Checkable Claim Detection at the Edge with CNN and Bert Models

James Byrne (byrnej@berkeley.edu)

Abstract

This paper presents models for detecting checkable claims in sentences using selected Neural Network architectures for NLP, with an aim to produce a reasonably accurate but computationally light-weight model for use at the edge. Mobile devices and browsers can then look for checkable claims and hand off to cloud-based knowledge-retrieval services. It compares CNN classification approaches with more computationally heavy transformers architectures (specifically Bert Base and DistilBERT) and evaluates them based on both computational costs and accuracy.

Introduction

There are multiple projects and organizations currently working on fact-checking approaches[1] to tackle the huge volume of "fake news" that has moved from the world of fringe conspiracy theory to mainstream media in the past 6 years[2] Projects such as PolitiFact, Snopes and *The Washington Post*'s Fact Checker have had some success, but all are based on significant manual research efforts.

Manual research is often essential for complex fact-checking, but it is impractical to apply to the millions of statements made in various forums each day. To tackle this, there have been multiple approaches tried to automate fact checking over the years, some of which have attempted to fact-check sentences directly and others that have taken a more modular approach involving separate knowledge bases. The latter includes the ClaimBuster project[3] which now offers live, on-demand checking of inputted text. The original 2017 ClaimBuster paper from *Hassan et.al* [1] inspired this project, and I will be investigating claim detection (ClaimBuster refers to this as "claim spotting"), the automated classification of sentences that contain *checkable* claims of fact. It would exclude non-claim sentences (e.g. "Where is the salt?", "Will Susan support the initiative?") and subjective or ephemeral statements that cannot be independently verified using objective evidence (e.g. "John likes fish." or "Serenity is all.").

Core to this paper is the aim to move claim detection to the edge. Instead of attempting to attempting to check on creation; i.e. to capture, claim-detect and fact-check the 3+ exabytes of information generated globally every day, I examine the feasibility of a check-on-read approach where edge devices (mobiles and browsers) identify a checkable claim only when presented to the user that can be verified on the cloud. For this, the memory and compute resources needed for claim detection are at least as important as the accuracy of the model.

Background

At the time of publication of the *Hassan et al 2017* paper, several major advances in technique, including transformers were not available or were not mature enough for practical use. In the years since, the ClaimBuster project has released SVM, BiLSTM and Adversarial Transformer approaches for their "claim spotter" module, and others have expanded on their work including *Hansen et al* in 2019[2], and *Meng et al* in 2020[3] both focusing on the use of transformer models for detection.

While these and other papers show success with large models, pushing F1- scores into the low-90% range, their size and computational demands make claim detection not tractable on edge devices.

[1] An excellent summary of projects can be found here: https://www.dw.com/en/fact-checking-a-curated-guide-to-resources-and-ideas/a-54509776
[2] See: https://www.economist.com/finance-and-economics/2018/04/05/fake-news-flourishes-when-partisan-audiences-crave-it
[3] http://Claimbuster.org

Method & Approach

The ClaimBuster Datasets and Data Augmentation

From the data sets made available by the ClaimBuster project, I used one (3xNCS.json) created for *Meng et al 2020* [3] for training and assessment, and a larger one (crowdsourced.csv) from *Benchmark Dataset of Check-worthy Factual Claims, Arslan et* al [4] for performance testing. Both contain labeled sentences from Presidential debate transcripts from the 1960's through to 2016.

The training dataset was curated with a 1:3 ratio of checkable claims to non-checkable statements, and in initial analysis and CNN attempts, the imbalance resulted in lower-than-desired validation accuracy.

Investigating the available data augmentation techniques for NLP to address this raised some questions. There are clear challenges with some of the techniques such as synonym substitution. For example, "cheap" and "coach" are both synonyms for "economy" in shopping and travel contexts respectively. "The coach is growing" and "the cheap is growing" no longer deserve the "checkable" label of "the economy is growing" and would contaminate the training data. Considering this, my first approach was to simply add additional copies of the positive samples to the training dataset to create a near 1:1 ratio. This was surprisingly successful, pushing validation accuracy above 90% even before tuning, so it was not necessary to attempt more complex augmentations.

Models and Hyperparameter Optimization

For CNN models, individual epochs during training were training in 2.5 seconds or less, so it was practical to use a grid-search methodology, varying the number, sizes and counts of the convolutional filters as well as the dropout factors and the size and depth of the dense layer(s) used to complete the classification.

For the Bert and DistilBERT models, the libraries provided out of the box models for sequence classification which added single dense layers to classify the input sentences. The Bert models epochs were taking in excess of two minutes each, so even though I had fewer hyperparameters to tune (essentially only learning rate and epsilon) a grid optimization approach would have required excessive run time, so I turned to the Keras tuner module with both random and Baysean search approaches.

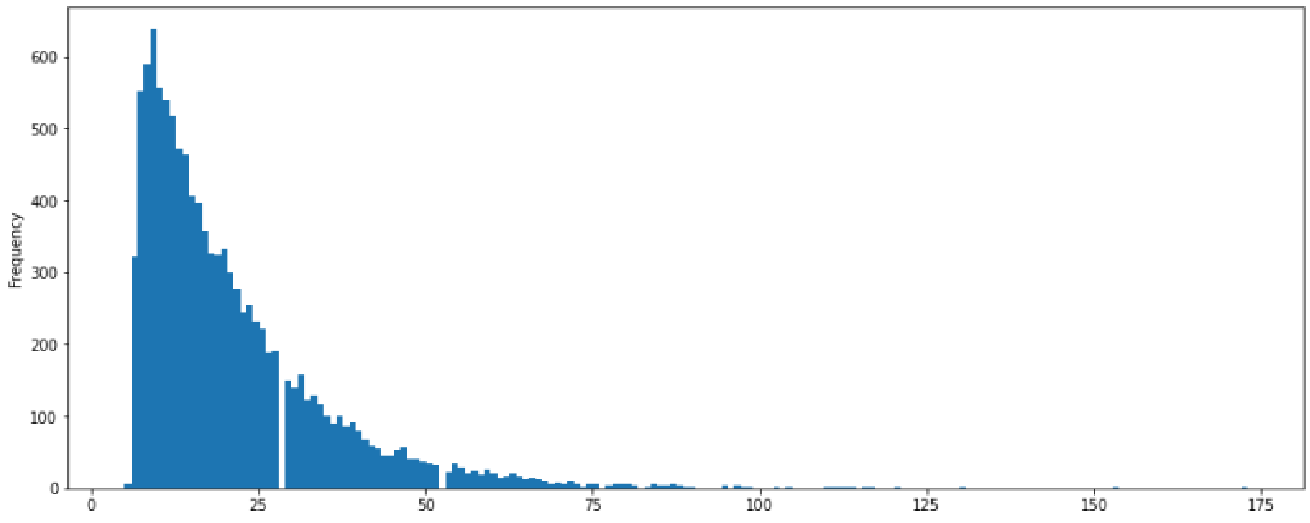

Figure 1 – Frequency vs data set token sequence lengths

Bert vs CNN for Edge Claim Detection

Even once tuned, the Bert and DistilBert models were not providing accuracy comparable with that of the CNN models. It is possible that we had insufficient data to effectively tune the Bert parameters, but in most of the tuning runs, I was seeing validation loss reach a minimum after a relatively low number of epochs and then start to grow again, suggesting that this was not the case. The following is a comparison of accuracy in the best-performing models of each architecture after hyperparameter tuning. All the neural models use a maximum length of 100 tokens. A simple NLTK-based bag-of-words (BOW) classification model is included as a baseline.

Model Architecture	Filters (#filters x #tokens)	Dense Dimension(s)	Model Parameters (in 1000s)	Validation Accuracy
BOW (Baseline)	-	-	-	0.69801
CNN	64x4, 64x8, 64x16	Single Layer of 8 nodes	501	**0.97559**
Bert	-	-	109,483	0.93273
DistilBERT	-	-	66,955	0.92722

Table 1 – Comparison of tuned CNN, Bert and DistilBERT to BOW baseline

Not only does the CNN model outperform both of the Bert models, it does so by over four percent with a model two or three orders of magnitude simpler than the transformers. This disqualified the Bert approach as a viable candidate for detection at the edge. There has been significant advances in the ability of transformers in the claim checking space – e.g. in *Hansen et al 2019* – but only at the expense of still larger models that require significant compute resources.

Based on this, the subsequent investigation of CNNs was focused on optimizing model size and inference times while maintaining acceptable accuracy.

Model Size Optimization

The next phase of investigation looked to reduce the overall model size without compromising the accuracy of the model. The main lever for reducing parameters in the models is the maximum length of the token sequences (max_len) used. The data followed a typical pattern for English sentences, with a peak sentence length around 10 tokens, with frequency falling off rapidly as sequence length gets longer. Reducing this max_len hyperparameter offers significant benefits in terms of model complexity.

I ran experiments at sequence lengths of 100, 50, 21 (the rounded average sequence length) and 17 (the median length). The following table shows the most accurate models for each sequence length after hyperparameter optimization. The models are in descending order of accuracy.

max_len	Filters (# filters x #tokens)	Dense Layer(s)	Model Parameters in 1,000s	Validation Accuracy
50	96x8, 96x16, 96x32	Single Layer of 8 nodes	681	**0.97649**
100	64x4, 64x8, 64x16	Single Layer of 8 nodes	**501**	0.97559
17	96x8, 96x12, 96x16	Single Layer of 32 nodes	592	0.97062
21	64x8, 64x12, 64x16	Single Layer of 8 nodes	526	0.96926

Table 2 – Comparison of tuned CNN models for different lengths of input sequence

The testing produced interesting results when I looked at models with the fewest tokens and smallest filter sizes considered during tuning. The best performing small model remained within 1.5% of the overall best performing model while reducing the model size by 35%. With additional optimization and pruning of low-significance features in the network, even further improvements might be possible.

max_len	Filters (#filters x #tokens)	Dense Layer(s)	Model Parameters in 1000s	Validation Accuracy
17	16x8, 32x16	Single Layer of 8 nodes	442	0.96067

Table 3 – Results of the best-performing small model

Inference Speed

As the final evaluation of an edge-ready model, I ran a large set of sequences through inference – in this case 100 iterations of the crowdsource.csv data set to determine the fastest model. I tested the overall best four models after hyperparameter tuning, plus the best small model discussed above.

The tests were run on bare-metal hardware using the following core library versions and Jupyter notebooks:

- x86_64 PC running Ubuntu 20.04.1 LTS 64-bit:
    - 1 AMD Ryzen TR 3970X 32-Core Processor with Hyperthreading (64 threads)
    - 1 NVidia RTX2080-Super GPU
    - Memory: 64GB RAM
    - Storage: PCI-E 4.0 NVME m.2 SSD
- Anaconda distribution of Python 3.8.2
- Tensorflow 2.4.1

The performance tests were run using the GPU, then with CPU only and then, to simulate low-powered, edge devices, with the CPU restricted to four or two total threads.

max_len	Filters (number x #tokens)	Dense Layer	# Params in 1000s	Val. Accy	Inference Performance (inferences per second)			
					GPU	CPU (All)	CPU (4 th.)	CPU (2 th.)
50	96x8, 96x16, 96x32	[8]	681	**0.97649**	64,616	41,295	12,776	7,727
100	64x4, 64x8, 64x16	[8]	501	0.97559	63,795	44,210	10,570	6,669
17	96x8, 96x12, 96x16	[32]	592	0.97062	78,362	93,645	59,758	49,467
21	64x8, 64x12, 64x16	[8]	526	0.96926	77,721	84,235	50,666	42,139
17	16x8, 32x16	[8]	**442**	0.96067	**83,241**	**164,808**	**132,122**	**144,605**

Table 4 – Inference Performance

While it is not surprising that the simplest model gave the fastest inference throughput, the scale of the differences is noteworthy. I note the following observations:

1. That CPU inference is significantly faster than GPU inference for models with shorter sequences. Further analysis of data throughput within the GPU; between main memory, to the CPU on to the GPU; and between main memory and CPU only; would be needed to speculate on an explanation.

2. The smallest model did not benefit as much as other models from the high core and thread counts available. I speculate that this is due in part to fewer opportunities to run tensor calculations in parallel.

3. These fewer possible parallel operations could also explain why the smallest model did not show the same orders-of-magnitude speed improvements over the larger ones when running in the massively parallel GPU hardware vs. the CPU.

Conclusion and Further Research

Based on the experiments I have run, it seems eminently feasible to move the claim detection portion of automated fact checking to the edge, embedding it into mobile and desktop browsers on even very low-powered hardware, allowing the automated cloud fact-checking services to only verify claims when and if they are presented to the user. This offers significant opportunity as the services would only scale hardware with the number of user impressions instead of the size of generated data (a low proportion of which is ever subsequently read), reducing costs for providing the knowledge-base and claim-checking portions of fact-checking.

To further optimize for the edge, I would look to conduct testing using pre-trained models running on edge devices under the tensorRT run-time-only library instead of the larger tensorflow framework, and to examine the features captured in the models to prune parameters that do not contribute to the inferred results.

References

[1] Hassan, Naeemul and Arslan, Fatma and Li, Chengkai and Tremayne, Mark (2017) **Toward Automated Fact-Checking: Detecting Check-worthy Factual Claims by ClaimBuster**: KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 2017 Pages 1803–1812; https://doi.org/10.1145/3097983.3098131

[2] Casper Hansen, Christian Hansen, Stephen Alstrup, Jakob Grue Simonsen, Christina Lioma (2019); **Neural Check-Worthiness Ranking with Weak Supervision: Finding Sentences for Fact-Checking**:; WWW '19: Companion Proceedings of The 2019 World Wide Web ConferenceMay 2019 Pages 994–1000; https://doi.org/10.1145/3308560.3316736

[3] Kevin Meng, Damian Jimenez, Fatma Arslan, Jacob Daniel Devasier, Daniel Obembe, Chengkai Li (2020); **Gradient-Based Adversarial Training on Transformer Networks for Detecting Check-Worthy Factual Claims**: arXiv:2002.07725 [cs.CL]; https://arxiv.org/abs/2002.07725

[4] Arslan, Fatma and Hassan, Naeemul and Li, Chengkai and Tremayne, Mark (2020): **A Benchmark Dataset of Check-worthy Factual Claims**:; 14th International AAAI Conference on Web and Social Media 2020; arXiv:2004.14425; https://arxiv.org/abs/2004.14425

Related Reading

1. Levy, R., Gretz, S., Sznajder, B., Hummel, S., Aharonov, R., & Slonim, N. (2017, September). **Unsupervised corpus–wide claim detection**. In *Proceedings of the 4th Workshop on Argument Mining* (pp. 79-84).
2. Duan, X., Liao, M., Zhao, X., Wu, W., & Lv, P. (2018, November). **An Unsupervised Joint Model for Claim Detection**. In *International Conference on Cognitive Systems and Signal Processing* (pp. 197-209). Springer, Singapore.
3. Allein, L., & Moens, M. F. (2020, October). **Checkworthiness in Automatic Claim Detection Models: Definitions and Analysis of Datasets**. In *Multidisciplinary International Symposium on Disinformation in Open Online Media* (pp. 1-17). Springer, Cham.
4. Konstantinovskiy, L., Price, O., Babakar, M., & Zubiaga, A. (2018). **Towards automated factchecking: Developing an annotation schema and benchmark for consistent automated claim detection**. *arXiv preprint arXiv:1809.08193*.
5. Hidey, C., Musi, E., Hwang, A., Muresan, S., & McKeown, K. (2017, September). **Analyzing the semantic types of claims and premises in an online persuasive forum**. In *Proceedings of the 4th Workshop on Argument Mining* (pp. 11-21).
6. Gencheva, P., Nakov, P., Màrquez, L., Barrón-Cedeño, A., & Koychev, I. (2017, September). **A context-aware approach for detecting worth-checking claims in political debates**. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017* (pp. 267-276).
7. Daxenberger, J., Eger, S., Habernal, I., Stab, C., & Gurevych, I. (2017). **What is the essence of a claim? Cross-domain claim identification.** *arXiv preprint arXiv:1704.07203*.
8. Cheema, G. S., Hakimov, S., & Ewerth, R. (2020). Check_square at CheckThat! 2020: **Claim Detection in Social Media via Fusion of Transformer and Syntactic Features**. *arXiv preprint arXiv:2007.10534*.
9. Chakrabarty, T., Hidey, C., & McKeown, K. (2019). **IMHO fine-tuning improves claim detection**. *arXiv preprint arXiv:1905.07000*.