Cross Domain Fake News Detection

Arnav Chakravarthy & Amrita Bhattacharjee

November 2019

1 Introduction

With the proliferation of social media among the general public, and the vast reachability that it provides, these platforms can be perfect for rapid dissemination of important information including news, natural calamity forecast, alerts and warnings. However, this has also been misused for malicious purposes by certain individuals and groups, to facilitate the spread of false and fabricated 'information', often on sensitive topics and issues. The consequences of this can be far reaching, with hundreds and thousands of people becoming unfortunate victims.

Recently there have been attempts to understand how to address this growing problem, including detection of fake news, and making attempts at thwarting the spread to some extent [2] [4]. However, since these studies require data that might often be very sensitive, finding relevant public datasets to train on is nearly impossible. This engenders the need to find a way to apply knowledge learned from data belonging to a different domain, and apply it to the data we want.

Auto-Encoders and Seq2Seq Models: Sequence to Sequence models have been widely used for a variety of NLP tasks, the most common ones being machine translation, text summarization, building conversational models etc. As the name suggests, this kind of model takes in a sequence as the input, and outputs a sequence (eg. a sequence of words). *Autoencoders* are a type of sequence to sequence models, and usually include two parts - *Encoder* and *Decoder*.

- *Encoder:* The encoder is usually implemented with a recurrent neural network, or the more common variants - LSTMs or GRUs. The encoder is responsible for transforming input words into a hidden vector representation.
- *Decoder:* The decoder is also implemented with recurrent neural networks - most often LSTMs and GRUs. The decoder takes in the hidden state vector and tries to reconstruct the input sequence.

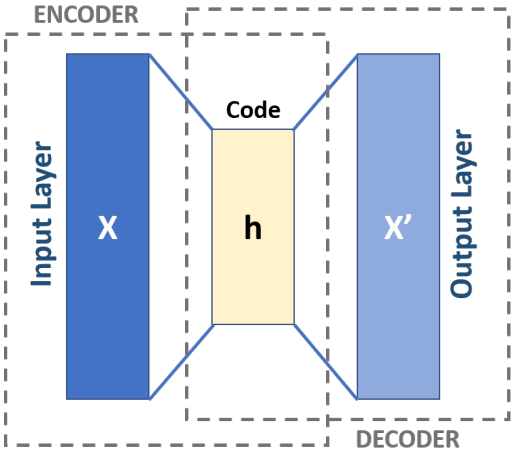

Fig. 1 Schema of an autoencoder. Image from [3]

2 Objective

We intend to use transfer learning to use the knowledge learned from one domain to detect fake news in another domain. We use the dataset described in [5]. The dataset consists of true and fake statements from two different sources - *GossipCop* and *PolitiFact*. Each data instance consists of a statement ie. text, and a label - 1, if it is fake, and 0 otherwise.

	Fake	Real
GossipCop	2230	3586
PolitiFact	270	145

The source domain we use consists of 'gossip' related to celebrities, and the target domain is related to political news. We see that the data from the *PolitiFact* domain is weakly labelled, ie. the number of samples is not sufficient for training. In such cases, transfer learning is ideal, whereby the model can be trained on data from a different domain, and then be applied to the domain corresponding to the weakly labelled data. To do this, we use the approach described in [1].

According to the authors, domain adaptation can be performed by removing domain-specific details, and then training the classifier on the de-biased data. The idea is that this will create more general classification models that would guarantee better performance. To do this, we develop a domain classifier and try to maximize the domain classification loss. This would ensure that the model does not learn any domain specific features.

In this context, we raise the following research questions:

1. *Can fake news be detected?*
2. *How does the performance of fake news detection models change when they are tested on a domain different from the one on which it was trained?*
3. *How do we characterize differences between domains, and how do we transfer knowledge learned from one domain to apply to another domain?*
4. *Does including the proposed auto-encoder in our model, help to improve classification accuracy?*

3 Methodology

3.1 Pre-processing and Feature Engineering

We perform a number of steps to clean and pre-process the text data to make it easier and more convenient to work with. We take the following steps:

3.1.1 Cleaning the text:

1. Expanding out shortened words like *I'm* into *I am*, *can't* into *cannot*, *it's* into *it is*, etc.
2. Removing special characters ie, any non-alphabet characters, like {, }, *[,], *, /* etc.

The next step is to create the vocabulary, and we do this by using the *Tokenizer* class in the *keras* text preprocessing library. This step tokenizes words and assigns indices to them. Also, all out of vocabulary words get tagged with the unknown ie. $< UNK >$ tag.

3.1.2 Lemmatization:

Lemmatization is a process in which words are converted to their base form. For example: *dancing* will be transformed to *dance*. Furthermore, lemmatization takes into consideration the context of the word. So *better* will be transformed to its lemma, *good*. This makes it easier to train models on natural text. There are several libraries and packages that are in use to perform this task - the notable ones being - WordNet Lemmatizer, Stanford CoreNLP Lemmatization, TreeTagger etc. For our implementation, we use the *WordNetLemmatizer* provided as a part of the *nltk* package in Python.

3.1.3 Stemming:

As described in the *nltk* documentation, *"Stemmers remove morphological affixes from words, leaving only the word stem."*

So, in a way similar to lemmatization, stemming reduces the word to a root, even if the root word does not exist in the language. For example: *generously* can be reduced to *gener*. There are several stemming algorithms, implementations of which are available in NLP packages. For our implementation, we use the *PorterStemmer* from the *nltk* package.

3.2 Model Architectures

We build three separate models in order to answer the research questions proposed above. These are *Domain specific classification model*, *Domain independent classification model with domain loss*, *Domain independent model with autoencoder*. We first describe model components and then explain the 3 models developed, and which components were used in which models.

3.2.1 Model components:

The first module is the *Input* module, that transforms each input sentence into a sample representation of the sentence. Each input sentence x_i is transformed into a sequence of vectors $\{v_i^j\}_{i=1}^n$. Here $v_i^j \in \mathbb{R}$, is a d-dimensional vector, where d is the number of dimensions of word embedding. n is the maximum sentence length used. We have used LSTMs in all our models for the embedding layer. An LSTM (or Long-Short Term Memory) is a type of a recurrent neural network, that has feedback connections, and can train on entire sequences of words or sentences. An LSTM unit consists of a cell, input gate, output gate and a forget gate. Advantages of LSTMs over other types of RNNs are the

fact that LSTMs can hold state over arbitrary intervals of time. LSTMs also do not suffer from the vanishing gradient problem, unlike traditional RNNs.

The second module is the *Sample Representation* which learns a representation of the sentences, given the output from the input layer. More precisely, here we do not use the encoder outputs; instead we use the states - both hidden and the cell states, as input to the decoder.

The third module is the *Output Module* which has 3 sub-modules:

Decoder Output: The objective of the decoder is to reconstruct the input sequence given the latent representation of the input, via the encoder states.

Content Classification: The objective of this module is to classify the content as *fake* or *real*.

Domain Classification: This module is trained to maximize the classification loss, ie. we do not want the model to learn domain specific features, so that this model can be applied across several domains.

3.2.2 Model 1 - Domain specific classification model:

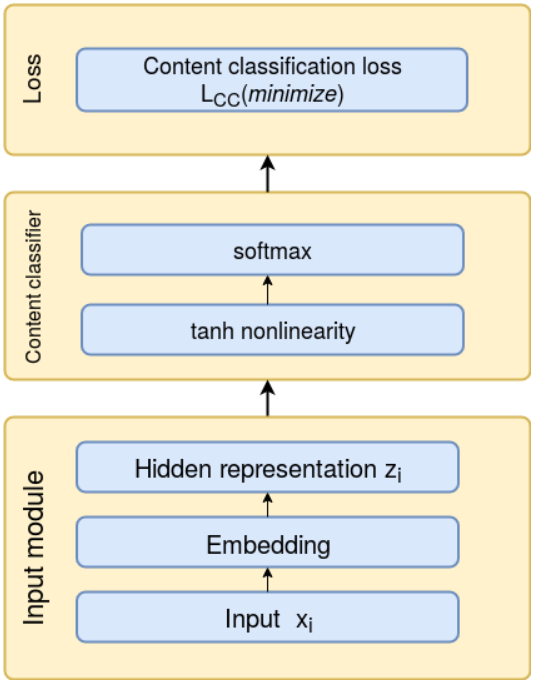

Fig. 2 Model 1 architecture

For this model we have a simple input layer, where the input sentences are embedded into hidden representations using an encoder, and then we perform content classification, in order to predict whether the given input sentence is fake or not.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_inputs (InputLayer)  (None, 100)               0
_________________________________________________________________
encoder_embedding (Embedding (None, 100, 100)          500200
_________________________________________________________________
lstm_1 (LSTM)                [(None, 64), (None, 64),  42240
_________________________________________________________________
non_linear_CC (Dense)        (None, 128)               8320
_________________________________________________________________
softmax_layer_CC (Dense)     (None, 2)                 258
=================================================================
Total params: 551,018
Trainable params: 551,018
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3 Summary for Model 1 - Domain Specific Classification

As shown in Fig 3, we have used LSTMs with 64 latent dimensions, as our encoder. Then we have a fully connected dense layer with 128 units, with a *tanh* activation. Finally to get the output labels we have an output layer with $softmax$ activation.

3.2.3 Model 2 - Domain Independent Model with Domain Loss

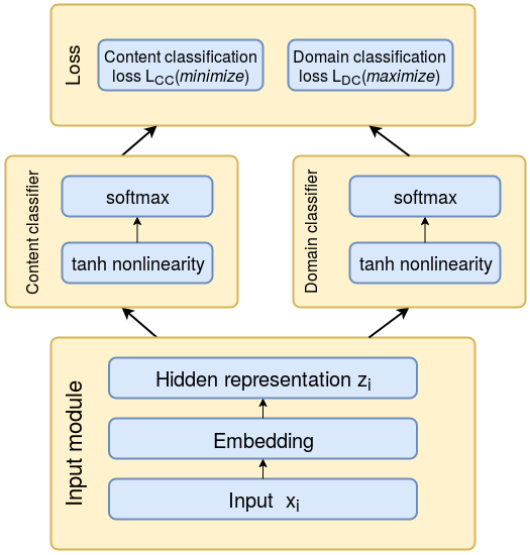

Fig. 4 Model 2 architecture

For the 2nd model, we train on both domains. Our objective is to attain satisfactory classification accuracy for both domains, and in order to do that we include a domain loss.

As shown in Fig 5, we again have LSTMs with 64 latent dimensions as the encoder. We then have two densely connected layers - one for content classification, and another for domain classification. So we have two kinds of losses - the content classification loss, that

Model: "model_2"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, 100)	0	
encoder_embedding (Embedding)	(None, 100, 100)	200200	encoder_inputs[0][0]
lstm_2 (LSTM)	[(None, 64), (None,	42240	encoder_embedding[0][0]
non_linear_CC (Dense)	(None, 128)	8320	lstm_2[0][0]
non_linear_DC (Dense)	(None, 128)	8320	lstm_2[0][0]
softmax_layer_CC (Dense)	(None, 2)	258	non_linear_CC[0][0]
softmax_layer_DC (Dense)	(None, 2)	258	non_linear_DC[0][0]

Total params: 259,596
Trainable params: 259,596
Non-trainable params: 0

Fig. 5 Model summary for Model 2

we try to minimize to get better accuracy, and the domain classification loss, which we try to *maximize*. This is because we do not want the model to learn domain specific features ie. the model should not be able to distinguish between samples from different domains. For both the losses, we have used *binary-crossentropy* as the loss function. For the output layer, we have used the $softmax$ activation here as well.

3.2.4 Model 3 - Domain Independent Model with Domain Loss and Autoencoder

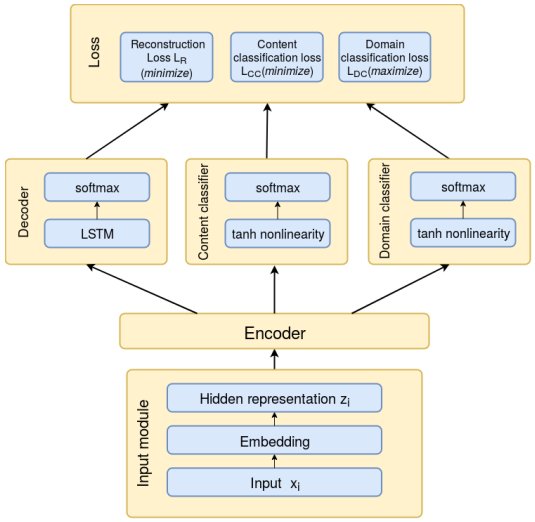

Fig. 6 Model 3 architecture

For this model, we have added an additional component to the previous model. Along with the encoder, we also have a decoder in this model. Our objective is to train the model in an end-to-end fashion on the input sentences, and we want to ensure that the learned representations of the input sequences are actually representative of the given input. To do this, we need to include the decoder in the training pipeline, to reconstruct the input sequence, and minimize a *reconstruction loss*, thereby ensuring that training the model for content classification is occurring on proper representations of the input data.

Model: "model_3"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, 100)	0	
encoder_embedding (Embedding)	(None, 100, 100)	200200	encoder_inputs[0][0]
lstm_3 (LSTM)	[(None, 64), (None,	42240	encoder_embedding[0][0]
repeat_vector_1 (RepeatVector)	(None, 100, 64)	0	lstm_3[0][0]
decoder_lstm (LSTM)	[(None, 100, 64), (N	33024	repeat_vector_1[0][0] lstm_3[0][1] lstm_3[0][2]
non_linear_CC (Dense)	(None, 128)	8320	lstm_3[0][0]
non_linear_DC (Dense)	(None, 128)	8320	lstm_3[0][0]
decoder_dense (Dense)	(None, 100, 2002)	130130	decoder_lstm[0][0]
softmax_layer_CC (Dense)	(None, 2)	258	non_linear_CC[0][0]
softmax_layer_DC (Dense)	(None, 2)	258	non_linear_DC[0][0]

Total params: 422,750
Trainable params: 422,750
Non-trainable params: 0

Fig. 7 Model summary for model 3

As shown in Fig 7, we used LSTMs with 64 latent dimensions for the encoder as well as the decoder. The fully connected dense layer used in the decoder has `vocab_length` + 1 number of units. This is because, intuitively, we can say that we have `vocab_len` + 1 number of possible labels for each word. The '+ 1' is because of the 'UNK' token, for out of dictionary words.

4 Experiments and Results

We perform 3 experiments using the 3 models developed, and evaluate the performance of each.

4.1 Experimental Setup & Environment

The deep learning library we use is *keras* with *TensorFlow* as the backend. We run our experiments on the *Google Colaboratory* platform, which is Google's free cloud service. It provides developers access to hosted runtimes, and options to accelerate execution using GPUs.

For all of our experiments, we have split our data into training, validation and test sets. 10% of the data is reserved as test data. Out of the remaining, data, 10% is validation data, and we train using the remaining. Furthermore, we specify the `random_state` to a fixed value for all 3 experiments, to avoid any biases that may creep in due to different validation sets.

To improve model performance, we use the following Callbacks:

1. *EarlyStopping:* This is a regularization method by which training is halted once the loss does not decrease significantly. This is done to prevent overfitting.

2. *ModelCheckpoint:* We use the ModelCheckpoint callback to save the best model, and the metric we monitor to check this is the validation loss.
3. *ReducedLROnPlateau:* This callback reduces the learning rate when the metric being monitored has stopped improving. This is done so that the model can reach the minima(for the loss function) and not miss it due to large step size. In our case, we monitor validation loss, and the learning rate is reduced automatically when the validation loss stops decreasing.

4.2 Experiment 1 - Domain Specific

For the first experiment, our intention is to classify content from each domain as 'real' or 'fake', and thereby attempt to answer our 1st research question. We train the model on the *GossipCop* data and we evaluate the performance for both *GossipCop* and *PolitiFact* fake news data during the testing phase.

4.2.1 Results for Experiment 1:

```
Prediction accuracy on same domain is 0.71
Precision on same domain is 0.6
Recall on same domain is 0.6
F1 on same domain is 0.6
```

	pred:Fake	pred:Real
true:Fake	128	87
true:Real	84	283

```
Prediction accuracy on different domain is 0.51
Precision on different domain is 0.71
Recall on different domain is 0.41
F1 on different domain is 0.52
```

	pred:Fake	pred:Real
true:Fake	110	160
true:Real	44	101

4.2.2 Inference

As expected, we see satisfactory performance when test data is from the same domain as the training domain, and unsatisfactory performance when test data is from a different domain. This answers our first research question - *Yes, fake news can be detected using sequence to sequence classification models, when trained properly.*

Furthermore, this experiment also directs us toward answering our second research question - *Performance of fake news detection models deteriorate when tested on a domain different from the training domain.*

4.3 Experiment 2 - Domain Independent (Without Autoencoder)

In this experiment, we test the 2nd model we developed. We use data from both the domain to train the model. We also shuffle the training data in order to remove any positional bias.

4.3.1 Results for Experiment 2:

```
Prediction accuracy on both domains is 0.68
Precision on both domains is 0.7
Recall on both domains is 0.41
F1 on both domains is 0.52
```

	pred:Fake	pred:Real
true:Fake	107	153
true:Real	46	318

4.3.2 Inference:

We see that the performance is quite satisfactory, although there has been a slight decrease in accuracy. Results from this experiment gives us hints toward answering our 3rd research question - *How do we characterize differences between domains, and how do we transfer knowledge learned from one domain to apply to another domain?* . We see that it is possible to successfully remove domain specific features, by maximizing the domain loss in the domain classifier and thereby perform transfer learning and successfully classify data from both domains, using one model.

4.4 Experiment 3 - Domain Independent with Domain Loss and Autoencoder

In this experiment, we test the 3rd model we developed. Just like the previous experiment, we use data from both domains, and shuffle the data before training the model.

4.4.1 Results for Experiment 3:

```
Prediction accuracy on both domains is 0.72
Precision on both domains is 0.66
Recall on both domains is 0.64
F1 on both domains is 0.65
```

	pred:Fake	pred:Real
true:Fake	167	95
true:Real	82	280

4.4.2 Inference:

We see there is some improvement after including the autoencoder as a part of the model. Accuracy, precision and recall have all increased from what we achieved with the previous model. This helps us to answer our 4th research question - *Does including the proposed autoencoder in our model, help to improve classification accuracy?* Yes, including the autoencoder did improve the classification accuracy.

5 Concluding Remarks

In this project, we tackled the increasingly relevant problem of fake news detection, and more precisely, cross domain fake news detection. We explore the idea of transfer learning across different domains in the context of classifictaion of fake news, and we propose some research questions to guide our exploration. In our attempt to arrive at answers to our research question, we develop 3 different models, adding more modules incrementally, and conduct experiments to test the models. We utilize advanced hyper-parameter tuning techniques and arrive at satisfactory results, and hence provide some answers to the proposed research questions.

For our future work, we intend to expand this work and experiment with different architectures and learning techniques. We also want to test out our model on datasets from other domains, evaluate the performance and get clues on how to create a more general fake news classification model.

References

1. Azarbonyad, H., Sim, R., White, R.W.: Domain adaptation for commitment detection in email. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 672–680. ACM (2019)
2. Conroy, N.J., Rubin, V.L., Chen, Y.: Automatic deception detection: Methods for finding fake news. Proceedings of the Association for Information Science and Technology **52**(1), 1–4 (2015)
3. Massi(CC), B.M.: Autoencoder schema. https://commons.wikimedia.org/w/index.php?curid=80177333. Accessed 30th November, 2019
4. Ruchansky, N., Seo, S., Liu, Y.: Csi: A hybrid deep model for fake news detection. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 797–806. ACM (2017)
5. Shu, K., Mahudeswaran, D., Wang, S., Lee, D., Liu, H.: Fakenewsnet: A data repository with news content, social context and dynamic information for studying fake news on social media. arXiv preprint arXiv:1809.01286 (2018)