Questionable Answers in Question Answering Research: Reproducibility and Variability of Published Results

Matt Crane

David R. Cheriton School of Computer Science, University of Waterloo

matt.crane@uwaterloo.ca

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

Abstract

"Based on theoretical reasoning it has been suggested that the reliability of findings published in the scientific literature decreases with the popularity of a research field" (Pfeiffer and Hoffmann, 2009). As we know, deep learning is very popular and the ability to reproduce results is an important part of science. There is growing concern within the deep learning community about the reproducibility of results that are presented. In this paper we present a number of controllable, yet unreported, effects that can substantially change the effectiveness of a sample model, and thusly the reproducibility of those results. Through these environmental effects we show that the commonly held belief that distribution of source code is all that is needed for reproducibility is not enough. Source code without a reproducible environment does not mean anything at all. In addition the range of results produced from these effects can be larger than the majority of incremental improvement reported.

1 Introduction

The recent "reproducibility crisis" (Baker, 2016) in various scientific fields (particularly Psychology and Social Sciences) indicates that some introspection is needed in all fields, particularly those that are experimental by nature. The efforts of Collberg's repeatability studies highlight the state of affairs within the computer systems research community (Moraila et al., 2014; Collberg et al., 2015).[1] Other fields have also begun to push for more stringent presentation of results, for example, the information retrieval community has been aware for some time of the issues surrounding weak baselines (Armstrong et al., 2009) and more recently reproducibility (Arguello et al., 2016; Lin et al., 2016).

The issue of reproducibility in the deep-learning community has also started to become a growing concern, with the need for replicable and reproducible results being included in a list of challenges for the ACL (Nivre, 2017). In reinforcement learning, Henderson et al. (2017) showed that there are a number of effects that would change the results obtained by published authors and call for more rigorous testing, and reporting, of state-of-the-art methods. There is also an ongoing project by OpenAI to provide baselines in reinforcement learning that are reproduced from published descriptions, but even they admit that their scores are only "roughly on par with the scores in published papers."[2] Reimers and Gurevych (2017) investigated over 50,000 combinations of hyper-parameter settings, such as word embedding sources and the optimizer across five different NLP tasks and found that these settings have a significant impact on both the variability, and the relative effectiveness of models.

In this paper we present a number of controllable environment settings that often go unreported, and illustrate that these are factors that can cause irreproducibility of results as presented in the literature. These environmental factors have an effect on the effectiveness of neural networks due to the non-convexity of the optimization surface, meaning that

[1] http://reproducibility.cs.arizona.edu

[2] https://blog.openai.com/openai-baselines-dqn/

Transactions of the Association for Computational Linguistics, vol. 6, pp. 241–252, 2018. Action Editor: Brian Roark.
Submission batch: 9/2017; Revision batch: 12/2017; Published 4/2018.
©2018 Association for Computational Linguistics. Distributed under a CC-BY 4.0 license.

even minor changes in computation can lead the network to fall into one of a multitude of local minima. Because these effect sizes are comparable to the largest incremental improvements that have been reported brings into question those improvements, and associated claims of progress.

2 Experimental Setup

In order to limit the scope of this paper, we specifically focus our efforts on a single natural language processing task—answer selection within question answering—elaborated upon in Section 2.1. We also further limit our discussion to look at how these environmental effects manifest in a single implementation of a single model, described in Section 2.2. These restrictions, however, do not mean that our results are only applicable to this model on this task, rather our discussion generalizes to all neural network based research.

To isolate the effect that each environmental factor has all other settings related to the network are fixed; that is, the hyper-parameters are static across all experiments, and only the environmental variable of interest is manipulated. Along with each of the presented factors we include suggestions on how to respond to these in order to best ensure that the work, as presented, is reproducible.

2.1 Exemplar Task

Answer selection is one important aspect of open-domain question answering. Given a question, q, and a set of candidate sentences, A, the answer selection task is to rank the sentences contained in A such that those candidates that answer the question are ranked at the top of the list. From this ranked list and assessments of whether the candidate contains an answer to the question, common information retrieval metrics average precision (AP) and reciprocal rank (RR) can be calculated to assess the effectiveness of the system. These metrics are the de facto metrics to evaluate answer selection, and as such the metrics are reported within this paper. Descriptions of these metrics are easily found in the literature.

Worryingly, in the literature, it is becoming increasingly common to not conduct statistical significance testing, rather a higher metric value is taken as evidence that the model performs better. Due to the nature of this paper, we only perform significance testing between results in the same condition, and not across conditions. Within each condition we identify a "baseline"/default setting to compare against. Conducting this many significance tests would normally call for a correction method to be applied, but we do not do so, as we only wish to indicate that selecting the higher number may result in an absolute difference, but not necessarily a statistically significantly one. To calculate significance we use a paired Wilcoxon signed rank test.

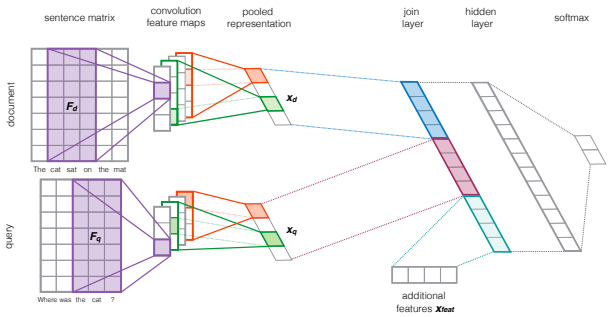

Figure 1: Exemplar model architecture diagram.

2.2 Exemplar Model

To perform our experiments we utilize the model released by Sequiera et al. (2017), a simplified PyTorch implementation of the model proposed by Severyn and Moschitti (2015). The model was chosen because of its simplicity, it is quick to train which supports a fast iteration of experiments, and it has also been reimplemented with similar effectiveness additional times (Rao et al., 2017). Figure 1 shows a diagram of the model, which adopts a "Siamese" structure with two sub-networks to process the question and candidate sentence.

We emphasize that this model was only selected to serve as an exemplar; the effects that are observed in relative performance will also be present in other models. Indeed, because of the simplicity of this model, it is likely that the environmental effects described will have a more substantial impact on the network effectiveness of more complicated models.

2.3 Datasets

The experiments reported in this paper are all performed against the TrecQA dataset that was first released by Wang et al. (2007) and further elab-

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

Split	Questions	Answers	
		Positive	Negative
TrecQA			
Train	1,229	6,403	47,014
Development	82	222	926
Test	100	284	1,233
Total	1,411	6,906	49,173
WikiQA			
Train	873	1,040	7,632
Development	126	140	990
Test	243	293	2,058
Total	1,242	1,473	10,680

Table 1: Dataset summaries.

orated upon by Yao et al. (2013), as well as the WikiQA dataset released by Yang et al. (2015). Both datasets consists of pre-determined training, development and test sets. For each question, each candidate answer is labelled positive if it contains the answer to the question, otherwise negative. The ratios of these labels and size of the splits for both datasets are shown in Table 1.

The TrecQA dataset has further diverged into two versions, named RAW and CLEAN. The CLEAN version has removed those questions that had no positive labelled answers. Results on these two variants are not directly comparable to each other (Rao et al., 2017), and experiments in this paper are performed against the RAW variant. Similar manipulation of the WikiQA dataset has also been performed, although no analysis of the comparability of the results has been conducted.

3 Weak Baselines

As observed by Armstrong et al. (2009) in the information retrieval field, the use of weak baselines is a factor that should be considered when discussing results as using a weak baseline shows a greater improvement than could otherwise be claimed. Table 2 shows the state-of-the-art results in answer selection, as replicated from the ACL Wiki, Table 3 likewise shows the (potentially incomplete) state-of-the-art results on the WikiQA dataset, sourced by inspection of relevant papers. The TrecQA dataset contains an additional row that presents a simple baseline—the sum of IDF weights for terms in both

Model	AP	RR	Δ AP	Δ RR
RAW DATASET				
Punyakanok et al. (2004)	0.419	0.494		
Cui et al. (2005)	0.427	0.526	0.008	0.032
Wang et al. (2007)	0.603	0.685	0.176	0.159
Heilman and Smith (2010)	0.609	0.692	0.006	0.007
Wang and Manning (2010)	0.595	0.695	−0.014	0.003
Yao et al. (2013)	0.631	0.748	0.022	0.053
Severyn and Moschitti (2013)	0.678	0.736	0.047	−0.012
Shnarch (2013)	0.686	0.754	0.008	0.006
IDF-Weighted Sum	0.701	0.769		
Yih et al. (2013)	0.709	0.770	0.023	0.016
Yu et al. (2014)	0.711	0.785	0.002	0.015
Wang and Nyberg (2015)	0.713	0.792	0.002	0.007
Feng et al. (2015)	0.711	0.800	−0.002	0.008
Severyn and Moschitti (2015)	0.746	0.808	0.033	0.008
Yang et al. (2016)	0.750	0.811	0.004	0.003
He et al. (2015)	0.762	0.830	0.012	0.019
He and Lin (2016)	0.758	0.822	−0.004	−0.008
Rao et al. (2016)	0.780	0.834	0.018	0.004
Chen et al. (2017b)	0.782	0.837	0.002	0.003
CLEAN DATASET				
Wang and Ittycheriah (2015)	0.746	0.820		
Tan et al. (2015)	0.728	0.832	−0.018	0.012
dos Santos et al. (2016)	0.753	0.851	0.007	0.019
Wang et al. (2016b)	0.771	0.845	0.018	−0.006
He et al. (2015)	0.777	0.836	0.006	−0.015
He and Lin (2016)	0.801	0.877	0.024	0.026
?)	0.802	0.875	0.001	−0.002

Table 2: State-of-the-art (replicated from ACL Wiki (ACL, 2017)) results on the TrecQA dataset versions, annotated with improvement over prior state-of-the-art results and a simple baseline.

the question and candidate sentence—that performs no learning of any sort.

$$S_{q,a} = \sum_{t \in q \cap a} \log \frac{|D|}{|\{d \in D : t \in d\}|} \qquad (1)$$

Equation 1 shows the function that produces these results, where D is the document collection, d is a document from this collection, q is the query, a is the candidate sentence, and t is a term. This calculation is done after removal of stopwords.[3] This baseline outperforms a number of the older state-of-the-art methods. Therefore these older results, and some results afterwards are comparing against a weak baseline. In fairness, this baseline was first reported in the literature by Yih et al. (2013), but is

[3]The stopword list contains 127 English words, sourced from the `nltk` Python Library.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

Model	AP	RR	Δ AP	Δ RR
Yu et al. (2014)	0.6190	0.6281		
Yang et al. (2015)	0.6520	0.6652	0.0330	0.0371
dos Santos et al. (2016)	0.6886	0.6957	0.0366	0.0305
Miao et al. (2016)	0.6886	0.7069	0.0000	0.0112
Yin et al. (2016)	0.6921	0.7108	0.0035	0.0039
Rao et al. (2016)	0.701	0.718	0.0080	0.0072
Wang et al. (2016b)	0.7058	0.7226	0.0048	0.0046
He and Lin (2016)	0.7090	0.7234	0.0032	0.0008
Yin and Schütze (2017)	0.7124	0.7237	0.0034	0.0003
Chen et al. (2017a)	0.7212	0.7312	0.0088	0.0075
Wang et al. (2016a)	0.7341	0.7418	0.0129	0.0106
Wang and Jiang (2016)	0.7433	0.7545	0.0092	0.0127

Table 3: State-of-the-art (gathered by manual inspection) results on the WikiQA dataset, annotated with improvement over prior state-of-the-art results.

a result that is frequently overlooked in the literature that followed. However, we also note that the results for this simple baseline differ between our reported value and that of Yih et al. (2013), which is substantially lower—0.6531 AP, 0.7071 RR. For these reasons we repeat this result here.

4 Confounding Variables

In this section we document a number of confounding variables that often go unreported in the literature, and can have a substantial effect on whether a result would be considered state-of-the-art or not, and the reproducibility of that result. These range from controllable factors, to factors that are not controllable, but need to be reported. To aid discussion, the state-of-the-art tables have been recreated and annotated with the change in AP and RR over the then state-of-the-art result (Table 2 and Table 3).

Unless otherwise stated, all experiments are performed using Docker containers that are derived from common, shared, base images. This substantially eases the fixing of all environment and versioning issues that can be observed when running under a native environment. All the data that is required to reproduce the results in this paper is publicly available. Including Docker images, scripts to create and use those images, and resulting pretrained model files.[4]

[4] https://github.com/snapbug/questionable-qa

Version	TrecQA AP	TrecQA RR	WikiQA AP	WikiQA RR
cf0e269	0.7495	0.8122	0.6732	0.6953
1f894ba				
171fee4	0.7495	0.8122	0.6732	0.6953
715502b	0.7495	0.8122	0.6732	0.6953
d99990b	0.7495	0.8122	0.6732	0.6953
70d7a03*	0.7495	0.8122	0.6732	0.6953
6d9d98f*+	0.7587	0.8225	0.6858	0.7065
5ef19a9*+	$0.6741^{\ddagger}$	$0.7519^{\ddagger}$	$0.5374^{\ddagger}$	$0.5422^{\ddagger}$
196f0aa*+	$0.6742^{\ddagger}$	$0.7519^{\ddagger}$	$0.5376^{\ddagger}$	$0.5424^{\ddagger}$
95ea349*+	$0.6713^{\ddagger}$	$0.7409^{\dagger}$	$0.5543^{\ddagger}$	$0.5579^{\ddagger}$

Table 4: Effect of the version of the model being used on model results. Only versions that modified the `py` files are included. A * indicates that the model at that changeset does not run under the created Docker environment, and results are taken from a native host, and a + indicates that the results from this version are themselves not reproducible, changing between runs. Version `1f894ba` does not complete due to a bug. A ‡ indicates that the result was statistically significantly different at the $p < 0.01$ level, and † at the $p < 0.05$ level, compared to `cf0e269`.

4.1 Software Versions

There are numerous points of software in which the version of the software being used can impact the end results substantially. These are the model definitions, they framework software and the libraries that the framework uses.

4.1.1 Model Definition

We refer to the code that is used to define the model and to run the experiments as the model definition. These are changing artifacts, and when the software is made available to researchers, then it *must* be accompanied by the version of that software being used. A cursory glance of the commit history of some of these repositories shows a non-zero amount of bug fixing commits. Because of the nature of deep learning, these bugs may actually *improve* effectiveness, as anecdotal evidence suggests.[5]

Whether the commits fix bugs or add features, the models being compared are inherently different and can result in different outcomes. Authors should specify which version of the code is being run to obtain the results presented. Table 4 shows the effect

[5] https://twitter.com/soumithchintala/status/910339781019791360

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

	TrecQA		WikiQA	
PyTorch	AP	RR	AP	RR
0.2.0	0.7234†	0.7866	0.6773	0.6980
0.1.12	0.7495	0.8122	0.6732	0.6953
0.1.11	0.7495	0.8122	0.6732	0.6953
0.1.10	0.7495	0.8122	0.6732	0.6953
0.1.9	0.7495	0.8122	0.6732	0.6953

Table 5: Effect of the version of PyTorch being used on model results. Version 0.1.8 and earlier would not run the sample model due to API changes. A † indicates that the results are statistically significantly different to 0.1.12 at the $p < 0.05$ level.

Library/Platform	AP	RR
TrecQA		
Intel MKL on Intel i7-6800K	0.7495	0.8122
Intel MKL on AMD FX-8370E	0.7487	0.8136
OpenBLAS on either	0.7307	0.8029
WikiQA		
Intel MKL on Intel i7-6800K	0.6732	0.6953
Intel MKL on AMD FX-8370E	0.6772	0.6981
OpenBLAS on either	0.6773	0.6980

Table 6: Effect of changing math library and architecture on model results, using PyTorch 0.1.12. None of the results are statistically significantly different to the i7-6800K.

of changing the version of the code on the model's effectiveness. As can be seen, there is a reasonable shift in results, and until authors specify the exact version of the model that is used for experimentation their results are non-reproducible. The version used for all further experiments in this paper is cf0e269.

4.1.2 Framework Version

Specifying the framework that is being used would be an important first step. Different framework versions could give different results for the same model code. To illustrate this we ran the sample model over a range of different versions of PyTorch.

Table 5 shows the impact of changing the version of PyTorch used in the training of the sample model. It shows that newer (0.2.0) is not necessarily better, although this depends on the dataset. The version used throughout the rest of the paper is 0.1.12, as this is the version that was used in prior work for this model. Version 0.1.8 and earlier would not run the sample model due to use of features introduced in 0.1.9. The results are stable for 0.1.x versions across datasets. One possible cause could be that the underlying libraries PyTorch relies on were pinned to specific versions across PyTorch versions. Alternatively, the model code may not be using features of PyTorch that were changing across these versions.

4.1.3 Framework Dependencies

While fixing the framework version is a good step, these frameworks often themselves rely on other libraries. Of particular interest to the neural network community is the math library that underpins all the matrix and vector operations. By default PyTorch installs a version of the library that is linked against Intel's Math Kernel Library (MKL). When running the sample model on different hardware, we identify the effectiveness of the model changes.

Table 6 shows the results of running the MKL-backed version on Intel and AMD hardware, compared to an OpenBLAS, which results in the same answers regardless of hardware. Intel also notes that the results of the same floating point calculation may be different across their own hardware.[6] It should not surprise the reader that Intel's math library gives different results on different architectures; after all, Intel knows with great detail the architecture of Intel chipsets and is not necessarily inclined to produce optimal code for competing platforms. The sensitivity of the network to the backing math library is dependent on the dataset. This difference in effectiveness is likely due to the relative non-convexity of the optimization surface for the two datasets, where the TrecQA surface has a large number of local minima.

Changing the library, or even the backend, within the same library, in which the model is implemented can substantially change the effectiveness of the model. For example Simon (2017) observed a 16% increase of test accuracy for the same model (from 0.5438 to 0.6197) by changing the computation backend from Tensorflow to MXNet.

[6] http://intel.ly/1b8Qrq6

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

	TrecQA		WikiQA	
Threads	AP	RR	AP	RR
1	0.7495	0.8122	0.6732	0.6953
2	0.7485	0.8145	0.6802	0.7022
3	0.7495	0.8122	0.6732	0.6953
4	0.7477	0.8096	0.6771	0.6983
5	0.7495	0.8122	0.6732	0.6953
6	0.7489	0.8162	0.6778	0.6992

Table 7: Effect of number of threads on model results using MKL-backed PyTorch `v0.1.12` on an Intel i7-6800K processor. None of the results are statistically significantly different to a single thread.

4.2 Threads

Threading introduces a number of possibilities for non-reproducible results, as results from threads can be returned in differing orders. This is because floating point arithmetic is non-associative as well as non-commutative; however, these effects can be controlled by using the appropriate functions and settings in the library. Training the sample model repeatedly achieves the same results, suggesting that these settings are being utilized inside the PyTorch library, although we implore readers to discover this for their library of choice.

However, while threading itself does not impact the results within PyTorch, the number of threads used does. Other than by never varying the number of threads used, this effect cannot be controlled for. The reason for this is related to the non-associativity of floating point maths. For example, given the mathematical relations $a + b = e$, and $c + d = f$, the floating point specification does not ensure that the mathematical equality $a + b + c + d = e + f$ holds. A result calculated on two threads may perform the $e + f$ calculation, while on four threads the $a+b+c+d$ calculation may be performed, resulting in potential differences.

For these experiments we use PyTorch `v0.1.12` with Intel's MKL library on an Intel i7-6800K processor. Using the `OMP_NUM_THREADS` and `MKL_NUM_THREADS` environment variables, as well as the `set_num_threads` function in PyTorch, we can control the number of threads used in training. We range this from 1–6 on our machine, as this is the number of hardware cores on the CPU, and therefore the maximum number of threads that OpenMP will spawn. Table 7 shows the results of this experiment. Interestingly the results are consistent within datasets when using an odd number of threads, although this is most likely coincidental. The range of differences is small, but is, again, larger than some of the incremental improvements reported in the literature. The exact environment variables, or code settings, that need to be modified will depend on the framework being used.

There is no solution to this given the non-associative nature of the floating-point and the splitting of workload among differing numbers of threads. The only recommendation is that authors report the number of threads used for training, although we do suggest a smaller number to err on the side of caution, as OpenMP will not create more threads than there are hardware cores.

4.3 GPU Computation

The variation of GPUs available for deep learning research is arguably larger than that of CPUs. There are many models, and each manufacturer is free to deviate from the reference models provided by nVidia or AMD, although it is unclear just how many choose to do so. There are also more uncontrollable factors, for instance, the number of threads that are used by the GPU is uncontrollable meaning that results are unlikely to be the same across different GPUs, unlike CPU training.

Table 8 shows the results of enabling GPU computation on the sample model. We report on both enabling the cuDNN backend, as this is the default, as well as disabling it globally. The cuDNN backend is known to contain some non-deterministic kernels.[7] In addition, nVidia provides a white-paper that describes some of the implementation details and compliance issues of the IEEE 754 floating point specification and their impact on nVidia GPUs (Whitehead and Fit-Florea, 2011). The paper also presents examples where compiling for a 32-bit `x86` architecture and a 64-bit `x86-64` architecture can yield different results.

In addition to running the experiment on our own GPU, an Asus branded nVidia GeForce 1080GTX (revision a1) we also repeated the experiment on

[7] `http://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/#reproducibility`

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

	TrecQA		WikiQA	
Computation Hardware	AP	RR	AP	RR
CPU				
Intel i7-6800K	0.7495	0.8122	0.6732	0.6953
GPU				
GeForce 1080GTX cuDNN	0.7277	0.7788	0.6604	0.6804
GeForce 1080GTX	0.7474	0.8044	0.6873	0.7054
Tesla K80 cuDNN	0.7527	0.8115	0.6852	0.7046
Tesla K80	0.7527	0.8115	0.6852	0.7046

Table 8: Effect of the computation hardware on model results. None of the results are statistically significantly different when compared to the Intel i7-6800K.

an Amazon EC2 p2.xlarge instance. This instance comes equipped with a single nVidia Tesla K80 GPU. Other instances come equipped with multiple GPUs, but as the model is both small and does not take advantage of multiple GPUs experiments were not performed on these instances. Of note is the presence or absence of the cuDNN library has no effect on the K80, but does on the 1080GTX GPU. We suspect that the reason for this is that the K80 is designed as a compute card, while the 1080GTX is primarily designed for graphics processing. This design difference could manifest itself in different instructions that can be taken advantage of by cuDNN kernels.

Even with just two GPUs and using the cuDNN backend, there is already evidence that the performance of the network depends on both the dataset and the underlying hardware. There is a clear dependence on the dataset for the relative performance. Further results reported on the GPU are reported on the GeForce 1080GTX with cuDNN disabled. Although this is not the default it maximizes reproducibility by avoiding non-reproducible kernels.

4.4 Random Seed

Perhaps the most obvious feature of machine learning that can impact the effectiveness is the random seed. Thus far the experiments in this paper have used a fixed seed, and like most prior research this was only implied rather than explicitly stated. The seed in question was 1234.

Randomness is a crucial part of machine learning and values from the random generator are widely used. For example, random values are used for the initial values of weights, for selecting which nodes to drop in drop-out layers, and for selecting set embeddings for terms that have no associated embeddings. As Goldberg (2017, Section 5.3.2 p59) rightly makes note of—"When debugging, *and for reproducibility of results* [emphasis added], it is advised to used a fixed random seed." Figure 2 shows the variance in AP and RR when specifying different seeds, for 200 randomly chosen seeds (selected using the bash (version 4.3.48(1)) `RANDOM` built-in, itself initialized/seeded to 1234 prior to performing runs).

Noting the generator of the random numbers is important, as different languages and libraries may use different generators. Most languages default to a pseudo-random generator for performance reasons, which carries the additional benefit that sequences can be reconstructed from a given start state, commonly referred to as a seed. For example, the bash version used to generate the seeds for Figure 2 uses a linear congruential generator (Park and Miller, 1988). A more commonly used generator is MT19937, a Mersenne Twister based on the Mersenne prime $2^{19937} - 1$, the standard implementation that uses a 32-bit word length. Another implementation, MT19937-64, uses a 64-bit word length and generates a different sequence. To specify the generator used it is often enough to specify the language version and platform being used. PyTorch and dependent libraries use the aforementioned MT19937 generator.

The spread of results shows that the results are either marginally worse than prior work (which would likely mean the result from this model would not be

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

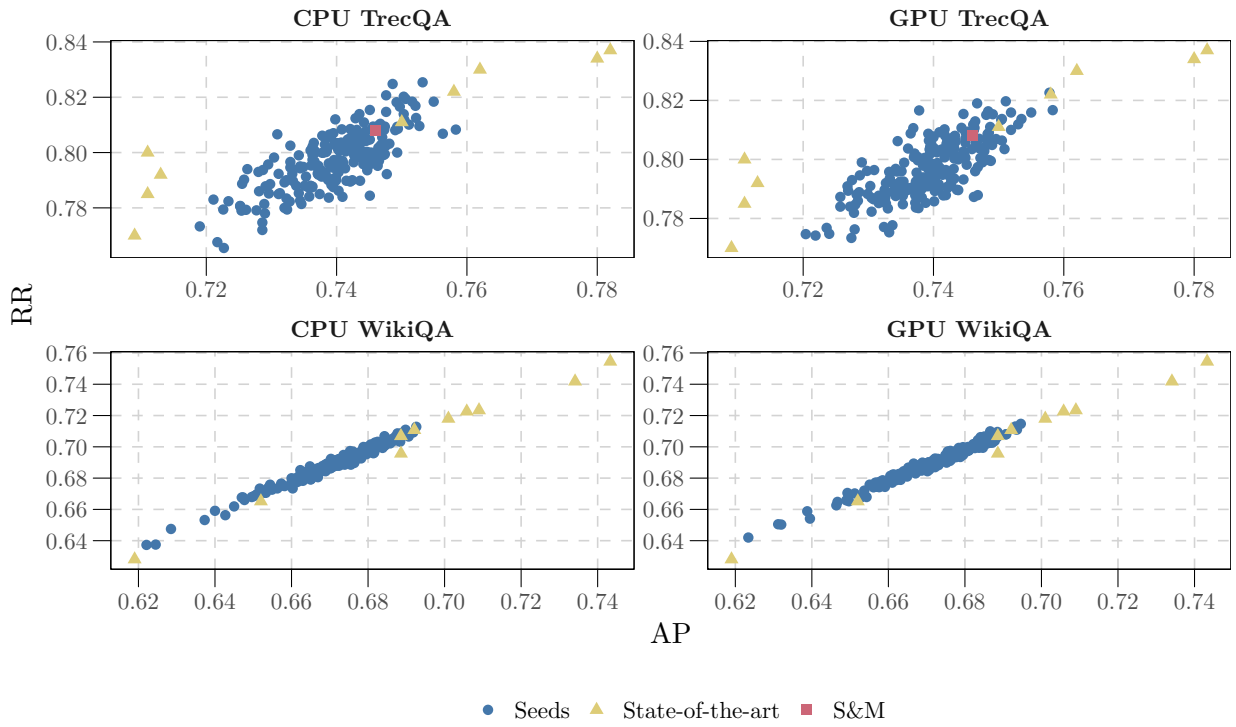

Figure 2: Variance in AP and RR due to the random seed specified when training on both the CPU & GPU. Version `cf0e269` of the model was used, with version `0.1.12` of PyTorch, and the Intel MKL math library. CPU training was performed on an Intel i7-6800K processor using one thread, and GPU training on an Asus branded nVidia 1080GTX (revision a1) with CUDA version 8.0.61. The 200 seeds were selected by the bash (version 4.3.48(1)) `RANDOM` builtin, itself initialized to 1234. The seeds selected were identical for all training conditions. The colours and shapes represent whether the model is the sample model (blue circles), the S&M model that is being reimplemented (red square), or another state-of-the-art result (yellow triangles). The TrecQA dataset is shown on the top, and the WikiQA dataset on the bottom.

published), or better than work that was reported on afterward (meaning these latter results may not have been published). Significance testing across these models was not performed.

Table 9 shows the agreements, calculated using Kendall's τ and Spearman's ρ, on rankings for each of the datasets, comparing the two metrics, and two computational backends used, all results shown are statistically significant at the $p < 0.01$ level.

On the TrecQA dataset, these variations in AP and RR show only moderate agreement in rankings of the model when the same training computational backend is used, and only weak agreement across computation backend. For the TrecQA dataset, the CPU results covered a range of 0.0393 for AP, and 0.0599 for RR, while the GPU covered 0.0379 AP, and 0.0492 RR respectively.

The WikiQA dataset exhibits stronger agreements about model rankings on the same computation backend, but similarly weak agreements when comparing across computational backends. The range of AP and RR values on this dataset are even larger, covering 0.0712 AP, and 0.0727 RR on GPU; and 0.0705 AP and 0.0755 RR on the CPU.

These ranges in AP and RR values are greater than a large proportion of incremental improvements reported in prior answer selection research (see Table 2 and Table 3), and indeed are an order of magnitude larger than a typically reported improvement in either metric on the WikiQA datasets. In these cases the model was trained to target AP, another setting that is not commonly reported. While some software for models made available specifies a seed, this detail is often omitted from the paper, making replication-from-paper efforts nigh on impossible.

Reagen et al. (2017, Chapter 4) discuss this variance in results from seeding, calling it Iso-Training Noise. They use this concept to frame discus-

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

TrecQA			
KENDALL'S τ	RR_{CPU}	AP_{GPU}	RR_{GPU}
AP_{CPU}	0.5514	0.2871	0.2069
RR_{CPU}		0.2148	0.2894
AP_{GPU}			0.5315
SPEARMAN'S ρ			
AP_{CPU}	0.7409	0.4125	0.3304
RR_{CPU}		0.3126	0.4205
AP_{GPU}			0.7171
WikiQA			
KENDALL'S τ	RR_{CPU}	AP_{GPU}	RR_{GPU}
AP_{CPU}	0.8842	0.3238	0.3358
RR_{CPU}		0.3096	0.3330
AP_{GPU}			0.9068
SPEARMAN'S ρ			
AP_{CPU}	0.9783	0.4622	0.4762
RR_{CPU}		0.4392	0.4690
AP_{GPU}			0.9868

Table 9: Kendall's τ and Spearman's ρ based on the rankings of model effectiveness on different seeds across metrics and training computation backend. All values are statistically significant at the $p < 0.01$ level.

sion over whether optimizations, such as using fixed point arithmetic over floating point, are safe to perform. They define an optimization as safe if the results are within one standard deviation of the mean of the results observed from multiple seeded runs.

We suggest that specifying the random seed used in training is the *bare minimum*, necessary step that should be taken, although given the potential for different pseudo-random generators, and differences in implementation, this may not be enough. Indeed, the best approach is to stop reporting single-value results, and instead report the distribution of results from a range of seeds. Doing so allows for a fairer comparison across models, by discarding potential comparisons of lucky and unlucky seeds. In addition, these result populations can be statistically compared for significance, allowing for stronger claims on improvement.

4.5 Interactions

Thus far this paper has presented a number of effects that can affect the results of a neural network. Each of these has been presented in isolation, after fixing the prior effects. These effects clearly have potential for interaction and the interaction is unpredictable. In this section we briefly examine one of these interactions, namely the seed selection combined with either CPU or GPU training.

The results presented in Table 8 show that for a given seed the models exhibit different effectiveness based on the hardware used for training. In Section 4.4 it was shown that the seed has a significant impact on the relative effectiveness of the model regardless of this computational backend. The correlation coefficients across devices presented in Table 9 leads us to suspect that there can be substantial changes in effectiveness when switching the backend from CPU to GPU and vice versa.

Figure 3 shows the effect of changing from CPU training to GPU training, using the same 200 seeds that were used in Section 4.4. The relationship observed in Figure 2 between AP and RR is still present, but there is no telling, given a fixed seed, whether training on GPU or CPU would result in better effectiveness. In addition, these deltas can be larger than a substantial number of incremental improvements reported. For example, a middling result on the CPU may be transformed to either a top or bottom result if switching to GPU training, with everything else fixed.

By reporting results as single numbers the variation due to the hardware on which the training is performed is hidden, and this could lead authors to conclude that their model is a substantial improvement on state-of-the-art. The changes in AP and RR that are observed are representative of even the larger improvements in state-of-the-art. However, when comparing the *distributions* of the scores across the backends by visual inspection of Figure 2 there is clearly not any difference in the populations. Statistical significance testing ($p \gg 0.05$ in a paired t-test, both two- and single-tailed) bears out this intuition. Using these population based results would then lead authors to a different conclusion than if the seed was "lucky" for the training hardware. This is a concrete example of the differences between reporting result distributions compared with single values.

4.6 Reporting Rounding

The final aspect of result reporting that is controllable for is the rounding of results. For example,

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

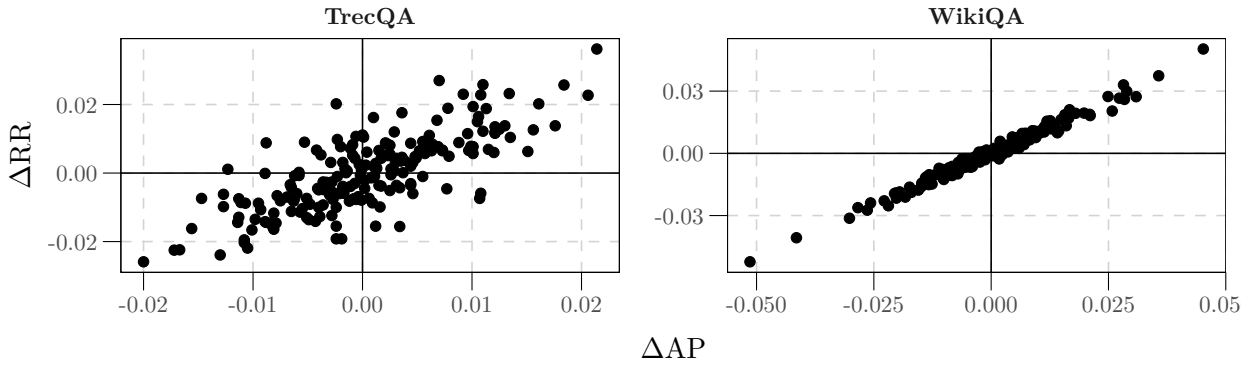

Figure 3: Change in AP and RR when switching to training the model on GPU from to CPU. Each dot represents a training run with the same seed provided to each of the training processes. The TrecQA dataset is shown on the left, and WikiQA on the right.

when using the default install options of the sample model, and fixing the other versions and settings, our sample model gives two observed separate results with CPU training—on the TrecQA dataset either an AP of 0.7485 or 0.7487 is obtained. While this difference of 0.0002 is *small*, there is a newer trend (present in the latter three papers in Table 2) of reporting results to three decimal points. In this case even such a minor difference can result in state-of-the-art or not, statistical significance notwithstanding. For example a result of 0.7484 would round down, while 0.7486 would round up, overemphasizing the difference by a factor of 5. We concede, however, that the same argument can be applied regardless of which decimal point cut-off is used, although we observe that `trec_eval`, the de facto tool used to calculate AP and RR, reports to four.

We recommend that reviewers be skeptical of such minor improvements on state-of-the-art when single results are reported, the recommendation here follows that of Section 4.4, in that ideally multiple seeds are used, and testing is performed on the population of results to determine improvement.

5 Conclusions

In this paper we have demonstrated a number of factors that are present during training of a model and affect the results of said model. These parameters, and their settings, often go unreported in the literature. The result is that a large amount of prior work in answer selection is inherently irreproducible. Furthermore, the differences in results illustrated by these effects can be much larger than the majority of improvements reported as gains in the literature.

The effects that we presented are not stand-alone effects. Interaction between effects also has an additional impact, one of which was discussed in Section 4.5. Other results presented in this paper do not consider this interaction. For example Table 6 suggested that a model trained using OpenBLAS produces worse results for the TrecQA dataset than one trained using Intel's MKL library, which is true… for that version of the model code, for that version of the framework, for that random seed, when trained on a single thread on that CPU, for that dataset. We reserve investigating the interaction effects of these individual effects for future work.

It is simply no longer adequate to report a single value when evaluating results from neural networks, *especially* without the presence of statistical testing on those results. By far the largest source of variability in the experiments presented in this paper was when the network was seeded with different random starting points. The range of results produced cover ranges of results that can be an order of magnitude larger than typically imported improvements.

As well as repeating experiments for multiple seeds, the specifications of the hardware on which the experiments were performed should be reported alongside the results, as changing the hardware can change the results by an order of magnitude. Additionally, the number of threads and the math library used impact on the results and should be reported.

Finally, beyond the hardware effects, the software that is used to both run the model, and define the model, has an impact. For this reason both the model

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

definition and library versions, as well as all the required dependencies, should be pinned to a specified version. These issues are easily avoidable by the use of common packaging tools such as Docker, which also provides opportunities to fix most of the non-versioning environmental issues as well.

In cases where authors are unable to provide a Docker image, or equivalent, then making the trained models available is one alternative. Loading pre-trained models is an action that is supported by a number of frameworks. PyTorch, for example, provides functions to load a model from a URL. The pre-trained models appear to provide consistent results even when the inference pass is performed using settings that would have provided different results in training.

A sentence is all that it takes to describe the environment used for training. For example: "our model was written against PyTorch v0.1.12, and training was conducted on an Intel i7-6800K using a single thread and Intel's Math Kernel Library". Beyond this we implore reviewers to be wary of such minor reported improvements in the light of these issues.

Acknowledgements

The author wishes to acknowledge the input and advice of (in alphabetical order) Gaurav Baruah, Jimmy Lin, Adam Roegiest, Royal Sequiera, and Michael Tu. Finally thanks to the reviewers and editors for their comments and suggestions to improve the paper.

References

ACL. 2017. Question Answering (State of the art). https://aclweb.org/w/index.php?title=Question_Answering_(State_of_the_art). Accessed: Sept. 7 2017.

Jaime Arguello, Matt Crane, Fernando Diaz, Jimmy Lin, and Andrew Trotman. 2016. Report on the SIGIR 2015 workshop on reproducibility, inexplicability, and generalizability of results (RIGOR). 49(2):107–116.

Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *SIGIR*, pages 601–610.

Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454.

Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, Liang He, and Weijie An. 2017a. Enhancing recurrent neural networks with positional attention for question answering. In *SIGIR*, pages 993–996.

Ruey-Cheng Chen, Evi Yulianti, Mark Sanderson, and W. Bruce Croft. 2017b. On the benefit of incorporating external features in a neural architecture for answer sentence selection. In *SIGIR*, pages 1017–1020.

Christian Collberg, Todd Proebsting, and Alex M. Warren. 2015. Repeatability and benefaction in computer systems research. *University of Arizona TR 14*.

Hang Cui, Renxu Sun, Keya Li, Min-Yan Kan, and Tat-Seng Chua. 2005. Question answering passage retrieval using dependency relations. In *SIGIR*, pages 400–407.

Cícero Nogueira dos Santos, Ming Tan, Bing Xiang, and Bowen Zhou. 2016. Attentive pooling networks. *arXiv*, abs/1602.03609v1.

Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. 2015. Applying deep learning to answer selection: A study and an open task. In *ASRU*, pages 813–820.

Yoav Goldberg. 2017. *Neural Network Methods for Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Hua He and Jimmy Lin. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *HLT-NAACL*, pages 937–948.

Hua He, Kevin Gimpel, and Jimmy Lin. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *EMNLP*, pages 1576–1586.

Michael Heilman and Noah A. Smith. 2010. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *HLT-NAACL*, pages 1011–1019.

Peter Henderson, Riashat Islam, Phillip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning that Matters. *arXiv*, abs/1709.06560v1.

Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig Macdonald, and Sebastiano Vigna. 2016. Toward reproducible baselines: The open-source IR reproducibility challenge. In *ECIR*, pages 408–420.

Yishu Miao, Lei Yu, and Phil Blunsom. 2016. Neural variational inference for text processing. In *ICML*, pages 1727–1736.

Gina Moraila, Akash Shankaran, Zuoming Shi, and Alex M. Warren. 2014. Measuring reproducibility in computer systems research. Technical report, University of Arizona.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024

Joakim Nivre. 2017. Challenges for ACL: ACL Presidential Address 2017. https://www.slideshare.net/aclanthology/joakim-nivre-2017-presidential-address-acl-2017-challenges-for-acl/. Accessed: 20 Sept. 2017.

Stephen K. Park and Keith W. Miller. 1988. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201.

Thomas Pfeiffer and Robert Hoffmann. 2009. Large-scale assessment of the effect of popularity on the reliability of research. *PLOS One*, 4(6):e5996.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2004. Mapping dependencies trees: An application to question answering. In *Proceedings of AI&Math 2004*, pages 1–10.

Jinfeng Rao, Hua He, and Jimmy Lin. 2016. Noise-contrastive estimation for answer selection with deep neural networks. In *CIKM*, pages 1913–1916.

Jinfeng Rao, Hua He, and Jimmy Lin. 2017. Experiments with convolutional neural network models for answer selection. In *SIGIR*, pages 1217–1220.

Brandon Reagen, Robert Adolf, Paul N. Whatmough, Gu-Yeon Wei, and David M. Brooks. 2017. *Deep Learning for Computer Architects*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers.

Nils Reimers and Iryna Gurevych. 2017. Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks. *arXiv*, abs/1707.06799v1.

Royal Sequiera, Gaurav Baruah, Zhucheng Tu, Salman Mohammed, Jinfeng Rao, Haotian Zhang, and Jimmy Lin. 2017. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. *arXiv*, abs/1707.07804v1.

Aliaksei Severyn and Alessandro Moschitti. 2013. Automatic feature engineering for answer selection and extraction. In *EMNLP*, volume 13, pages 458–467.

Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *SIGIR*, pages 373–382.

Eyal Shnarch. 2013. *Probabilistic Models for Lexical Inference*. Bar Ilan University.

Julien Simon. 2017. Keras shoot-out: TensorFlow vs MXNet. https://medium.com/@julsimon/keras-shoot-out-tensorflow-vs-mxnet-51ae2b30a9c0. Accessed: 5 Sept. 2017.

Ming Tan, Bing Xiang, and Bowen Zhou. 2015. LSTM-based deep learning models for non-factoid answer selection. *arXiv*, abs/1511.04108v4.

Zhiguo Wang and Abraham Ittycheriah. 2015. FAQ-based question answering via word alignment. *arXiv*, abs/1507.02628v1.

Shuohang Wang and Jing Jiang. 2016. A compare-aggregate model for matching text sequences. *arXiv*, abs/1611.01747v1.

Mengqiu Wang and Christopher D. Manning. 2010. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *COLING*, pages 1164–1172.

Di Wang and Eric Nyberg. 2015. A long short-term memory model for answer sentence selection in question answering. In *ACL*, pages 707–712.

Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. 2007. What is the jeopardy model? A quasi-synchronous grammar for QA. In *EMNLP-CoNLL*, volume 7, pages 22–32.

Bingning Wang, Kang Liu, and Jun Zhao. 2016a. Inner attention based recurrent neural networks for answer selection. In *ACL*, pages 1288–1297.

Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. 2016b. Sentence similarity learning by lexical decomposition and composition. In *COLING*, pages 1340–1349.

Nathan Whitehead and Alex Fit-Florea. 2011. Precision & performance: Floating point and IEEE 754 compliance for nVidia GPUs. Accessed: Sept. 7 2017, from https://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *EMNLP*, pages 2013–2018.

Liu Yang, Qingyao Ai, Jiafeng Guo, and W. Bruce Croft. 2016. aNMM: Ranking short answer texts with attention-based neural matching model. In *CIKM*, pages 287–296.

Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. 2013. Answer extraction as sequence tagging with tree edit distance. In *HLT-NAACL*, pages 858–867.

Scott Wen-tau Yih, Ming-Wei Chang, Chris Meek, and Andrzej Pastusiak. 2013. Question answering using enhanced lexical semantic models. In *ACL*, pages 1744–1753.

Wenpeng Yin and Hinrich Schütze. 2017. Task-specific attentive pooling of phrase alignments contributes to sentence matching. In *EACL*, pages 699–709.

Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. 2016. ABCNN: Attention-based convolutional neural network for modeling sentence pairs. *TACL*, 4(1):259–272.

Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. 2014. Deep learning for answer sentence selection. *arXiv*, abs/1412.1632v1.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00018/1567606/tacl_a_00018.pdf by guest on 18 July 2024