How Well Can a Genetic Algorithm Fine-tune Transformer Encoders? A First Approach

Vicente Ivan Sanchez Carmona and **Shanshan Jiang** and **Bin Dong**
Ricoh Software Research Center (Beijing) Co., Ltd
{Vicente.Carmona, Shanshan.Jiang, Bin.Dong}@cn.ricoh.com

Abstract

Genetic Algorithms (GAs) have been studied across different fields such as engineering or medicine to optimize diverse problems such as network routing, or medical image segmentation. Moreover, they have been used to automatically find optimal architectures for deep neural networks. However, to our knowledge, they have not been applied as a weight optimizer for the Transformer model. While gradient descent has been the main paradigm for this task, we believe that GAs have advantages to *bring to the table*. In this paper, we will show that even though GAs are capable of fine-tuning Transformer encoders, their generalization ability is considerably poorer than that from Adam; however, on a closer look, GAs ability to exploit knowledge from 2 different pretraining datasets surpasses Adam's ability to do so.

1 Introduction

Genetic Algorithms (GAs), a set of optimization methods, although widely studied in other fields such as electric engineering (Li and Ge, 2009; Sainath et al., 2021) or medicine (Ghosh et al., 2016), have not played a big role in the field of NLP as gradient descent algorithms. Indeed, a disadvantage of GAs is high running times when the search space is big. However, GAs possess advantages that make us reconsider their usefulness in NLP such as 1) algorithmic simplicity, 2) no vanishing- or exploding-gradient problems since no gradient signal is necessary, and 3) any mathematical expression can be optimized, such as Accuracy.

On the other hand, gradient descent approaches such as Adam (Kingma and Ba, 2015) are widely used not only due to the high fine-tuning scores they achieve for NLP models, but also due to a common –and barely challenged– assumption that prevails in the NLP field: fine-tuning a Transformer encoder that has been pretrained on two datasets will lead to considerably better scores than fine-tuning an encoder that was pretrained on either of the two pretraining datasets since, according to scaling laws (Kaplan et al., 2020), it is assumed that the former encoder has learned linguistic and (or) world knowledge from the two datasets, as opposed to the latter encoder which has only acquired knowledge from one dataset. However, we pose some skepticism on Adam's ability to efficiently exploit hidden knowledge from the 2 pretraining datasets encoded in such encoders.

In this paper, we propose a two-sided study of the ability of a GA to fine-tune pretrained Transformer encoders. Firstly, we study how well a GA can fine-tune pretrained encoders for the task of sentiment analysis across three datasets. And secondly, we put Adam to the test by comparing its ability to leverage pretrained knowledge from 2 pretraining datasets, at fine-tuning time, with respect to the ability of a GA to do so. Our main hypothesis is that the GA's crossover operator is the key factor to both fine-tune pretrained encoders and efficiently exploiting the knowledge from two pretraining datasets. To our knowledge, this is the first study of fine-tuning Transformer encoders via GAs.

Interestingly, our results are divided: we encountered both a negative and a positive result. On the one hand, although we confirm our hypothesis and show the ability of a GA to fine-tune Transformer encoders, we find two big deficiencies when compared against Adam: considerably higher training times (up to 46x) and a high drop in accuracy scores (up to 28 points) –a negative result. On the other hand, our results show that the GA outperforms Adam's ability to leverage knowledge from two pretraining datasets at fine-tuning time: fine-tuning encoders, pretrained on 2 datasets, via Adam leads to an average gain of 0.55 accuracy points with respect to fine-tuning encoders pretrained on only one pretraining dataset; but the GA's mean gain in performance under the same scenario is 1.65 points,

Proceedings of the Fifth Workshop on Insights from Negative Results in NLP, pages 25–33

a relative increase of 200% (and up to 1540% for a particular case) –a positive result.

Overall, we believe GAs hold as an efficient mechanism for knowledge recombination of Transformer encoders. We hope the community will follow our work to carry out a deeper exploration of GAs on more challenging tasks.

2 Related Work

2.1 Genetic Algorithms

We note that our work is not the first to use GAs to optimize the weights of a Neural Network (NN) (Lander and Shang, 2015; Vázquez-Fernández et al., 2012; David and Greental, 2014). However, previous works evolved NNs (mainly feedforward NNs) containing only a few thousand weights. Our Transformer encoders contain almost 9.5 million parameters. On the other hand, recently, Sobhanam and Prakash (2023) used GAs for BERT-based models such as RoBERTa to automatically search for the best hyperparameter values for an optimal fine-tuning, such as the layers to be fine-tuned, the batch size, the learning rate, and the most suitable activation function; however, in that work, the GA is not used for finding the optimal weights of the model but only optimal hyperparameters.

2.2 AutoML

This area, also called Neural Architecture Search (Elsken et al., 2019), aims to automatically discover optimal architectures for deep NNs via variations of GAs. Recent works have shown the ability of GAs to find architectures as optimal as those from human designers (Miikkulainen et al., 2019; Xie and Yuille, 2017; Liang et al., 2019) and architectures that obtained SOTA results (Real et al., 2019). But, to our knowledge, there is no previous work where a Transformer model was fine-tuned using GAs.

3 Methods and Datasets

3.1 Genetic Algorithm

We use a variant of the Eclectic Genetic Algorithm (EGA) (Kuri and Quezada, 1998; Kuri-Morales et al., 2013). We chose it due to 1) its optimal trade-off between complexity,[1] efficiency, and memory usage due to GPU restrictions, and 2) its resemblance to an ideal GA (Mitchell et al., 1993). EGA follows the usual cycle of GAs. A population of n individuals (pretrained Transformer encoders in our case) is evolved through generations (an operation that can be cast as fine-tuning). In each generation, individuals are ranked by their fitness score (accuracy score on the train set) and crossed[2] to produce offspring (new encoders), and some of these offspring will experience mutation in their chromosomes (sets of hidden vectors). To allow EGA to cross encoders (recombine the knowledge encoded in their parameters), we replaced its crossover operator with the simulated-binary crossover: (Wirsansky, 2020):

$$child_1 = 0.5[(1+\beta)parent_1 + (1-\beta)parent_2]$$

$$child_2 = 0.5[(1-\beta)parent_1 + (1+\beta)parent_2]$$

where β is a hyperparameter manually chosen; and $parent_1, parent_2$ correspond to the set of all hidden vectors of two Transformer encoders. The crossover operation is done layer by layer[3] of both parent encoders which results in $child_1, child_2$ being the recombination of both parent encoders' vectors. Then these two offspring are evaluated on the train set, their fitness score is compared with that from all candidate encoders in the population, and the cycle repeats.

To test our hypothesis, we fix the crossover's probability of occurrence to $p_{cross} = 1$ to fully test its effect; we set the probability of mutation to $p_{mut} = 0.2$ to control for its effect. To mutate an encoder, we add a randomly drawn number in the [-1, 1] interval to randomly chosen weights.

3.2 Datasets

For pretraining encoders, we use 2 popular datasets: WikiText-103 (wiki) (Merity et al., 2017) and 1-Billion-Word (lm1b) (Chelba et al., 2013). For fine-tuning, we use three popular sentiment analysis datasets: SST-2 (Socher et al., 2013), IMDB (Maas et al., 2011), and Yelp (Zhang et al., 2015). We chose these downstream datasets for interpretability of results as binary accuracy scores are obtained.

[1] More complex than the Canonical GA (CGA) (Sivanandam and Deepa, 2008) but simpler than latest GAs. We note that we also experimented with the CGA, but we obtained poor results due to its over-simplicity which refrained it to cope with the high-dimensional space of Transformer models.

[2] The individual in rank i is crossed with the individual in rank $n - i + 1$, i.e. the best individual is crossed with the worst one and so on.

[3] For example, the first attention layers of two encoders will be crossed to produce two attention layers, one for each child.

4 Experiments and Results

4.1 Experimental Setup

We used the Transformer encoder variant from the KerasNLP framework (Watson et al., 2022). We pretrained 10 different encoders with each pretraining dataset by varying random seeds;[4] we refer to them as either wiki or lm1b encoders according to the dataset used. We also pretrained 5 different encoders using both pretraining datasets; we call them Mixed encoders. For some experiments with EGA we used randomly initialized encoders; we call them random encoders. We note 1) the same pretrained encoders are used for both cases fine-tuning them via Adam (baselines) and fine-tuning them via EGA, except for the Mixed encoders which are used only as baselines; 2) for all experiments with EGA, the number of generations is set to 100, the population size to 20 encoders; to obtain means and standard deviations, we run EGA 3 times for each sentiment analysis dataset using different random seeds; 3) for both Adam and EGA, to compute downstream mean scores we use validation or test accuracy scores (depending if the dataset has a test set) of the encoders with highest validation score.

4.1.1 Gains in Accuracy Score

For Adam, we define *gain in accuracy score* as the amount of performance increase in accuracy points obtained by fine-tuning encoders pretrained on 2 datasets with respect to the score obtained by fine-tuning encoders pretrained on only one of the two pretraining datasets. For example, the points increased by fine-tuning Mixed encoders with respect to fine-tuning wiki encoders on the SST-2 data. We refer to this gain as $gain_{Adam}$.

For EGA, we define gain in accuracy as the gain in points obtained by evolving (fine-tuning) wiki and lm1b encoders in the same population with respect to the score obtained by evolving only encoders of a single type (wiki or lm1b) which is the equivalent figure of comparing leveraging two pretraining datasets at fine-tuning time vs. only one dataset; we refer to this as $gain_{EGA}$.

We compute gains in accuracy score as follows: $gain_{Adam} = acc_{Mixed\_enc} - acc_{single\_type\_enc}$ $gain_{EGA} = acc_{wiki+lm1b\_enc} - acc_{single\_type\_enc}$ where acc means accuracy and $single\_type\_enc$ refers to either wiki or lm1b encoders. To compare EGA's gains in performance with those from Adam, we compute the relative increase in gain provided by EGA:

$$\frac{gain_{EGA} - gain_{Adam}}{|gain_{Adam}|} \times 100\% \quad (1)$$

4.1.2 Effect of Number and Type of Encoder

To fully test EGA's ability to recombine knowledge from encoders, we carry out experiments across 6 levels where we vary the number and type of pretrained encoders. At Level 1, populations consist of 10 different lm1b encoders and 10 random encoders; and similarly for Level 2 where instead of lm1b we use wiki encoders. Populations at Levels 3, 4, and 5 contain 5, 10, and 15 pretrained encoders, respectively; but, different from Levels 1 and 2, we use both lm1b and wiki encoders (50% wiki and 50% lm1b), and the rest of the population are random encoders. Finally, populations at Level 6 consist only of pretrained encoders (10 wiki and 10 lm1b).

4.2 Results

4.2.1 Baselines

Fine-tuning encoders pretrained on both datasets via Adam leads to two substantial gains in score on SST-2 data: 1.69 and 1.1 points (Tables 2 and 3) with respect to wiki and lm1b encoders, respectively, which are the differences in accuracy from Mixed encoders and wiki, lm1b encoders in Table 4. However, on the other downstream datasets, this leads to minor gains: a gain of 0.36 points for IMDB data when lm1b and Mixed encoders are measured against each other, and gains of 0.1 and 0.58 points for Yelp data (Tables 2 and 3). Surprisingly, we observe a drop in gain of 0.52 points for IMDB data (Table 2): wiki encoders achieve a superior accuracy score (85.03, Table 4) than Mixed encoders (84.51) (we provide a possible explanation for this finding in Section 5).

By averaging all gains in score from Adam in Tables 2 and 3, we observe that Mixed encoders lead to a mean increase of only 0.55 points. However, we do not jump straightaway to the conclusion that, at fine-tuning time, Adam's ability to leverage knowledge from encoders pretrained on 2 datasets is not as impactful as we expected since these results could be obscured by a *ceiling effect*, as we discuss in Section 5.

4.2.2 Genetic Algorithm Results

SST-2: As shown in Table 1, EGA's best score on SST-2 data (59.15 points) comes from a mixed

[4] We believe that by doing so the encoders can pick different patterns even if pretrained on the same data.

Dataset	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
SST-2	57.25(0.021)	57.71(0.010)	57.91(0.011)	59.15(0.003)	58.69(0.016)	57.98(0.020)
IMDB	60.09(0.001)	57.0(5e-4)	57.41(0.002)	57.06(0.001)	57.87(0.023)	59.59(0.015)
Yelp	56.82(0.008)	58.0(0.001)	59.43(0.044)	59.04(0.010)	59.64(0.014)	57.33(0.006)

Table 1: Mean validation or test accuracy scores across three different random seeds of fine-tuning pretrained encoders via EGA (standard deviations in parenthesis) for different Levels as described in Section 4.1.2.

Dataset	Adam	EGA	Relative increase
SST-2	1.69	1.44	-14.79%
IMDB	-0.52	2.59	598.07%
Yelp	0.1	1.64	1540%

Table 2: Gains in accuracy points by Adam and EGA: two pretraining datasets (wiki+lm1b) vs. wiki dataset. Column Relative increase shows the increase of performance of EGA over Adam as in Section 4.1.1.

Dataset	Adam	EGA	Relative increase
SST-2	1.1	1.9	72.72%
IMDB	0.36	-0.5	-238.88%
Yelp	0.58	2.82	386.20%

Table 3: Gains in accuracy points by Adam and EGA: two pretraining datasets (wiki+lm1b) vs. lm1b dataset. Column Relative increase shows the increase of performance of EGA over Adam as in Section 4.1.1.

Dataset	Wiki enc	Lm1b enc	Mixed enc
SST-2	75.12(0.014)	75.71(0.013)	76.81(0.010)
IMDB	85.03(0.006)	84.15(0.006)	84.51(0.003)
Yelp	87.62(0.006)	87.14(0.003)	87.72(0.004)

Table 4: Mean validation or test accuracy scores across encoders (standard deviations in parenthesis) of fine-tuning pretrained encoders via Adam.

population (Level 4): 5 wiki and 5 lm1b encoders (and the rest random encoders). On the other hand, the lowest accuracy scores come from populations where only one type of encoder is used: Level 1 (only lm1b) and Level 2 (only wiki). As we see, recombining the hidden knowledge from wiki and lm1b encoders leads to substantial gains of 1.44 and 1.9 points (Tables 2 and 3) compared to crossing either only wiki or only lm1b encoders. Comparing EGA's gains vs. Adam's gains, we see that Adam obtains a bigger gain than EGA when using two pretraining datasets as opposed to only wiki data, as shown in Table 2: 1.69 vs. 1.44 points; however, this figure turns around for lm1b encoders where EGA's increase in gain is superior to that of Adam by 72.72%.

IMDB: We observe a clear gain in performance when knowledge from wiki encoders is mixed with that from lm1b encoders: a rise of 2.59 points (Table 2) which is the difference between crossing only wiki encoders (Level 2, Table 1) and crossing both encoders type (Level 6). Compared to Adam's gain in score (-0.52 points) EGA achieves a superior relative increase in performance of 598%. Nevertheless, similar to Adam, we see a drop in gain: lm1b encoders provide better results for EGA than wiki+lm1b encoders by 0.5 points (Level 1 vs. Level 6) as shown in Table 3.

Yelp: The best scenario comes from mixing knowledge from both encoder types (Table 1, Level 5): 59.64 points, providing a gain in score of 1.64 points with respect to fine-tuning only wiki encoders (Level 2), representing a remarkable relative increase of performance of 1540% with respect to the gain obtained by Adam of only 0.1 points (Table 2). From Table 3 we see the biggest gain in accuracy score obtained by EGA across all sentiment analysis datasets: 2.82 points increase by, again, crossing wiki with lm1b encoders (Level 5) as opposed to only lm1b encoders (Level 1).

Fine-tuning times: As we see in Table 5, EGA takes considerably more time than Adam (at least 31 times more) representing a disadvantage.

Dataset	Adam	EGA	Factor
SST-2	1.57	51.59	33x
IMDB	3.42	158.92	46x
Yelp	2.9	90.13	31x

Table 5: Average time in min. that Adam and EGA take to obtain an encoder with the highest validation score, and the factor of difference between them.

5 Discussion and Conclusions

How well can EGA fine-tune Transformer encoders? We observed in Table 1 that EGA is able to fine-tune the encoders on the sentiment analysis datasets with scores reaching, or close to, the

60 points threshold. Although there is a wide gap compared to Adam's scores (up to 28 points), we believe these results show the capability of GAs for fine-tuning Transformer encoders.

Exploiting knowledge from pretraining datasets: Another important aspect of fine-tuning is leveraging pretrained knowledge from two datasets. We observed in Tables 2 and 3 how Adam more often than not achieves small gains in performance with an average gain across datasets of only 0.55 points. Remarkably, EGA better exploits the hidden knowledge from wiki and lm1b encoders by obtaining and average gain of 1.65 points (3 times Adam's gain). On a closer look, we observe substantial relative increases of performance from EGA of up to 1540% as shown for the Yelp dataset.

Caution must be applied: We interpret these results with precaution. We cannot firmly conclude that Adam's capability of leveraging knowledge from encoders pretrained on 2 datasets will invariably lead to such a small average gain in accuracy for any other task or dataset since we may be facing a ceiling effect (Cohen, 1995). This effect happens either when the NLP model is too close to a perfect score (which is not our case) or when the NLP model is too close to its maximum capability in solving a task, which may be our case. It is possible that the variant of Transformer encoder we used, when pretrained on only one dataset, is already close to the maximum score it can achieve; thus, knowledge from another pretraining dataset helps but not as much as it would for a more difficult task where the initial score is small enough to leave room for improvement.

2 is not always better than 1: We saw the interesting finding that fine-tuning encoders pretrained on only 1 dataset led to the best results for the IMDB dataset, for both Adam (via wiki encoders) and EGA (via lm1b encoders). To provide a plausible rationale, we manually reviewed IMDB, wiki, and lm1b instances to find any qualitative patterns. Our first observation is the similarity in writing style between IMDB and wiki instances: long texts with a clear description of an item, entity, or event supported by facts or arguments and followed by a conclusion –patterns that Adam may have recovered from wiki encoders. On the other hand, we notice the newspaper writing style in lm1b instances which somehow differs from those in IMDB; probably, EGA exploited factuality and cultural patterns from the news articles in lm1b that helped to classify IMDB instances since both datasets are contemporary with a short time gap. Moreover, we believe that the different patterns in wiki and lm1b instances, rather than complementing to each other to improve on the downstream scores, as in the case for the SST-2 and yelp data, they are at odds with each other for the IMDB dataset; however, it is unclear exactly in which way. We believe this finding requires a deeper analysis given the complexity of the IMDB instances (Otterbacher, 2013).

Future work: We delimited our work to a specific choice of Transformer encoder, Genetic Algorithm, pretraining data, and downstream task. Naturally, further experimentation is necessary to generalize our results, such as studying more complex GAs and hybrid approaches that take advantage of the strengths of both Adam (high scores in low time) and GAs (ability to exploit different pretraining data) for fine-tuning more complex NLP models such as BERT (Devlin et al., 2019); test on other pretraining datasets, such as the BookCorpus (Zhu et al., 2015) or C4 (Raffel et al., 2020); and test the hypotheses proposed in this work on more complex downstream tasks and datasets to either confirm our results and elaborate upon them, or to pinpoint possible ceiling effects.

Acknowledgements

We thank the anonymous reviewers for their insightful comments which helped improve the Introduction section and the interpretation of results of this work.

References

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *Interspeech*.

Paul R. Cohen. 1995. *Empirical methods for artificial intelligence*. MIT Press, Cambridge, MA, USA.

Omid E David and Iddo Greental. 2014. Genetic algorithms for evolving deep neural networks. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1451–1452.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yadolah Dodge. 2008. *Coefficient of Variation*, pages 95–96. Springer New York, New York, NY.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(1):1997–2017.

Payel Ghosh, Melanie Mitchell, James A. Tanyi, and Arthur Y. Hung. 2016. Incorporating priors for medical image segmentation using a genetic algorithm. *Neurocomputing*, 195:181–194. Learning for Medical Imaging.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeff Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *ArXiv*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Angel Kuri and Carlos Quezada. 1998. A universal eclectic genetic algorithm for constrained optimization. *Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT'98*.

Angel Fernando Kuri-Morales, Edwin Aldana-Bobadilla, and Ignacio López-Peña. 2013. The best genetic algorithm ii. In *Advances in Soft Computing and Its Applications*, pages 16–29, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sean Lander and Yi Shang. 2015. Evoae–a new evolutionary method for training autoencoders for deep learning networks. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pages 790–795. IEEE.

Taoshen Li and Zhihui Ge. 2009. A multiple qos anycast routing algorithm based adaptive genetic algorithm. In *2009 Third International Conference on Genetic and Evolutionary Computing*, pages 89–92.

Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. 2019. Evolutionary neural automl for deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 401–409.

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

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. *International Conference on Learning Representations*.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2019. Chapter 15 - evolving deep neural networks. In Robert Kozma, Cesare Alippi, Yoonsuck Choe, and Francesco Carlo Morabito, editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Academic Press.

Melanie Mitchell, John Holland, and Stephanie Forrest. 1993. When will a genetic algorithm outperform hill climbing. *Advances in neural information processing systems*, 6.

Jahna Otterbacher. 2013. Gender, writing and ranking in review forums: a case study of the imdb. *Knowledge and Information Systems*, 35:645–664.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

G. Sainath, S. Vignesh, S. Siddarth, and G. Suganya. 2021. Application of neuroevolution in autonomous cars. In *International Virtual Conference on Industry 4.0*, pages 301–311, Singapore. Springer Singapore.

S.N. Sivanandam and S.N. Deepa. 2008. *Introduction to Genetic Algorithms*. Springer Berlin, Heidelberg.

H. Sobhanam and J. Prakash. 2023. Analysis of fine tuning the hyper parameters in roberta model using genetic algorithm for text classification. *International Journal of Information Technology*, 15:3669–3677.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Eduardo Vázquez-Fernández, Carlos A. Coello Coello, and Feliú D. Sagols Troncoso. 2012. Assessing the positional values of chess pieces by tuning neural networks' weights with an evolutionary algorithm. In *World Automation Congress 2012*, pages 1–6.

Matthew Watson, Chen Qian, Jonathan Bischof, François Chollet, et al. 2022. Kerasnlp. https://github.com/keras-team/keras-nlp.

Eyal Wirsansky. 2020. *Hands-On Genetic Algorithms with Python*. Packt Publishing, Birmingham, UK.

Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

A Appendix

A.1 Population-based Analysis

In this subsection, we show an additional analysis at the population level across the last 4 levels, where populations consist of both wiki and lm1b encoders, for the SST-2 dataset. Figures 1, 2, 3, 4 show the evolution of accuracy scores through generations. In each figure, training curve_1, validation curve_1 correspond to the evolution of one population. Thus, we show the evolution of 5 populations per level where each population is evolved using a different random seed. In each curve, each point represents the average accuracy score of one population for a given generation number.

As we see across all figures, our belief that Transformer encoders pretrained on the same dataset, but using a different random seed for pretraining, can capture different linguistic or world knowledge seems to be supported by these plots since at the beginning of all evolution processes the standard deviations for each population are very wide, which means that accuracy scores across each individual vary to a great extent which seems to imply that individuals encode different knowledge (some of them having learned patterns more useful for the SST-2 data than others) which is reflected in their different chromosomes.

Also, we observe in Figures 1 and 2 that for Levels 3 and 4, around generation $gen = 40$, most of

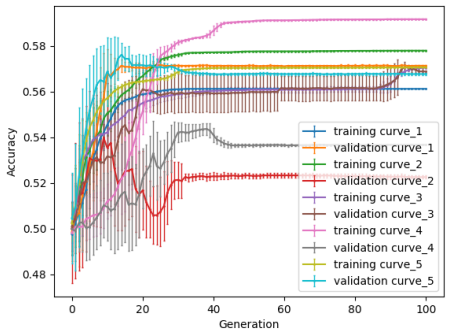

Figure 1: Average accuracy scores at the population level across generations for Level 3. Bars represent standard deviations.

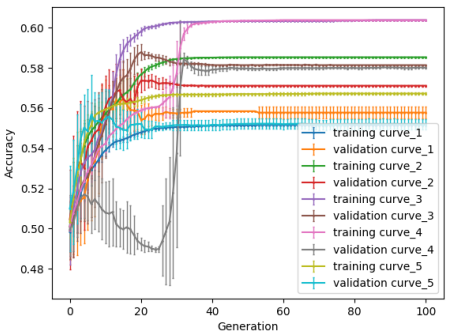

Figure 2: Average accuracy scores at the population level across generations for Level 4. Bars represent standard deviations.

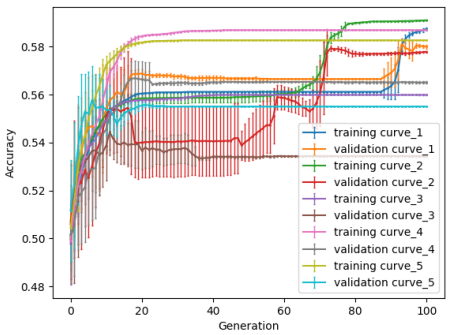

Figure 3: Average accuracy scores at the population level across generations for Level 5. Bars represent standard deviations.

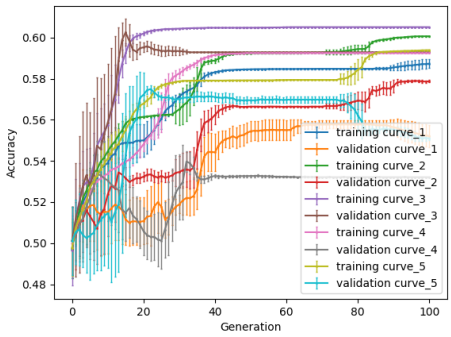

Figure 4: Average accuracy scores at the population level across generations for Level 6. Bars represent standard deviations.

the populations tend to converge to the final average accuracy, and in several cases the variation is minimal which means that individuals should share a large part of their genetic material with each other. This is a well-known effect in GAs and it tends to lead to local optima. However, for Levels 5 and 6 where most or all encoders are pretrained, stability for some populations tends to arrive at the last generations as there are cases where populations still see an increase in their average scores by almost the end of the run; this could mean that a bigger diversity of both wiki and lm1b encoders is helpful for avoiding or escaping local minima.

A.2 Genetic Analysis of the Best Individual

The best individual from all our experiments with the SST-2 dataset comes from Level 4; this encoder achieved a validation score of $val = 0.6135$. We traced back all its parents up to the first generation to have an idea of how its chromosome is formed. Not surprisingly, half of its genetic material is formed by weight vectors from wiki encoders and half from lm1b encoders. This piece of evidence further supports our hypothesis; recombining knowledge from different types of encoders leads to optimal individuals. It seems that weight vectors from different encoder types may encode different type of linguistic or world knowledge and when recombined they produce parameters more fit to the task at hand. We leave this hypothesis to be tested in future work.

A.3 Robustness to Variability

We measured how robust is each optimization method to the impact of random seed variation on the downstream scores; ideally, optimization methods would provide a robust estimate of the accuracy which translates into low variability. To measure this property, we computed the Coefficient of Variation (CV) (Dodge, 2008) since directly comparing the standard deviations from Adam and EGA is not a reliable approach due to the wide gap between mean accuracy scores from both methods. The coefficient of variation is a standardized measure that takes into account the size of the mean scores as follows:

$$CV = \frac{standard\_deviation}{mean} \times 100\% \quad (2)$$

Thus, higher CV values represent a higher degree of variability. We compute coefficients of variation using means and standard deviations from Tables 4 and 1 for Adam and EGA, respectively. We find that while Adam's CV values range from 0.007% to 0.018%, EGA's CV values falls in the 0.0009%-0.07% range, both intervals containing extremely low signs of variation showing that both methods exhibit comparably high and robust estimates of accuracy.

A.4 Sampling of SST-2

To allow for a faster (and more environmentally friendly) training on the SST-2 data with EGA, we investigated if we could reduce its train set size through a learning curve. The learning curve in Figure 5 was obtained by evolving 20 randomly initialized encoders for 100 generations across 5 different random seeds. It shows that the best validation scores come from using approx. 5% of the train set (3072 instances).[5] Thus, we chose to use a random sample of size 3072 for our experiments with EGA and Adam.

A.5 Transformer Model and Training Details

Our target model is the Transformer encoder variant implemented in the KerasNLP framework which is roughly the equivalent of a half-size Transformer encoder from the original Transformer model in (Vaswani et al., 2017). We chose this variant mainly for memory consumption reasons when fine-tuning it with the genetic algorithm. We used same settings and hyperparameters as in (Watson et al., 2022) to have a fully reproducible baseline. Also, for some of our experiments we used the same dataset for pretraining (WikiText-103 dataset) and the same dataset for fine-tuning (SST-2) as

[5]We used up to approx. 88% of the dataset to keep it balanced between positive and negative labels since we are optimizing accuracy scores.

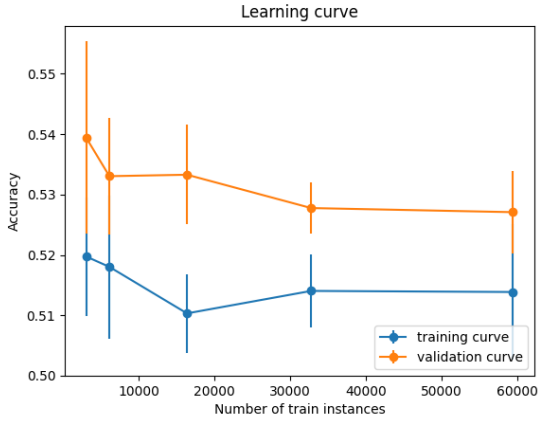

Figure 5: Learning curve. Each point is averaged on 5 different runs. Bars represent standard deviations.

those used in the KerasNLP original implementation.

More concretely, this Transformer implementation consists of 3 encoder blocks each with 4 attention heads; feedforward layer size is 512; token and learnable position embeddings are of dimension 256; sequence length of 128 tokens, and Word Piece Tokenizer. Total number of parameters is almost 9.5 million trainable weights. Pretraining batch size is 128, fine-tuning batch size is 32, sequence length is set to 128, mask rate is set to 0.25, dropout rate is set to 0.1, epsilon is set to $1e-5$, pretraining learning rate is $5e-4$, fine-tuning learning rate is $5e-5$, and pretraining epochs is set to 8. There is a parameter which we change from the original implementation; we increased the number of fine-tuning epochs to 15 since before the 15th epoch validation scores go down.

A.6 Hardware and Software Used

We used Tensorflow version 2.10.1, KerasNLP version 0.3.1, python version 3.10.9. To run our experiments we used an Nvidia RTX3060 GPU. The total time that all our experiments took to run was 495.55 hrs. (20.64 days).