Is this enough sarcasm? Investigating the performance of a pre-trained BERT model on a sarcasm detection task as the amount of data used to fine-tune the model is reduced

Joe Farrington
Dept. of Computer Science
University College London
ucabjmf@ucl.ac.uk

Jeremy Dang
Dept. of Computer Science
University College London
ucabjd5@ucl.ac.uk

Benjamin Klasmer
Dept. of Computer Science
University College London
zcapbak@ucl.ac.uk

Abstract

One of the most difficult and expensive parts of the machine learning process can be obtaining good quality labelled data. This is especially true when the task is difficult for humans to perform, such as sarcasm detection.

Research in both computer vision and NLP has shown that in some circumstances fine-tuning a pre-trained model can achieve the same performance as a model trained from scratch, but with far fewer labelled training examples.

We apply the recently released pre-trained model, $\mathrm{BERT}_{\mathrm{BASE}}$ to the problem of sarcasm detection. We achieve an overall test set accuracy score of 73.7% on the the SARC 2.0 /pol balanced dataset, and 81.4% accuracy when predicting which of two paired responses in the test set is sarcastic, a state-of-the-art result when compared to other models that also do not incorporate representations of person who wrote the remark.

We also compare how the performance of the pre-trained $\mathrm{BERT}_{\mathrm{BASE}}$ model changes as we reduce the number of training examples used to fine-tune the model with the performance of three baseline models: logistic regression classifiers trained using a Bag-of-Words, sentence embeddings based on GloVe and sentence embeddings based on ELMo. We observe that $\mathrm{BERT}_{\mathrm{BASE}}$ is capable of performing better than the baseline models when fine-tuned with only a quarter as much training data.

1 Introduction

1.1 Context and motivation

Sarcasm is defined as a remark that means the opposite of what is said, generally used to mock or convey contempt. It is not always easy to know if someone is being sarcastic, and this is especially the case when the remark is made in writing rather than verbally. When delivered in person, sarcasm is easier to pick up on since there are more cues, including facial expression and tone. In conversation with people they know, humans will also rely on their knowledge of that person. Without this additional information, separate from the words themselves, the task can be very difficult.

A major challenge in sarcasm detection has been compiling training datasets, because of the difficulty humans have in performing the task accurately. Recently, taking advantage of self-annotations through the use of hashtags or other indicators on platforms such as Twitter and Reddit has led to the production of large datasets without the need for individual hand labelling. However, these self-annotations are not available for many types of data for which a person or company may wish to detect sarcasm. Moreover, the difficulty of getting labelled data is common to many problems in NLP: it can be expensive and time consuming, while using services such as Amazon's Mechanical Turk can lead to biases when participants adopt simple heuristics to perform the task more quickly.

Until recently models for sarcasm detection, in common with models for many other NLP tasks, have been built from scratch using hand-engineered features. An emerging trend in NLP, following the success of the method in computer vision applications, is the use of transfer learning. The key to this approach is that lots of the information a model needs to learn for different tasks is the same. In a computer vision model, for example, the early layers of a model learns about edges and basic shapes, knowledge of which will be required for most vision tasks. For NLP, it will be helpful for most tasks to understand what different words mean, how they relate to one another, and have an understanding of grammar. Models can be pre-trained by institutions with significant resources on very large datasets to learn these common features, and then fine-tuned with much less data, using much less time and computation power, for specific tasks.

1.2 Research question and hypothesis

In this paper we investigate how a fine-tuned, pre-trained $\text{BERT}_{\text{BASE}}$ model ("the BERT model") performs on sarcasm detection, and how reducing the number of training examples used to fine-tune the BERT model affects its test set performance compared to models with less (or no) information from pre-training: logistic regression classifiers applied to Bag-of-Words features, sentence embeddings based on GloVe word embeddings and sentence embeddings generated by a pre-trained ELMo model. This question is important because establishing an adequate amount of training data required to fine-tune a pre-trained model may prevent wasted effort labelling additional data that makes little difference to model quality.

2 Background and related work

2.1 The SARC dataset

Joshi et al. (2017) reviewed the field of sarcasm detection and found researchers using a wide range of different datasets, drawn from a range of different sources and annotated in different ways. This made comparing the relative merits of the different approaches difficult.

One development since that review that has made it easier to compare different models is the release, and use by other reseachers, of the Self-Annotated Reddit Corpus (SARC) dataset (Khodak et al., 2017). SARC is a corpus of 533-million statements (including 1.53-million sarcastic statements) scraped from the social media platform Reddit. The dataset has been labelled through self-annotation, based on a convention adopted by users to add '/s' to the end of sarcastic posts. The dataset is structured such that there are ancestor posts, and responses to these posts which are classified as sarcastic or not. An example of this format is:

Ancestor: "*Jindal being super smug about volcano monitoring*"

Response labelled as sarcastic: "*Who ever heard of a volcano causing problems for people?*"

Response labelled as not sarcastic: "*you'd think someone from Louisiana, of all places, would understand the importance of early warnings for natural disasters*"

In this example it is relatively easy for a human to identify the sarcastic comment, even without the context provided by the ancestor, but this is not always the case.

The SARC dataset provides two different tasks. In the 'unbalanced task, there are multiple responses to an ancestor post and the classifier must determine which are sarcastic, with non-sarcastic posts significantly outnumbering sarcastic posts. In the 'balanced task, the classifier must determine which of two responses to an ancestor post is sarcastic. Training and test set data are available to perform these tasks over multiple subreddits (forums), or for a single politics based subreddit: /pol.

2.2 Recent results using the SARC dataset

Recently, in common with other areas of NLP, deep learning methods have been applied to the problem of sarcasm detection. In 2018, two papers claimed state of the art results on sarcasm detection using deep learning, and the SARC dataset as a benchmark: ContextuAl SarCasm DEtector (CASCADE) (Hazarika et al., 2018) and ELMo4Irony (Ilić et al., 2018).

CASCADE uses a combination of both content of the comments and context about the user and discussion. For a user that posts a comment, the contextual side comprises of a user embedding model that looks at the historical posts of the user to learn about their personality and writing style. This is combined with a document model that looks at the discussion forum where the comment of interest was posted to extract the background, contextual and topical information. To model the content, a convolutional neural network (CNN) is used to extract syntactic features. All three of these outputs are then concatenated together and used for classification. Hazarika et al. (2018) also analyse the importance of each of feature and show that the inclusion of user embeddings boosts the accuracy performance of CASCADE by 8-12%. Although user embeddings have been shown here to be vital part of the model, there are many circumstances when the data is not readily available.

ELMo4Irony (Ilić et al. (2018)) is based on Embeddings from Language Model or ELMo for short (Peters et al. (2018)). The authors suggest that character level cues such as words written in all caps, quotation marks and emojis which may be used to infer sarcasm. They build word representations from a character level using ELMo. These embeddings are then passed onto a BiLSTM layer with 2,048 hidden units. The hidden units of

Method	all-bal	pol-bal	pol-unbal
Bag-of-Words (a)	73.2	75.9	27.0
Bag-of-Bigrams (a)	75.8	76.5	24.9
Sentence Embedding (a)	71.0	76.0	26.7
CASCADE[†](b)	77.0	74.0	-
ELMo4Irony (c)	77.3	78.5	-
Human (Majority) (a)	92.0	85.0	-

Table 1: Comparison of the accuracy performance of different models tested on the SARC dataset. Results marked (a) are from Khodak et al. (2017), (b) are from Hazarika et al. (2018) and (c) are from (Ilić et al., 2018).
[†] Result may not be directly comparable to the others, a different evaluation method appears to have been used compared to Khodak et al. (2017).

the LSTMs are aggregrated together using max-pooling, which the authors claim improves the performance. The resulting vector is then passed through a 2-layer dense feed forward network that contains 512 hidden units each. Finally, the output is classified using a final dense layer that performs binary classification.

In Table 1 we present a set of the results on the SARC dataset from Khodak et al. (2017), Hazarika et al. (2018) and Ilić et al. (2018). We note that the figures presented in Hazarika et al. (2018) do not appear to be directly comparable to those in the other two papers because it appears they report plain test set accuracy using the balanced datasets rather than the performance on the balanced task as described in Khodak et al. (2017). This is implied by the comparatively poor result reported by Hazarika et al. (2018) for their Bag-of-Words benchmark, and the fact that their reported F1 scores are not the same as their accuracy scores.[1]

2.3 Transfer learning and fine-tuning

One of the most important trends in computer vision models over the last five years has been the increasingly widespread use of transfer learning, specifically taking large networks pre-trained on massive datasets and fine-tuning them for different tasks. For example Zeiler and Fergus (2014) beat the previous best method on Caltech-256 (using 60 training images per class) was beaten by a CNN (pre-trained on the Imagenet 2012dataset of 1.3M images) using only 6 images per class.

Until relatively recently, similar approaches were not widely adopted in NLP. Embeddings, such as Word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) are pre-trained, dense feature representations for individual words or tokens which have largely replaced the traditional representation of words/tokens with one-hot vectors. ELMo (Peters et al., 2018) can be used to generate embeddings for words or sentences based on their context. However, in contrast to the computer vision examples above, embeddings effectively only provide a pre-trained first layer for the model and the remaining layers must be trained from scratch.

One successful implementation of transfer learning of a full model on NLP tasks was Universal Language Model Fine-tuning (ULMFiT) (Howard and Ruder, 2018). A very interesting observation from ULMFiT, which leads into our work, is that on the IMDb sentiment analysis dataset (Maas et al., 2011) and the AG news (Zhang et al., 2015) topic classification dataset, the same performance was observed when using 100 data points with the pre-trained language model as was observed when training the model from scratch with 10x (for the IMDb dataset) and 20x (for the AG news dataset) the number of training examples. This suggests that the process of fine-tuning language models can be very effective at getting strong results even in circumstances where obtaining labelled data can be difficult (such as for sarcasm detection).

More recently, the Google AI language team claimed state-of-the-art results on 11 NLP datasets, for tasks including text classification, but also question answering and named-entity recognition using Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018). BERT uses the Transformer architecture (Vaswani et al., 2017) to encode sequences, rather than the LSTM layers used of ULMFiT. BERT differs from other language models in that it was trained bi-directionally on a task which predicts only masked words in a sentence rather than the next (or, if running backwards, previous) word. The BERT models were pre-trained on a much larger corpus than ULMFiT and the suggested procedure for fine-tuning a BERT model does not include a separate step for fine-tuning the language model before fine-tuning the whole model (including the new task-specific layer).

There is no discussion in the paper comparing

[1] We would expect the F1 score to be the same as the accuracy for the balanced task because every false negative will have a corresponding false positive.

performance on subsets of the training data, but the authors note that fine-tuning $\text{BERT}_{\text{LARGE}}$, the larger of the two pre-trained versions of BERT available and the one that was used to achieve the headline state-of-the-art results, with smaller datasets was sometimes unstable. The implementation guidance on GitHub [2] that accompanies the pretrained models also states that high variance (of several percentage points) has been observed with small datasets (e.g. MRPC in the GLUE benchmark, which has 3.7k training examples).

We have not been able to identify any papers that have used either ULMFiT or BERT for sarcasm detection, or any papers that investigate the effects of training data size on the performance achieved using BERT, and therefore we intend to investigate how the performance of a fine-tuned BERT model compares to model specifically designed to solve the problem of sarcasm detection, and how that performance changes (compared to models with less or no pre-training) when the number of training examples is reduced.

3 Methods and Experiments

3.1 Data

We use the balanced version of the /pol subset of the SARC 2.0 dataset[3], which was introduced by Khodak et al. (2017) so that we can have a consistent approach and can compare results to existing research.

The balanced version of the dataset consists of 'ancestor' comments and a pair of responses for each: one labelled as sarcastic and one labelled as not-sarcastic. The training set consists of 6,834 ancestor comments, while the test set consists of 1,703 ancestor comments, with each set having a pair of responses for every ancestor comment.

We created a validation set in order to perform hyperparameter tuning on the BERT model by randomly selecting 20% of the ancestors (and their corresponding responses) from the training set. We refer to the remaining 80% of the training set as the 'project training set' to distinguish it from the whole SARC 2.0 /pol balanced training set from which it has been sampled.

In order to assess the impact of reducing the number of training examples, we split the project training set down further, randomly sampling (without replacement) subsets consisting of 50%, 25%, 12.5% and 6.25% of the ancestors in the project training set and their corresponding responses.

Project training set	Number of responses
100%	10,933
50%	5,465
25%	2,731
12.5%	1,365
6.25%	681

Table 2: Number of responses in each of the samples of the project training set. In each case, 50% of the responses are labelled sarcastic and 50% are labelled as not sarcastic.

We split using ancestors to ensure that the validation set looked similar to the test set and each set either included both of the relevant responses or neither. We only used the responses when training each model. Each response was included as a separate training examples with a label of 1 for sarcastic and 0 for not-sarcastic.

3.2 Repetition

As explained above in Section 2.3, Devlin et al. (2018) and the implementation guidance on GitHub suggests that BERT can be unstable with small datasets. In order to investigate the impact of this on our results, we have trained multiple models at each stage of the process. The basic principle is that by averaging the results over multiple runs, we should increase the chance of picking a stable combination of hyperparameters. Similarly, at test time, it is interesting to assess how different the outcomes are when training with the same training data and the same hyperparameters. In the context of determining whether a sample size is adequate, it is important to know what the impact of that reduction is on the consistency of the results achieved.

This is discussed in more detail below, for each individual model.

We chose to use one version of each size training set (rather than drawing a different sample for each repeat) in order to avoid conflating different sources of variance. The composition, as well as the number, of examples in the training set is likely to have some affect on the performance of the models. If we had used different samples for each repetition, this would likely have led to an increased variance for the smaller datasets than the higher datasets (because the 100% project training

[2] https://github.com/google-research/bert

[3] http://nlp.cs.princeton.edu/SARC/2.0/

sets would contain much more overlap each time than the 6.25% project training sets).

3.3 BERT

We used the PyTorch (Paszke et al., 2017) implementation of $\text{BERT}_{\text{BASE}}$, converted from the original TensorFlow implementation by HuggingFace Inc[4]. $\text{BERT}_{\text{BASE}}$ is the smaller of the two models described in Devlin et al. (2018) and consists of 12 stacked Transformer blocks, with a hidden-unit size of 768 and 12 self attention heads.

We followed the implementation guidance in (Trivedi, 2019), an article about using this implementation of BERT for multi-class classification, and part of our work is based on the associated code. Specifically, we used the model class "BertForSequenceClassification" which is set up for fine-tuning on classification tasks using the cross-entropy loss. This adds a linear layer on top of the pre-trained model, mapping from the 768 dimensions of the hidden state to the number of labels. The weight matrix of the linear layer is initialized randomly from a normal distribution of 0 and standard deviation 0.02, while the bias is initialized to 0.

We performed tokenization using the PyTorch implementation of the BertTokenizer, which uses Wordpieces (Wu et al., 2016) to encode words outside the 30,000 word vocabulary using known word components. We convert all of the characters to lower case based on our initial experiments which found no significant differences in the use of capitalisation in sarcastic and non-sarcastic responses and slightly improved validation set performance when using lower case tokens. We used a maximum sequence length of 50 tokens because fewer than 1% of the responses in the whole /pol training set had a length greater than 50 when processed with the BERT tokenizer. We judged that this was a reasonable cut-off, a balance between losing too much information and wasting too much computation time processing padding.

In Table 3 we show the suggested range of hyperparmaters for fine-tuning suggested by Devlin et al. (2018). For each of the five subsets of the project training set (100%, 50%, 25%, 12.5% and 6.25%), we performed five runs with each of the six combinations of batch size and learning rate in Table 3 for five epochs using the BERTAdam optimizer. We calculated the mean accuracy on the validation set after each epoch over the five runs, and selected the combination of learning rate, batch size, and number of epochs that achieved the highest mean validation set accuracy for each training set size.[5]

Hyperparameter	Suggested range
Batch Size	16 or 32
Learning Rate	5e-5, 3e-5 or 2e-5
Number of Epochs	3 or 4

Table 3: Suggested hyperparameters for fine-tuning the pre-trained BERT language model from Devlin et al. (2018).

For each of the five subsets of the project training set we trained five models using the best hyperparameters identified using the validation set and calculated our three evaluation metrics on each model using the test set.

3.4 Baseline models: Bag-of-Words and Sentence Embeddings

We used two of the baseline models included in Khodak et al. (2017) as comparison points for our results with training sets of different sizes. We used the code shared in support of the paper, which is available on GitHub[6], which uses scikit-learn (Pedregosa et al., 2011) and NLTK (Bird et al., 2009).

The responses were tokenized by splitting on punctuation. Following the provided code we did not lower case the tokens, apply a token limit and any restrictions (size, frequency) for including words in the vocabulary.

For the Bag-of-Words model, the responses are converted to a sparse vector: each row corresponds to a word in the training set vocabulary and the entry in each row is number of times that word appears in the response.

For the Sentence Embedding model, the responses are converted into a dense, 1600 dimensional vector which is the element-wise sum (over each word in the response) of the GloVe representations pre-trained on the Amazon product corpus.

In each case, a simple logistic regression classifier was trained on the feature representations using the cross-entropy loss using the sci-kit learn

[4]https://github.com/huggingface/pytorch-pretrained-BERT

[5]See the supplied source code for the best hyperparameters we identified for each training set size

[6]https://github.com/NLPrinceton/SARC

(Pedregosa et al., 2011) LogisticRegressionCV function. Two-fold cross-validation was used to tune the inverse regularisation strengths over a range 10^{-2} to 10^{3} (in multiples of 10), and a final classifier is returned by LogisticRegressionCV based on the level of inverse regularisation strength that achieves the best average cross-validation error over the two folds. As with BERT, we trained five models with each of the training sets using five different random states - but each repeat produced the same results on the test set.

3.5 Using sentence embeddings from ELMo

We initially attempted to apply a small multi-layer perceptron (with a 256-unit hidden layer, and a 2 node output layer) to ELMo sentence embeddings generated by the pre-trained model available on TensorFlow Hub[7]. However, we had two problems with this method: generating the ELMo embeddings made each epoch slow, and we were unable to find a set of hyperparameters that led to a model that could perform competitively with the Bag-of-Words baseline.

As a risk mitigation step we therefore decided to use the PyTorch ElmoEmbedder[8] to generate ELMo embeddings once for each model, store them as an array, and then apply the same logistic regression classifier from Khodak et al. (2017) that was used with the Bag-of-Words and GloVe features for consistency and speed of training.

We followed the method described above for the baseline models above, with some differences to account with the different method of encoding. We set a maximum sequence length of 50 tokens (which, as we explain above in Section 3.3, will affect fewer than 1% of the training examples) to reduce the amount of padding required when generating the ELMo embeddings. The ELMo embeddings are returned as a vector with dimensions ($3\times$ sequence length $\times 1024$): we performed mean pooling over the different layers (3) and the tokens to obtain a single 1024-dimensional dense vector for each response. Unlike GloVe word embeddings or the Bag-of-Words features, ELMo embeddings are not always the same for the same sentence. We generated one set of embeddings for the test set which were used for each training set size. For each run of the model on the different training set sizes we created new ELMo embeddings so that our results would reflect this potential source of variance.

[7] https://tfhub.dev/google/elmo/2

[8] https://github.com/allenai/allennlp/blob/master/allennlp/commands/elmo.py

3.6 Metrics

For each of the models, in order to compare our results with Khodak et al. (2017) and Ilić et al. (2018) we calculate the accuracy on the balanced task of predicting which of each pair of responses is sarcastic. In addition, in order to compare our results with Hazarika et al. (2018) who, as we explain above, appear not to have used the same evaluation task, we will also calculate the accuracy and F1 score on the test set without considering which responses are paired.

To calculate the accuracy on the balanced task, in line with the description in Khodak et al. (2017), we consider the pair of responses for each ancestor in the test set in turn. The response assigned the highest probability by the model of being sarcastic was labelled as such, and the other response is labelled as not sarcastic. For BERT, the logits output by the model are not normalized, and therefore we apply the softmax function to the logits before determining which response has a higher probability of being sarcastic.

To calculate the accuracy and F1 score on the test set to compare with Hazarika et al. (2018), we assign each response the label for which it has the highest probability, without considering which responses were paired.

For the BERT model, the reported metrics for each size test set are the arithmetic mean over the five models trained with the best hyperparameters for that training set. We also calculate the standard deviation for each metric over the five repeats.

For the two Sentence Embedding models and Bag-of-Words model the results were consistent between repeats with the same training set size and therefore the mean over the five models does not differ from the result from a single model and the standard deviations were zero.

4 Results and discussion

4.1 BERT compared to existing models

In Table 4, we present a comparison between previous models, the mean test set performance our our fine-tuned BERT model trained on the 100% project training set, and the majority-vote human performance reported by Khodak et al. (2017).

The BERT model outperforms all of the models that reported a score on the balanced task. Ad-

Method	Balanced task	Accuracy	F1 Score
Bag-of-Words (a)	75.9	-	-
Bag-of-Bigrams (a)	76.5	-	-
Sentence Embedding (a)	76.0	-	-
ELMo4Irony (c)	78.5	-	-
CASCADE - no personality features (b)	-	68.0	70.0
BERT (this paper)	**81.4**	**73.7**	**73.4**
CASCADE - with personality features (b)	-	74.0	75.0
Human (Majority) (a)	85.0	-	-

Table 4: Comparison of the performance of different models on the balanced subset of the SARC 2.0 /pol dataset. Results marked (a) are from Khodak et al. (2017), (b) are from Hazarika et al. (2018) and (c) are from (Ilić et al., 2018).

ditionally, the model outperforms the version of CASCADE that did not include personality feature about the individual users. The results in Hazarika et al. (2018) are only presented to two significant figures so our model appears to be competitive in overall accuracy with CASCADE when the personality features are included, but has a worse F1 score. When not considering paired responses, the BERT model makes more false negative errors (classifying a sarcastic response as not-sarcastic) than false positive errors.

These results demonstrate the power of fine-tuning a pre-trained language model: the performance of the BERT model using just the responses is less than one percentage point worse than the performance of CASCADE, which has been specifically created for the task and includes additional information about the discussion and the history of the users. We also note that the CASACDE model was trained on more data (approximately 1.3k additional training examples), because Hazarika et al. (2018) only held back 10% of the training data as a validation set compared to our 20%.

The human majority still significantly outperform the BERT model, but had the advantage of reviewing the whole context of the message when classifying the remarks.

4.2 The effect of reducing the amount of training data

In Figures 1, 2 and 3 we plot how the mean model performance on the test set measured by accuracy on the balanced task, the plain accuracy, and the F1 score changed as we reduced the training set size. These results are the mean metrics over five models trained with the same size training set. We plot the standard deviation of the results for BERT as error bars; the other models returned consistent results and therefore the standard deviation over the metrics was zero.

BERT consistently achieves the best performance across the metrics and training set sizes. As we discuss in Section 2.3, (Devlin et al., 2018) suggest training multiple models when using $\text{BERT}_{\text{LARGE}}$ with small datasets because fine-tuning with small dataset sometimes led to poor results. The large standard deviation on our BERT model metrics using the 25% training

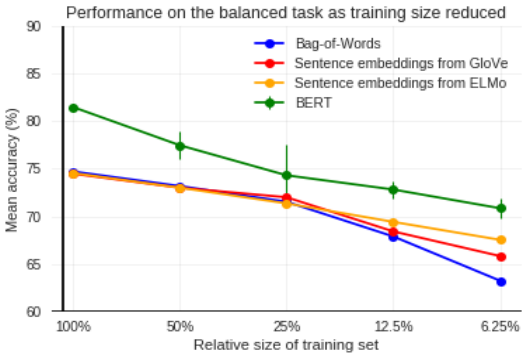

Figure 1: Balanced task accuracy on the test set as the training set size is reduced

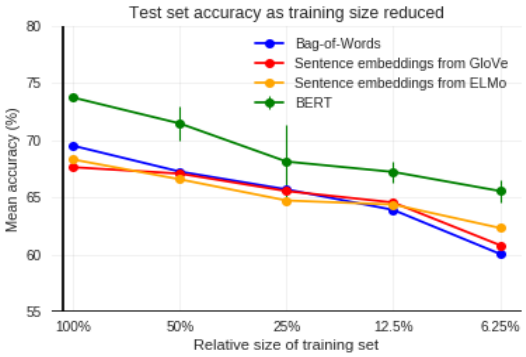

Figure 2: Accuracy on the test set as the training set size is reduced

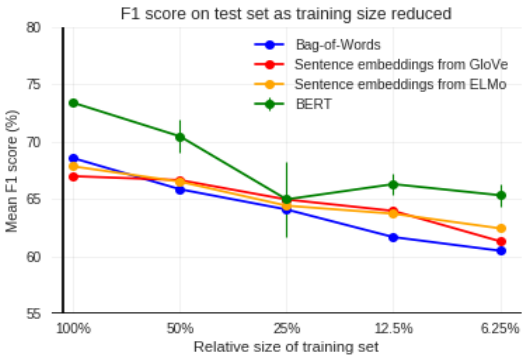

Figure 3: F1 score on the test set as the training set size is reduced

set size suggests $\text{BERT}_{\text{BASE}}$ can suffer from the same issue. Four of the models achieved a balanced task accuracy of 75-76% on the test set, but one achieved only 67%: worse than the performance of any of the baseline models with the same amount of training data. This poor performance was also reflected in the validation set performance of this model, and this suggests that even if using $\text{BERT}_{\text{BASE}}$ on a small training set it is advisable to train multiple models and compare the validation set performance rather than training a single model and being unable to tell whether it is underperforming. This contrasts with the consistency observed in the logistic regression models. The key sources of variance during training of BERT are the random initialization of the classification layer, shuffling of the training set and the dropout layers.

BERT is capable of outperforming all of the baseline models when trained with only 25% of the data. This is a significant advantage and would represent a large cost saving if humans were being used to label the data and performance at the level of the baseline models was adequate for the task.

While our BERT model outperforms ELMO4Irony when trained with the whole project training set (equivalent in size to the training set used by Ilić et al. (2018): 10.9k responses), this advantage is lost when BERT is trained on the 50% project training set. Our BERT model continues to outperform CASCADE without personality embeddings when it is trained with the 25% project training set (2.7k responses), which is equivalent to 22% of the data used to train CASCADE by Hazarika et al. (2018).

The three baseline models perform very similarly on the full project training set. Their performance starts to diverge on training set sizes below 25%. On the 6.25% training set, the the sentence embedding model using ELMo outperforms that using GloVe, which in turn outperforms that using a Bag-of-Words. The likely explanation for this is that as the training set gets smaller, the pre-trained knowledge about the similarity between different words means that the model can transfer its knowledge from an example in the training set to examples in the test set with synonyms, while a Bag-of-Words model would not be able to make those connections.

Interestingly, the additional training data used for fine-tuning in the 50% and 100% training set sizes has a much more positive effect on the performance of the BERT models than the baseline models. This is potentially counter-intuitive: we might expect the models with less pre-trained information to benefit more from the increased number of training examples. This appears to be the case up to the 25% project training set, and then the pattern reverses. One potential explanation that we have investigated is that the increased data leads to less overfitting by the BERT model, but we have not identified a consistent pattern in the differences in the training set performance between BERT models fine-tuned with the 100%, 50% and 25% project training sets.

5 Conclusions and further work

We have demonstrated the effectiveness of fine-tuning a pre-trained general model on sarcasm detection. A pre-trained $\text{BERT}_{\text{BASE}}$ model fine-tuned using only the response comments achieves similar test set performance to CASCADE - a model designed specifically for sarcasm detection and incorporating additional features including information about the context of the discussion and the personality of users.

We have also shown that BERT is capable out outperforming baseline Bag-of-Words and sentence embedding models when trained with only 25% of the data.

There are several potential avenues for extending this work. Firstly, we could investigate the effect of increasing the maximum sample size (beyond our 100% project training set size) using the main balanced set of the SARC dataset to establish the point at which more training data used for fine-tuning does not improve the BERT model's test set performance. Secondly, we could investigate the effect of variation in the content of the training sets by drawing different sized samples from the much larger, main SARC dataset while using models with the same initialized weights on each repeat. Thirdly, we could investigate the relative impact of the different sources of variance in training BERT: the initialization of the weights of the classification layer, the shuffling of the training set and dropout. And, finally, we could perform a similar analysis on other tasks in NLP where getting good quality labelled examples is challenging.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Devamanyu Hazarika, Soujanya Poria, Sruthi Gorantla, Erik Cambria, Roger Zimmermann, and Rada Mihalcea. 2018. CASCADE: contextual sarcasm detection in online discussion forums. *CoRR*, abs/1805.06413.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Suzana Ilić, Edison Marrese-Taylor, Jorge Balazs, and Yutaka Matsuo. 2018. Deep contextualized word representations for detecting sarcasm and irony. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 2–7. Association for Computational Linguistics.

Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. 2017. Automatic sarcasm detection: A survey. *ACM Comput. Surv.*, 50(5):73:1–73:22.

Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. 2017. A large self-annotated corpus for sarcasm. *CoRR*, abs/1704.05579.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237.

Kaushal Trivedi. 2019. Multi-label text classification using BERT The Mighty Transformer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 649–657, Cambridge, MA, USA. MIT Press.