Given the rise of deep learning, our work encompasses both neural and non-neural models, and aims to find an efficient and reliable way to detect authorship. We also try to get an idea of the stylistic features needed to be learned in order to accurately distinguish between authors. Our models are run on a subset of the Gutenberg Dataset, along with a custom Spooky Author Dataset from Kaggle, and the Reuters dataset. In addition to several recurrent models, we also construct a probabilistic Variational AutoEncoder - Recurrent model for classification tasks.

Experimental results show that detection of style over long sentences is a hard task even for complex Long Short Term Memory networks (LSTMs), however, they work better than simple ngram models. Our Recurrent Neural Network (RNN) Encoder Decoder model coupled with self-attention is able to outperform the ngram (most common words) model by a minimum of 6% across the datasets.

The source code, preprocessed datasets and additional model information is available at:

`https://github.com/VeritasAuthorship/veritas`.

KEYWORDS

vae, rnn, gutenberg, reuters, authorship, attention

1 INTRODUCTION

Identifying and accurately discerning authorship of text is a key task that can be used in plagiarism detection as well as allowing relevant bodies (including research institutions and police forces) to create author profiles. Over the past couple decades, understanding stylistic characteristics of texts has become a well researched field. Ramnial et al. [16] in their work utilize text alignment and traditional machine learning algorithms such as k-nearest neighbors[6] to determine stylistic preferences for an author. Others have utilized a crowdsourcing approach to help annotate and distinguish authors based on writing style [17].

Stylometry: Approaches to the problem of authorship detection focus on constructing both stylistic and topic-based features. Stylistic features are generally based on occurrence of "function words" (articles, prepositions, conjunctions, etc.), while topic-based features attempt to capture the content focus of a document. Coulthard [5] in their paper, used idioletic and linguistic uniqueness to distinguish authors and determine authorship of unclassified documents. Similar stylstic approaches have also been used for forensic studies [4]. Stylistic analysis, though intriguing from a linguistic standpoint, requires a great deal of research and knowledge to do well manually. Sari et al. [18] analyzed feature-based classification approaches across multiple datasets, finding that the optimal choice of features depends heavily on characteristics of the dataset. For datasets with high topic variance per-author (e.g. news articles), topic features prove very useful, whereas on corpora with shorter texts, or high content similarity (e.g. IMDb reviews, legal judgments), function word-based features were most useful.

Recurrent Neural Networks: In recent years, with the advent and growth of deep learning, recurrent neural architectures have come out as the useful for understanding relationship between words in a sentence. While much work has been done applying recurrent network approaches to classification, much less has focused on authorship identification. Qian [15] investigate the efficiency of Gated Recurrent Units (GRUs) [4] and Long Short Term Memory networks [10] in understanding language semantics in sentences. They found that GRUs do a better job than LSTMs in recognizing authorship, especially on longer sentences or articles. The authors also found that recognizing authorship via articles was much more accurate than recognizing authorship based on individual sentences. This could primarily have been due to not enough stylistic information being present in a short sentence. Bagnall [1] used a character level "multi-headed" recurrent network to maintain a character-to-character language model for each author independently (this requires restricting the character set) to achieve state-of-the-art results on the PAN 2015 Author Identification task. A recent breakthrough in recurrent approaches is the "attention" mechanism (Bahdanau et al. [2]), which allows a recurrent network model to learn to focus only on relevant subsets of the input. Du and Huang [7] provided an attention-based method for sequence classification, which achieved state of the art results on multiple classification problems. Others, such as Yang et al. [25] have also delved into utilizing multiple layers of attention, in the hopes of better classifying longer passages of texts (documents).

Recurrent networks have also enabled development of models for tasks like text generation and sequence labeling. Models for these problems fit into an "encoder-decoder" framework(Sutskever et al. [21]). An "encoder" neural network converts an input sequence into a fixed size vector, and a decoder network (given this vector and other information from the encoder as input) can generate arbitrary-length output. A decoder which is able to produce multiple outputs might be helpful in determining authorship in multi-author documents.

Probabilistic Models: Probabilistic approaches with neural networks for text generation have gained popularity in recent years[24][19]. The idea is that learning a latent representation based on a probabilistic distribution of the underlying data helps the neural model learn better and not be overly sensitive to variance in input data. Xu and Sun [23] employed Variational AutoEncoders (VAE) to classify text in an unsupervised manner. They train a VAE and pass the output to a classifier network to distinguish movie text on the Large Movie Review Dataset (IMDb). The authors utilized a Gaussian latent space and showcased that their VAE model sampled positive reviews as positive values in the Gaussian space and negative reviews as negative values in the Gaussian space. While utilizing VAEs for classification tasks is not widely studied, this paper gives glimpses into their potential for being useful in such tasks.

The contributions of our work are summarized as follows.

- Mapped performance of traditional machine learning models such as N-grams and Logistic Regression for the task of authorship identification.
- Constructed Long Short Term Network (LSTM) classifer, an Encoder Decoder model with attention, and a Variational AutoEncoder - Recurrent Neural Network Decoder model (VAE-RNN). Cross examined performances of Neural models in comparison to traditional machine learning models.
- Performed authorship analysis on a custom subset of the Gutenberg dataset, both at a sentence based and article based level. Furthermore, also performed authorship analysis on Kaggle's Spooky dataset and the Reuters_50_50 (C50) dataset.

2 METHODOLOGIES

2.1 Datasets

For the task of authorship attribution, we restricted ourselves to only classifying works written in English, for this paper. In order to gain an insight of performance on both modern and older English, we selected and customized The Gutenberg Dataset, the Spooky Authorship Identification Dataset on Kaggle, and a subset of the Reuters C50 dataset.

2.1.1 ***Gutenberg Dataset.*** The Gutenberg dataset contains books and writings of many authors. We restricted ourselves to a selection of 6 authors (all British, with similar writing styles). Since each text is long, we extract short passages from each text to use as training examples (3-5 sentences, with a minimum of 50 characters, and a maximum of 500 characters) Importantly, our test set consists of passages from texts that are not in our training set, so the models cannot learn only topic-based features to score high. This does make the task much harder, though, for all the models.

2.1.2 ***Spooky Authorship Identification.*** The Spooky Author Identification consists of examples from the works of Edgar Allen Poe, Mary Shelley, and HP Lovecraft (all from horror stories). The passages, in this case are single sentences. We cannot use the provided blind test set for evaluation, so we use a 30% test split on the provided training set. We did submit our model's results to Kaggle to see the relative performance of our model on the blind test set.

2.1.3 ***Reuters C50 Dataset.*** The Reuters C50 Dataset consists of news articles written by 50 different writers. The train and tests sets each have 2500 texts (50 per author, 50 authors). We select subsets of 5-10 authors to test with, varying the number of texts per author that we train the model with.

2.2 Feature Sets

- **N-grams over words** - N-grams are useful in creating context rich feature sets, since they keep track of co-occurences of n words in sequence. We utilize bigram and trigram features over tokens, and have found these to give a significant improvement over unigram features
- **Part-Of-Speech** - Part-Of-Speech (POS) tags are widely used in Natural Language Processing to discern semantic information from words [13] [22]. We investigate whether stylistic information can be extracted from the a sentence by utilizing N-gram features over POS tags. We accomplish this by replacing each word with its POS tag, and construct N-gram features over the POS tags.
- **Word Embeddings** - With the rise of Neural Networks, creating vectorized representations of feature sets using word embeddings has become a widely studied technique. Based on work of Pennington et al. [14], we utilize 300 dimension Glove word embeddings trained on the Wikipedia and Gigaword corpus in our Neural models.

2.3 Baselines

2.3.1 ***Most Frequent Words.*** We utilize the most frequent words used by an author as a baseline to determine authorship. Previous works have shown that authors utilize unique linguistic traits in their works [4]. In order to utilize this uniqueness, we construct an N-gram feature bank for each author at train time. During our experiments, we found bigrams to be most effective. Part-of-speech (POS) analysis has also proven to be useful while discerning language identity[9]. Hence, in addition to creating N-gram of words used by authors, we also create N-grams of POS tags for each author.

Upon training on a large set of text, we utilize our constructed features for comparison against any new body of text to find the most probable author. Based on empirical evidence, we found N-gram of words to be much more effective than N-grams of POS tags

2.3.2 ***Logistic Regression.*** A second baseline we used was logistic regression over N-gram features. Consider multiclass logistic regression over K output classes, with an input size of d. To perform logistic regression, we use a $K \times d$ weight matrix $\mathbf{W}$ to parameterize our output hypothesis. On an input example x we represent the *likelihood* of an output class y for x by

$$\begin{aligned}\Pr(y = k|\mathbf{x}) &= \mathrm{softmax}(\mathbf{Wx}) \\ &= \frac{e^{(\mathbf{Wx})_k}}{\sum_{i=1}^{K} e^{(\mathbf{Wx})_\mathrm{i}}}\end{aligned}$$

We want to choose $\mathbf{W}$ to maximize this likelihood over our training set. We can reformulate this as a loss minimization problem, using the **softmax cross-entropy loss**:

$$\ell(\mathbf{Wx}, y^*) = -\log \Pr(y = y^*|\mathbf{x}) = -\log(\text{softmax}(\mathbf{Wx})_{y^*})$$

Now we can use any optimizer to minimize this loss, and compute the optimal $\hat{\mathbf{W}}$

$$\hat{\mathbf{W}} \leftarrow \arg\min_{\mathbf{W}} \frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{Wx}, y_i)$$

So that our final hypothesis $f_{\hat{\mathbf{W}}}$ is

$$f_{\hat{\mathbf{W}}}(x) = \arg\max_{k \in [K]} \text{softmax}(\hat{\mathbf{W}}x)_k$$

2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) provide a framework for deep learning, in the case that input or output data is sequential. We hypothesized that for the task of authorship detection, recurrent models would be useful since they are perform well on the important task of understanding and retaining stylistic information over a body of text. Recurrent architectures are defined by specific recurrent units (GRU, LSTM, etc.), which are applied (or "unrolled") over the length of a sequence. RNNs model the probability of a "next label", given a sequence of input examples.

Specifically, given a hidden state h_{t-1} and a vector $\mathbf{x}$, a recurrent unit computes a new hidden state h_t. During prediction, the new hidden state h_t is used to compute probabilities for the choices of the label y_t, as follows.

At timestep t:

$$\begin{cases} h_t = \sigma(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \\ \Pr(y_t|\mathbf{x}, y_1, \ldots, y_{t-1}) = \text{softmax}(W_{yh}h_t) \end{cases}$$

W_{hh}, W_{hx}, and W_{yh} are weight matrices which parameterize affine transforms on the vectors x_t and h_t, allowing vector representations to be mapped across dimensions. The exact values of these are arbitrary, and can be learned via "backpropagation through time".

There are a few issues, however, with this simple recurrent architecture. When training this recurrent network on long input sequences using backpropagation, the *vanishing* and *exploding gradient* problems will keep the gradient update steps from having their intended effects. More sophisticated recurrent units have been devised, which attempt to moderate these gradient issues, and more strictly model information transfer between input and output hidden states; most notably, the Gated Recurrent Unit (GRU), and the Long Short-Term Memory Unit (LSTM).

2.4.1 Gated Recurrent Unit (GRU). In a GRU cell, the computation of h_t is given by the following:

$$\begin{cases} z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1} + b^{(z)}) \\ r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} + b^{(r)}) \\ \tilde{h}_t = \tanh(r_t \circ U^{(h)}h_{t-1} + W^{(h)}x_t + b^{(h)}) \\ h_t = (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} \end{cases}$$

The update gate z_t, decides how much information to update h_{t-1} with, while the reset gate, r_t, decides how much past information should be kept in h_t.

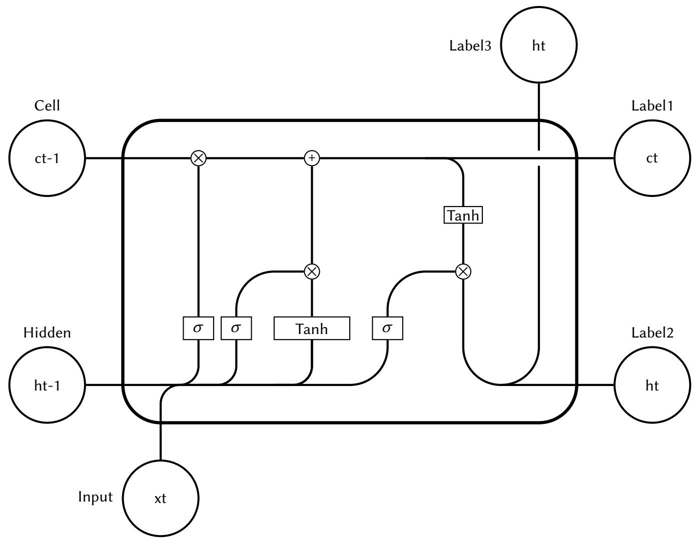

Figure 1: LSTM Cell

2.4.2 Long Short Term Memory Unit (LSTM). The LSTM units are slightly more complex mathematically; the sequence of steps to update h_t is given by

$$\begin{cases} i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}) \\ f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}) \\ o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(c)}) \\ \tilde{c}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1} + b^{(c)}) \\ c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t = o_t \circ \tanh(c_t) \end{cases}$$

The LSTM hidden state consists both of c_t and h_t. The cell state, c_t, is introduced as a mechanism for preserving "long-term memory". f_t (forget gate), i_t (input gate), and o_t (output gate) compute a series of, essentially, soft probability masks, which are applied to h_{t-1} and c_{t-1}. $\tilde{c}_t$ represents new information, computed from x_t and h_{t-1}. c_t is constructed by combining this new information ($i_t \circ \tilde{c}_t$) with the old memory ($f_t \circ c_{t-1}$), where f_t allows information to be forgotten from old memory, and i_t allows information to be left out of the current step. Finally h_t is computed by masking $\tanh c_t$, which essentially maps c_t back to the appropriate range, and provides an extra importance mask to ensure h_t retains current information.

2.5 Long Short-Term Memory Classifier

As mentioned above, an LSTM unit consists of a cell and multiple gates including an input gate, an output gate and a forget gate. The gates allow the unit to retain cell state over an arbitrary period of time, thus potentially understanding contextual information.

We use an LSTM classifier as one of our methods to extract informative meaning out of a sentence (see 2). We utilize a bidirectional LSTM with a hidden size of 200 and batch size of 1. Dropout is also utilized to prevent overfitting of data and to enhance the training process[20]. We utilize a dropout rate of 0.2 for all neural models

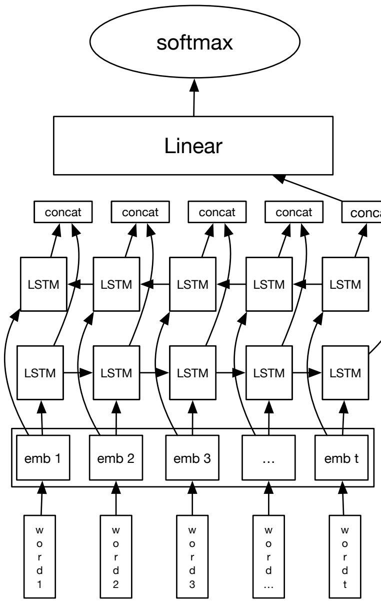

Figure 2: Bidirectional-LSTM Classifier Architecture

in this paper. To reduce the weights of the bidirectional LSTM, we use two hidden fully connected layers, initialized using Glorot initializer(Glorot and Bengio [8]). In addition, we also utilize an Embedding Layer initialized with Glove pretrained embeddings to learn word-embeddings for each input token. For the encoder LSTM as well as the embedding layer, we use the Adam Optimizer[11], with learning rate of 0.0005. This rate was chosen experimentally to learn the weights effectively and efficiently. The encoder takes in a sentence (list of tokens) and returns the LSTM output, along with the hidden and cell states of the LSTM. The hidden state was passed on to a fully connected Neural network layer to reduce its dimensionality. This is a common technique used in classification tasks, since a gradual reduction in dimensionality helps retain important features better. A distribution of authorship likelihood is finally generated by passing the output of the neural net to a Softmax layer.

2.6 LSTM Classifier

2.7 Encoder Decoder with Attention

With the growth of research on sequence-to-sequence problems, such as machine translation, encoder-decoder models have proven to be extremely successful, and widely used. We propose using an encoder-decoder model, with an attention mechanism for the problem of authorship attribution: theoretically, using an Encoder-Decoder structure, we might be able to predict authorship in documents with multiple authorsfi!?say by having the decoder return a sequence of authors, in decreasing order of likelihood match with the input document. In order for this to work, the recurrent model would have to be able to capture style in such a way that a decoder could extract authorship information from the encoder hidden states. In this work however, we devote our efforts into

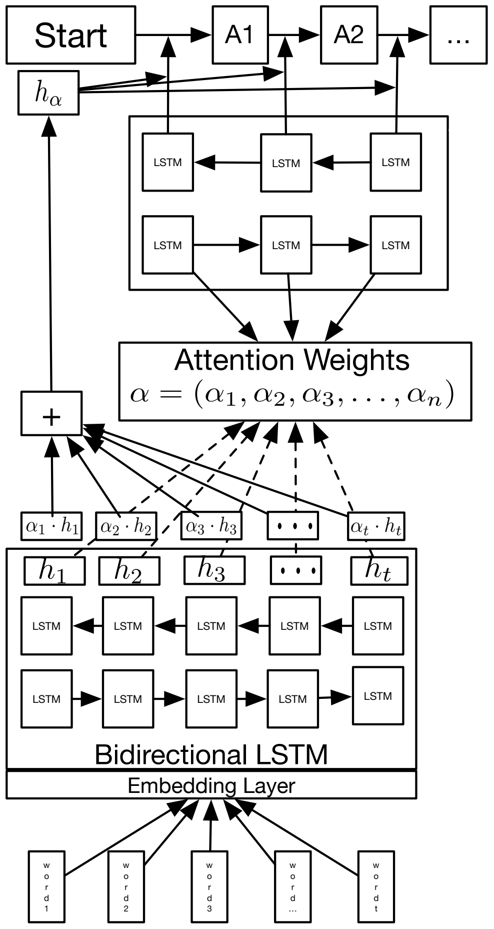

Figure 3: Encoder-Decoder with Attention Classifier Architecture (Multiple Author Classification)

analyzing this model in the single-author case. Even if we run the decoding step only a single time, the attention mechanism should still help, and learn to focus on the relevant parts of the input sentence in predicting authorship.

2.7.1 Encoder. For the encoder model, we utilize an LSTM encoder similar to the one trained for the LSTM classifier. The key difference is that unlike the classifier where we pass the hidden state to a fully connected neural network and subsequently retrieve a distribution over authors, in the encoder-decoder model, we save the hidden state. The hidden state is important since it provides a rich set of contextual feature information about the input to the decoder.

2.7.2 Decoder With Attention. For the decoder model, we utilize an LSTM cell with a hidden size of 200, dropout rate of 0.2 and a batch size of 1. We utilize an embedding layer to learn the embeddings for each of the authors to in the output set. Similar to the Encoder, we utilize the Adam optimizer[11] and found a learning rate of 0.0005 to be most effective. The need for separate embedding layers for the encoder and decoder steps arises due to the sentence inputs to the encoder being different from the decoder inputs (author embeddings).

Recurrent Neural Networks often have difficulty retaining contextual information over long sentences/passages of text. Attention helps counter this by retaining context from each part of the input sentence. We use the Attention mechanism proposed by Bahdanau et al. [2], as seen in fig 3. An attention computation, more generally, describes a way to condense or summarize a sequence of vectors into a single vector, with the intent of preserving the effect of the "most important ones" (in our case, the words in a sentence which determine authorship most effectively). The goal is to learn weights for this process which correspond to these importance entities. The decoder architecture proposed by Bahdanau et al. [2] allows these attention weights to be computed at each decoding step i, based on the decoder's hidden state $\tilde{h}_i$. Mathematically, it can be described as follows.

e_{ij} is a form of a similarity measure, between a decoded hidden state i, and an encoded hidden state j. It helps convey information about how relevant the input word j is to the current decoding step i. We compute the attention weights by computing a softmax over these importances, over all encoded hidden states. Using these attention weights, we compute a context vector c_i as the weighted average of the encoded hidden states, according to the attention weights αij.

We train our combined encoder decoder model together using NLLLoss loss.

2.8 Variational AutoEncoder - Recurrent Neural Network

2.8.1 ***Variational AutoEncoder.*** Variational Autoencoder have become popular as generative models in recent times. Bowman et al. [3], in their work utilized Variational Autoencoders (VAE) for textual generation. The generative concept of VAE is relatively unembellished; firstly a latent space is sampled from a distribution, usually a Gaussian distribution, and then an output is constructed/sampled from a posterior distribution parameterized by a neural network (decoder), with respect to the latent space.

The core idea is that the by sampling to create the latent space /representation, VAEs are more resistant to fluctuations and variations in the data. The choice of posterior representation and the decoder architecture depends on the data being modelled, with the intent of capturing the true identity of the underlying data in the posterior representation. Yang et al. [24] showcased that VAEs were able to accurately learn a latent representation for various kinds of textual data, utilizing a Multivariate Gaussian distribution for encoder sampling.

Our architecture consists of a VAE encoder and decoder, along with an RNN classifier. For the VAE encoder, we first construct a latent vector representation z utilizing a multivariate Gaussian prior and then sample to get the encoder output $p_\theta(z)$.

For the decoder step, since we wish to segregate the final output into various clusters denoting authors, we utilize a Binomial distribution to sample from. The decoder emits text from a conditional distribution ($p_\theta(\mathbf{x}|\mathbf{z})$).

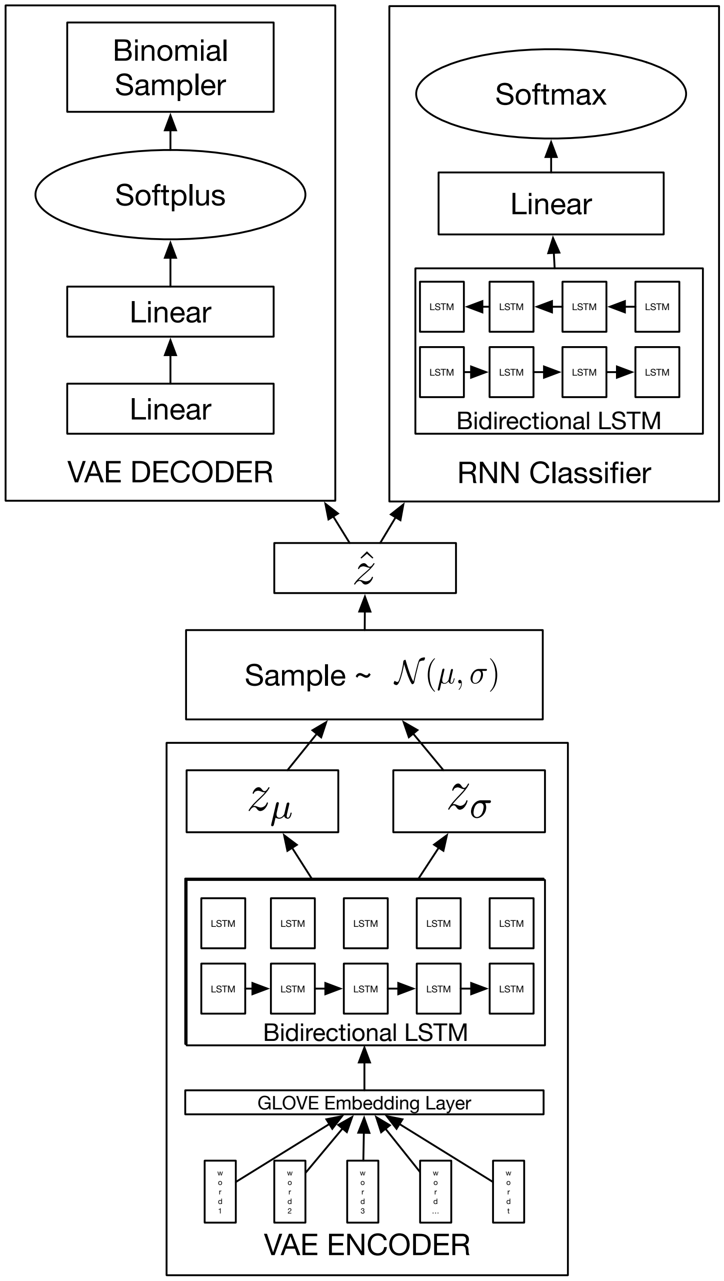

Figure 4: Variational AutoEncoder - RNN Architecture

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_\theta (x_t | x_1, x_2, \ldots, x_{t-1}, \mathbf{z})$$

The training process uses a variational lower bound to estimate parameters;we use the objective previously defined by Kingma and Welling [12].

$$\log p_\theta(\mathbf{x}) = -\log \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) dz \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$$
$$-KL(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\phi(\mathbf{z}))$$

2.8.2 ***RNN Classifier.*** Since we wish to utilize VAEs for a classification task instead of a generation task, we utilize an RNN layer as our classifier, as shown in fig 4. The RNN layer takes in the encoder output from the VAE and aims to classify text based on the information present in the latent representation .

We utilize this approach since we believe that the latent representation would be able to capture stylistic information about an author. We pass the latent representation to an LSTM unit with hidden size of 200, dropout rate of 0.2 and trained using Adam optimizer[11], with the learning rate of 1e-3. The RNN classifier uses NLLLoss for calculating loss. The LSTM output is passed to a fully connected neural layer followed by a Softmax layer, which calculates a distribution over the authors.

It is important to note that the loss for the VAE and the RNN classifier are calculated separately.

3 EXPERIMENTS

3.1 Dataset Setup

3.1.1 Gutenberg. For the Gutenberg dataset, we chose 6 British authors with similar writing styles for comparison. Furthermore, in order to test our models with various types of inputs, we run experiments with sentence-based classification and passage-based classification.

- Passage-based - Passage-based classification involves passing an entire passage to the model, and predicting authorship based on model output. As described earlier, we describe a passage as a collection of 3-5 contiguous sentences.
- Sentence-based - Sentence-based classification involves sentence-wise passing of all sentences to a model and retrieving authorship predictions for each sentence. Once we retrieve predictions for each sentence, we ascribe authorship for the passage to the max retrieved authors from the sentences.

Based on our experiments, we found a sentence-wise approach to work better overall, especially with neural models. This likely can be attributed to recurrent models being unable to retain state over long passages.

3.1.2 Spooky. The Spooky dataset is already split into sentences, and hence we assign authorship for each sentence in the training and test sets

3.1.3 Reuters. To run our classifiers on the Reuters dataset, we use sentence-based classification, as this was shown to work much better, on the Gutenberg passages.

Additionally, we run our model on articles from a subset of the authors: we select 3-5 authors to test with, training and testing on the same number of passages for each author.

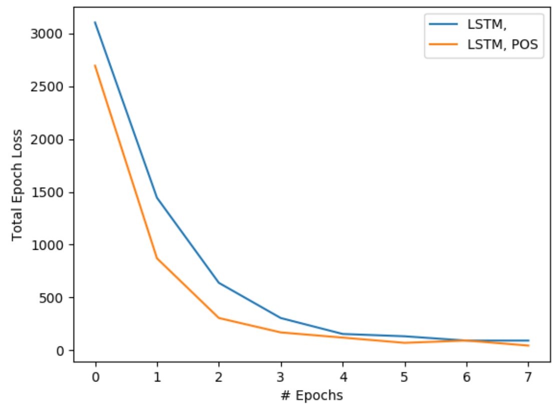

Figure 5: LSTM Model Losses (Glove Embeddings, and POS Tags) on Reuters Dataset

3.2 Baseline

We ran varied experiments on our baseline models to measure performance. For the most common words per author model as well as the logistic regression model, we found N-gram size of 2 to be most effective. Furthermore, we also tested our baselines with both N-gram of words as well as N-gram of POS tags of the words.

For all datasets, we found a significant decrease in performance when using N-gram of POS tags. For the Reuters dataset, the accuracy of both most common words as well as logistic regression fell from over 90% accuracy to sub 30% accuracy. This may be attributed to losing author specific vocabulary information on conversion of words to POS tags. Logistic regression outperformed most common words baseline and was able to get as high as 97.8% correctness on the Reuters dataset (in comparison to 90%).

Both models struggled to get good accuracy on the Gutenberg dataset. This may be attributed to the fact that the test set was from books not trained on, as well as the author writing styles and word choices being extremely similar. We saw this trend in all our neural models as well.

Our LSTM classifier was trained for 8 epochs for all our experiments. Furthermore, we utilized pretrained Glove embeddings for initialization. We found that running the neural model sentence-wise on the Gutenberg dataset yielded the best results. We also found that for longer passages, the LSTM classifier struggles in accurately classifying the author.

During training, we found that our loss decreased nicely, as shown in fig 5 . We found a learning rate of 0.0005 to be most effective. Running the classifier on the Reuters dataset, we were able to achieve an accuracy of 96%, while only achieving a 35.6% accuracy on the Gutenberg dataset.

We compared these results with those we get by replacing the LSTM cells with GRU cells in this instead of LSTM recurrent cells. It seems that the results do not differ too much, but the LSTM model does better across datasets. The GRU model achieves 76.6% accuracy on the Spooky dataset, 30.8% on the Gutenberg dataset, and 94.6% on the Reuters dataset.

Table 1: Experiment Results.

Model	Gutenberg (British, 5 authors)	Spooky (3 Authors)	Reuters (3 Authors)
N-gram	0.297	0.648	0.9
N-gram (pos tags)	0.295	0.4	0.333
Logistic Regression	**0.413**	0.80	0.978
Logistic Regression (pos tags)	0.39	0.53	0.32
LSTM	0.358	0.757	0.96
Encoder-Decoder with Attention (GRU)	0.308	0.766	0.946
Encoder-Decoder with Attention (LSTM)	0.353	0.777	**0.961**
Encoder - Decoder (pos tags) with attention (LSTM)	0.354	**0.803**	0.933
VAE-RNN	0.2	0.40	0.33

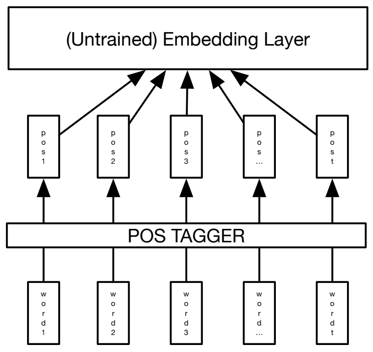

Figure 6: Learning POS tag embeddings

3.3 Encoder-Decoder with Attention

The Encoder Decoder model is frequently used as a generative model. One major benefit it holds over other neural models utilized is its ability to detect and decode multiple authors. For our experiments, we used singular author documents and hence hand-trained the model to only decode one author for each example.

We experimented with training the encoder decoder with both word embeddings as well as N-gram POS tag embeddings. While the word embeddings were initialized using Glove, the N-gram POS tag embeddings were learned from scratch (see figure 6).

Interestingly, unlike the baselines, the POS tag embeddings showed a slight improvement in results and increased our score on the Spooky dataset from 77.7% to 80.3%. However, these gains were not replicated for other datasets, with accuracies being mostly unaffected by using POS tag embeddings. Interestingly, though, the LSTM models work much better than the baseline models, when training using POS tags; for the recurrent models, the accuracies decrease only by a few percentage points (or increase), while the baseline models drop drastically in performance. This suggests that there *is* stylistic information to be learned that has to do only with each author's choice of grammar, and which is being captured by the learned word embeddings in the recurrent models.

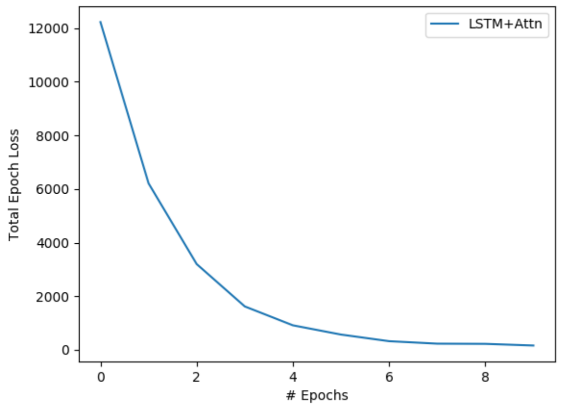

Figure 7: Encoder Decoder Model Loss: Spooky Dataset, using Glove Embeddings

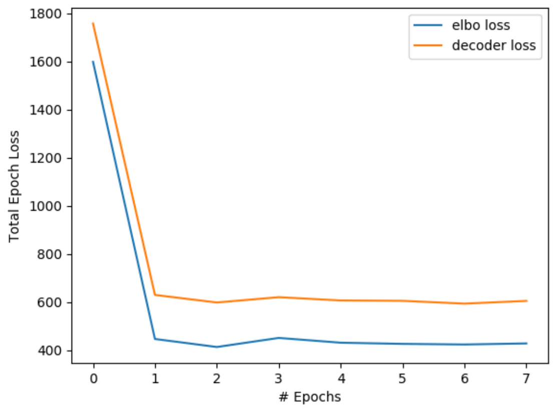

Figure 8: VAE Model Losses (ELBO, Rnn Decoder) on Reuters Dataset

3.4 VAE —RNN

The VAE—RNN training comprises two parts. The first involves training the Variational Autoencoder, and then utilizing the encoder part of the VAE, we train the RNN classifier. We train the VAE in an unsupervised manner, utilizing ELBO objective function (as a lower bound of log likelihood), and Adam optimizer for 8 epochs. Based on our experiments, we found that the VAE loss decreased well, however, this was not complemented by a smooth decrease in RNN loss. This meant that the VAE encoder was not learning a valuable enough latent representation to accurately help predict authorship. Since the RNN classifier did not learn well, the architecture ended up performing poorly. Looking at the loss diagrams for any of the datasets (see fig 8 regarding the REUTERS dataset), the losses do not decrease significantly after the first epoch.

4 FUTURE WORK

In this work we studied various recurrent architectures and found some approaches that worked and some that did not fare as well. We wish to supplement this work by diving deeper into probabilistic models for classification and make our VAE-RNN model perform well for textual classification.An interesting result we found during our experiments is that there seems to be enough stylistic information purely in the grammar (as represented by POS tag sequences in english) for recurrent networks to nearly match the optimal performance among these models. It would be really interesting to explore this and other feature sets, and investigate further how style gets captured in the recurrent networks and how to exploit the representation more effectively.

Some languages have well defined stylistic features, whereas others are more subtle in word choice and style. In future works, we wish to investigate the task of authorship recognition on works in other languages such as French, German and compare the results we receive to our current results. Finally, we wish to expand on our work of detecting one author, to multi-author detection. This is extremely useful in analyzing both court ruling (and learning more about a judge's disposition) as well as ascribing authorship in academic/historical texts.

REFERENCES

[1] Douglas Bagnall. 2015. Author Identification using Multi-headed Recurrent Neural Networks. *CoRR* abs/1506.04891 (2015). http://dblp.uni-trier.de/db/journals/corr/corr1506.html#Bagnall15

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014). http://arxiv.org/abs/1409.0473

[3] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2015. Generating Sentences from a Continuous Space. *CoRR* abs/1511.06349 (2015). http://arxiv.org/abs/1511.06349

[4] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *SSST@EMNLP*.

[5] Malcolm Coulthard. 2004. Author identification, idiolect, and linguistic uniqueness. *Applied linguistics* 25, 4 (2004), 431–447.

[6] T. Cover and P. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (January 1967), 21–27. DOI: http://dx.doi.org/10.1109/TIT.1967.1053964

[7] Changshun Du and Lei Huang. 2018. Text Classification Research with Attention-based Recurrent Neural Networks. *International Journal of Computers Communications Control* 13 (02 2018), 50. DOI: http://dx.doi.org/10.15837/ijccc.2018.1.3142

[8] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Yee Whye Teh and Mike Titterington (Eds.), Vol. 9. PMLR, Chia Laguna Resort, Sardinia, Italy, 249–256. http://proceedings.mlr.press/v9/glorot10a.html

[9] Fahim Muhammad Hasan, Naushad UzZaman, and Mumit Khan. 2007. Comparison of different POS Tagging Techniques (n-gram, HMM and Brill's tagger) for Bangla. In *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, Khaled Elleithy (Ed.). Springer Netherlands, Dordrecht, 121–126.

[10] Sepp Hochreiter and Jrgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation* 9 (12 1997), 1735–80. DOI: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[13] Julian Kupiec. 1992. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech & Language* 6, 3 (1992), 225–242.

[14] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[15] Zhang Qian, He. Deep Learning based Authorship Identification. (????). https://web.stanford.edu/class/cs224n/reports/2760185.pdf

[16] Hoshiladevi Ramnial, Shireen Panchoo, and Sameerchand Pudaruth. 2016. Authorship Attribution Using Stylometry and Machine Learning Techniques. In *Intelligent Systems Technologies and Applications*, Stefano Berretti, Sabu M. Thampi, and Praveen Ranjan Srivastava (Eds.). Springer International Publishing, Cham, 113–125.

[17] Andi Rexha, Mark Kröll, Hermann Ziak, and Roman Kern. 2018. Authorship identification of documents with high content similarity. *Scientometrics* 115, 1 (01 Apr 2018), 223–237. DOI: http://dx.doi.org/10.1007/s11192-018-2661-6

[18] Yunita Sari, Mark Stevenson, and Andreas Vlachos. 2018. Topic or Style? Exploring the Most Useful Features for Authorship Attribution. In *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, 343–353. http://aclweb.org/anthology/C18-1029

[19] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2017. A Hybrid Convolutional Variational Autoencoder for Text Generation. *CoRR* abs/1702.02390 (2017). http://arxiv.org/abs/1702.02390

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR* abs/1409.3215 (2014). http://arxiv.org/abs/1409.3215

[22] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 173–180.

[23] Weidi Xu and Haoze Sun. 2016. Semi-supervised Variational Autoencoders for Sequence Classification. *CoRR* abs/1603.02514 (2016). http://arxiv.org/abs/1603.02514

[24] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. *CoRR* abs/1702.08139 (2017). http://arxiv.org/abs/1702.08139

[25] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *HLT-NAACL*.