Hyper-parameter tuning and empirical evaluation in Natural Language Processing

Marco Concetto Rudilosso, Sadir Abdul Hadi, Sachchit Prasad, Pranav Nashikkar, Miron Zelina

***Abstract*—As the field of Natural Language Processing (NLP) continues to mature, with more claimed state-of-the-art results, it has become increasingly difficult to discern between model architecture strength and irrelevant factors. We see our works being developed into a fully integrated framework, which would allow us to compare architectures on different data-sets with automatic hyper-parameter tuning. We introduce a systematic and reproducible method, which allows us to reduce bias. Our black-box optimisation not only leads to computational speedup, but also achieves stronger results than the word2vec default hyper-parameters; achieving a significant 15.9% improvement on a word similarity evaluation metric. We aim to usher in a new trend where future research has greater emphasis on careful empirical experimentation, leading to superior architectures.**

I. Introduction

NATURAL Language Processing, sitting at the intersection of computer science, artificial intelligence and computational linguistics, captures the essence of human language to computationally interpret, manipulate and generate natural language; essentially finding how computers can derive useful meaning from it. The human brain itself is not well understood in functions such as thought processes and manipulation of information, meaning that replicating such behaviour in machines poses difficult challenges. As a result, the current state-of-art is often not competitive with human performance.

How does one represent something as fundamental as a word? Finding a method by which to represent words such that they encode as much information as possible is one of the key factors pursued in the field.

Following the works of Mikolov et al. [1], and their word2vec system, there has been a flurry of research into the use of word embeddings for NLP tasks. A word embedding is a mathematical object, often taking the form of a vector, which has an association with that particular word. Each dimension has a value which corresponds to a feature of that word and can convey both grammatical and semantic interpretations.

It is difficult to develop a metric for the quality of word embeddings which constitutes good understanding for a word. Furthermore, different NLP domains may require different syntactic and semantic properties, making it challenging to find an all-encompassing evaluation task.

With a steady influx of 'state-of-the-art' results on language modelling benchmarks, an issue which has arisen is that many of these methods have been evaluated on different data-sets and with varied computational resources. Inaccuracies in reported results are significant because as the field matures, it is hard to make significant gains in successive proposed models and as gains become progressively smaller, it becomes difficult to distinguish these from irrelevant factors.

Model hyper-parameters are often poorly controlled, making them a great contributor to such errors. Using the right set of hyper-parameters radically changes the ranking between models when making comparisons, however the process of tuning these can be time consuming, especially if the computational resources available are low. Claims of results therefore must be reconsidered due to this large degree of experimental variation - making it harder to discern what constitutes the true state-of-the-art.

The solution which we propose makes it simpler and more reliable to compare and contrast models given the biases and noise which are inherently present. We draw upon advances from Melis et al. [2] who demonstrated the value of controlling for hyper-parameters and found that simpler, traditional models outperformed more complex models which were thought to be superior. Thus we also aim to automatically find an optimal hyper-parameter configuration for our model and allow for a set of custom evaluation tasks to capture more syntactic and semantic properties.

To establish the effectiveness of the framework, we train word2vec on a corpus of text for a given evaluation task. The original hyper-parameters for word2vec are compared to our experimental method which uses black-box tuning. The aims of this are to explore the effects of black-box hyper-parameter optimisation on the results of the model. This data-driven method allows us to more systematically identify optimised hyper-parameters, which may constitute a general improvement on the embeddings or an improvement for evaluation tasks.

Our work allows for the replication of the experimental methodology for optimising hyper-parameters on different tasks for analysis, resulting in a general purpose framework for any configuration.

II. Related Work

A. Word Embeddings

When we solve a problem using computational techniques, such as neural networks, representing input in a mathematical form is a necessity. Dealing with large vocabularies, it can be seen there is not enough information to capture everything which is said with it, meaning that there are many real phrases which are never observed in the training data. These issues can be seen in some of the representations which were used earlier, ranging from cluster-based methods [3] to distributional methods, which were memory-inefficient [4].

In order to improve the efficiency of word representations, researchers have developed low-dimensional and dense vectors, called word embeddings. They have multiple advantages,

including the need for less computational power, and a greater generalisation power, which signifies that information can be more easily transferred between similar words [5]. Our work makes use of one specific method of obtaining word embeddings: word2vec.

Word2vec, introduced by Mikolov et al. in 2013 [1], is a popular program for generating word embeddings that uses the Skip-Gram with negative-sampling (SGNS) algorithm, which tries to maximise the probability of predicting the context around a specific word. Despite word2vec's success in terms of efficiency and expressive power, it has been criticised for being somewhat obscure and not well-grounded, especially when it comes to formulas which were found heuristically [6]. Furthermore, the model's hyper-parameters were manually tuned, and the justification behind it was that "the choice of the training algorithm and the hyper-parameter selection is a task specific decision, as we found that different problems have different optimal hyper-parameter configurations" [1]. We therefore see an opportunity here to develop a system which enables us to set the hyper-parameters in an optimal way depending on the task.

B. Hyper-parameters and Optimisation

1) Hyper-parameters: A hyper-parameter is a parameter of the model itself (e.g. number of layers, type of activation function, number of units in a layer), in contrast to model parameters which are learned directly from data and change during training (e.g. the weights in a neural network). The choice of these hyper-parameters can strongly affect results, but is often an inadequately controlled source of variation, that can lead to empirically unsound claims thus obscuring the underlying architecture strength.

2) Bayesian Optimisation: The optimisation of hyper-parameters is difficult, as it is not a differentiable or convex problem, but they must still be tuned for optimal performance. Standard approaches include grid searches, and human intuition which can be inefficient and prone to bias respectively. A better solution would be a black-box optimisation method such as Bayesian Optimisation. This involves sampling the function, and using knowledge gained about it with each iteration to try to work out the global structure (and optima). Rather than evaluating the true function, which can be expensive (e.g. fully training a neural network), a proxy function is used to approximate it and optimise in its place. Using this, one can efficiently explore the optimal hyper-parameter search space while retaining strong performance by making use of all knowledge to make the best guess for each sample.

3) Parallelising Bayesian Optimisation: Parallel computing is all-pervasive, and we would be wise to reap its benefits for a massive computational speedup in our system. Such a speedup can help to increase overall performance and accuracy as more sophisticated models can be tested, and the convergence to the optima can be faster. Desautels et al. [7] outline a Bayesian optimisation method in which they apply a Gaussian process to each sample point, with the assumption that this will approximately fit the true function, with each sample reducing the potential variance around the samples. This process is parallelised by batching these samples and running experiments in parallel; this does mean that each point only considers the information from the previous batch rather than all points, leading to slower propagation of knowledge.

4) Vizier: A system which becomes easier to experiment on rather than understand becomes more akin to that of a black-box. Vizier [8] is a tool which uses black-box optimisation to tune hyper-parameters automatically. Using techniques such as automated stopping on the performance Curve Stopping Rule, it achieves optimality gaps similar to those without the stopping rule, using 50% fewer CPU-hours for hyper-parameter tuning for deep neural networks.

C. Previous Systems

Several previous works have also been conducted in a similar manner to our own, but did not provide the features that we sought.

1) MLComp: MLComp was an open platform for testing data-sets and algorithms [9]. New models could be uploaded and tested against new or existing data-sets, then would be freely executed, and the results, algorithms, and data would all be publicly available for further re-use. The system supported different types of tasks to ensure compatibility between models and data-sets, as well as custom metrics. The automation of hyper-parameter refinement and reproducibility is lower than our system, as only grid search was natively supported and the lack of containerisation due to the age of the system introduced another potential source of variation; especially in the long term as libraries are modified and optimised. As of March 1st 2018, MLComp is no longer online.

2) CodaLab: CodaLab is a system designed to enable research to be easily distributed in a notebook style environment and is the successor to MLComp [10]. Experiments are run inside a docker image, and the server automatically assigns workers to unexecuted tasks, with support for parallel runs (even ones that have other runs as dependencies). The system is similar to ours, but less focused; our approach is to have full scale automation of experiments, hyper-parameter tuning and Bayesian optimisation, with support for plugging in evaluations. In contrast, CodaLab focuses on integrating containers with a notebook style document, to encapsulate specific research papers, without as much focus on automatic hyper-parameter tuning.

III. System Overview

In this section we will discuss the design and the structure of the system that we have developed for performing hyper-parameter optimisation on word-embedding models.

A. The Goal and the Context

The users of the system should be able to easily set up experiments to tune the hyper-parameters of a specific model, trained on a user-specified data-set, on a specific evaluation task. The system should facilitate exploration of unseen combinations and better understand the effects on one another.

Ideally our system should:

1) Allow users to specify a data-set to be trained on as well as a model to train and an evaluation task which the system will tune hyper-parameters towards
2) Allow users to specify which hyper-parameters they want to be tuned
3) Run multiple jobs at a time
4) Run the training and evaluation jobs in a loop, to tune the hyper-parameters

The computational resources that we had access to were:

- A simple virtual machine (VM) running Linux
- A high-performance computing (HPC) heterogeneous cluster that could schedule our most expensive jobs

This specific set-up has allowed us to run a scheduling server on the VM that would create and en-queue jobs, like training the models and evaluating them, on the HPC cluster, so that we could harness the ability to run multiple resource-heavy jobs in parallel, with high-performance hardware.

In order to meet the desired requirements for our system, we had to overcome different engineering challenges:

1) Making it easy for the end user to use dependencies that are not already available and installed on the cluster.
2) Creating a future-proof API on top of an old job scheduling system. This is due to the nature of the software that allows users to schedule jobs on the cluster. It requires them to create an SSH connection and then submit a job in the form of a Bash script, with a special syntax for comments that allows them to specify the system requirements, such as RAM and disk storage.
3) Handling the unreliability of the cluster, which does not always provide a perfectly clean environment. In fact, jobs can fail for a variety of reasons, such as not having enough disk storage or not being allocated as much RAM as expected.

B. The Solution

This section will cover the design of our system, exploring the reasons behind specific decisions, as well as the new possibilities that our work may create. We will discuss how we have overcome the aforementioned problems and how our system can be used in a range of different scenarios.

1) The Dependencies Problem: A recurring issue in software development, is the one of having reproducible and portable software. This is particularly problematic in our case, as we want to offer our users the possibility to use the technology stack they prefer, but the HPC cluster nodes that will run the experiments might not have all the dependencies they need.

Predicting the need of all the possible experiments would be an impossible task, thus we have decided to make our users define the environment of their experiments using Singularity containers [11], a containerisation technology that has been developed with HPC clusters in mind.

This allows us to offer our users as much freedom as we can and it makes sure that when running the same experiments multiple times, the context in which it will run will always have the right dependencies.

2) A programmatic approach for HPC job scheduling: The current solution, installed on the HPC cluster that we had at our disposal, exposes an interface for submitting jobs that highly relies on human intervention. In order to submit a job, one has to first establish a remote connection with a cluster log-in node, through SSH, and then submit a job by using a CLI command that accepts a Bash script, whose contents are: the requirements of the job, such as the maximum run-time, RAM and storage needed, and the job itself.

Controlling the state of a job involves as much human interaction as submitting one, thus making it difficult to use the current system to run a set of experiments in a loop.

In order to overcome this issue, we have created our own scheduling service, which is a persistent server on a separate VM. In order to schedule an experiment to tune the hyper-parameters of some model, the user sends an HTTP request to our service, containing a set of information in JSON form.

The user can specify a variety of parameters for the experiment they want to run, such as the training and the evaluation Singularity containers to run, the amount of RAM, disk and run-time they need and the hyper-parameters they want to tune.

3) Making our System More Reliable: Even with the system that we have just described, user interaction is still needed when an error occurs during the execution of a job on the cluster. The causes can be of different nature, one of which is the cluster itself. It may happen that the node your job is scheduled on does not have the amount of disk storage or RAM you have asked for, thus adding a source of unreliability in our system.

In our effort to minimise the user's need to control the status of their experiments, we have developed InVita, a CLI command that runs any command that you would run in a terminal, but it also checks its status and reports errors to the main scheduling service, so that it can restart a job.

4) The Tuning: With the different parts that we have so far described, we are able to easily schedule jobs on the cluster and check for errors, thus making it more reliable to run experiments. Nonetheless, this system would not be able to accomplish our main goal, tuning the hyper-parameters for word-embedding models.

In order to perform black-box optimisation we have decided to create a JSONRPC wrapper on top of Spearmint [12], a library to perform Bayesian optimisation. We have done this because the out of the box version of this software does not provide a programmatic interface that can be used to start interact with the optimiser. Thanks to our work, we can now setup a new experiment, deposit results and get new hyper-parameters through a simple API.

C. The System at a Glance

Our final system (refer to Fig. 1) schedules training and evaluation tasks in a containerised manner, while utilising Spearmint to perform Bayesian optimisation. The system runs the tasks in parallel whenever possible to accelerate the rate at which results are obtained.

In order to start an experiment the user sends a HTTP request to our scheduling service, specifying the parameters for their experiment.

When the system receives the request, it queries Spearmint for a configurable number of hyper-parameters and it will then proceed to schedule on the HPC cluster a number of different training jobs. When one of them finishes, an evaluation job is scheduled, at the end of which the scheduling service will notify Spearmint with the results and get a new set of hyper-parameters for which it will start another train-eval cycle.

Every time a job finishes or produces an error, the InVita CLI sends its data to the scheduling service.

D. Implementations Used

- We have used a publicly available implementation of word2vec from: https://github.com/dav/word2vec.
- We have also used an open-source implementation for evaluating on the SimLex dataset: https://github.com/nmrksic/eval-multilingual-simlex.
- Our Bayesian Optimisation library was Spearmint, available from: https://github.com/JasperSnoek/spearmint.
- The preprocessing script for our text corpus is located here: http://mattmahoney.net/dc/textdata.html.

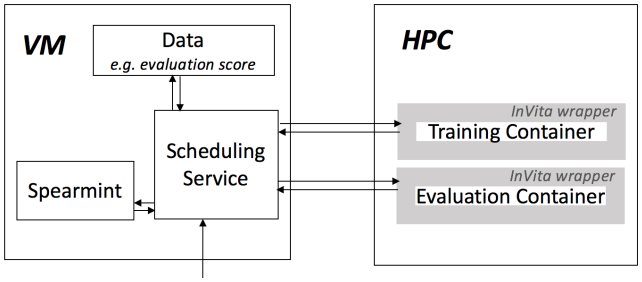


Fig. 1: The system architecture as described earlier

IV. Experimental Setup

To run our experiments, a suitable text corpus and evaluation task had to be chosen. In order to produce meaningful results, we have chosen tasks and corpora used widely across different domains.

One such popular benchmark for evaluating word embeddings is the word similarity evaluation task. This task evaluates word embeddings based on their ability to group similar words together in the output vector space. For example, the words ”boat” and ”ship” should have vectors which are not too far apart (using some distance measure).

Specifically, we chose the SimLex-999 data-set [13], for its ability to capture a wide range of semantic similarities, and also its separation of word similarity and word relatedness, which recent work has discussed as a possible source of bias [14]. We split the original SimLex-999 data-set into a 2:1 split for the purposes of tuning the hyper-parameters (SimLex-666, dev set) and for the purposes of final testing (SimLex-333, test set). The cosine distance of word vectors is commonly used as the similarity measure, and we have also used this method. The Spearman rank correlation coefficient (implementation link mentioned above) was used as the metric to tune on, as it provided a single real number in the range $[-1, 1]$, which could be returned directly to Spearmint.

As our text corpus, the first two gigabytes of the 20171201 Wikipedia English dump have been used. This dump was then processed using the same script as Mikolov et al. [1] have used (can be found above), which removes all formatting, converts every word to lowercase and exchanges digits for their word representations.

Ideally, we would explore all points in the hyper-parameter space, however due to practical, computational constraints, this is not possible - as even by conservative estimates, exploring the full space would take hundreds of years of compute time. Bayesian optimisation is an efficient method for exploring the search space. For this reason, Spearmint was used to optimise using the results of each run (262 in total) and allowed us to locate the optimum in the hyper-parameter search space. The results, hyper-parameters and vectors were stored at the end of each run for further analysis.

V. Results

A. Evaluations Over Time

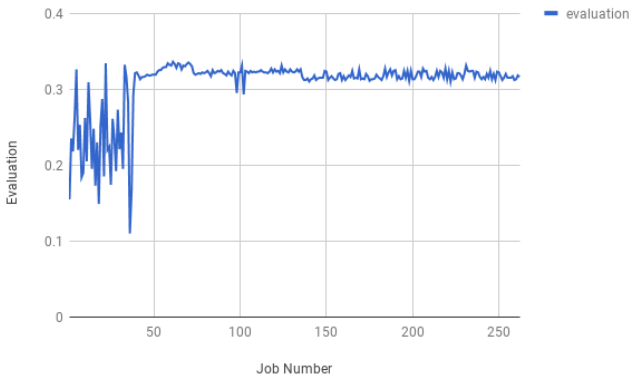
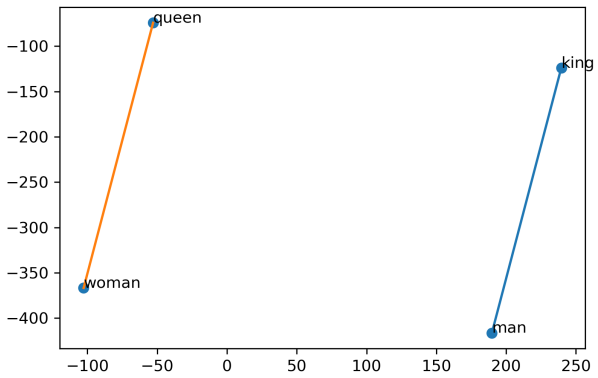
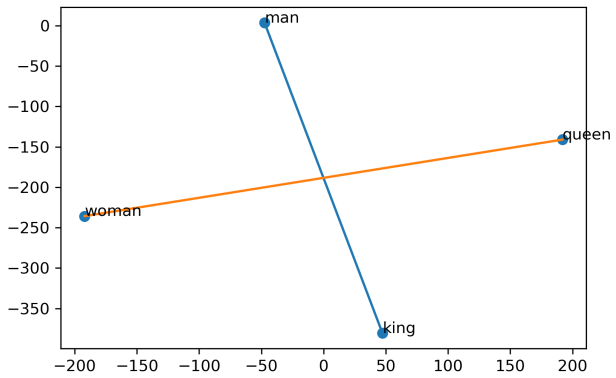


Fig. 2: The performance of 262 iterations over time, as Spearmint attempts to find the optimal word2vec hyper-parameters

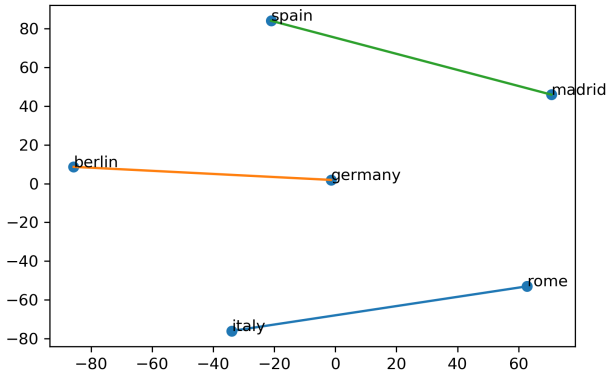
Through the use of our system, we were able to successfully achieve a significant edge over the defaults provided by Mikolov et al. [1] (albeit the defaults were for a larger version of our data-set), an improvement of 16.3%. Thus the system is clearly capable of achieving comparable if not superior results to a human experimenter, but achieves this in a repeatable and systematic manner. Furthermore, our results show that word2vec is strongly sensitive to hyper-parameter configurations. Using our system, it took several days to run the experiment and would be difficult to do this without proper computational resources, thus leading to impropriety in model evaluation. This is proven by the large range of different results that the model achieves in our experiment, with a low of 0.11 and a high of 0.336 (a factor of 3 difference). The wide range of results achieved and the superior performance to the word2vec defaults demonstrate the importance of careful hyper-parameter tuning.



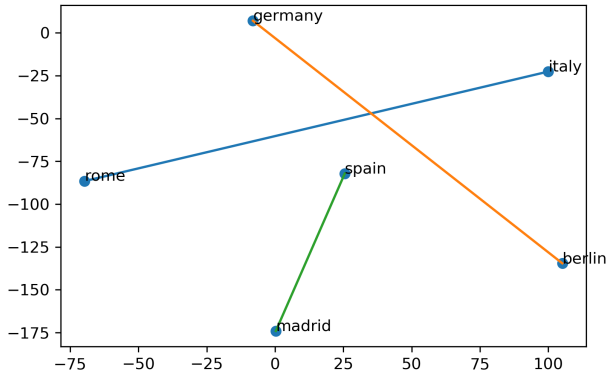
(a) King and queen visualisation with best performing vectors



(b) King and queen visualisation with worst performing vectors



(c) State and capital visualisation with worst performing vectors



(d) State and capital visualisation with best performing vectors

Fig. 3: Using t-SNE to visualise the relationships between pair of words. The figures illustrate the difference between the worst and best word embeddings vector that the experiment produced. It is clear that the best result is much better at representing the relationship between correlated words.

Figure 2 displays properties of finding a local maximum, with all values being in the 0.31-0.34 range after about 40 iterations, suggesting the framework is now choosing to exploit what it knows rather than further explore the search space. A strong result was achieved early on, but the system continued to try other areas to try and discover a global optimum, demonstrated by the high variance section at start, while the hyper-parameter space was being mapped out. In this case, only minor refinements were achieved over the early successful results (best run was only 0.01 better than 6th run), but it is possible for significant gains to be made at this stage for some tasks.

Figure 3 (on the next page) clearly displays Mikolov et al.'s much lauded linear relationships between word vectors on our best run and the distinct lack of relation on our worst. Through use of our system we have been able to prove that the ability to learn the implicit relationships between words is highly dependent on these hyper-parameters. Our system was also able to achieve an improvement of over a factor of 2 from the initial to the best result.

B. Visualising High-Dimensional Space

The hyper-parameter space can vary in 6 dimensions, making it extremely difficult to analyse its topography to understand what precisely the system has done. To this end, there exist techniques for dimensionality reduction that aim to preserve as much original information as possible but transform the data into a lower dimensional space. Principal Components Analysis (PCA) [15] is one such method, which works by calculating the eigenvectors and values of the data matrix, and projects the data onto the selected number of reduced orthogonal dimensions using them, reducing variation. For best performance, data should first be normalised so that the scale of the dimensions does not skew the output.

C. PCA Hyper-parameter Visualisation

Figure 4 shows the system at work; the spread of points at the bottom shows the exploration phase as it attempts to map out the search space. The slope towards the optimum is where the system chose to exploit its knowledge, with the concentration of points near the maximum. This suggests that

it was correctly able to identify and make use of the strongest results.

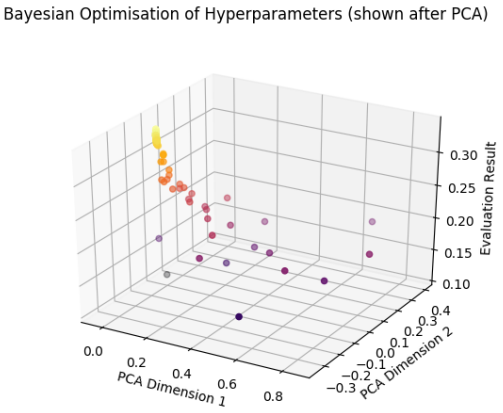


Fig. 4: The results of the experiment after the hyper-parameters have been normalised and dimensionally reduced to 2. This has an explained variance ratio of 0.968, so is a good representation of the higher dimensional data. Brighter points correspond to better evaluation results.

D. Accounting for Time

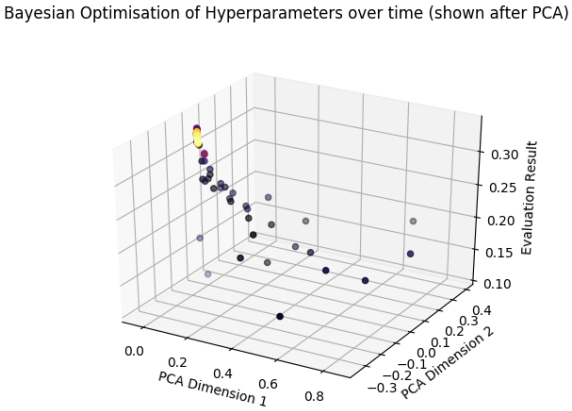


Fig. 5: The graph with colours at each point according to when it was evaluated, with brighter colours signifying later runs.

Figure 5 shows the evolution of the framework's knowledge of the search space over time, with an early discovery of the optimum as the dark slope upwards shows. Despite this, the lighter spots scattered around the space show that it still attempted to look for other potential avenues but ended up focusing its efforts near the maximum. The dark spot at the very top of the graph seems to suggest it has found a local optimum; it did not beat this result even with many subsequent runs having exhausted this area of space.

E. Parallel Coordinates Visualisation

The parallel coordinates visualisation (refer to Fig. 6 is another way of analysing high dimensional spaces [17]. The

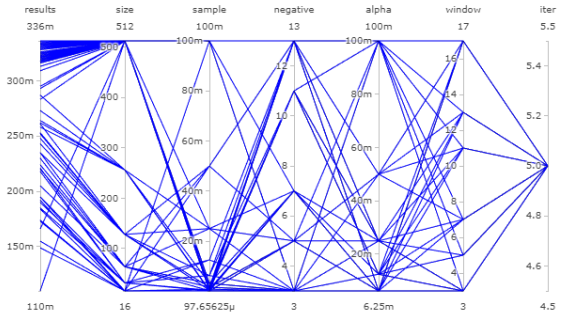


Fig. 6: The parallel coordinates visualisation of the hyper-parameters and results. Each axis represents a dimension with each line being a single run. Built with plotly [16] N.B. 'm' refers to milli prefix.

strongest runs all use the maximum "size" (vector dimensionality), and most of these also use the lowest "sample" (word occurrence threshold for data-set inclusion), which suggests that our results are being artificially bounded by our own computational constraints. It would be worth investigating values beyond our bounds for further potential evaluation gains.

F. Varying Vector Dimensionality

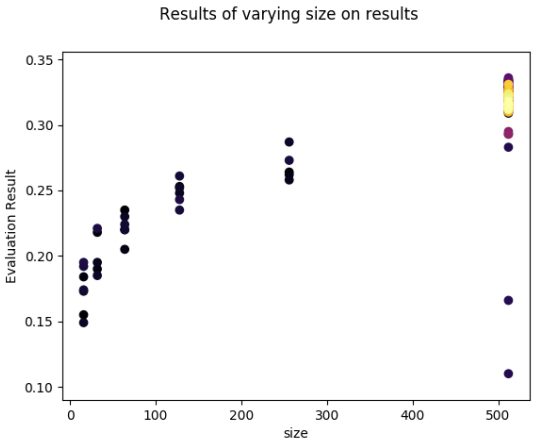


Fig. 7: The effects of varying dimensionalities for the word vectors on the results. Brighter colours represent later runs.

Taking a closer look at the effects of varying size on results in Figure 7, word2vec seems particularly sensitive to size with no singular other hyper-parameter exhibiting such a clear relationship. This is not surprising; higher dimensionality word embeddings theoretically allow for more information. Since only a discrete set of values were allowed, the curve is not smooth, but a distinct pattern seems present that appears to be logarithmic in nature. Intuitively this appears reasonable, more information density per word has diminishing returns in terms of importance.

This graph also demonstrates Bayesian optimisation working well; the earlier darker points in sub-optimal regions, with the bright concentration of points at the optimum. Our system has quickly identified and exploited this hyper-parameter, quickly encountering our own hard limit.

G. Evaluation Result Distribution

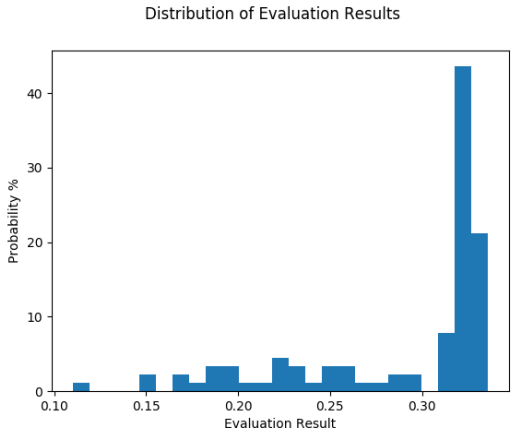


Fig. 8: The distribution of evaluation results

Figure 8 displays the concentration of results around the optimum, with a roughly uniform distribution elsewhere. The system explored a range of values, and as Fig. 4 shows, focused on the single most promising area. It is unclear whether this is due to the search space actually being somewhat convex or whether the optimiser could be tweaked to explore more before specialising.

VI. Future Works

Our work has great potential for expansion in multiple different areas. The system itself can be reused and modified, while the experiments and results can offer new avenues for further research and inquiry.

Our results show that Bayesian optimisation performs very well, even outperforming the default word2vec hand-tuned hyper-parameters on our data-set. Further runs and experiments would help to investigate and further corroborate the consistency of the framework. The 6^{th} run happened to be an extremely strong result, leading us to question how performance would look if a very strong result wasn't achieved early on and whether the system would continue to explore sufficiently to try and find the same (or better) optimum anyway. This further led to the question of how important the initialisation values were.

To examine the convexity of our specific search space, a grid search would be interesting to map out as much as we could compute and compare which path our optimiser followed using the true topography to ascertain its optimality. Techniques from computer vision exist to reduce the computation required while achieving higher resolution imagery, with geometric interpolation of the search surface based on polygonal reconstruction techniques [18]. Another viable method would be to merely adapt our Gaussian process based system's acquisition function to always sample the highest variance point; thus leading to as high a resolution a map as possible with constrained computation.

Our work has also presented a clear trend with the word2vec model; word dimensionality and word occurrence thresholds were hitting our cap with strong runs being associated with high and low values of these dimensions respectively. By removing our limits and starting our system with current positions, perhaps it is the case that we have only reached an artificial local optimum and the true value could still be improved. An investigation into a mechanism to automatically push the limit somewhat would be worthwhile, to investigate the idea of a 'softer' limit or none at all to make such a scenario less likely while preserving much of the computational speedup of limits by only doing so if necessary.

The system was designed to be extensible, resulting in many modular parts of the system which could be changed as required to build upon existing results. The main areas where changes can be explored include the evaluation tasks, data-sets used, the training models employed and being able to decide which hyper-parameters are tuned.

Since the system has made use of containerisation, it is a relatively simple task to add further downstream evaluation tasks such as Named-Entity Recognition (NER) or Natural Language Inference [19] - as extensive further setup is not required to test these new metrics. In fact, task-based evaluation has been a topic of interest recently, and multiple papers have explored the performance of models in particular tasks, such as NER [20]. Nayak et al. [21] developed a system which produces reports to highlight the comparative performance of some models in different tasks. Our system can complement their work, as their paper highlights that they have overcome the problem of tuning hyper-parameters by limiting their number.

With regards to data-sets, the experimental setup used in this project only made use of a snapshot from the Wikipedia text corpus for training, however, a larger and topically different corpora could be used and the effect of the changing sizes of the corpora could be explored. It has often been found that the data-sets used in machine learning techniques have a major impact upon the final result [22] and often can improve performance when being modified. The framework would allow more work to be conducted in this space by better isolating the effects of the data-set from the noise. Another interesting avenue to explore with our tool would be the effect of having different types of data-sets for different tasks. It has been shown, for example, that we can create equally good representations of clinical terms whether we use clinical notes or bio-medical research papers [23]. Further research on similar questions can be supported by our system.

The current training model used in the existing setup was word2vec. This may be switched out for other training models which would allow for comparisons to be made between their algorithms and drawing parallels between the hyper-parameters used, as Berardi et al. have tried to do in their paper [24].

While these suggestions comprise of possible ways to use the framework and its "plug-and-play" nature to explore how

other components behave - other extensions would look at how the framework as a whole can be used and for which purposes the results would be useful.

The availability of a standardised evaluation tool, bearing similarities to the one proposed by Nayak et al. [21] would enable the research community to compare results to a better degree of certainty and make it easier to validate the results of their work.

As Melis et al. have mentioned in their paper [2], the difference between varying experimental set-ups has made comparisons between models difficult - using the existing framework provided would allow for further research into comparison between models and their uses. Since it has been observed that different data-sets require different hyper-parameters, being able to find the best range of hyper-parameters to use classified by the model and also the purpose of its use would constitute a worthwhile avenue to explore.

With successive gains between models becoming smaller, this allows for one way with which to control for the problematic hyper-parameter noise. Melis et al. [2] have highlighted that standard LSTMs could be used over other state-of-the-art methods to provide more reliable results with hyper-parameter tuning - using our framework would allow for more further formal work to be completed in this space.

VII. Conclusion

Our work has two main contributions - we demonstrate the value of using Bayesian optimisation for hyper-parameter tuning which results in a computational speedup and a reduction in bias. Furthermore, we propose a framework with which to evaluate architecture performance on a variety of different data-sets in a systematic manner.

Our experiment involved training word2vec on the Wikipedia corpus and Simlex. After each run, Spearmint optimised the search space to identify the most promising hyper-parameters. We had 262 runs of this loop - during which significant improvement was made on evaluation from 0.155 to 0.366; a significant edge (+0.046) over Mikolov et al.'s word2vec default.

We observed that the optimiser performed a broad exploration of the search space, before exploiting the information gleaned. Therefore, Spearmint is more likely to find a global optimum, while also not missing out on opportunities for final incremental improvements.

Interesting avenues to investigate would include rerunning our experiments multiple times to see if Bayesian optimisation is constrained by local optima and how effectively it can explore the search space.

By testing different models and data-sets, it would bring us closer to being able to identify more optimal solutions to different NLP scenarios. Therefore, we expect more breakthroughs like Melis et al. [2], which will dramatically change the landscape of machine learning and Natural Language Processing.

Our experimental setup from this report, can be found at https://gitlab.com/ninjin/comp3096_2018.

References

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[2] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," *arXiv preprint arXiv:1707.05589*, 2017.

[3] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: A simple and general method for semi-supervised learning," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ser. ACL '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 384–394. [Online]. Available: http://dl.acm.org/citation.cfm?id=1858681.1858721

[4] T. Kober, J. Weeds, J. Reffin, and D. J. Weir, "Improving sparse word representations with distributional inference for semantic composition," *CoRR*, vol. abs/1608.06794, 2016. [Online]. Available: http://arxiv.org/abs/1608.06794

[5] Y. Goldberg, "A primer on neural network models for natural language processing," *CoRR*, vol. abs/1510.00726, 2015. [Online]. Available: http://arxiv.org/abs/1510.00726

[6] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *CoRR*, vol. abs/1402.3722, 2014. [Online]. Available: http://arxiv.org/abs/1402.3722

[7] T. Desautels, A. Krause, and J. W. Burdick, "Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3873–3923, 2014.

[8] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1487–1495.

[9] P. Liang and J. Abernethy, "Mlcomp - home," 2018. [Online]. Available: http://mlcomp.org/

[10] P. Liang, I. Guyon, E. Viegas, S. Escalera, X. Solé, and E. Carmichael, "Codalab," 2018. [Online]. Available: http://codalab.org/

[11] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PloS one*, vol. 12, no. 5, p. e0177459, 2017.

[12] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[13] F. Hill, R. Reichart, and A. Korhonen, "Simlex-999: Evaluating semantic models with (genuine) similarity estimation," *Computational Linguistics*, vol. 41, no. 4, pp. 665–695, 2015.

[14] E. Agirre, E. Alfonseca, K. Hall, J. Kravalova, M. Paşca, and A. Soroa, "A study on similarity and relatedness using distributional and wordnet-based approaches," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 19–27.

[15] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, p. 559–572, 1901.

[16] P. T. Inc. (2015) Collaborative data science. Montreal, QC. [Online]. Available: https://plot.ly

[17] J. Heinrich and D. Weiskopf, "State of the art of parallel coordinates." in *Eurographics (STARs)*, 2013, pp. 95–116.

[18] E. Boyer and S. Petitjean, "Curve and surface reconstruction from regular and non regular point sets," *Computational Geometry*, vol. 19, p. 101–126, 2001.

[19] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims, "Evaluation methods for unsupervised word embeddings," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 298–307.

[20] M. Seok, H.-J. Song, C. Park, J.-D. Kim, and Y.-S. Kim, "Comparison of ner performance using word embedding," pp. 784–788, 12 2015.

[21] N. Nayak, G. Angeli, and C. D. Manning, "Evaluating word embeddings using a representative suite of practical tasks," *ACL 2016*, p. 19, 2016.

[22] P. Kaushik Mudrakarta, A. Taly, M. Sundararajan, and K. Dhamdhere, "It was the training data pruning too!" *ArXiv e-prints*, 2018.

[23] S. V. S. Pakhomov, G. P. Finley, R. McEwan, Y. Wang, and G. B. Melton, "Corpus domain effects on distributional semantic modeling of medical terms," *Bioinformatics*, vol. 32 23, pp. 3635–3644, 2016.

[24] G. Berardi, A. Esuli, and D. Marcheggiani, "Word embeddings go to italy: A comparison of models and training datasets," in *IIR*, 2015.