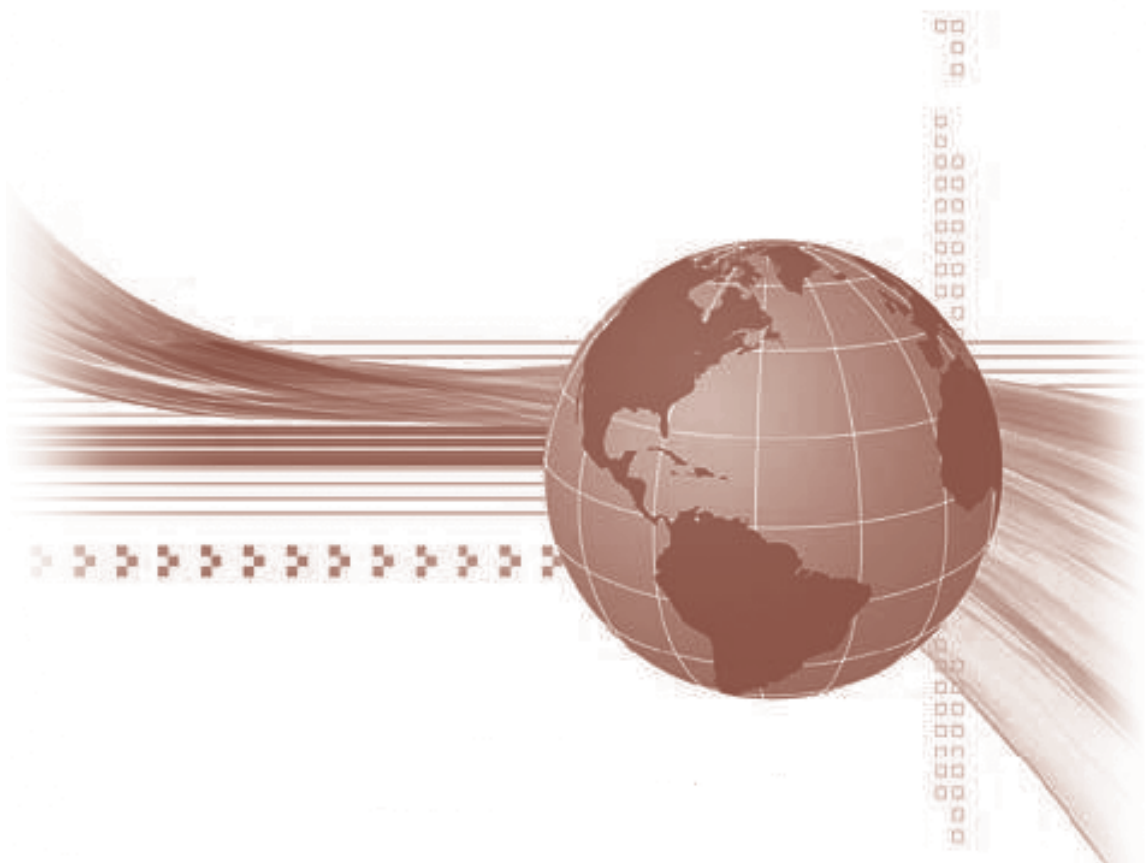
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

2/2020

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

No. 2/2020

July - December

EDITORIAL BOARD

YEAR ..... **VOLUME 65 (LXV) 2020**
MONTH **DECEMBER**
ISSUE **2**

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

2

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

Á. Szijártó, P. Lehotay-Kéry, A. Kiss, *Experimental Study of Some Properties of Knowledge Distillation* 5

M.-M. Mircea, *Employing Long Short-Term Memory Networks in Trigger Detection for Emetophobia* 17

T.V. Pricope, *A View on Deep Reinforcement Learning in Imperfect Information Games* 31

H.B. Mureșan, A.D. Călin, A.M. Coroiu, *Overview of Recent Deep Learning Methods Applied in Fruit Counting for Yield Estimation* 50

M. Petrescu, R. Petrescu, *Log replication in Raft vs Kafka* 66

C.M. Tiutin, M.-T. Trifan, A. Vescan, *Defect Prediction-Based Test Case Prioritization* 78

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 2, 2020
DOI: 10.24193/subbi.2020.2.01

EXPERIMENTAL STUDY OF SOME PROPERTIES OF KNOWLEDGE DISTILLATION

ÁDÁM SZIJÁRTÓ, PÉTER LEHOTAY-KÉRY, AND ATTILA KISS

Abstract. For more complex classification problems it is inevitable that we use increasingly complex and cumbersome classifying models. However, often we do not have the space or processing power to deploy these models.

Knowledge distillation is an effective way to improve the accuracy of an otherwise smaller, simpler model using a more complex teacher network or ensemble of networks. This way we can have a classifier with an accuracy that is comparable to the accuracy of the teacher while small enough to deploy.

In this paper we evaluate certain features of this distilling method, while trying to improve its results. These experiments and examinations and the discovered properties may also help to further develop this operation.

1. Introduction

Knowledge distillation is a method to transfer the knowledge of an already trained neural network to another, possibly a smaller one. The benefit of this is we can achieve a higher accuracy for the student models as opposed to training it on their own.

This simple method can significantly boost the accuracy of a model in a way that could not be achieved with normal training methods or hard outputs. This distillation technique is especially useful when we have an accurate, but cumbersome neural network (or ensemble of networks); but do not have the resources to deploy it.

In this paper, we examine certain features of this method, such as transitivity and symmetry, by conducting experiments to prove or refute these

Received by the editors: 7 August 2020.

2010 *Mathematics Subject Classification.* 68T05, 68T30.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*;

Key words and phrases. Artificial Intelligence, Convolutional Neural Networks, Deep learning, Knowledge distillation, Neural networks.

properties. In addition, we evaluate how this method performs using an ensemble of student networks and how its parameters affect the results by further experiments.

We are going to evaluate the symmetry of knowledge distillation by distilling the knowledge back from a trained student network to an untrained teacher network. We are going to conclude that knowledge distillation is a symmetric operation if the new model performs better than the student model and its performance is close to the original model.

We are going to evaluate the transitivity of knowledge distillation by distilling the knowledge from the teacher model to a less complex middle model, and then distill it further to the original student model. We are going to conclude that knowledge distillation is a transitive operation if there is no significant difference between the accuracy of the middle model and the original student model.

2. Related works

In their 1998 paper titled "Neural network ensembles" [1], Lars Kai and Peter Salamon argue that building multiple classifier models and evaluating their results to a given classifying problem can vastly outperform a single model even if those models are significantly simpler and individually do not perform as well as the single model. They also found that cross validation could greatly reduce overfitting while training these models.

This idea was further elaborated in the 2000 paper "Ensemble methods in machine learning" [2] by T. G. Dietterich, who reviewed these methods and explained why ensembles could often perform better than any single classifier. Furthermore, the author reviewed some previous studies comparing ensemble methods and presented some new experiments.

Taking this as a basis in their 2015 paper "Distilling the knowledge in a neural network" [3] Hinton, Vinyals and Dean found that the "knowledge" from a trained complex model or even an ensemble of models could be distilled down into a much simpler model without compromising performance and accuracy. They argue that the training and deployment of a classifier are two completely different problems with different requirements. We should not use the same model, but use a cumbersome one for the training and – as the computational complexity is a huge factor for end users – we should use a distilled simpler model for deployment.

This idea has been further improved in the 2017 paper titled "A gift from knowledge transfer distillation: Fast optimization, network minimization and transfer learning" [4], which proposed a new solution: the knowledge from a pretrained deep neural network (DNN) is distilled and transferred to another

DNN. The method uses FSP (flow of solution procedure) matrix, representing the distilled knowledge from the teacher DNN.

In the "Distillation as a defense to adversarial perturbations against deep neural networks" [5] paper, the authors found that this method was also effective against adversarial attacks. They found that training a model, then distilling its knowledge to one that is structured the same way can significantly increase its robustness.

However, since then methods have been found for adversarial attacks against which this kind of distillation does not work. It has been elaborated in the paper titled "Towards Evaluating the Robustness of Neural Networks" [6] in 2017, where the authors introduced three new attack algorithms that were successful on both distilled and undistilled neural networks with 100% probability.

In addition to these, the idea of generating softened outputs with a trained classifier in order to enhance the performance of another one goes beyond neural networks. In their 2017 paper "Distilling a Neural Network Into a Soft Decision Tree" [7], Nicholas Frosst and Geoffrey Hinton argue that this method can be applied when distilling knowledge from a neural network to a decision tree.

"Residual Knowledge Distillation" [8] further distills the knowledge by introducing an assistant which learns residual errors. The experiments of the authors showed that their approach achieved appealing results on popular classification datasets.

The human visual system relies on temporal dependencies among frames from the visual input to conduct recognition. Based on this observation, "Tkd: Temporal knowledge distillation for active perception" [9] proposes the Temporal Knowledge Distillation framework, which distills the temporal knowledge from a neural network-based model over selected video frames to a light model. Results of the authors showed consistent improvement in accuracy-speed trade-offs for object detection, compared to other modern object recognition methods.

"Explaining Knowledge Distillation by Quantifying the Knowledge" [10] presents a method to qualify and analyze task-relevant and task-irrelevant visual concepts that are encoded in intermediate layers of a Deep Neural Network. Authors designed mathematical metrics to evaluate feature representations of the Deep Neural Network and diagnosed Deep Neural Networks as experiments.

"Learning an Evolutionary Embedding via Massive Knowledge Distillation" [11] proposes an Evolutionary Embedding Learning framework to learn a fast and accurate student network for open-set problems via Massive Knowledge

Distillation. Authors introduced a novel correlated embedding loss to match embedding spaces between the teacher and student network. EEL achieved better performance with other state-of-the-art methods for various large-scale open-set problems.

"Feature-map-level Online Adversarial Knowledge Distillation" [12] proposes an online knowledge distillation method that transfers the knowledge of the feature map using the adversarial training framework. Authors trained multiple networks simultaneously by employing discriminators to distinguish the feature map distributions of different networks. Furthermore, they proposed a novel cyclic learning scheme for training more than two networks together.

3. Background

3.1. **Convolutional Neural Networks.** For our experiments we used a CNN (Convolutional Neural Network)[13][14], which is a class of deep neural networks, a regularized multilayer perceptron. They are most often applied to analyze images, by learning filters independently from prior knowledge. CNNs consist of an input, an output and multiple hidden layers.

In neural networks, each neuron produces the output value by applying a function to the input values that come from the previous layer. Weights and biases determine this function and their iterative adjustments progress the learning.

In CNN, most of the hidden layers are convolutions, which are special linear operations. When data are passing through a convolutional layer, it becomes abstracted to a feature map.

CNNs may also include some pooling layers to reduce the dimensions of data. In our experiments we used max pooling[15][16]. Pooling combines the outputs of neurons in one layer into a single neuron in the next layer. Max pooling uses the maximum value as combination.

In order to reduce overfitting, we used Dropout [17][18] in each layer. Dropout means that at each training stage, nodes together with their edges are dropped out of the net with probability 1-p, so that a reduced network is left. Only this network is trained on the data at this stage. The removed nodes and edges are reinserted at the next stage.

3.2. **Knowledge distillation.** The knowledge distillation method uses a special activation function to produce "softened" probabilities, which are then used to train the student network, on which we also apply the previously mentioned activation function. This special function is a parameterized version of the widely used softmax[19] function, which is used to convert the last layer of the network into probabilities.

Softmax can be given in the following form[3]:

$$q_i = \frac{exp(\frac{z_i}{t})}{\sum_j exp(\frac{z_j}{t})}$$

where t is a parameter called temperature, which converts z_i logit value to q_i probability. For a standard softmax, t is normally set to 1. The higher we set this parameter, the softer the output probabilities are going to be, and this way we can preserve more features of the input than the teacher net learned, meaning that the student receives more information as opposed to using hard outputs.

The distilled model will be the smaller network we have trained on a transfer set, which is not the same dataset as the one we used to train the larger model. As loss function, cross entropy is used between the output of the distilled model and the output of the larger model.

4. Experiments

For the experiments we used the GTSRB (German Traffic Sign Recognition Benchmark) dataset[20]. The teacher net was a CNN (Convolutional Neural Network) with three layers, each with 128 nodes, using rectified linear activation functions.

With this model we managed to achieve an 0.9473 accuracy on the test set. This served as a baseline for our further experiments. As for the student, we used a dense neural network with one hidden layer with rectified linear activation function.

Training it normally with the hard outputs and traditional *softmax* output layer, we had an accuracy on the test set that is not higher than 0.1635. The results of the distillation process, in relation to the temperature parameter, can be seen in **Figure 1**.

Compared to the traditional training approach, we can clearly see a significant improvement in the graph. However, the temperature parameter does not seem to show much influence on the results if it is greater than 4. In fact the accuracy appears to be quite random between the range of 0.5 and 0.8. It will serve as a baseline in our further experimentation.

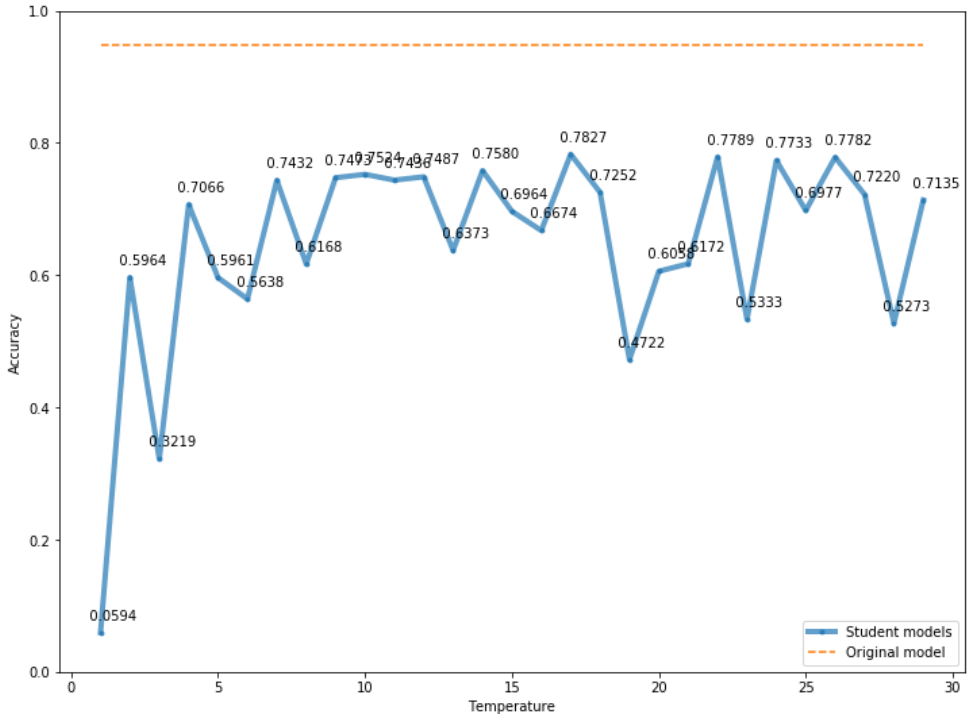


FIGURE 1. Results of the distillation

4.1. **Symmetry.** To test the symmetry of the method, we investigated if we could reverse the distillation process by taking a student model that had been trained with this technique, then we distilled its knowledge into the original (untrained) teacher model. For this experiment we took the best performing student network – which we received with temperature 17, and had an accuracy of 0.7827 – then used it to generate the softened outputs.

They were used to train the teacher model with the modified softmax output layer. If we presume that the distillation process is symmetric, we expect the new model to perform better than the student model, and nearly as good as the original one we started with. We trained 10 models going from 1 to 10. The results can be seen in **Figure 2.**

We can see the accuracy is significantly better than our best student model with an average accuracy of around 0.84. However, it is not even close to the original accuracy of 0.9473.

It is also important to note that there is no significant deviation among the performance of the models, meaning the temperature parameter has little to

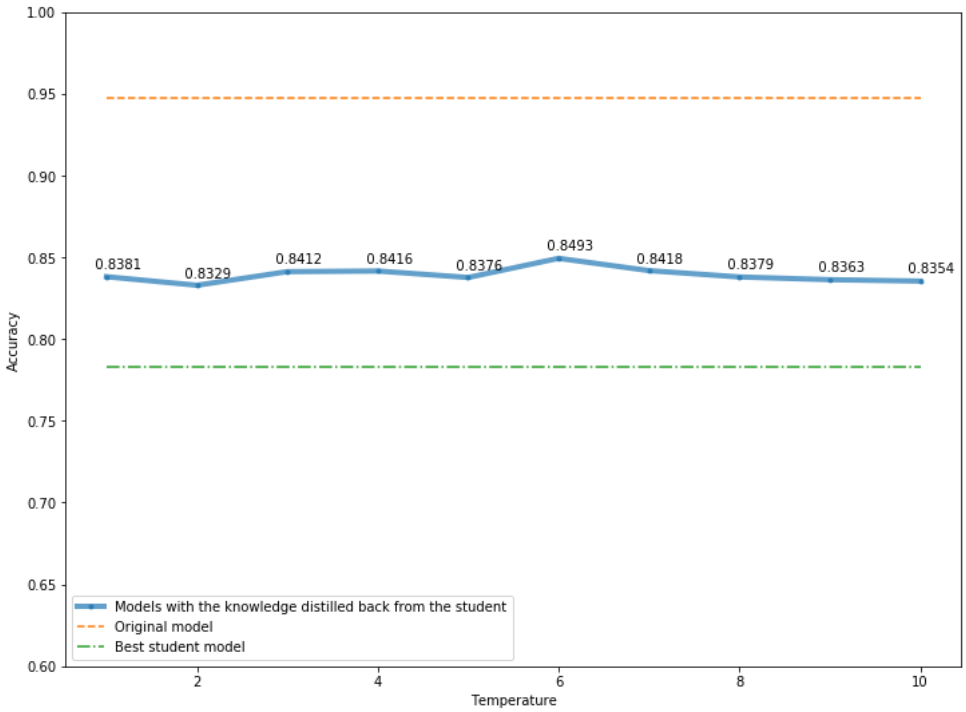


FIGURE 2. Results of distilling the knowledge back to the original model

no influence on the accuracy of the given model, and using this special parameterized softmax function provides no improvement as opposed to training traditionally on the output of the student model. With all that said, we can conclude that this method does not in fact keep symmetry.

4.2. **Student ensembles.** In this paragraph we are discussing whether we can improve the accuracy of the network in which we distilled the knowledge to, by creating an ensemble of networks of the same architecture, but using different distilling temperature parameter. To generate the predictions of the ensembles, we used a simple majority voting.

Using all 29 student networks, after evaluation, we achieved an overall accuracy of 0.7712 on our test set, which is certainly worse than our best student network (0.7827), but better than the average accuracy (0.65014). In order to improve this, we evaluated the best N networks. The results are shown in **Figure 3**.

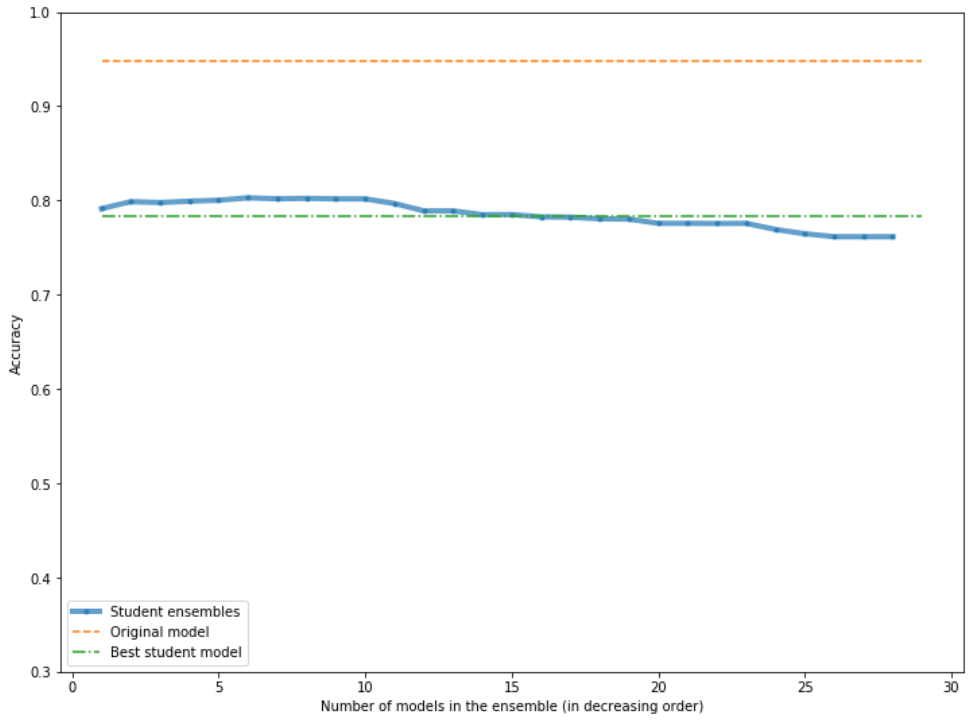


FIGURE 3. Results of distilling the knowledge back to the original model

Here we can clearly see that after 10 networks the accuracy is slowly, but steadily declining, and after 18 it goes below the best result of our individual student networks. It might be due to the fact that the models in our ensemble are structurally quite similar (the only difference is the temperature parameter) and the fact that they all were trained on the same data results in models that mostly make the same mistake during classification.

Considering that even if we find the best student models in relation to the temperature, then find the ideal number of networks for the ensemble and increase the complexity of the model, the boost in accuracy is not significant enough for this kind of trade off.

4.3. **Transitivity.** To test the transitivity of this technique, we first created a new model structure, which stood between the teacher and the student model in terms of complexity. It is a deep neural network with 2 hidden layers, with 50 nodes each. Then we distilled the knowledge with the discussed technique to the middle model.

21 different models were trained; one in the traditional way with hard outputs, and 20 with distillation with the temperature parameter ranging from 1 to 20. We then took the best performing model and distilled its knowledge further to the original student model.

Ideally, these results are comparable to the ones we received from directly distilling the knowledge to the student model. The results of the performance of the middle models in relation to the temperature can be seen in **Figure 4** (0 being the one trained on hard outputs).

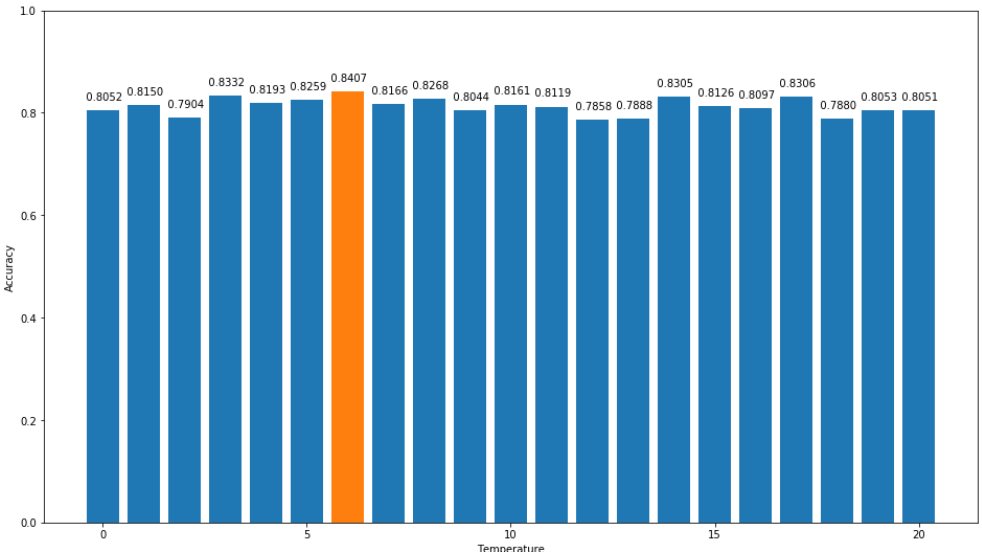


FIGURE 4. Results of distilling the knowledge to the middle model

After the experiment we saw that the best performing model was the one with temperature 6, which we used further in this experiment. Interesting to note that the improvement provided by the distillation method was insignificant as the performance of the non-distilled model was just slightly lower than the average of the distilled ones with very little standard deviation.

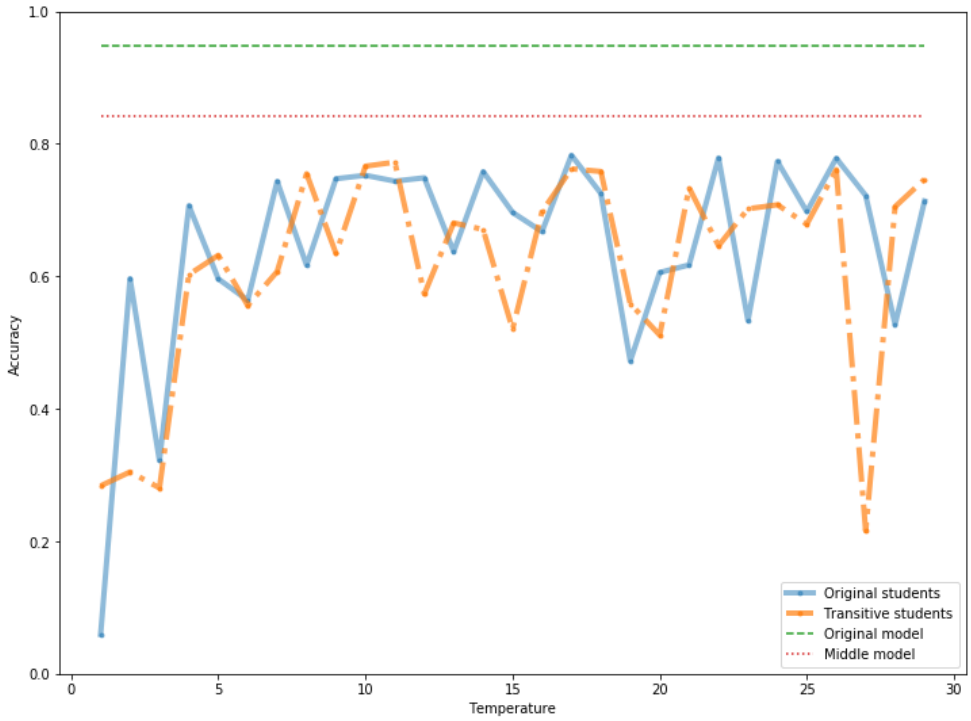


FIGURE 5. Results of the distillation

In **Figure 5** we can see the accuracy of the transitively distilled models in relation to the originals, as well as the baseline model and our highest performing middle model. Although the numbers are not exact, the overall distribution of the results are actually quite similar to the ones we had from the direct distillation.

We can claim that as long as the middle model achieves good enough accuracy – close to the original one – this distillation method keeps the transitive property.

5. Conclusion

In this paper we investigated multiple features and behaviours of the knowledge distillation method. We experimented with symmetry by distilling the knowledge back from our trained student network to our untrained teacher network. We conclude that even though it outperformed the best student, it did not come close to the model trained in the traditional way, and acted more as a noise rather than useful additional information, proving that this method is not symmetric.

Experiments for creating an ensemble of student networks were also conducted by using student networks trained with different temperatures. We were able to achieve very little improvement, which is due to the fact that besides the temperature there were no structural differences between the models, which resulted in similar cases of misclassification in every net. This leads us to believe that even though the temperature can affect the performance of the model, it has little to no effect on the behaviour.

Lastly, we investigated the transitive feature of this method by distilling the knowledge to a slightly more complex model than our student model, then distilled it further to our original student model. According to our experiment, the difference was not remarkable between these students and our baseline students, proving that this method is transitive.

6. Acknowledgements

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[2] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[3] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[4] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4133–4141, 2017.

[5] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[7] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[8] Mengya Gao, Yujun Shen, Quanquan Li, and Chen Change Loy. Residual knowledge distillation. *arXiv preprint arXiv:2002.09168*, 2020.

[9] Mohammad Farhadi Bajestani and Yezhou Yang. Tkd: Temporal knowledge distillation for active perception. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 953–962, 2020.

[10] Xu Cheng, Zhefan Rao, Yilan Chen, and Quanshi Zhang. Explaining knowledge distillation by quantifying the knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12925–12935, 2020.

[11] Xiang Wu, Ran He, Yibo Hu, and Zhenan Sun. Learning an evolutionary embedding via massive knowledge distillation. *International Journal of Computer Vision*, pages 1–18, 2020.

[12] Inseop Chung, SeongUk Park, Jangho Kim, and Nojun Kwak. Feature-map-level online adversarial knowledge distillation. *arXiv preprint arXiv:2002.01775*, 2020.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[15] Kouichi Yamaguchi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. A neural network for speaker-independent isolated word recognition. In *First International Conference on Spoken Language Processing*, pages 1077–1080, 1990.

[16] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[18] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in neural information processing systems*, pages 3084–3092, 2013.

[19] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[20] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

Department of Information Systems, Faculty of Informatics, ELTE Eötvös Loránd University, Budapest, Hungary

Email address: c4442f@inf.elte.hu, lkp@caesar.elte.hu, kiss@inf.elte.hu

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 2, 2020
DOI: 10.24193/subbi.2020.2.02

EMPLOYING LONG SHORT-TERM MEMORY NETWORKS IN TRIGGER DETECTION FOR EMETOPHOBIA

MARIA-MĂDĂLINA MIRCEA

Abstract. Research focused on mental health-related issues is vital to the modern person's life. Specific phobias are part of the anxiety disorder umbrella and they are distressing afflictions. *Emetophobia* is the rarely known, yet fairly common and highly disruptive specific phobia of vomiting. Unlike other phobias, emetophobia is triggered not only by the object of the specific fear, but also by verbal and written mentions of said object. This paper proposes and compares ten neural network-based architectures that discern between triggering and non-triggering groups of written words. An interface is created, where the best models can be used in emetophobia-related applications. This interface is then integrated into an application that can be used by emetophobes to censor online content such that the exposure to triggers is controlled, patient-centered, and patient-paced.

1. Introduction

For the longest time, mental health has been a largely overlooked aspect of people's lives. This fact is worrisome, especially given the fact that globally, over 10% of the population suffers from at least one mental health disorder [9]. Ranging from mild anxiety to substance abuse and severe depression, mental health problems affect daily activities and even become life-threatening if left untreated.

The specific phobia, a very common affliction, is part of the most widely-spread set of mental health conditions, namely, anxiety disorders. *Emetophobia*, the specific phobia of vomiting and sickness, is fairly familiar to the average person, but scarcely by name. It is, however, a life-altering, highly disruptive phobia, with around 2% of male and 7% of female sufferers [10].

Received by the editors: 24 August 2020.

2010 *Mathematics Subject Classification.* 68T05, 68T50.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Induction*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing – *Text analysis*.

Key words and phrases. machine learning, text classification, long short-term memory, trigger detection, natural language processing, neural networks.

Emetophobia is highly disruptive, mostly because the main trigger comes from inside the body, which people cannot "escape". The act of being physically ill is usually the main trigger for emetophobes, but it is definitely not the only one. What is uncommon about this affliction is the fact that triggers also include verbal or written communications, images, audio recordings or videos that suggest, either implicitly or explicitly, the act of being ill.

The purpose of the current paper is to demonstrate the use of modern Machine Learning (ML) techniques in written trigger identification for emetophobia, going into detail about the technical aspects, from the architectures we used to the plugin we developed. We propose and compare multiple neural-network architectures, from simple *artificial neural networks* (ANNs) to Bidirectional Long Short-Term Memory Networks (BiLSTMs). We then choose the best performing ensemble of models and create an application that is useful for the therapy of people with emetophobia.

Emetophobia is rarely studied, especially in relation to Computer Science or Artificial Intelligence. The task we focus on is, to the best of our knowledge, the first of its kind. Text classification, however, is not new and BiLSTM nerworks are state-of-the-art in this regard. This motivates our chosen approach. We developed our own models using Tensorflow and Keras, and tested 10 architectures. We achieved outstanding results on our dataset, which were also tested "in the wild" within our own API and trigger-censoring application.

The rest of the paper is organized as follows. Section 2 provides more details about the issue at hand. Section 3 introduces the methodology we propose for emetophobia trigger identification and details the process of collecting the dataset, constructing and comparing the models, and implementing the API and the final application. Section 4 provides an analysis of our results. Section 5 presents what we intend to do in the future to provide more help for people suffering from this condition.

2. Background

This section provides details about emetophobia and people suffering from it. Then, we describe the ML techniques we used and the reason they were the best choice for the task. Finally, we provide an analysis of related work.

2.1. **Emetophobia.**

Emetophobia is not usually regarded as serious or worthy of extensive research. The main reason for this is that "No one likes being sick". It is a very common perception that emetophobia is just that: fear of being sick. However, it is much more than that. People suffering from emetophobia construct their whole lives around their avoidance and safety-seeking behaviours. Be it founded or completely unfounded, the fear is constant and

consistent throughout their day and the anxiety is scarcely reduced in intensity.

Avoidance behaviours include, but are not limited to, staying inside, not eating at restaurants, rarely attending events, avoiding places where people usually drink, etc. Safety behaviours include carrying plastic bags in case they feel sick, taking many pills, sanitizing their body and surroundings too often, burning their food to make sure it is cooked, etc.

The exact situation emetophobes fear varies. A 2011 Dutch study [10] arrived to the conclusion that a large majority of people fear vomiting themselves, but another large percentage fear vomiting around others or seeing others vomit. Around one-fifth of the participants fear all 3 of these events [10]. Another interesting characteristic of many people suffering from this phobia is that they, ironically, experience nausea when they are anxious or scared, which makes for a gruesome vicious cycle.

Avoidance and safety-seeking behaviours seem to be the primary factor that makes one's fear stagnate or increase with time [10]. Since censoring triggers can be considered avoidance, we feel it is necessary to explain the primary purpose of our work. The application we developed should be used in order for emetophobes to be exposed to triggers gradually, in a controlled environment, without fear that their free time spent online will trigger their phobia and disrupt their regular day. This application is not meant to replace therapy or encourage avoidance of any kind. On top of that, the API has additional uses, which we will discuss in Section 5.

2.2. **Machine Learning Techniques.** *Artificial Neural Networks* (ANNs) started to be hypothesized and, later on, implemented, in the 1940s. They are considered by many to be the basis of Artificial Intelligence (AI). ANNs loosely take after anatomical neurons and synapses, in an attempt to eventually reach human-level perception and understanding. Even though AI is still far from this goal, ANNs are useful in many Computer Science tasks and applications, from stock market predictions to image recognition and cybersecurity.

As an improvement to simple ANNs, *Recurrent Neural Networks* (RNNs) were introduced to allow the algorithm to analyse sequences of information of arbitrary length. RNNs, however, come with a high risk of exploding or vanishing gradient, which is where Long Short-Term Memory Networks (LSTMs) step in. LSTMs choose when to remember and when to forget information they parsed. They, however, only analyse information from the past. Bidirectional LSTMs are an enhancement of LSTMs, where information is parsed forwards, as well as backwards, in order to provide a better understanding of the input.

Machine Learning techniques have been growing in efficiency, as well as complexity, since their very beginning. At this moment in time, BiLSTMs are considered to be state-of-the-art in sequence analysis tasks, such as text classification.

2.3. **Related work.** Since emetophobia is scarcely studied in relation to Computer Science and Machine Learning, the next best comparison would be hate speech detection, followed closely by sentiment analysis. With the freedom of speech offered by the internet and, more specifically, social media, the issue of hate speech has become more and more prevalent and worrisome. Automated detection of hate speech is a need humanity could not have predicted a number of years ago. This task has been tackled by many researchers lately, with relevant work starting to emerge in 2015 or even earlier.

Davidson et al. [3] used 3 output classes (Clean, Offensive, and Hate) instead of the previously-popular 2 (Clean vs Hate) to differentiate between hate speech and offensive language. This is because many offensive terms are used online daily in a non-pejorative manner and the distinction is crucial. The dataset was made up of 25 thousand examples of tweets. The tweets were labeled by hand, by at least 3 workers, the final label being decided in a "most votes" fashion. The ML models compared in this article were Logistic Regression, Naive Bayes, Support Vector Machines, etc. [3].

Badjatiya et al. [1] compared different combinations of neural networks and embeddings to find the most appropriate one for tweet classification. The 3 classes used for the task are sexist, racist, and neither sexist nor racist. The dataset contained 16k instances of annotated tweets. Training was performed using 10-fold cross-validation. One of the best combinations proved to be LSTM with Random Embeddings, improved with Gradient Boosted Decision Trees. CNNs and LSTMs also performed admirably when combined with GloVe embeddings, and even better when also using GBDT [1].

Recently, Do et al. [5] employed a Bidirectional LSTM network to identify hate speech in Vietnamese social media text. The dataset used contained over 25 thousand Twitter comments, about 20,000 being used for training and 5,000 for testing. Each item was assigned a label from 1 to 3, 1 meaning CLEAN, 2 meaning OFFENSIVE, and 3 meaning HATE [5]. The model was compared with Support Vector Machines, Logistic Regression, and Gated Recurrent Units, this comparison clearly showcasing that the BiLSTM was the best choice for the task.

Other recent articles used SVMs [11], Bag-of-words [4], and other similar techniques, which are not necessarily comparable to our proposed approach.

3. Methodology

This section introduces the methodology we propose for emetophobia trigger identification. As with any similar endeavour, the first step was to extract a specific dataset of words and phrases, then to develop ML models capable of discerning between triggering and non-triggering phrases, followed by a thorough evaluation of the models. The final steps consisted of developing an API and a useful application where this API is integrated.

3.1. **Data set.** The dataset was chosen manually by searching the internet, specifically social media websites like Facebook and Instagram, and extracting words and phrases. Social Media was chosen as source for the data because nowadays, most people have a habit of spending a long time on the internet, hence this is where the probability of an encounter with a trigger rises drastically. The main reason for manually creating a new dataset is that research in emetophobia in relation to Computer Science is scarce, so there is no dataset that we were able to find that can be used in the task at hand.

We first selected 600 sentences, which were split into 300 triggers and 300 non-triggers. Triggers included sentences like "I feel sick" or "my child just threw up". Non-triggers included everything from "Happy birthday" to "I am traveling to Europe". The first version of the dataset contained phrases of up to 17 words. This version could have worked well with other approaches to the same task, but not to the one we chose. For example, if the models were directly trained to take as input longer texts of varying lengths and highlight the triggering parts of each text, then this dataset would have been useful. However, this is not the approach we chose.

Our models were trained to determine if a short piece of text is triggering or not, using an output value from 0 (meaning not triggering) to 1 (meaning triggering). The model has no information about the text as a whole. It does not receive a whole webpage, whole paragraphs or even whole sentences. The model only receives a small segment of the text and determines if that segment is triggering, then it receives the next segment of the text, and so on. The segment of the text the model receives is controlled from the outside by the algorithm in the API, which uses a sliding window approach, as detailed in Section 4.1.

Since the model receives short segments of the text and determines if they are triggering or not triggering, we found it fitting to modify the dataset so that the training is performed on appropriate sentences. If the model received a sentence of 17 words which was marked as triggering, it would be impossible for it to determine exactly which part of the sentence makes it triggering. It is unlikely that all of the 17 words are, in themselves, triggering. However, for shorter sentences, say, up to 5 words, the model can assume that all of

the words form a triggering phrase and, thus, it can label the sentence as a whole as triggering. This is why we decided to manually parse our dataset and extract the precise 5-word segment of each entry that determines if it is triggering. For example, the sentence "I traveled to Europe last month and the turbulence made me sick, but I felt fine later" became "the turbulence made me sick". The final dataset contains 600 sentences of up to 5 words.

The pre-processing of the dataset consisted of making all letters into lowercase, removing punctuation, numbers, emojis, special characters, repeating white space, etc. The next step parsed the dataset and replaced each word with its lemma using the *morphy* function [6] provided by Princeton's WordNet. The tokenization was performed using the *TwitterTokenizer* provided by NLTK [2]. Finally, the triggering phrases were marked with a value of 1, while the non-triggering phrases were marked with 0. Two fragments from the dataset can be seen in Figures 1 and 2. The dataset was made publicly available on Github [7].

travel to europe and	0
people who understand us	0
to make new friends	0
it is a beautiful country	0
north east of england	0
as i can remember	0
ocd and anxiety too	0
i love to help others	0

FIGURE 1. Non-triggering phrases in the dataset

the actual vomit itself	1
sick when i am drunk	1
it does make me gag	1
threw up from being sick	1
a bug and threw up	1
i did not throw up	1
puke after every feed	1
throw up after meals	1
threw up twice	1
never a stomach bug	1

FIGURE 2. Triggering phrases in the dataset

3.2. **Proposed ML models.** Since trigger identification requires analysis of word sequences, LSTMs are the most suitable ML technique for the task. Most of the model architectures used regression. However, to achieve as complete a

picture as possible, we also considered simple ANN and BiLSTM architectures, as well as classification models for each of these approaches. Some models use the NLTK stopwords dictionary during the tokenization process, while others do not. Other differences between models consisted of different dropout values, a different number of epochs, different learning rates, etc. Something that all of the models had in common was the first layer (excluding the input layer), which was an Embedding layer. In all instances, this layer used Stanford's GloVe Twitter pre-trained word vectors [8] with 100 dimensions.

The first architecture we trained and tested was a LSTM model with one layer of 100 units and a dropout value of 25%. This layer was then flattened and connected to a fully connected layer of 100 units, a fully connected layer of 25 units, and, finally, the output layer with 1 unit. This model used the stopwords dictionary, 100 training epochs and the default TensorFlow value for the learning rate. The architecture is illustrated in Figure 3. This model is henceforth denoted LSTM_1. For the second model, denoted LSTM_2, the first architecture was modified by adding one more LSTM layer with 100 units and a dropout value of 25%.

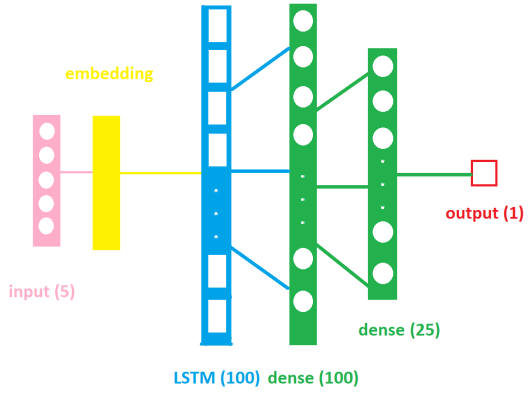


FIGURE 3. The architecture of the LSTM_1 model

The following 3 proposed models are all BiLSTM models. They use the stopwords dictionary modified by removing the word "up". This modification was performed because phrases like "throw up" were deemed important for this study. The first BiLSTM model (denoted BiLSTM_1) consisted of the input layer, the embedding layer, one BiLSTM layer with 200 units, which was then flattened and fed into a fully connected layer with 100 neurons, and then into the output layer. This model was trained for 100 epochs with the default learning rate value. The second BiLSTM model (denoted BiLSTM_2) had different fully connected layers, (one with 100 neurons, one dropout layer with

a value of 25%, one fully connected layer with 25 neurons). The architecture, as given by the *summary()* method provided by Keras, is displayed in Figure 4. This model was trained for 200 epochs with a learning rate of 0.0001.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 5)]               0
_________________________________________________________________
embedding (Embedding)        (None, 5, 100)            85600
_________________________________________________________________
bidirectional (Bidirectional (None, 5, 200)            160800
_________________________________________________________________
time_distributed (TimeDistri (None, 5, 100)            20100
_________________________________________________________________
flatten (Flatten)            (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               50100
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 25)                2525
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 26
=================================================================
Total params: 319,151
Trainable params: 233,551
Non-trainable params: 85,600
_________________________________________________________________
```

FIGURE 4. The Keras architecture of the BiLSTM_2 model

The final BiLSTM regression model (denoted BiLSTM_3) was also the largest and most computationally expensive one. The number of BiLSTM layers was doubled, and one more dropout layer was added between the last fully connected layer and the output neuron. This model was trained with the modified stop words dictionary, for 200 epochs, with a learning rate of 0.00005. Finally, two simple ANN models were constructed. The first model is illustrated in Figure 5. It was trained with the modified stop words dictionary, for 100 epochs. The second ANN model had the same structure, with additional dropout layers.

The classifications models we tested were variations of the architectures we already described. The main difference was at the output layer level, where two neurons were used instead of one. The output, which had so far been Boolean in nature, was one-hot-encoded to fit the new output layer structure.

After the best architecture was chosen, an ensemble of models was created. Each of the models in the ensemble was given a weight based on the metrics obtained in the testing phase and the final output was decided by computing a weighted average of all of the output values.

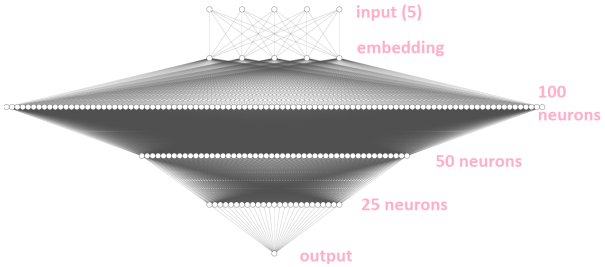


FIGURE 5. The architecture of the ANN_1 model

3.3. **Evaluation.** For the classification models, we used Binary Crossentropy as the loss function and accuracy for metrics. For the regression models, the main loss function used was *Mean Squared Error* (MSE). We implemented an accuracy measure for the regression models as well. The principle we used for this accuracy measure is illustrated in Figure 6. The same principle was applied in the final API.

We used 10-fold cross-validation, so that the result obtained was as objective as possible, with a lower probability of it becoming overfitted or biased towards parts of the dataset. One instance of each model was randomly given one fold for testing, and the other 9 folds for training. Thus, resulted were 10 different instances per architecture. When computing the best architecture, all of these models were taken into account. The 10 models were ordered ascendingly by loss. Then, the first 3 models (with the lowest loss) were assigned a weight of 0.1, the following 4, a weight of 0.2, and the final 3 were assigned a weight of 0.3. The weights given to the instances within an ensemble are displayed in Figure 7.

	Expected value = 0	Expected value = 1
Output < 0.5	Correct	Incorrect
Output >= 0.5	Incorrect	Correct

FIGURE 6. Accuracy measure for the regression models

4. RESULTS AND DISCUSSION

A comparison of the results obtained for all of the architectures can be seen in Figure 8. This table shows the model name, the model accuracy, and the model loss for all of the architectures. For the three BiLSTM regression architectures, more measures are displayed: Mean-Squared Error (denoted MSE), Precision (P), Recall (R), and F1 (the F1 score). These measures were obtained for the ensembles on the 600 entries in the whole dataset. All of

10	0.1	0.004117
5	0.1	0.013662
6	0.1	0.014248
9	0.2	0.020728
7	0.2	0.022361
1	0.2	0.025597
4	0.2	0.028322
3	0.3	0.029072
2	0.3	0.031339
8	0.3	0.055445
Total		0.028681

FIGURE 7. Final loss of an ensemble - The first column represents the order number of the instance; the second column represents the weight assigned to the instance; the third column represents the loss of the instance; The last line in the table, labeled "Total", shows the final loss of the ensemble, computed using a weighted sum of the model losses (i.e. $Total = \frac{\sum_{m=1}^{10} loss(m) * weight(m)}{2}$, where m=model)

the models performed admirably, but the best performance was measured for the BiLSTM network with one layer (denoted BiLSTM_1). Even though the accuracy was comparable for most of the models, the loss for this architecture was lower than the other regression models and significantly lower than the classification models. After selecting this model as the winner, we constructed the ensemble, assigning weights to the instances following the rules described above.

Figure 9 illustrates a graph of the loss with respect to the training epoch for the 10 instances of the chosen architecture (namely, BiLSTM_1). Similar to the other architectures, this model learned quickly within the first 10 to 20 epochs, then slowed down until the final epoch. All of the 10 instances seem to have learned at the same rate, achieving a similar loss by the end of the training phase. We tried to slow down the learning process in hopes of achieving an even better result by decreasing the learning rate and increasing the number of epochs, but the performance did not exceed the best performance achieved previously.

4.1. **API and Application.** We developed a server written in Flask (Python) that implements the approach described in the previous sections. The main endpoint of the API receives a POST request with the text to analyse. The

	Accuracy	Loss	MSE	P	R	F1
ANN_1	0.938333327	0.07811134				
ANN_2	0.939999989	0.063035519				
BiLSTM_1	0.976666662	0.028680518	0.006194	0.996667	0.99335	0.995008
BiLSTM_2	0.965833321	0.040919585	0.01008	0.996667	0.996667	0.996667
BiLSTM_3	0.962499994	0.040133117	0.009327	0.993333	0.996656	0.994992
LSTM_1	0.964999998	0.038441581				
LSTM_2	0.963333318	0.036721471				
BiLSTM_C_1	0.979999992	0.294667984				
BiLSTM_C_2	0.972499999	0.374570083				
LSTM_C_1	0.970000002	0.378273888				

FIGURE 8. Comparison of the metrics for the 10 architectures

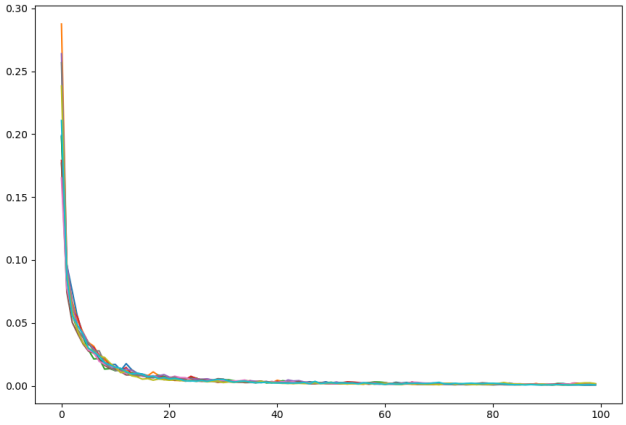


FIGURE 9. Loss graph for the winning model

response contains a text of the same length, with the triggering phrases censored using the asterisk character. The text is parsed using a "sliding-window" approach, detailed in what follows. The first 5 words are analysed. If they are triggering, they are replaced with asterisks and the algorithm goes on to the next 5 words, i.e. words 5-10 of the text. If they are non-triggering, the algorithm moves 3 words to the right, thus keeping the last two words from the previous input to ensure that these words were not part of a triggering phrase.

As an example, the text "I traveled to Europe last month and the turbulence made me sick, but I felt fine later" will be parsed as follows. First, the words "I traveled to Europe last" will be analysed and they are labeled as negative.

The algorithm appends the words "I traveled to" to the response and moves on to analyse the phrase "Europe last month and the". It is not triggering, so the response becomes "I traveled to Europe last month" and the algorithm goes on to analyse the phrase "and the turbulence made me". The input is not triggering, so the response becomes "I traveled to Europe last month and the turbulence". The algorithm now analyses the phrase "made me sick, but I", which is labeled as positive. The response thus becomes "I traveled to Europe last month and the turbulence **** ** ****, *** *". The algorithm now analyses the phrase "felt fine later", which is labeled as negative. The response "I traveled to Europe last month and the turbulence **** ** ****, *** * felt fine later" is returned by the endpoint.

The API was utilized in a Chrome extension that censors written triggers encountered online. Figure 10 depicts a webpage with and without the extension. The algorithm is very strict with variations of the word "eat". For our purposes, this could be considered extreme. The same applies for mentions of the word "anxiety". As mentioned above, the word "up" causes problems for the algorithm. These issues are all mild since they cause excessive censorship, which is better than no censorship at all.

False positives are to be preferred in such a scenario, where the sufferer would rather see less of the non-triggering text of a webpage, rather than more of the triggering text. Furthermore, there are extreme cases where even the words "eat" or "anxiety" cause an increase in the stress levels of an individual, so such censorship, even when not expected or purposeful, could end up adding to the positive characteristics of the application. False negatives, on the other hand, would be a much graver issue. The problem of false negatives is one that, if present, should be brought to a minimum right away. The false negatives we encountered were only labeled as negative because of improper representation in the dataset.

4.2. **Comparison to related work.** Emetophobia is rarely focused on in research, especially with respect to Computer Science and Artificial Intelligence. This is the reason why we were unable to find papers on this topic that we could compare our results to. Our solution to this issue was to compare with similar works in slightly different domains. Hate speech detection is also an up and coming topic in research, since the time spent online by individuals is increasing and this leads to more and more exposure to hateful and rude comments. We deemed this task similar enough to emetophobia trigger detection to address comparable works in this section.

The pre-processing phase we employed in our approach is similar to the approaches we analysed. GloVe embeddings are also frequent in research that

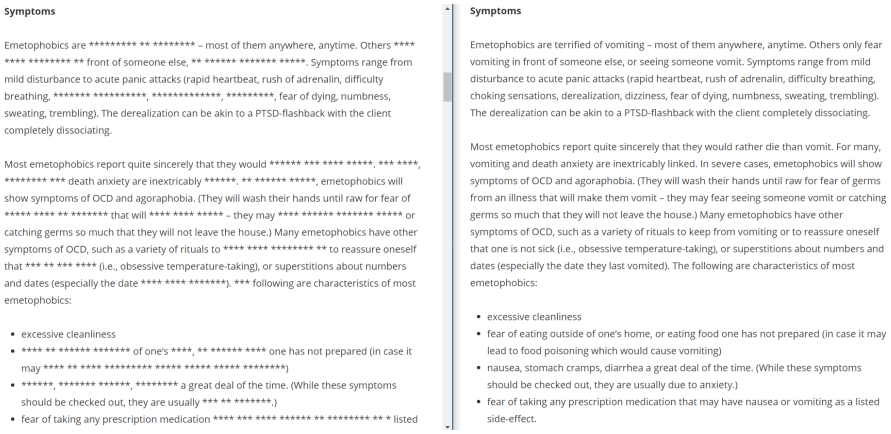


FIGURE 10. Comparison of a webpage with (left) and without (right) the extension enabled

utilizes social media text since these embeddings are trained on billions of tweets and have learned colloquial language well.

Perhaps the most obvious difference between the proposed approach and other comparable works is the dataset. Most similar articles had access to thousands of instances, while we only used 600. Some had workers manually label their data according to strict rules, thus ensuring accurate and consistent labels, while we relied on our own perception when extracting relevant phrases. Our dataset can and will be improved upon. Another important distinction is that most similar articles use classification models, while our proposed approach highlights regression models as better-performing for this specific task. Classification models were similar in accuracy, but with a significantly higher loss than regression models.

5. Conclusions and future work

This paper presented our approach to the issue of censoring online content that can be triggering for people with emetophobia (the extreme fear of vomiting). We collected a dataset, constructed different neural network architectures, and compared their performance on our dataset. The architecture with the best performance was turned into an ensemble. This ensemble was utilized in an API, which was then incorporated into a Chrome extension that censors triggering content on the internet.

Future improvements to the existing API will commence with expanding the dataset to include a larger number of instances. Tweaks need to be made to make sure that the model can discern between triggering and non-triggering

uses of the word "up", "anxiety", "throw", etc. Additional endpoints of the API can be used in an upcoming project, a Virtual Reality therapy application currently in development. The amount of triggering phrases utilized by the user will decide if they are ready to pass to the next level or not.

Modern ML techniques have many uses in everyday life. While some uses are purely for entertainment or comfort, others can drastically improve the life of people suffering with certain afflictions. Since mental health is such an important aspect of one's life, applications that aid therapy or improve a patient's quality of life will remain a crucial part of Computer Science research for years to come.

References

[1] Badjatiya, P., Gupta, S., Gupta, M., and Varma, V. Deep learning for hate speech detection in tweets. In *Proceedings of ACM WWW'17 Companion* (Republic and Canton of Geneva, CHE, 2017), WWW '17 Companion, International World Wide Web Conferences Steering Committee, p. 759–760.

[2] Bird, S., Klein, E., and Loper, E. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[3] Davidson, T., Warmsley, D., Macy, M., and Weber, I. Automated hate speech detection and the problem of offensive language. *ICWSM* (2017), 512–515.

[4] Djuric, N., Zhou, J., Morris, R., Grbovic, M., Radosavljevic, V., and Bhamidipati, N. Hate speech detection with comment embeddings. In *Proceedings of the 24th International Conference on World Wide Web* (New York, NY, USA, 2015), WWW '15 Companion, Association for Computing Machinery, p. 29–30.

[5] Do, H. T.-T., Huynh, H. D., Van Nguyen, K., Nguyen, N. L.-T., and Nguyen, A. G.-T. Hate speech detection on vietnamese social media text using the bidirectional-lstm model. In *Proceedings of VLSP 2019* (2019).

[6] Fellbaum, C. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[7] Mircea, M.-M. Emetophobia dataset, Aug. 2020.

[8] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *In EMNLP* (2014).

[9] Ritchie, H., and Roser, M. Mental health. *Our World in Data* (2018). https://ourworldindata.org/mental-health.

[10] van Hout, W. J. P. J., and Bouman, T. K. Clinical features, prevalence and psychiatric complaints in subjects with fear of vomiting. *Clinical Psychology & Psychotherapy 19*, 6 (2012), 531–539.

[11] Warner, W., and Hirschberg, J. Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media* (Montréal, Canada, June 2012), Association for Computational Linguistics, pp. 19–26.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu St., 400084 Cluj-Napoca, Romania

Email address: mmic2002@scs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 2, 2020
DOI: 10.24193/subbi.2020.2.03

A VIEW ON DEEP REINFORCEMENT LEARNING IN IMPERFECT INFORMATION GAMES

TIDOR-VLAD PRICOPE

Abstract. Many real-world applications can be described as large-scale games of imperfect information. This kind of games is particularly harder than the deterministic one as the search space is even more sizeable. In this paper, I want to explore the power of reinforcement learning in such an environment; that is why I take a look at one of the most popular game of such type, no limit Texas Hold'em Poker, yet unsolved, developing multiple agents with different learning paradigms and techniques and then comparing their respective performances. When applied to no-limit Hold'em Poker, deep reinforcement learning agents clearly outperform agents with a more traditional approach. Moreover, if these last agents rival a human beginner level of play, the ones based on reinforcement learning compare to an amateur human player. The main algorithm uses Fictitious Play in combination with ANNs and some handcrafted metrics. We also applied the main algorithm to another game of imperfect information, less complex than Poker, in order to show the scalability of this solution and the increase in performance when put neck in neck with established classical approaches from the reinforcement learning literature.

1. Introduction

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning [15]. We are thinking about games as simulations of our real world with special, particular features and rules, that is why, lately, this field represented the perfect playground for machine learning research. Solving particular game environments can lead to solutions that scale to more complex, real-word challenges such as airport and network security, financial trading, traffic control, routing ([8],

Received by the editors: 27 July 2020.
2010 *Mathematics Subject Classification.* 68T05.
1998 *CR Categories and Descriptors.* I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems – *Games*.
Key words and phrases. Artificial Intelligence, Computer Poker, Adaptive Learning, Fictitious Play, Deep Reinforcement Learning, Neural Networks.

[10], [17]). We witnessed the rapid development of computer AI with the massive success in perfect-information games like Chess and Go (AlphaGo Zero, 2014 [13]; LeelaChessZero 2016 [14],), but researchers have yet to reach the same progress in imperfect-information games (AlphaStar, Deepmind, [1]).

Before the 2000s, poker solving approaches could have been categorized (from an architecture point of view) in 3 main classes: expert system, game-theoretic optimal play and simulations based on enumerations [2]. One of the most successful poker bots at the time was Loki [2] (the same group later developed DeepStack [9]), which used the above methods combined with parametric models for opponent modelling. Although this approach is very far from a Nash-equilibrium, it finds locally optimal solutions to certain situations and its performance can be used as a threshold when comparing our modern ways of solving imperfect information games. Therefore, I will develop my own version of Loki Poker bot, as our first agent, in order to use for testing.

Fictitious play [4] is a popular method for achieving Nash Equilibria in normal-form (single-step) games. For our deep reinforcement agent, we try to add on a variant of Fictitious play, normally used in self-play scenarios (Neural Fictitious Self-Play [7]) and show how this approach can be also re-modelled and applied to a one-player environment. It was proven that NFSP provides poor performance in games with large-scale search space and search depth [21], because of the complexity of opponents' strategy and the fact that a DQN (Deep Q Network [11]) learns in offline mode and it uses only raw, crude data as input. Moreover, NFSP wasn't tested on the more complex variant no-limit. I try to address these issues by considering some high-level hand-crafted heuristics to go alongside raw data from a state of play. My approach uses, in addition, hard coded rankings of card combinations and Monte-Carlo heuristics for assessing an approximate strength of the opponent hand. This will represent the main idea behind the second agent I am going to build.

I empirically evaluate each agent in two-player (heads up) zero-sum computer poker games and explain how each one can work even in a multiple-player scheme with limited performance loss.

2. Background

In this section I provide an overview of reinforcement learning and fictitious play in extensive-form games. I am going to mark some important mathematical elements here as they will be used for reference in the next sections.

2.1. **Reinforcement Learning.**

Reinforcement learning [15] agents typically learn to maximize their expected future rewards from interaction with an environment. The environment is usually modelled as a *Markov decision process (MDP)*. Reinforcement

learning algorithms can learn in many ways, but we are interested in the ones that learn from sequential experience in the form of transition tuples from one state (s) to another taking into account the action (a) necessary to reach the new state and the respective reward of that operation (r): $(s_t, a_t, r_{t+1}, s_{t+1})$. The goal of the agents is to maximize their rewards, this is typically done by learning the *action-value function* Q, defined as the expected gain of taking action a in state s and following the policy π: $Q(s,a) = E^{\pi}[G_t | S_t = s, A_t = a]$. Here, $G_t = \sum_{i=t}^{T} R_{i+1}$is a random variable of the agent's cumulative future rewards starting from time t [15]. From this, it easily follows that we may want to take the action of the highest estimated value Q, that's why *Q-learning* [20] was invented as a way to learn about the greedy policy storing and replaying past experience. To approximate the *action-value function*, a neural network can be used and this approach is one of the most popular when dealing with more complex games and the system is called a *DQN* [18].

2.2. **Fictitious Play.**

Fictitious play (FP) [4] is a game-theoretic model of learning from self-play. Fictitious play is commonly defined in normal form (single-step games), which is exponentially less efficient for extensive-form games (multi-step games). To provide more context, fictitious players choose their best response against the other players' (opponents') average behaviour; in normal-form, this defines a player's behavioural strategy $\hat{\pi}$ as a probability distribution over all the possible actions. Heinrich et al. (2015) [6] introduced *Full-Width Extensive-Form Fictitious Play (XFP)* that enables fictitious players to update their strategies in behavioral, extensive form, resulting in linear time and space complexity. In extensive-form fictitious play, we have a convex combination of normal-form strategies $\hat{\sigma} = \lambda_1 \hat{\pi}_1 + \lambda_2 \hat{\pi}_2$, that was proven it can achieve a realization-equivalent behavioral strategy σ, by setting it to be proportional to the respective convex combination of realization-probabilities: $\sigma(s,a) \propto \lambda_1 x_{\pi_1}(s)\pi_1(s,a) + \lambda_2 x_{\pi_2}(s)\pi_2(s,a) \ \forall s, a$ [6] [7], where $\lambda_1 x_{\pi_1}(s) + \lambda_2 x_{\pi_2}(s)$ is the normalizing constant for the strategy at information state s. This is important as it provides strong theoretical background for approximating the behavioural normal-form strategies in order for a convex combination to work in extensive-form games.

In order to define a family of probability distributions, let $\Delta(n)$ a standard simplex in R^n, $v_i \in \Delta(n)$ being the i-th vertex and let $H : \mathrm{Int}(\Delta(n)) \to R$ the entropy function $H(p) = -p^T \log(p)$. In a two-player game, where there is the concept of an opponent, each player chooses its strategy $p_i \in \Delta(m_i)$, $m_i \in N^*$ and collects the associate reward given by the value-function: $V_i(p_i, p_{-i}) = {p_i}^T M_i p_{-i} + \tau \cdot H(p_i)$, where $-i$, $i \in \{1, 2, ..., n\}$ refers to the complementary set

$\{1, 2, ..., i-1, i+1, ...n\}$ [12]. M_i is the game-specific reward matrix of shape (m_i, m_{-i}) that holds entries with the numerical compensation for player 1 having the strategy p_i and the opponent having the strategy p_2. Note that we shall use reinforcement learning to approximate this value-function through sampling and observing the reward at each state. If follows that we can define player i 's best response as a function $\beta_i : \Delta(m_{-i}) \to \Delta(m_i)$, $\beta_i(p_{-i}) = \arg\max V(p_i, p_{-i})$ and player i's average response until step k in the game as empirical frequencies $\pi_i(k) : N \to \Delta(m_i)$ of player P_i [12].

Depending of the game type, there are multiple Fictitious Play (FP) abstractions: in discrete time, continuous and dynamic continuous. For *discrete time FP*, we can define the strategy at step k as the best response to the empirical frequencies of opponent actions:

$$p_i(k) = \beta_i(\pi_{-i}(k)) \quad (1)$$

In *continuous time FP*, the following equations are used:

$$\frac{d}{dt}\pi_i = \beta_i(\pi_{-i}(t)) - \pi_i(t), i = \overline{1,2} \quad (2)$$

The difference that comes with the third type of abstraction, in which Poker falls in (as a multi-step, multi-player game in which our strategy and our opponent's are continuously and dynamically changing at each step), is that each player has access to the derivative of his empirical frequency $\frac{d}{dt}\pi_i$, therefore the strategy at moment t can be defined as:

$$p_i(t) = \beta_i\left(\pi_{-i}(t) + \eta\tfrac{d}{dt}\pi_{-i}(t)\right), \text{ with } \eta \text{ positive parameter} \quad (3)$$

We interpret this formula as a player choosing his best response based on current opponent's average strategy profile combined with a possible change of it that may appear in the future.

The authors of this study, *anticipatory dynamics of continuous-time dynamic fictitious play* [12] show that, depending on the game, for a good choice of η, the stability in Nash equilibrium points can be improved. The challenge that comes with it though is the fact that the derivative cannot be directly measured and needs to be approximated or reconstructed by empirical frequencies measurements.

Formula (3) will stay at the basis of our Reinforcement Learning agent, in the following sections, we will explain how we tackle the aforementioned problem and how we define the best and average response through supervised and reinforcement learning.

3. Developing the Agents

In this section, I am going to address the technical details and the main process of building my 2 agents mentioned in the introduction. Therefore, my

first agent should be a reinforcement learning free one, that's why I am going to build it as my own mini remake version of Loki [2] featuring betting decisions with card heuristics and opponent-modelling. The second agent will learn Poker training with the first agent trying to consistently beat him, treating the opponent as part of the environment.

3.1. **Agent 1.**

We construct this agent mainly as an expert system at its core with heuristics for betting decisions and opponent-modelling for exploitations. Agent 1 defines his policy π depending on the street of the game, on the hand strength metrics and whether an opponent model was found or not. We got a look-up table for the preflop stage containing the rankings of all 2-card pairs. Starting with the flop, we maliciously evaluate the win rate in a particular situation by enumerating all possible 5-card combinations with the current board and using another look-up table that contains the rankings of all *7462* such distinct combinations of card, with 1 being a *royal flush* and 7462 being *7-5-4-3-2* with at least 2 different *suits*.

This is not really enough as we need to take into account possible future hand strength increase or decrease. That's why we also compute positive hand potential by *Monte-Carlo* simulation of states that can derive from the current one and assessing those with the look-up table mentioned earlier. In order to be completely sure when to place a bet/raise, we need to analyze the limit break points in such a hand strength metric distribution. For that, I simulated 1000 games of Poker and assessed the hand strength of the best 5-card combination that includes the two hole cards. The results can be observed in the figure 1. As expected, most of the hands are really weak, but we can expect great results of our hand strength metric indicates at least *0.8/1.*

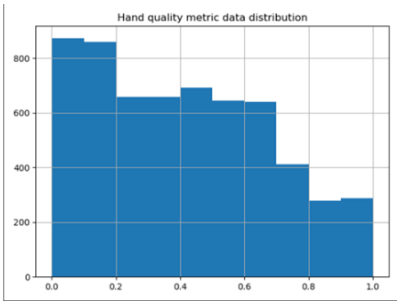


FIGURE 1. Hand strength distribution over 1000 games

Opponent modelling

The goal of this part is to find a good approximation of opponent average strategy π^{-i} with a good accuracy of predicting the *fold* moves. I do that by training two separate supervised classification models which are active during the games, collecting data about the opponent.

I use a naïve Bayes classifier (to replicate the Bayesian analysis presented in the Loki paper), after a certain number of actions taken, *min_steps_bayes*, the

model will train and try to guess opponent's next move. The input for this classifier consists of a 1D array containing an expected average hand strength of the opponent (obtained from Monte-Carlo simulations), raise demand, the opponent stack, the number of consecutive and same-suits cards on the board and the street number.

I also use a deep neural network as our second classifier. I decided to use a CNN architecture, the input being represented as an image of the current board state alongside some of the scalar features mentioned at the other classifier. This will also have a *min_steps_CNN* parameter set at the beginning, usually at least two times higher than *min_steps_bayes*, after which, the model will be ready to start predicting.

The main reason that I use this configuration is that the Bayes model shines when less data is available, taking into consideration class probabilities but then is really outperformed by a neural network when much more data units are available, so in the long run, we shall keep the neural model active as we deactivate the first one.

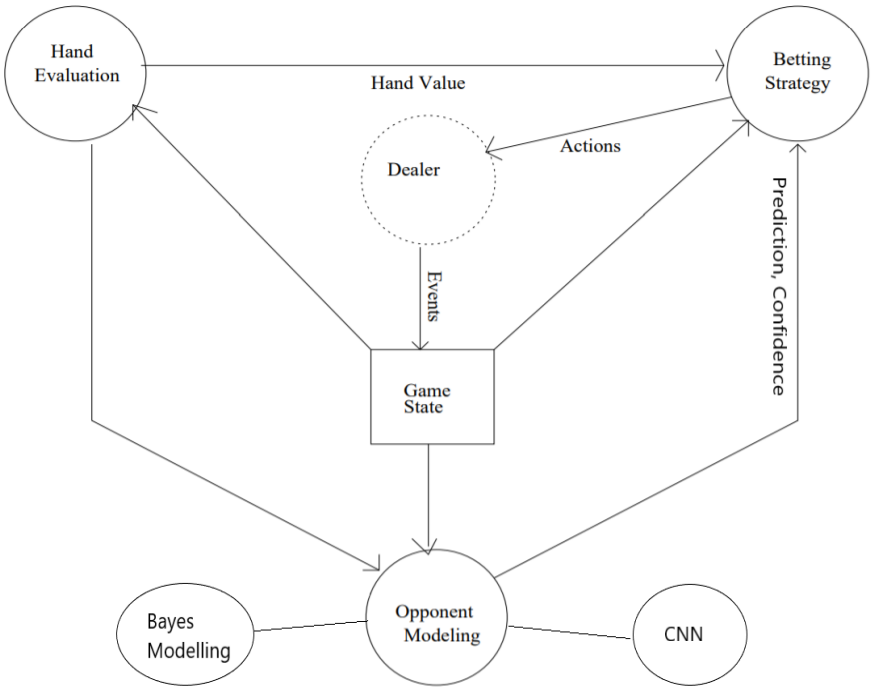


FIGURE 2. Agent 1 architecture (our mini version of Loki.)

Algorithm 1 | Agent 1, expert system with neural opponent-modelling and Bayes classifier method

---

Initialize Bayes memory M_{Bayes}
Initialize CNN memory M_{CNN}
Initialize $acc \leftarrow 0$
for 1 : nogames do
  Initialize new game G and execute agent via RUNAGENT for each player in the game
function RUNAGENT(G)
  wr $\leftarrow$ getwinrate (currentposition)
  if $a \neq \hat{a}$ then
    $nomistakes \leftarrow nomistakes + 1$
  Set policy $\sigma \leftarrow \begin{cases} acc - greedy\,(\text{exploiting} - \text{expert} - \text{system}\,(wr)\ \text{policy}), & \text{with probability } acc \\ \text{simple} - \text{expert} - \text{system}\,(wr)\ \text{policy}, & \text{with probability } 1 - acc \end{cases}$
  Observe initial information state s and opponent action a
  Store behaviour tuple (s, a) in supervized learning memory M_{Bayes} and M_{CNN}
  if $\mathrm{M}_{Bayes}.size$ % minstepsbayes then :
    $acc \leftarrow$ train Naive $-$ Bayes $-$ Classifier
  if $\mathrm{M}_{CNN}.size$ % minstepscnn then :
    $acc \leftarrow$ train Neural $-$ Network $-$ Classifier
  if agent follows targeted response policy σ=$acc - greedy$ then
    Save our prediction $\hat{a}$
end function

The agent functions as an expert system with betting decisions based on mentioned hand strength heuristics at first and then, after it collects enough data to start the Bayes classifier it changes its policy (in an accuracy greedy way) to another expert system for exploitation. This system will be deactivated as we gather enough information to start the neural model and use this one for opponent's moves prediction. We shall choose to use the opponent modelling part based on the number of mistakes the classifies make during a few games.

This agent will be important in testing by adversarial reasoning, as it offers a measurement to opponent's exploitability through the opponent modelling part.

3.2. **Agent 2.**

This deep reinforcement learning agent will continuously learn to play Poker by training with **Agent 1** from scratch. Its strategy of play combines the greedy strategy β offered by the action-value function with the average strategy π obtained though supervised classification.

Recall the equation (3), subtracting π_i from both sides and using (1) yields:

$$\frac{d}{dt}\pi_i = \beta_i \left(\pi_{-i}(t) + \eta \frac{d}{dt}\pi_{-i}(t) \right) - \pi_i(t) \qquad (4)$$

In *NSFP* [7], the authors chose a discrete time approximation of the derivative: ${\beta^{t+1}}_i - {\pi_i}^t \approx \frac{d}{dt}{\pi_i}^t$. Their motivation for this is the fact that a change in $\hat{\pi}_i$ at step $t+1$ is proportional to ${\beta^{t+1}}_i - {\pi_i}^t$ which is the normal-form update direction of discrete-time FP. If substituted in (4) yields:

$$\begin{array}{rcl} p_i(t) & \approx & \beta_i(\pi_{-i}(t) + \eta(\beta_i(\pi_{-i}(t+1)) - \pi_{-i}(t))) \quad \Leftrightarrow \\ p_i(t) & \approx & \beta_i((1-\eta)\pi_{-i}(t) + \eta\beta_i(\pi_{-i}(t+1))) \end{array}$$

and this is how we arrive at the combined policy approach $\sigma \equiv (1-\eta)\hat{\pi} + \eta\hat{\beta}$ which was proven to be really good in practice [7], being tested successfully on Leduc and Limit version Hold'em Poker for self-play agents. We will use this formula for our Agent 2, however compared to the referenced experiments, this time, we are applying it to the no-limit version of the game with hand-crafted inputs and we are going to treat it as a single player game considering the opponent as part of the environment. The definition of the combined policy approach, in theory, allows for such a change of perspective and to my knowledge, these exact experiments haven't been conducted in Texas Hold'em Poker.

Therefore, Agent 2 uses 3 neural networks. First, a *DDQN* system [18] with a *value network* $Q\left(s, a\left|\theta^Q\right.\right)$ for predicting the Q values for each action based on data from M_{RL}. It trains through backpropagation using the *Bellman equation* with future Q values obtained through a *target network* $Q'\left(s, a|\theta^{Q'}\right)$. Secondly, we use a *policy network* $\Pi\left(s, a\left|\theta^{\Pi}\right.\right)$ to define our agent's average response based on data from M_{SL}. Note that M_{RL} and M_{SL} are two reservoirs of data that are updated frequently in the game, the first one storing transitions and the second one storing state-action tuples used for supervised classification (Algorithm 2). M_{RL} is implemented as a circular buffer as it needs much more memory to operate. We choose our main policy σ from a mixture of strategies: $\beta = \varepsilon - greedy(Q)$ and $\pi = \Pi$: $\sigma \equiv (1-\eta)\hat{\pi} + \eta\hat{\beta}$, $\eta \in (0, 1]$.
Observe that the algorithm used has general scope and may be used in other games, MDPs with imperfect information or to practical real-life problems.

Note that we can also set the main policy σ every step t in the game for a more stochastic approach. I will actually do that in the experiments to test this small change to the NFSP algorithm.
The neural networks for the two strategies are implemented as CNNs and will have mainly the same architecture, the only difference appearing at the last layer

Algorithm 2 | Agent 2, reinforcement learning agent with fitted Q-learning

for 1 : nogames do
Initialize new game G and execute agent via RUNAGENT for each player in the game
function RUNAGENT(G)
Initialize replay memories M_{RL} (*circular* buffer) and M_{SL} (own behaviour dataset)
Initialize average − policy network $\Pi\left(s, a|\theta^{\Pi}\right)$ with random weights θ^{Π}
Initialize action − value network $\mathrm{Q}\left(s, a|\theta^{Q}\right)$ with random weights θ^{Q}
Initialize target network with weights $\theta^{Q'} \leftarrow \theta^{Q}$
Initialize $\pi - \beta$ parameter η
for each episode do

$$\text{Set policy } \sigma \leftarrow \begin{cases} \varepsilon - greedy\,(Q)\,, & \text{with probability } \eta \\ \Pi, & \text{with probability } 1-\eta \end{cases}$$

Observe initial information state s_1 and reward r_1
for $\mathrm{t} = 1, \mathrm{min}$replaymemorysize do
Sample action a_t from policy σ
Execute action a_t in emulator and observe reward r_{t+1} and next information state s_{t+1}
Store transition $\left(s_t, a_t, r_{t+1}, s_{t+1}\right)$ in reinforcement learning memeory M_{RL}
if agent follows best response policy $\sigma = \beta\,(= \varepsilon - greedy\,(Q))$ then :
Store behaviour tuple $\left(s_t, a_t\right)$ in supervised learning memory M_{SL}
Update θ^{Π} with gradient descent on loss
$L\left(\theta^{\Pi}\right) = \mathrm{E}_{(s,a)\sim M_{SL}}\left[KL\ Divergence\ \Pi\left(s, a|\theta^{\Pi}\right)\right]$
Update θ^{Q} with gradient descent on loss

$$L\left(\theta^{Q}\right) = \mathrm{E}_{(s,a,r,s')\sim M_{RL}}\left[\left(r + \max_{a'}\ Q\left(s', a'|\theta^{Q'}\right) - Q\left(s, a|\theta^{Q}\right)\right)^{2}\right]$$

Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
end function

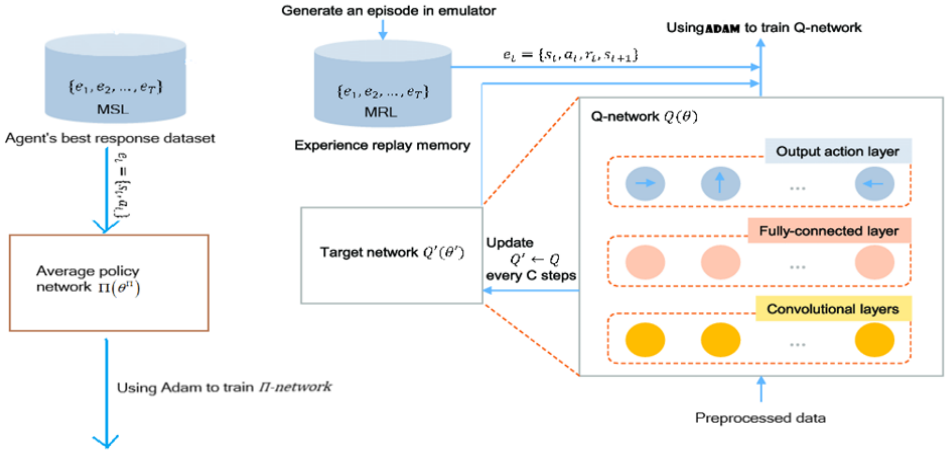


FIGURE 3. Overview of Agent 2 architecture.

The input is represented as a *17x17x9* 3D array containing the images of the last two board states joined by the scalar features we mentioned at **Agent 1** where we add the opponent last action. The fact that we add the last board state and the opponent last action is due to wanting to test an *attention mechanism* similar to the one used for *AlphaGo Zero* [13]. The CNN is composed of 5 hidden layers: 4 layers of convolution, 2 MaxPooling and 1 fully-connected. The loss for the value network remains the classic *MSE* and for the policy network, we use *KL divergence*. M_{SL} will be updated using *reservoir sampling* [19] and M_{RL} will function as a *circular buffer*. Above (figure 3), we can see the architecture of this agent.

4. Experiments

I am mainly focused on no-limit variant of Poker for experiments, but I am also going to test the algorithm on another imperfect information game to solidify our claim of general scalability and applicability. I devised a less complex game than Poker and verify the necessity of the essential components by rigorously evaluating the respective performances. In case of Poker, we are going to measure each agent's performance against some generic players and against each other.

4.1. **A pilot experiment.**

Introducing *Blop game* (figure 4), originally a perfect information game that consists of a quadratic matrix/image where a pixel is colored as blue (our player), another as green (the exit) and the third one as red (the enemy). The player can move in all 8 directions associated with the grid, or may choose to stay still. The player receives a negative 1 reward for moving in any direction and a positive 20 for reaching the exit, but it gains negative 300 if it hits the enemy moment when also the game ends. The objective of this game is to reach the destination as efficient as possible.

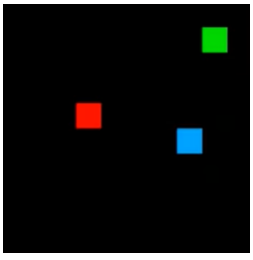


Figure 4. Blop Game.

Without any additional rules, this forms a deterministic game that can be solved quite easily by popular search methods such as A^*. Because we want to observe the functionality of the algorithm in the field of games with imperfect information, we will make a **fundamental change** in terms of the base rules used. Thus, we will ***poison*** 5 out of the

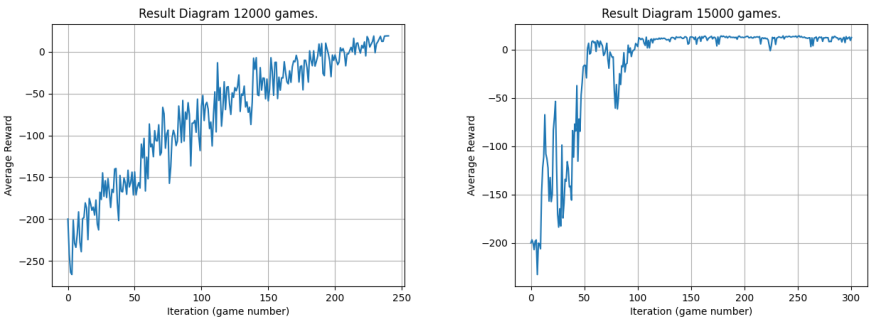
9 basic moves: when the player tries to stand still or move vertically or horizontally, he will make a random move instead. Thus, only the **4 movements** (diagonally) would work as intended.

For this experiment, the initial configuration uses a 10x10 image and is always the same: the player in the upper left corner, the exit in the center and with enemy steady near the exit on the segment formed by the initial points of the player and the exit. This way, the max reward that we are aiming for is **20**, obtained only through diagonal moves.

We use *algorithm 2*, all we feed the algorithm is the **RGB** images of states, we therefore use convolutional neural networks with 3 hidden layers (2 convolutions and 1 fully-connected). The parameter η was set to **0.1**, ε to **0.12**, max size of $M_{SL}$to **2m** and for M_{RL} to **20k**. We updated the parameters of Q and Π networks once every **4** steps (for each one) and the *target network* parameters were reset once every **5** episodes.

In order to study the performance and the speed of convergence, I compare the results with the results obtained by a standard *Double DQN* [11], a very popular system for solving games. This was implemented through setting the parameter η to **1** (always selecting the greedy strategy).

Figure 5 shows a crushing victory for our implementation that uses a combination of greedy and average strategies.



(a) Standard DDQN training process. Each stat point represents the aggregate performance in last 50 episodes

(b) Algorithm 2 training process. Each stat point represents the aggregate performance in last 50 episodes

FIGURE 5. Comparison between algorithm 2 performance in Blop Game and a standard method of solving games from literature.

It learns much quicker that it should not rely on anything apart from diagonal moves for reaching the goal - note that the very first reward is different.

Both greedy and average strategies of Agent 2 converge to the same optimal solution of reaching the exit in 5 moves with only diagonal moves. However, while at episode 5000, the algorithm 2 was pretty much done, the DDQN aggregate reward was still in the negatives 50.

I then modified the game to allow the enemy to randomly move, as the player moves, in order to add even more imperfect information to the game. This did not pose a challenge to our agent though, as it solved the game almost as fast as the previous version with quicker convergence speeds than DDQN's.

4.2. **General specifications for the Poker games.**

The format I am using for the games is heads-up, no-limit with **100** chips as starting stack and **5** chips small blind. For performance evaluations I am using two metrics: ***average stack*** over a fixed number of games and ***mbb/h*** (milli big blinds per hand = 1/1000 of a big blind). This is normally the metric to use for addressing performance in Poker, very many articles use this one for their experiments ([21], [7], [5], [3]). To provide some intuition, the values for a mbb/h metric will usually stay in the interval [-750, 750] and a human professional player would aim for winnings of **40-50 mbb/h**. An average stack of over 100 guarantees, most of the time, a match **win rate** of at least 50%.

The generic players used are the following: *Randomplayer* (a player that chooses call 3 times out of 5 and the other actions 2 times out of 5 with equal probable chance), a *Callplayer* (a players that always calls) and *HeuristicMCplayer* (a player that chooses its actions based only on Monte-Carlo simulations and not look-up tables). We expect the last generic artificial player described above to be the strongest challenger as simulations are generally very useful in Poker because it is important to know, objectively, what are your chances to win to make a bet, excluding the physchological element and the concept of *bluffing*. However, the way you use that information is also crucial, that's why this player is still not that great - it will always raise when the simulations show that it's winning.

4.3. **Agent 1.**

I am going to refer to the *Agent 1* without opponent modelling as BaseAgent1 player. We can clearly see an improvement in the performance of Agent 1 (Table 1), using the opponent modelling part compared to when we don't use it. Although I expected a higher gain in winnings, we should not forget that we are limited by how good and exploitable the expert systems behind Agent 1 are. After 250 games against *HeuristicMC*, we got **85.71%** test set accuracy for predicting moves which provided a **2%** increase in performance during this length of play.

Results after 250 games Texas Hold'em Poker per player-match, statistical error +/- 2			
Player 1	**Player 2**	Avg. Stack (Player 1)	**Result (winrate P1)**
HeuristicMCplayer	**Randomplayer**	107	52%
BaseAgent1player	**Randomplayer**	141	72%
BaseAgent1player	**Callplayer**	163	81.6%
Agent 1	**Callplayer**	167	84.8%
BaseAgent1player	**HeuristicMCplayer**	111	57.6%
Agent 1	**HeuristicMCplayer**	119	59.2%

Table 1 Results of some experiments with artificial players

Experiment with a human player

I've invited a friend, Catalin, to take on this first agent. The test subject has an **advanced beginner** to **low intermediate level** at Poker, he knows the rules of the game very well and can make educated decisions during most of the situations, but lacks the experience of more advanced players. Catalin accepted to play a total of **29** games against **Agent 1** in which he adopted an anti-computer strategy, constantly changing his style of play and testing for bluffs.

With all of that said, **Agent 1** managed to beat him both in the first 22 games where an *opponent model* wasn't available and in the next 7 games at full power. Even on such a small sample size of games, the neural network signaled a **60%** accuracy in predicting the opponent's next move.

Player 1	Score	Player 2	No games	Win-rate	
Base Agent 1	+13 -9	**Catalin**	22	59.09%	
Agent 1	+6 -1	**Catalin**	7	85.71%	
Agent 1 total	+19 -10	**Catalin**	29	65.51%	Total

Table 2 Agent 1 and BaseAgent1 performance against a human (low intermediate) player

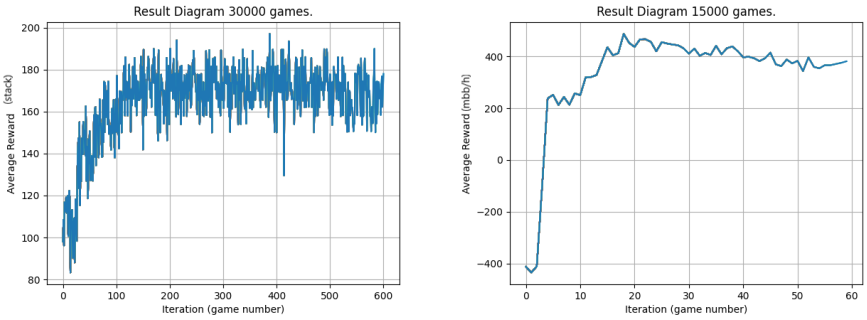
The majority of losses came from *all-ins* in the preflop stage of the game, but as the model learnt more about Catalin's playing style, it became more resilient in calling bluffs and started to aim for a turn-river finish. The mbb/h winnings were over **150 mbb/h** for the artificial player. One other thing that Catalin told us is that he became very surprised of the playing style in the last 7 games, during which the agent tried to exploit him.

4.4. **Agent 2.**

We are ready to apply the algorithm in no-limit Texas Hold'em Poker, by considering the opponent as part of the environment and trying to consistently beat him. η and ε were both set to 0.1, max length of M_{RL} to 300k and for M_{SL} to 1.2 m, the *learning rate* fir reinforcement learning and supervised learning were set to 0.05, 0.005, respectively. The exploration rate ε decays to 0 proportionally to the inverse square root of the number of games in the

training process. The agent performs 2 stochastic gradient updates of mini-batch size 256 per network for every game. The *target network-ul* parameters were reset once every 128 hands of play.
The training process was performed several times from scratch to confirm that the results are indeed consistent. The ultimate goal of this agent is to beat Agent 1 in at least 250 games match, for this we first trained an agent to beat *Random_*player (to first get a small sense of how Poker works) and then, we saved that version to next train with BaseAgent1_player.
Finally, this version of Agent 2 will be the starting point to train against Agent 1.

Below *(figure 6)* we can see the performance of algorithm 2 training against the Random_player, it quickly crushes him. In the testing phase afterwards, the average strategy of this agent recorded an average stack of **173.02** (84.8%-win rate), while the combined strategies approach recorded a close **166.23**.



(a) Agent 2 training process. Each stat point = the aggregate performance (stack) in last 50 episodes

(b) Agent 2 training process. Each stat point = the aggregate performance (mbb/h) in last 250

FIGURE 6. Measuring Agent 2 training performance in stack and mbb metrics. The axis are different for each one.

The greedy strategy (Q-network strength) is a little lower at **137.82**. It is nice to see that already **Agent 2** became more successful in defeating *Random_player* than **Agent 1** ever was. Below (figure 7), we can clearly see how the agent won, by analyzing his play style.

It seems that in general, the agent is aggressive, always trying to increase the pot and earn more. This is indeed the **right strategy** against a player who does not rely on any relevant game information. But obviously, *call* or *fold* decisions must be made at least occasionally, when the game hand / current

situation of the board is unlucky for us (in order to stop the opponent from winning through luck).

Going after Agent 1 now *(figure 8)*, we can see that in the first 15k games, the performance is pretty much similar to BaseAgent1player, but after another 15k games of training Agent 2 completely outshines him. He wins, apparently by finding a way to exploit the expert systems that both BaseAgent1 and Agent 1 are based on. This effect seems more severe when training against Agent 1 where the winnings cross over 600 mbb/h. We can deduct that from this agent playing style, using only raise average, calls and folds. On the other hand, the version of the Agent 2 that won against BaseAgent1 has a more balanced playing style and it is more destined to do well against human players. The strategies do not converge to the same locally optimal one, which means there is still space for improvement by increasing the number of iterations. However, due to the nature of the study and limited resources, the current results are good enough to call a victory for reinforcement learning.

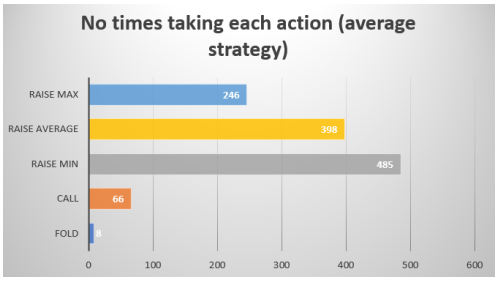
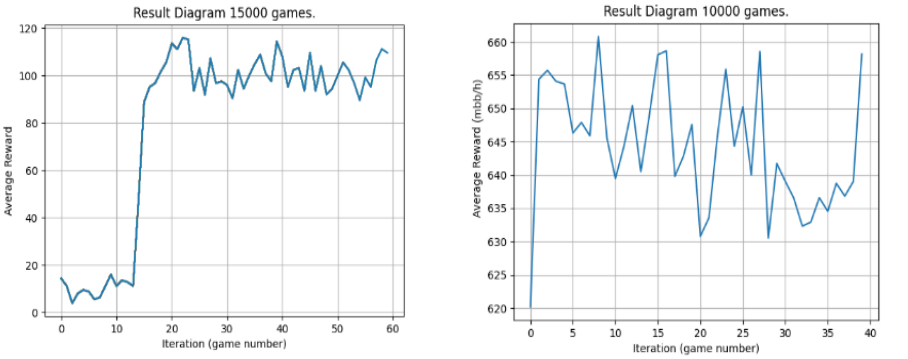


FIGURE 7. Play style in 250 games vs Randomplayer



(a) Training vs BaseAgent1, first 15k games (b) Training vs Agent 1 after 10k games

FIGURE 8. Training evolution against the two versions of Agent 1. Note how the scale for (b) is so much higher.

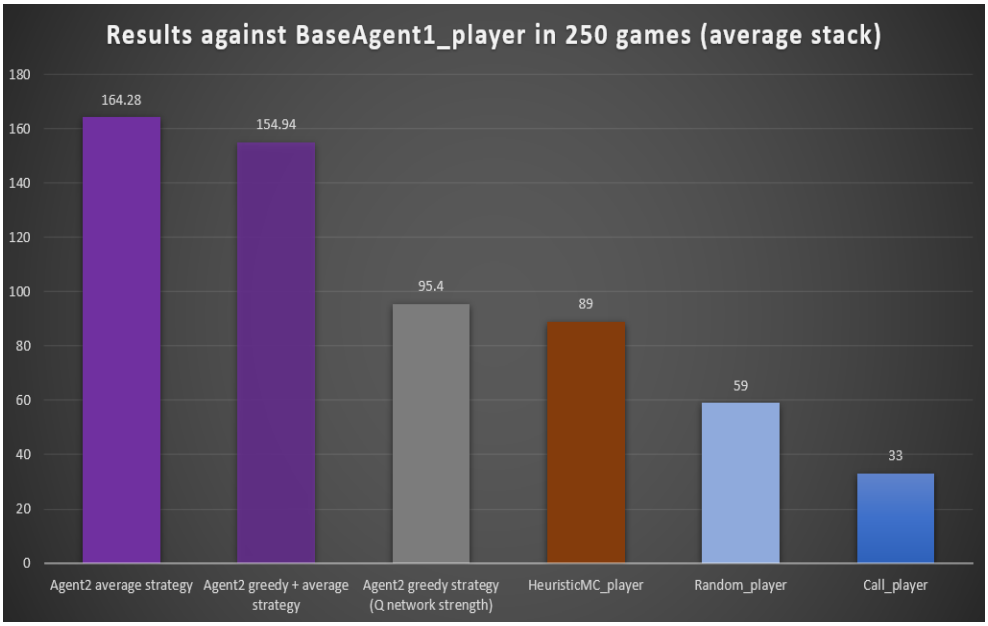


FIGURE 9. Results of some previous players against BaseAgent1player compared with Agent 2; statistical error +/-10.

It is interesting to visualize the overall performance of all the agents against one of the best build until now *(figure 9)*. The version used in that chart was the one trained for 30k games against Agent 1.

As we can see, the one agent developed through reinforcement learning completely outperforms the other ones in head-up no-limit Texas Hold'em Poker.

Experiment with a human player

I've invited another friend who has a much higher level than Catalin at Poker. The test subject has an **amateur** level of play, advanced intermediate to advanced. He is experienced, but lacks the real money high stakes experience of play that professional players possess. Expectations are not high, because in the end, our artificial player knows the game of poker only by training with other artificial players. Do note that I am using the version of Agent 2 whose performance can be visualized in figure 9.

In **10** games arranged for this match, Agent 2 won **7** and lost **3**, with an estimated winnings of **120** mbb/h. First of all, analyzing the games, Agent 2 really taught himself the basic, trivial strategies of the game:

- never fold if the opponent does not raise after preflop.
- mostly raise a good hand but also bluff from time to time if you have good potential for the next streets.
- usually all-in when the hand is very good and the pot is significant.
- never fold in the next round after a big raise of your own, if the opponent does not put pressure.

Secondly, I noticed a very aggressive tendency in *preflop*, which is also present in other artificial players such as **Cepheus** [3], but what is even more interesting, although the *AI* prefers to *raise* in this part of the game, in most cases he does not accept this exact behavior from his opponent, folding to a raise greater than 20-30 in preflop (but not all the time).

Third, it's pretty tricky to accurately report whether the AI really does intentionally bluff or not, but from what I've noticed, even when going all-in on the flop or turn, it always has at least one pair, probably fail-safe. There were, however, a few isolated cases, when he put a lot of pressure but had nothing in his hand, probably estimating that the opponent, most likely, has nothing.

This agent can actually play a **multi-player Poker game**, although not as well as in heads-up, by making a small change in our inputs when we use the predict function to get a move. The only input components that we use, relevant to a multi-player game, is the average estimated opponent strength, which can be recomputed with respect to the number of players through Monte-Carlo simulations and the opponent's stack which can be substituted with the average stack of all the opponents.

Note that for these experiments, I used a *NVIDIA Tesla T4 Workstation* with *32*GB of RAM and a *NVIDIA GTX 1050ti w*ith *16*GB of RAM, but the resulting artificial players can be run on a less impressive machine even without a GPU, with 8GB of RAM.

5. Conclusion and Further Research

I have successfully showed the power and utility of deep reinforcement learning in imperfect information games, compared to other methods. When applied to no-limit hold'em Poker, deep reinforcement learning agents clearly outperform agents with a more traditional approach.

The human experiments, although successful, were conducted on a really small scale, where statistical error or selection bias may have played a role in the outcome. However, in future work, we can switch our current aim (that being to observe, intuitively, how our agents might fair against a human opponent) to an extensive testing against more professional individuals and over

a very high number of games. Experiments against state-of-the-art artificial poker players would also represent something to be considered in the future.

Further research on this matter may consists in developing a Poker agent trained completely through self-play. It would be interesting to see how an artificial player that learns only by playing with a decent opponent to get better at a game stands against a player trained by playing only against itself.

Acknowledgment

I want to thank Professor Laura Diosan and Lecturer Gabriel Mircea (Babes-Bolyai University) for supervising this project. As an inexperienced undergraduate, I received massive advice from both of them to complete this research project that is strongly tied to my 70-page thesis (*Deep Reinforcement Learning in Imperfect Information Games: Texas Hold'em Poker* [16]) which contains many more details about the implementation and methods used for each Poker agent.

References

[1] Arulkumaran, K., Cully, A., and Togelius, J. Alphastar: An evolutionary computation perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2019), pp. 314–315.

[2] Billings, D., Papp, D., Schaeffer, J., and Szafron, D. Opponent modeling in poker. *Aaai/iaai 493* (1998), 499.

[3] Bowling, M., Burch, N., Johanson, M., and Tammelin, O. Heads-up limit hold'em poker is solved. *Science 347*, 6218 (2015), 145–149.

[4] Brown, G. W. Iterative solution of games by fictitious play. *Activity analysis of production and allocation 13*, 1 (1951), 374–376.

[5] Brown, N., and Sandholm, T. Superhuman ai for multiplayer poker. *Science 365*, 6456 (2019), 885–890.

[6] Heinrich, J., Lanctot, M., and Silver, D. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning* (2015), pp. 805–813.

[7] Heinrich, J., and Silver, D. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121* (2016).

[8] Lambert Iii, T. J., Epelman, M. A., and Smith, R. L. A fictitious play approach to large-scale optimization. *Operations Research 53*, 3 (2005), 477–489.

[9] Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science 356*, 6337 (2017), 508–513.

[10] Nevmyvaka, Y., Feng, Y., and Kearns, M. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning* (2006), pp. 673–680.

[11] Sewak, M. Deep q network (dqn), double dqn and dueling dqn. In *Deep Reinforcement Learning*. Springer, 2019, pp. 95–108.

[12] Shamma, J. S., and Arslan, G. Dynamic fictitious play, dynamic gradient play, and distributed convergence to nash equilibria. *IEEE Transactions on Automatic Control 50*, 3 (2005), 312–327.

[13] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature 550*, 7676 (2017), 354–359.

[14] Stipić, A., Bronzin, T., Prole, B., and Pap, K. Deep learning advancements: closing the gap. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (2019), IEEE, pp. 1087–1092.

[15] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] Tidor-Vlad, P. Deep reinforcement learning in imperfect information games: Texas hold'em poker, 7 2020. Bachelor's Thesis.

[17] Urieli, D., and Stone, P. Tactex'13: a champion adaptive power trading agent. In *Twenty-Eighth AAAI Conference on Artificial Intelligence* (2014).

[18] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence* (2016).

[19] Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS) 11*, 1 (1985), 37–57.

[20] Watkins, C. J., and Dayan, P. Q-learning. *Machine learning 8*, 3-4 (1992), 279–292.

[21] Zhang, L., Wang, W., Li, S., and Pan, G. Monte carlo neural fictitious self-play: Approach to approximate nash equilibrium of imperfect-information games. *arXiv preprint arXiv:1903.09569* (2019).

University of Edinburgh, School of Informatics, Informatics Forum, 10 Crichton Street, Edinburgh, UK, EH8 9AB$^{\smile}$

Email address: T.V.Pricope@sms.ed.ac.uk

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 2, 2020
DOI: 10.24193/subbi.2020.2.04

OVERVIEW OF RECENT DEEP LEARNING METHODS APPLIED IN FRUIT COUNTING FOR YIELD ESTIMATION

H. B. MUREŞAN, A. D. CĂLIN, AND A. M. COROIU

Abstract. This paper is an overview of the latest advancements of image recognition for fruit counting and yield estimation. Considering this domain is developing rapidly, we have considered the cutting-edge literature in the field, for the last 5 years, focused on the task of yield estimation by detecting and counting fruit in the tree canopy. This is a much more complex task than the classification of fruit post-harvesting, which has been more widely reviewed. Moreover, we identify the major challenges and propose the next steps for advancing this research field.

1. Introduction

This paper presents state of the art models and methods based on artificial intelligence for detecting fruits in orchards and on plantations. A system that can accurately and automatically detect and count fruit before harvest gives agricultural enterprises the ability to optimize and streamline their harvest process. Through a better understanding of the variability of yield across their farmlands, growers can make more informed and cost-effective decisions for labor allotment, storage, packaging, and transportation. While this process is performed manually, it involves a very high labour cost, which can be reduced using automated fruit counting computer vision systems.

Therefore, we analysed several papers tackling this issue using deep learning techniques. We selected papers of the latest 5 years of research in the field of fruit counting in tree canopies for yield estimation. We have searched for precision and digital agriculture publications using ACM digital library,

Received by the editors: 10 November 2020.

2010 *Mathematics Subject Classification.* 68T45.

1998 *CR Categories and Descriptors.* I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis – *Object recognition*; I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding – Intensity, color, photometry, and thresholding*.

Key words and phrases. smart-agriculture, deep learning, yield estimation, transfer learning, intersection over union, F1-score.

Science direct, IEEE and Google Scholar platforms, and used keywords such as "fruit counting", "deep learning", and "yield estimation".

(A) (B) (C)

(A) (B) (C)

FIGURE 1. Grapes, apricots and tomatoes in different lighting conditions, backgrounds and occlusions

We have identified review papers focused on agri-tech that present a very broad overview of applications of deep learning in various fields of agriculture [4]. Other papers present the methods overview for the specific domain of fruit or image classification [6]. However, most papers deal with post-harvesting classification of fruits for packaging or similar purposes. In this review, we aim to narrow the focus specifically on the task of pre-harvest yield estimation of fruit in an orchard, which is a problem of localising and counting fruit in the tree canopy, with different backgrounds, occlusions and lighting conditions. This enables farmers to estimate their yield and plan resources required for harvesting, storage, processing/packaging accordingly. Furthermore, with accurate computer vision technology, the harvesting could be performed by robots, with increased efficiency.

The main goals of the reviewed papers presented here are:

- accurately predicting the number of fruits in an image under various illumination conditions and different levels of fruit occlusion in the

tree canopy (this is to optimise and streamline the harvesting process and the fruit distribution for commercialisation or processing)
- reducing the labor costs required to perform yield estimation or harvest (representing 50-70% of the total costs [22])
- correctly estimating the size of the fruits from images
- adapting existing classification techniques for automatic robot harvesting or low-power devices (mobile phone).

The paper is structured as follows: firstly, we will present the methods identified in the papers, then we will discuss the datasets particularities used for training and testing performance. Next, we present results and the conclusions, as well as the challenges that remain in this research area and possible steps that can be taken to address them.

2. State of the art

2.1. **Methods.** The main methods we have analysed can be split into three main categories:

- **Deep learning with simple pre-processing** [13] - These methods involve the use of convolutional neural networks (CNNs) for the task of object detection (fruits in this case). They apply basic image pre-processing, such as rotations, vertical/horizontal flips, random zoom level, image cropping and colorspace conversion to augment the training dataset. Typically such methods require a large number of images to train the model.
- **Deep learning with complex pre-processing** [11, 2, 1, 5] - In addition to the previous category, complex image pre-processing, involving filtering features (such as background), colorspace changing and colorband isolation, or applying other intelligent algorithms for feature selection, is applied to enhance the training dataset before running a CNN on it.
- **Transfer learning** [21, 3, 23] - This method uses existing trained models and replaces the top layers to retrain them on a particular dataset. As opposed to previous methods, it can yield good results with reduced input data.

A novel approach for counting the number of tomato fruits is presented by Rahnemoonfar [13]. The authors proposed an approach based on a convolutional neural network (CNN) for object counting (different scales and occlusions) rather than area calculation (average pixel coverage), incorporating a modified Inception-ResNet-A module and a scaling module (17×17 to 8×8). The latter involves calculating the total pixel coverage of the target fruit and

then divide it to an average pixel coverage of a single fruit. The former typically requires a detection step, but it is not influenced by image scale and occlusion. Furthermore, as the proposed goal was only to provide a fruit count per image, rather than the actual location of fruits, the bounding box proposal was skipped, decreasing the inference time. The proposed model was a CNN which incorporated modified Inception-ResNet-A modules [18] and a 17×17 to 8×8 reduction module. The authors also use convolutional layers with large kernels towards the top of the network, to extract large scale features. The output of the network is given by a fully connected layer with a single output, the number of predicted fruits.

The model presented was compared with an area-based method, in terms of accuracy and speed. The accuracy for predicting the number of fruits in an image was defined as follows:

$$\left(1 - \frac{|predicted\_count - actual\_count|}{actual\_count}\right) \times 100 \tag{1}$$

In paper Mao [11], the authors presented a novel approach for detecting cucumbers. On top of the common difficulties of object detection, cucumbers easily blend in with the background due to their green color, further complicating the task. In order to compensate for this, the authors devised a four-component system that aimed to improve the accuracy of the detection, containing:

(1) ***Color component selection:*** From a total of 15 color components from 5 color spaces (RGB, HSI, YCbCr, Lab, YIQ), this extracts the top 3 that make it the easiest to differentiate the cucumber from the background. For this, the I-RELIEF [17] algorithm was used, which calculates weights for given features. The most relevant color bands selected were red and green from RGB color space and the intensity from the HSI color-space.

(2) ***Background pre-processing:*** The green component (from the RGB representation) was smoothed using a 3×3 median filter. Afterwards, the OTSU [20] method was applied to obtain a filtered background, and, finally, the Maximally Stable Extremal Regions [12] were used to eliminate the leaves from the background.

(3) ***Deep learning-based feature extraction:*** The authors segmented the original input image into small areas, applied pixel interpolation, and used LeNet5 [9] model, which support input data size 32×32. Each color component was passed through a separate instance of LeNet5 and fused, creating a multipath CNN. For this step, the authors elected to segment the original input image into small areas

and apply pixel interpolation on the image. The result of the interpolation is a collection of 32×32 areas. As noted in the paper, using the VGG [16] or AlexNet [7] models would require resizing the areas with a factor of 20, which could produce image distortion. Thus, the LeNet5 [9] model was used, which has a 32×32 input size, so no resizing is necessary. To fully make use of the selected color components, each one was passed through a separate instance of LeNet5, fusing the output of the last layers of the models, creating a multipath convolutional neural network (MPCNN).

(4) ***Cucumber region detection:*** The feature maps produced by the convolutional neural networks were merged and a Principal Component Analysis [8] algorithm was applied to reduce their dimensionality. The classification was done by a Support Vector Machine approach.

A fruit yield estimation pipeline that can map fruit counts from an input image is created in Chen [2]. The pipeline includes:

(1) ***Data labeling:*** makes use of a crowd-sourcing web platform for data labeling. Images for labeling are subdivided into several windows, each window annotated by 3 different users.

(2) ***Blob extraction:*** trains a fully convolutional network to extract candidate regions (blobs). Input is an image $h \times w \times 3$ and the output a score tensor $h \times w \times n$ where n is the object class (the probability that the pixel may contain a fruit or not, using a softmax function).

(3) ***Fruit counting:*** uses a second CNN algorithm trained for counting fruit in each region. For each blob, the output is a number representing the fruit count. The fine-tuning process involves running the blob detection network on the training images to obtain segmented images and bounding boxes, which are resized to 128×128. Next, ground truth counts are associated with the count network.

(4) ***Count mapping:*** maps a linear regression model between fruit count estimates and final fruit count. This trains a linear regression to intermediate count estimates with human-generated labels as ground truth, minimising the loss function between the count network and the blob network.

This pipeline is evaluated using two datasets (oranges in day and apples at night) and human-generated count and labeling for ground truth. For each image x_i, we have the actual number of fruit z_i and the human ground truth $\widetilde{z_i}$. If $f(x_i)$ is the algorithm generated count, the problem is to minimise the l^2 error:

$$l^2 = \sqrt{\sum_{i=1}^{n} (f(x_i) - \widetilde{z_i})^2} \tag{2}$$

Paper Bargoti [1] is focused on developing an image processing framework for fruit detection and counting using orchard image data. They use a general-purpose image segmentation approach, including two feature learning algorithms: multiscale multilayered perceptrons (MLP) and convolutional neural networks (CNN). These networks were extended by including metadata which correlates with appearance variations and/or class distributions. Further, the authors utilised watershed segmentation (WS) and circular Hough transform (CHT) algorithms to process image pixels, and then detect and count fruits. Finally, the counts from each row in the orchard were summed up and compared with the total post-harvest counts (done by a grading and counting machine).

In paper Kang [5], authors developed a real-time apple detector based on the LedNet architecture. The presented model uses the Feature Pyramid Network (FPN) [10] and Atrous Spatial Pyramid Pooling (ASPP) algorithms. The one-stage model was chosen by the authors as it offers the same, or superior performance to two-stage detectors, but with fewer network parameters. The FPN used in LedNet fuses feature maps at three levels of downsampling (1/8, 1/16, 1/32) to increase the model's capability of detecting objects at various ranges. The ASPP technique was employed to process multi-scale features. The custom ResNet backbone was a light-weight version of a typical ResNet architecture to reduce the inference time on an embedded system, such as an autonomous robot.

One study, Xiang [21], presents fruit image classification using a lightweight neural network MobileNetV2 [15] (pretrained using ImageNet dataset, for feature extraction). Here, the top layer was replaced with a conventional convolution layer (conv2d) and a Softmax classifier (for feature classification into 5 classes of fruits) [21]. They also applied dropout to the new-added conv2d at the same time to reduce overfitting. The new model was trained and fine-tuned in two stages, using Adam optimizer of different learning rate, and batch size of 64. TensorFlow 1.14 stable was used for performance evaluation. Compared to others, this method can be deployed in low-power and limited-computing devices such as mobile phones.

Another study uses machine vision to accurately identify and localise grapes and apples, Fourie [3]. With the advantage of less time need for training and good performance with limited training data, transfer learning was used, based on deep convolutional neural network (DCNN). The authors pre-trained the

InceptionV3 model [19] on the ImageNet database, as a generic image feature extractor. Next, their classifiers were added to separate fruit and background features. Further, a final layer was replaced with one trained on a custom dataset of apple trees and vines, acting as a classification head, specialising the network with custom images training separately. For the last step of localising and counting fruit new layers were added. The output of the last convolution and the remaining spatial correlated outputs are pooled into a single high-dimensional feature vector, linked to the classification head. The localizer outputs a grid of confidence scores that indicate the fruit localisation in the image.

A more advanced study, focuses on detecting six different types of fruits: lime, lemon, apple, mandarin, tomato and orange in orchard settings, Yu [23]. The algorithms used are color based - Faster R-CNN (Convolutional Neural Networks, two stage region-based model) and SSD (Single Shot Detector, which is a region free method) applying transfer learning for fruit detection and counting.

2.2. **Datasets.** The studies we present make use of public datasets with fruits, where available (for example, ImageNet, COCO or dataset in [14]). Others have either collected their own datasets of specific images in orchards under various lighting conditions or used unspecific images from the web, retrieved with a web crawler.

In Rahnemoonfar [13], the lack of available public datasets with annotated images of tomatoes was handled using an interesting approach. They created their own, consisting of fully synthetic images. Their images were created as follows: firstly, they added green and brown circles for background, then applied a Gaussian filter to blur them, and finally, added red circles to simulate the tomato fruits. The authors also took into consideration variations in fruit size, scale, illumination and overlap, generating 24000 images for training and 2400 for validation, and using 100 real images for testing.

For study Mao [11], 225 images were collected from a cucumber planting base, in Shouguang, China, Shandong. The images were taken between 7 am to 10 am and from 3 pm to 6 pm to reduce the impact of illumination conditions. The images were resized from 4032×3016 to 1024×768.

The experiments in Chen [2] used two datasets that differ from the perspective of lighting conditions, occlusion levels, resolution and camera type. The first dataset contains orange images of size 1280×960 and was collected during the day, using a steady camera carried by a human operator at walking speed. The orange trees were in a nontrellis arrangement. There were 5000 images, labeled by 22 users. The second one, an apple dataset, collected at

night using an external flash, with images of size 1920 × 1200, from a utility vehicle at the speed of $1m/s$, with trees in a trellis arrangement.

In Bargoti [1], the dataset was collected in a commercial orchard of Kanzi and Pink Lady apple varieties, over a 0.5ha block of v-trellis arrangement of 17 rows, using a teleoperated vehicle, in daylight. The set contains more than 8,000 images of size 1232 ×1616. For experiments, random sub-sampling was used, dividing each image into 32 parts of size 308 × 202, manually annotated to binary fruit and non-fruit classes.

In Kang [5], 800 apple images were collected from an orchard in Qingdao, China, using a Kinect-v2, from a distance between 0.5 - 1.5 meters. An additional set of 300 images of apples in different scenes were collected to diversify the dataset. Due to the distance at which the images were taken, the apples would be represented largely in the small scale features. To avoid this imbalance, the authors applied a crop-and-resize algorithm. The labelling process was done with the help of a clustering region-based neural network. The model would extract multi-scale features, proposing potential regions of interest (ROI). The pixels of the ROIs were segmented using pixel-connection into independent candidate patches and bounding box coordinates were assigned to it. With the help of this model, the labelling of training data was done in only two days.

In Xiang [21] the ImageNet dataset was used, a large dataset of 1.4M images [14], and 1000 classes of web images having complex backgrounds: 3,670 images of 5 fruits collected from the Internet, including apples (633), bananas (898), carambola (641), guava (699) and kiwi (799). For the experiment, these were split into two subsets, 3,213 images for training and 457 images for validation. Images were adjusted to size 224×224 as required by the MobileNetV2 model.

In Fourie [3], authors used the ImageNet dataset, and collected their own datasets from an apple orchard and a vineyard. The apple set contaied 21 images (2000 × 3000 px), with various light conditions and view angles. Images were normalised to the same mean RGB intensity. 442 areas of interest were extracted for training and augmented through random transformations. 20% of the data was used for testing. The vineyard set was also split into a training set (95 images), and a testing set (52 images). Testing images were captured under different light conditions than those in the training set.

In Yu [23] authors used a Python Web Crawler to create a 2000 dataset of images. They augmented the set by rotations and RGB adjustments with different brightness and saturation, obtaining 2995 images (tomato 124, mandarin 301, orange 377, apple 680, lemon 605, lime 909).

2.3. **Results.** The models are often evaluated using testing data, measuring especially accuracy and processing time, but also loss, F1 score, Recall, True Positives, False Positives, and other specific measures defined by the authors.

As seen in Table 1, the proposed method by Rahnemoonfar [13] is significantly better in terms of accuracy than an area-based counting method. From a processing time perspective, the proposed method and the area-based are both much faster than manual counting.

Method	Avg accuracy	Stdev	Avg time/image
CNN based	91.03%	2.5	0.006
Area based	66.16%	7.9	0.05
Manual count	-	-	6.5

TABLE 1. Average accuracy and time over 100 test images of the methods studied in Rahnemoonfar [13]

The proposed method in Mao [11] was compared with one that uses an MPCNN with the red, green, blue channels and another that uses a single CNN for an RGB image. The best results were obtained by the model that was using the color bands selected by the color selection component (red, green, intensity), presented in Table 2. The metrics depicted are:

- correct recognition rate (CRR) - ratio between the number of true positive(TP) pixels and the total number of pixels in an image
- false recognition rate (FRR) - ratio between the number of false positive(FP) pixels and the total number of pixels in an image
- correct tot false ratio (CFR) - ratio between the CRR and FRR

Another observation was that the multi-path convolutional neural network performed strictly better than the regular convolutional network. This showed that applying late fusion instead of early fusion on multiple color components yields better results.

The results of Chen [2] in terms of reducing the l^2 error were obtained for the combination of blob + count + regression, values obtained are shown in

Methods	TP	FP	TP + FP	CRR	FRR	CFR
RGB + CNN + softmax	76,954	25,431	102,385	92.77%	24.84%	3.73
RGB + CNN + SVM	77,436	17,627	95,063	93.36%	18.54%	5.04
RGB + MPCNN + SVM	78,688	15,884	94,572	94.87%	16.80%	5.65
RGI + MPCNN + SVM	80,670	10,571	91,241	97.25%	11.59%	8.39

TABLE 2. The performance of the methods proposed in paper Mao [11]

Table 3. To evaluate pixel-wise accuracy, Intersection over Union and ROC curves were used. There were in total 7200 oranges in 71 images and 1749 apples in 21 images in the testing sets.

Model	l^2 error	Ratio Counted	Std Dev
Orange blob	16.9	0.935	15.6
Orange blob+regression	15.9	0.999	15.9
Orange blob+count	19.2	0.851	12.7
Orange blob+count+regression	**13.8**	**0.968**	13.5
Apple blob	46.5	1.475	24.9
Apple blob+regression	20.4	1.025	20.3
Apple blob+count	20.9	0.767	8.4
Apple blob+count+regression	**10.5**	**0.913**	7.7

TABLE 3. Count accuracy of the CNN proposed in Chen [2] for orange and apple set.

The metrics used for evaluating the proposed model in paper Kang [5] were the inference time, the number of parameters, F1-score, precision, recall, intersection over union (IoU), area under the curve - which was denoted as AP_m (where m is the threshold for IoU used to accept or reject proposed regions of interest). A comparison between the crop-and-resize augmentation process with the regular data augmentation method is presented in Table 4. As anticipated, the model performs poorly on medium and large fruits when augmenting images with rotates and color/brightness alteration due to the size imbalance. It can be noted that the model trained on data augmented with the crop-and-resize operation performed well regardless of the object size. Several popular architectures were compared with the proposed model (Table 5). The LedNet with the light-weight backbone performed just as well as the other much larger networks, while having the fastest inference time.

The results in the MLP network in Bargoti [1] improved after including the metadata, which can be observed in Table 6. Extending this with CNNs, the best pixel-wise F1-score of 0.791 was achieved, while the WS produced

Method	AP_{50}	AP_{small}	AP_{med}	AP_{large}	IoU
Crop and Resize	0.826	0.832	0.817	0.763	86.7%
Standard	0.797	0.818	0.778	0.652	78.3%

TABLE 4. The impact of the two augmentation methods utilised in Kang [5] on the performance of the LedNet model.

Method	AP_{50}	F1	Recall	Acc	IoU	Time	Params
LedNet(LW-Net)	0.826	0.834	0.821	0.853	86.3%	28 ms	7.4 M
LedNet (ResNet-101)	0.843	0.849	0.841	0.864	87.2%	46 ms	188 M
YOLOv3	0.803	0.803	0.801	0.82	84.2%	45 ms	248 M
YOLOv3 (Tiny)	0.782	0.783	0.776	0.796	82.4%	30 ms	35.4 M
Faster-RCNN (VGG)	0.814	0.818	0.814	0.835	86.3%	145 ms	533 M

TABLE 5. Evaluation the 5 different backbones used for the detector in Kang [5] .

the best results, with a detection F1-score of 0.861. Comparing the count estimates using CNN and WS with the base counting the squared correlation coefficient obtained $r^2 = 0.826$.

	Both	ms-MLP	CNN	Neither
ms-MLP	0.834	0.860	0.739	0.709
CNN	0.921	0.843	0.849	0.731

TABLE 6. Comparing ms-MLP and CNN approaches for fruit detection with image segmentation output and WS detection algorithm in Bargoti [1]

In Xiang [21], the classification accuracy obtained for the 5 fruits was 85.12%. To demonstrate its effectiveness on source-limited platforms, the models were deployed also on an Android smartphone (Honor 10 by Huawei). Through transfer learning, the new model was able to accelerate and optimise the learning process (MobileNetV2 as the best running time, as described in Table 7).

Apple classification accuracy was 98% (deciding if an area of interest contained an apple or background) in Fourie [3]. For the vine set, the network could correctly classify 99% of the testing areas of interest if they contain grape bunches. Next step would be to correlate counting and yield estimation.

Model	Training		Validation		Run
	Loss	Acc	Loss	Acc	(sec)
MobileNetV2	0.0109	0.9984	0.4719	0.8512	327
MobileNetV1	0.0335	0.9960	1.3527	0.7352	1618
InceptionV3	0.0071	0.9994	0.6322	0.8578	670
DenseNet121	0.0036	0.9994	0.3695	0.8906	7965

TABLE 7. Loss and Accuracy on the training and validation sets for the proposed models in Xiang [21].

The results for Yu [23] show that the accuracy of the model trained by Faster R-CNN was higher (at 89%) that for the model trained by SSD (at 82%). The average speed per image in seconds 8.21 (Faster-RCNN) and 6.70 (SSD) respectively (see Table 8).

Fruit	Orange	Mandarin	Lemon	Apple	Tomato
SSD	0.86	0.85	0.81	0.81	0.77
Faster R-CNN	0.91	0.90	0.89	0.89	0.87

TABLE 8. Accuracy comparison between the Faster R-CNN and SSD models on 5 different fruit classes, as described by Yu [23]

3. Discussion

3.1. **Results analysis.** Figure 2 presents the accuracy values of each reviewed model against the size of the dataset and number of classes. All but one of the studied papers (Bargoti [1], Mao [11], Rahnemoonfar [13], Kang [5], Fourie [3], Chen [2] and Yu [23]) implement deep learning detectors. These networks have the advantage of providing both class prediction as well as coordinates to locate the object in the image, making them better for fruit counting and yield estimation. The downside is that they also must be trained using data containing the same information, data which is scarce, they are more complex than classifiers and thus require more training resources.

The model in Xiang [21] is a classifier, which is simpler to train and deploy, compared to a detector, however it provides only class predictions. Despite this, the model was outclassed by all other works. The reason behind this is very likely to be the dataset obtained by scraping images from the Internet, which would contain a high degree of variance and potentially too few samples per variant.

The average accuracy across these works is 91.98%. The datasets of images used by authors range in size (from 168 to 24000), depending on the method used, and high accuracy has been obtained for low and high number of testing images, in association with the proper model.

One observation is that models that were trained to classify or detect 5 or more classes of fruits have not achieved an accuracy over 90% (89% and 85.10%, respectively).

3.2. **Challenges imposed by datasets.** Although popular datasets (ImageNet, COCO) are often used for transfer learning, they do not contain real field image data of occluded fruits and various lighting conditions throughout

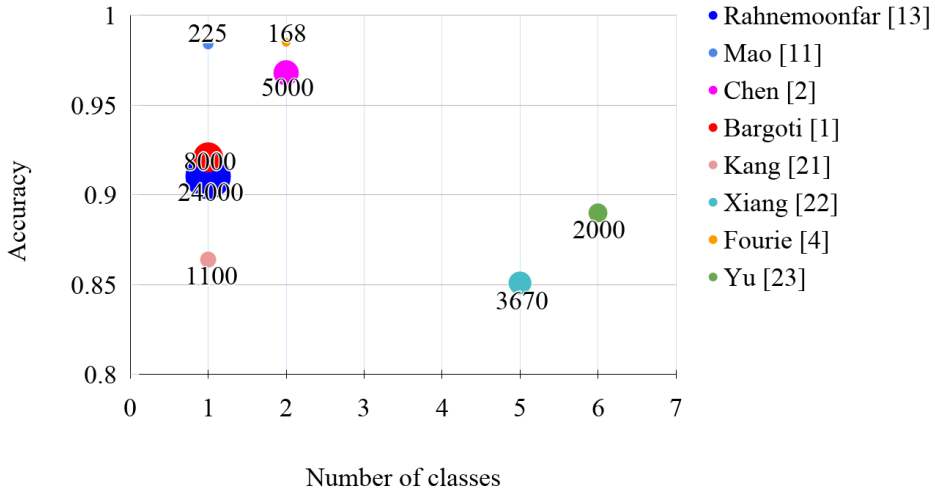


FIGURE 2. Accuracy in relation with the Number of Classes and Dataset Size: Chart that helps visualize the best performing models from each of the studied papers. We took into account the accuracy, number of classes and images that the model was trained on.

the day. As such, a big disadvantage is that every research group focused on fruit detection needs to create their own dataset to suit their needs. Creating such a dataset is both a difficult and time cosuming process for the following reasons:

- The collected images must contain samples in each lighting conditions under which the model is expected to perform (e.g. sunny, rain, cloudy, early morning, late evening)
- The images should capture as many variations of the targeted fruit classes as possible (each fruit class contains different amounts of variations in shape, size and colour, and in some cases manifest visual deformations or defects)
- The background of the images must be relevant to the desired application (e.g. if the model should detect ripe fruits on trees in orchards, the background should include tree branches and leaves)
- The dataset must be annotated with the class name for classification, to which bounding box annotations must be added for detection, an overall time consuming process if done manually

A publicly available dataset with such images would be advantageous to achieve shorter research times. Alternatively, it has been shown that synthetic images do not degrade the performance of trained models and are much easier to create than real images. Perhaps a combination of a synthetic image generation algorithm together with a small dataset of real images for fine-tuning can serve as a starting point.

Another subject of research is the impact of illumination conditions on images due to the position on the globe. Specifically, if the images are taken in an area close to the Ecuador during the daytime, they will be differently lit than images taken during the daytime in areas closer to the poles. Thus, the goal is to investigate whether a model trained on images from one of these areas performs equally well on images from the other area.

3.3. **Model optimisation.** One more possible direction is increasing of the accuracy of models with a new approach based on the collected methods already existed in literature. It was proven that convolutional neural networks achieve better performance than the alternative methods in tasks of fruit detection. However, there are tasks that are still challenging for this class of algorithms, detecting partially occluded fruits or correctly counting grouped fruits being among the more frequent ones.

Since the majority of reviewed papers proposed models that use images or frames extracted from videos, the area of video analysis remains largely unexplored. The LSTM architecture is well suited to process time series, and videos can be seen as a series of frames. This approach could further address the aforementioned issues as the video could cover trees/plants from multiple angles.

4. Conclusions and next steps

In this paper, we presented an overview of the latest research involving deep learning for fruit yield estimation using orchard images. Some very good results (up to 97% accuracy) were obtained using simple or complex pre-processing techniques and large data for CNN training Mao [11]. Some successful attempts have used transfer learning for limited training data, proving that there is good potential even for low-resource platforms to be used Fourie [3], Xiang [21]. Some studies focus on mapping and adjusting, based on the manual count, the algorithm generated count Bargoti [1], Rahnemoonfar [13], with specific applications in an orchard (results of up 96.8% accuracy in counting Chen [2]).

We have also highlighted the limitations of these studies and possible directions of research, derived from challenges posed by the need to use real

field data with various fruits, and the need improve the models by increasing accuracy of detection for a larger variety of fruits.

Overall, the results show a very good potential for further research and improvement up to the use in practical settings for pre-harvest yield estimation and designing harvesting robots.

This paper is a very useful initial step for a more elaborate project, the role of the current paper is to set a ground from we can develop particular approaches.

References

[1] Bargoti, S., and Underwood, J. P. Image segmentation for fruit detection and yield estimation in apple orchards. *Journal of Field Robotics 34*, 6 (2017), 1039–1060.

[2] Chen, S. W., Shivakumar, S. S., Dcunha, S., Das, J., Okon, E., Qu, C., Taylor, C. J., and Kumar, V. Counting apples and oranges with deep learning: A data-driven approach. *IEEE Robotics and Automation Letters 2*, 2 (2017), 781–788.

[3] Fourie, J., Hsaio, J., and Werner, A. Crop yield estimation using deep learning. In *7th Asian-Australasian Conference on Precision Agriculture* (2017), pp. 1–10.

[4] Kamilaris, A., and Prenafeta-Boldú, F. X. Deep learning in agriculture: A survey. *Computers and electronics in agriculture 147* (2018), 70–90.

[5] Kang, H., and Chen, C. Fast implementation of real-time fruit detection in apple orchards using deep learning. *Computers and Electronics in Agriculture 168* (2020), 105108.

[6] Koirala, A., Walsh, K. B., Wang, Z., and McCarthy, C. Deep learning–method overview and review of use for fruit detection and yield estimation. *Computers and Electronics in Agriculture 162* (2019), 219–234.

[7] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM 60*, 6 (May 2017), 84–90.

[8] Lavania, S., and Matey, P. S. Novel method for weed classification in maize field using otsu and pca implementation. In *2015 IEEE International Conference on Computational Intelligence Communication Technology* (2 2015), pp. 534–537.

[9] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (6 1998), 2278–2324.

[10] Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (7 2017), pp. 936–944.

[11] Mao, S., Li, Y., Ma, Y., Zhang, B., Zhou, J., and Wang, K. Automatic cucumber recognition algorithm for harvesting robots in the natural environment using deep learning and multi-feature fusion. *Computers and Electronics in Agriculture 170* (2020), 105254.

[12] Nistér, D., and Stewénius, H. Linear time maximally stable extremal regions. In *Computer Vision – ECCV 2008* (Berlin, Heidelberg, 2008), D. Forsyth, P. Torr, and A. Zisserman, Eds., Springer Berlin Heidelberg, pp. 183–196.

[13] Rahnemoonfar, M., and Sheppard, C. Real-time yield estimation based on deep learning. In *Autonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping II* (2017), J. A. Thomasson, M. McKee, and R. J. Moorhead, Eds., vol. 10218, International Society for Optics and Photonics, SPIE, pp. 59 – 65.

[14] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision 115*, 3 (2015), 211–252.

[15] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 4510–4520.

[16] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).

[17] Sun, Y. Iterative relief for feature weighting: Algorithms, theories, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 6 (6 2007), 1035–1051.

[18] Szegedy, C., Ioffe, S., and Vanhoucke, V. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR abs/1602.07261* (2016).

[19] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2818–2826.

[20] Wang, L., and Duan, H.-c. Application of otsu'method in multi-threshold image segmentation [j]. *Computer Engineering and Design 11* (2008), 2844–2845.

[21] Xiang, Q., Wang, X., Li, R., Zhang, G., Lai, J., and Hu, Q. Fruit image classification based on mobilenetv2 with transfer learning technique. In *Proceedings of the 3rd International Conference on Computer Science and Application Engineering* (2019), pp. 1–7.

[22] Yang, R., Wu, M., Bao, Z., and Zhang, P. Cherry recognition based on color channel transform. In *Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science* (2019), pp. 292–296.

[23] Yu, H., Song, S., Ma, S., and Sinnott, R. O. Estimating fruit crop yield through deep learning. In *Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies* (2019), pp. 145–148.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu St., 400084 Cluj-Napoca, Romania

Email address: {horea.muresan, alinacalin, adrianac}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 2, 2020
DOI: 10.24193/subbi.2020.2.05

LOG REPLICATION IN RAFT VS KAFKA

MANUELA PETRESCU AND RĂZVAN PETRESCU

Abstract. The implementation of a fault-tolerant system requires some type of consensus algorithm for correct operation. From Paxos to Viewstamped Replication and Raft multiple algorithms have been developed to handle this problem. This paper presents and compares the Raft algorithm and Apache Kafka, a distributed messaging system which, although at a higher level, implements many concepts present in Raft (strong leadership, append-only log, log compaction, etc.).

This shows that mechanisms conceived to handle one class of problems (consensus algorithms) are very useful to handle a larger category in the context of distributed systems.

1. INTRODUCTION

Due to the increased volumes and fault tolerance requested by the real-life applications, new methods emerged to implement replicated servicers and coordination client interaction with server replicas. From Paxos algorithm, to Raft and Kafka, many applications use different models in order to provide solutions for these requirements.

This paper is analyzing differences and similarities between Raft consensus algorithm and Kafka methods and consensus. Why Raft and why Kafka? Raft is a consensus algorithm that was created as an alternative to Paxos algorithm, comparable with it in terms of fault tolerance and performance, having a different structure allowing it to be easier to understand and to implement. For both of them (Kafka and Raft), logs play an essential role.

Kafka is a distributed software platform originally developed by Linked, based on log replication that allows client processes to publish and consume data. It is an alternative to the traditional queues-based messages systems (ActiveMQ).

Received by the editors: 25 September 2020.

2010 *Mathematics Subject Classification.* 68U99.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer Communication Systems**]: Distributed Systems – *Distributed applications*.

Key words and phrases. comparison, Raft, Kafka, consensus algorithm, replicated log.

Previous Work There are a lot of papers in the international scientific databases that are related to RAFT and to KAFTA or ZooKeeper. There are also papers that compare the algorithms, but the last ones are focusing on a comparison between the algorithm's implementation in Hyperledger Fabric [14, 15], a blockchain platform. The current paper focuses on Raft and Kafka as Apache plans to replace ZooKeeper with a RAFT based algorithm. The plans take the form of an approved change request, currently under development.

The paper is structured in four parts. The first part presents the Raft algorithm: the key aspects and the most important features are detailed, focusing on leader election and log replication algorithm. The second part presents Kafka, the leader election and log replication processes. The third part analyzes the differences and the similarities between Kafka and Raft, the last part is for conclusion and future work.

2. RAFT

Raft is an algorithm for managing replicated logs as the consensus is implemented by selecting a leader. The leader has the full responsibility of managing the logs without consulting the other servers. The typical number of servers used in a Raft cluster is five, because having five servers a system can tolerate up to two node failures.

In real life applications appeared the need for a collection of machines (nodes) to work in a coherent manner, to perform operations on reliable data which is constantly updating and to provide methods in case of node failure. One algorithm that tried to provide a solution for these requirements was Paxos algorithm, published in 1998 by Lamport.[1] The approach, based on a state machine replication, ensured that all cases in a distributed, fault-tolerant implementations are handled safely. However, according to Diego Ongaro and John Ousterhout from Stanford University [3], Paxos was neither easy to understand nor to implement. They have conducted a series of experiments with students which concluded the Raft is much easier to understand as Raft separates the key elements of consensus (such as leader election, log replication and contingency measures) and reduces the number of states in the system (for example, the logs are not allowed to have holes). The Paxos implementation in real systems such as Chubby [7], encountered many problems. Chubby implementation had to modify the Paxos algorithm, and the final solution was so different from the original algorithms that the comments from its developers are relevant in this context: "There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system . . . the final system will be based on an unproven protocol" [7].

The key features of Raft comparing with Paxos are:[3]

- The strong leader: the form of leadership is stronger than other forms of leadership in other consensus algorithms as Raft allows one-way updates. The log is replicated only from leader to the nodes.
- Raft allows one-way updates. The log is replicated only from leader to the other nodes called followers.
- Membership changes:the mechanism used for changing the set of servers in the cluster is based on a joint consensus method.

2.1. **Leader Election.** A server in the cluster can be in one of the three states (leader, follower or candidate to be a leader). By default, when the system is started a leader is selected, and a new leader is selected in case the current leader fails. The follower's servers are passive, they just respond to the requests from the leader and the candidates and apply the log changes send by the leader. All the client's requests are handled by the leader, and in the event that a client sends the request to a follower, the request is redirected to the leader.

Raft algorithm introduces in the election process the "term" concept. Raft splits time into terms of variable length, each term is identified by a consecutive integer. The election is started when the followers do not receive messages from the leader in a period of time called election timeout, they enter in candidate state and send messages to other nodes in order to be elected. Only one vote is allowed per server, and the vote uses the "first come, first served" methodology. A condition for a server to be voted - check for stale servers- is that the candidate term is greater or equal than its own (terms act as a logical clock and are sent in in the communication between servers). If a deadlock occurs (two candidates get the same number of votes), the election closes and a new election is started, each candidate initiating the selection process after a randomly timeout has passed (150-300ms)- the random timeout is used to increase the possibility that a leader is elected in the next election process. The following image displays the term representation in Raft[3]:

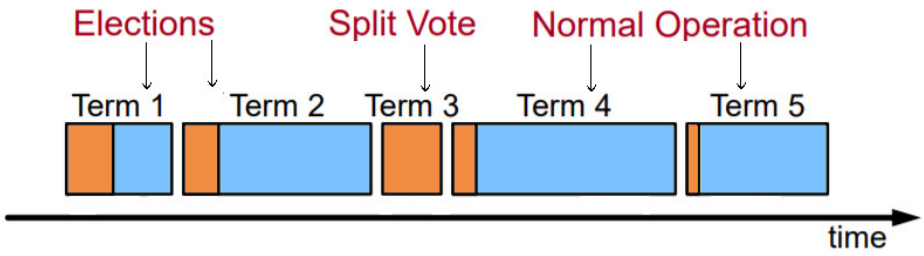


FIGURE 1. Term representation in Raft

2.2. **Log Replication.** Once a new leader is elected, it starts serving client's requests. Each request contains a command with data to be applied to the state machine. The leader appends the command to its log and starts the notification process for the other servers. After the log entry was replicated on the majority of servers, the leader applies /executes the command (the entry is committed), responds to the client and informs the followers. In case of network errors, the leader retries indefinitely, and based on the timers, the follower's logs will contain valid data. Once the follower is informed that a log entry was committed, it updates using the entry its own state machine. The inconsistencies between the leader and the follower's logs are solved by forcing the follower's log to update to the leader's log version. The following image presents the possible status of a replicated log in different nodes.

Due to the log management replication, the size of the logs is increasing in a rapid manner, so the log replicated systems and log replicated algorithms (including Raft) use compacting features. The compacting mechanism is based on the snapshot solution, where the entire system state is copied to a stable storage, and then the logs delete all the information up to that point.

Contingency measures in case of node failure, are different according to the node's role and can be structured as follows:

- if the leader node has failed, then a new election has to be performed.
- if the node's role is candidate or follower, the communication (using RPC) fails and the leader will send messages continuously until the server is back on track. At this point, the log will be updated according to the entries from the leader's log and the commands will be applied on the state machine.

3. KAFKA

Apache Kafka is a distributed streaming platform, based on a commit log that started as an internal LinkedIn project designed to provide a low-latency,

high-throughput platform for manipulating real-time data feeds. Kafka was designed to be used for two classes of applications that require building real-time streaming data pipelines used to get data between other systems in a reliable manner and for applications that react or manipulate the streams of data. Kafka is used for publishing and subscribing to streams of data, to store the data in a fault-tolerant way and to guarantee the order of the messages.

Kafka runs as a cluster of servers (same as Raft), and although the size of the cluster can have any size from 1 to (probably) hundreds, Kafka uses an additional software component for metadata management. This component, called Apache Zookeeper employs a quorum-based voting for leader election and always requires an odd number of nodes. In order to survive a single node failure, the minimum number of nodes is 3. To handle a two-node failure, it requires a cluster made of 5 nodes similar to what Raft recommends - number of 5 servers for a two-node failure). Kafka stores the stream of data in topics, each topic can be split in different partitions. In order to avoid duplicate writes from the client, the Kafka client library assigns a unique Id to each client process and a sequential number to every record submitted for processing. Thus, the Kafka leader can perform duplicate checks to avoid multiple writes [6]. Zookeeper is a distributed coordination service that provides synchronization and group services. Zookeeper uses an atomic broadcast protocol called Zab that ensures data to be kept consistent while trying to achieve a high performance primary-backup system. The protocol, its correctness and its performance were analysed in papers [10, 11, 12]; however for distributed systems, in which data accuracy is mandatory, Zookeeper might face some issues. The high efficiency in Kafka is based on the fact that the follower servers serve the read requests, and forward the write requests to the leader. That implies that there is the possibility that a read request can be served by providing stale data [13]. However, Kafka plans to switch from ZooKeeper to a Raft based implementation. The plans take the form of a change request currently under development, to implement a controller quorum based on Raft for leader election [16]. Using a different algorithm, the cases when the system will return stale values for read requests will be solved and fixed and it will also remove the dependency from a single service.

Kafka topics are just a scalability and performance design decision, as the order of operations are guaranteed only for data written in a single partition. The Kafka partition is similar to Raft log. The following image presents the structure of a topic and the method to handle write requests[1].

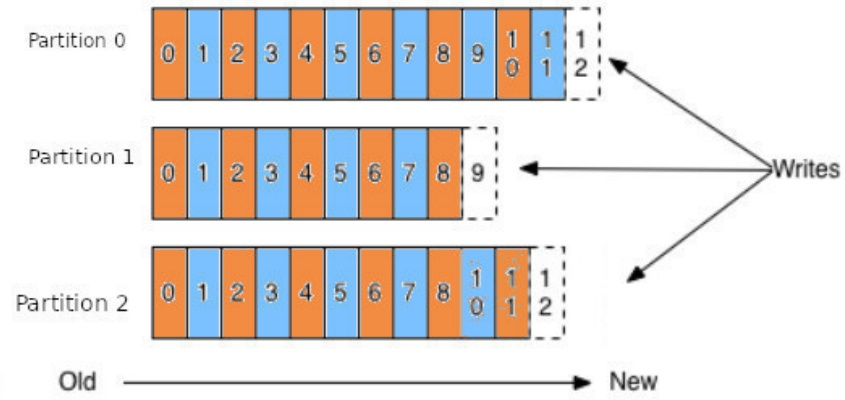


FIGURE 2. The structure of a topic [1]

Published data is written to the disk and then is replicated to ensure fault-tolerance. The producer is informed that the data was acknowledged only after it is replicated across the other server's partitions. The replication factor can be controlled for each topic.

3.1. **Leaders.** Each partition has a leader. This flexibility increases the performance, contributing to low latency and high-throughput. Kafka tries to avoid having the same leader node for a large number of partitions, so it enforces a balance mechanism. The leaders for partitions are distributed among different servers in the cluster. In Kafka, the number of replicas for each partition is configurable (so, it allows to have a different number of replicas based on a topic basis).

A node is considered to be "alive" if it is able to maintain its session with ZooKeeper and to replicate the leader's write in a reasonable time frame. The leader keeps track of all its "alive" or "in sync" nodes also known as the ISR (In-Sync Replicas) list, and when a follower falls behind or dies it is removed from the list.

When a server falls (that is not a leader for a specific partition), the leader will continue to notify and wait for that node. However, when the leader for a specific partition falls, a new server has to be selected from the followers as the leader for that partition. The constraint related to the follower is that it has to be up-to-date, as the crucial guarantee for a log replication algorithm is the following: if the client receives the commitment acknowledgment for a message, and the leader fails, that new selected leader must contain the message. The

leader has to wait for a sufficient number of followers to acknowledge they received the message before the leader declares the message is committed.

For cluster coordination tasks, Kafka uses ZooKeeper as a single control broker. It stores information about partition's leaders and is responsible for selecting a new leader. In the unlikely event that all the follower's node die, two methods can be used:

- the system can wait for a follower/replica from the preferred replica set to recover and to choose it as leader (hoping that the data is valid).
- or to use the first replica that is available (even if it is not in the preferred replica set)- this method is a trade-off between not having any data or having some "older" data.

3.2. **Communication and logs.** Kafka architecture consists of a cluster having minimum 3 servers (needed for ZooKeeper), but is scalable to hundreds for Kafka (as a note, Kafka can also run on a single server). On each server there are multiple partitions that could also be split into topics. Each producer can publish to any topic, it can publish to all the partitions of the topic in a round-robin way or it can add an id/key to the message in order to publish to the same partition. Consumers on the other hand are grouped in clusters, Kafka allows multiple consumer clusters. Consumers can be separate processes or separate machines, and each record published for topic is delivered to each subscribing group. Only one customer instance is allowed for a subscribing group. The only metadata used for each consumer is the position of the consumer in the log. This approach offers a higher flexibility compared to queue messaging, as it offers each client the possibility to re-read an older message(s) or to skip others to get to the up-to-date index [4].

Kafka persists the messages for a configurable time period, and manages to perform constantly with respect to the data log size. Storing data for a long time only requires more space on the disks. However, Kafka offers a method to compact the logs - the process is based on copying the file but saving only the last value of a message.

The replication algorithm ensures that all the logs from different servers have the same offsets for the same messages and the same order for the messages. The only exception from this rule occurs in the following situation: the messages in the leader log that are in the process of replication and were not saved in the follow's files.

In order to improve performance, to reduce the number of I/O requests and the overhead of the network round-trip, Kafka implements the "message set" abstraction that groups the messages, using bulk messages whenever is possible. Kafka tries to send larger packets instead of sending smaller messages one

by one, as they were published by the producers. Another method to improve performance is appending in the log in bulk mode. The servers append in the log in one go in continuous memory blocks (bulk mode) and the consumer receives a larger linear chunk every time. As a note, Kafka uses "push" type for producer- allowing them to send as many messages as they can produce, but for customers, it works with a "pull" method. The pull method implies that the messages are not sent to the consumers (the consumers might get overwhelmed and could not process them in a timely manner), so Kafka waits for the consumer to request for new messages.

4. KEY SIMILARITIES BETWEEN RAFT AND KAFKA

4.1. **Leaders.** Both solutions use the concept of Leaders for long-term, steady operations. This decision is in contrast with Paxos family of algorithms where each operation is voted by a majority of nodes a method which requires more round-trip communications between nodes. Using a master node, on the other hand, involves a much simpler communication between the leader and its followers. The leaders are elected using a consensus algorithm between the candidates or the up-to-date replicas.

4.2. **Log replication.** Again, both Raft and Kafka rely on a consistent, ordered log of operations that is replicated from the leader node to the followers. An operation (in Raft) or a message (in Kafka) is reported to the client as committed only after it is confirmed by the followers. Both behave in the same manner: first they apply the operation/message in their own logs, then try to replicate on the other nodes, and finally wait for the nodes to confirm the writing before sending a confirmation message to the producer.

Both systems provide the same guarantees regarding the replicated log: same positions in the replicated logs will contain the same, identical information.

Both solutions support log compaction by using periodical snapshots to reduce the size of the logs.

4.3. **Failover and leader election.** Both Raft and Kafka support automatic failover by executing an election algorithm among the up-to-date follower nodes that are still alive. The actual failure detection method is different, but the high-level election method is similar. Both solutions only handle non-Byzantine failures.

In case of a follower node failure, there is no impact in operation and the restarted follower or candidate node will simply replay all the operations from the leader node until it becomes up to date.

4.4. **Additional Kafka features. Topics and Partitions** Even if Kafka is based on a replicated log, it's architecture that uses topics enables a higher degree of parallelism and efficiency as requests are split between the servers in the cluster.

The fundamental condition for a log replication-based system is that the messages persist and that are sorted in the log in the same order as they were received. In Kafka, each partition is ordered and it consists in an immutable sequence of records. Each record in the partition is uniquely identified by a sequential id number called offset (the consumer only has to remember the offset of the last record he got in order to "move" in the log and to read records from the past). A consumer is able to deliberately go back to an old offset to re-process the data. This ability, even if is not specific for queues, can bring a lot of benefits to the consumers [8].

Load balancing

A no routing tier is used in Kafka, all the requests are sent directly to the broker that is the leader of the partition, avoiding a possible bottle neck in the routing tier. In order for a producer to know which is the leader of a specific topic, the servers in Kafka must be able to answer which servers are alive and where are located the leader nodes for each partition.

Moreover, the parallelism and load balancing is increased as each topic is split into partitions, and each publisher can write on a partition based on a round-robin method or can write to a specific partition (using the exposed interface for semantic partitioning which allows the publisher to specify a key to the partition) [4]

Consumer Groups

Kafka introduce the notion of consumer groups "a consumer group" is a cluster of consumers or processes in an application. For example, if the messages in a system must be consumed by two modules or processes of the same application, these processes should be configured in separate groups. The configuration is due to the fact that a message will be sent only once for each consumer group.

Highly Configurable (number of replicas, recovery policy, batch parameters)

Designed to be addressed to a large type of applications, Kafka is quite configurable, as it offers support by parameters to configure the number of replicas for each partition, and to modify it (for example: increasing the replication factor of an existing partition). The batch mode parameters are also configurable (time of waiting or number of messages per batch), and the recovery policy for falling nodes.

Batching

Batching is a feature that contributes a lot to the efficiency of the system. Batching in Kafka can be configured, to pile up and to send no more than a specified number of messages or to wait only a specific amount of time. This allows sending out large chunks of data through the network and fewer I/O operations on the server (even if they are larger) increasing performance. As a note, in exprimental validation there was a stong correlation betweend the packet sizes and the time (packet send interval), and the time needed for the packages to fit a phase-type distribution. The performance metrics are influenced by the various configuration and by the network latency [9].

5. Conclusion and Future Work

Kafka and Raft were designed in parallel in relatively the same period (Kafka was open sourced in 2011 and came out from Apache incubator in late 2012 as Raft was the subject of a lecture in march 2013 [5]) and they have a lot of similarities. Kafka can be used as an implementation of consensus distributed algorithms and can be a solution to develop a replicated state machine.

As a note, Kafka plans to remove the ZooKeeper dependency, in order to manage metadata in a more robust and scalable way using a RAFT based algorithm. Using ZooKeeper has some drawbacks because ZooKeeper is a separate system. Unifying the systems and the configurations between Kafka and ZooKeeper would ease Kafka usage and also would improve performance. The plans take the form of a change request, currently under development: it implements a controller quorum based on Raft for leader election [2].

The future work would be to develop a benchmark test to compare the performance between a Raft implementation and Apache Kafka. Both solutions provide a set of configuration parameters which can heavily influence the performance of each implementation. As an example: the replication factor set in Kafka can influence the efficiency. The hardware can also play an important role: does the system run on physical or on virtual machines (physical machines influence network delays as virtual processing introduces other types of delays).

References

[1] Apache Kafka. http://kafka.apache.org
[2] C. McCabe, Kafka improvements proposals" (having status accepted 2020, Retrieved from https://cwiki.apache.org/confluence/display/KAFKA/KIP-500
[3] D. Ongaro, J. Ousterhout (2013), In Search of an Understandable Consensus Algorithm, USENIX ATC 14, Proceedings of the 2014 USENIX Annual Technical Conference, June 2014 Pages 305-320, Retrieved from https://web.stanford.edu/ ouster/cgi-bin/papers/raft-atc14

[4] J. Kreps, Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines), Retrieved from https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines, Retrieved
[5] J. Ousterhout, Lecture for the Raft User Study, March 2013, Retrieved from https://raft.github.io/slides/raftuserstudy2013.pdf
[6] L. Lamport, The Part-Time Parliament, ACM Transactions on Computer Systems, 16 (2): 133-169. DOI:10.1145/279227.279229.
[7] T. D Chandra, R. Griesemer, J.Redstone, Paxos made live: an engineering perspective. In Proc. PODC'07, ACM Symposium on Principles of Distributed Computing (2007), ACM, pp. 398-407.
[8] W. Guozhang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, J. Stei, Building a Replicated Logging System with Apache Kafka, Proceedings of the VLDB Endowment, Retrieved from http://www.vldb.org/pvldb/vol8/p1654-wang.pdf, DOI:10.14778/2824032.2824063
[9] W. Han Z. Shang, K. Wolter, Performance Prediction for the Apache Kafka Messaging System, 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City, DOI: 10.1109/HPCC/SmartCity/DSS.2019.00036
[10] B. Reed, F Junqueira, A simple totally ordered broadcast protocol, Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware, September 2008, Article No. 2, Pages 1-6, doi:10.1145/1529974.1529978,
[11] A. Medeiros, ZooKeeper's atomic broadcast protocol: Theory and practice, Helsinki University of Technology, 2012, Retrieved from https://www.semanticscholar.org/paper/ZooKeeper-
[12] F. P. Junqueira, B. C. Reed and M. Serafini, Zab: High-performance broadcast for primary-backup systems, 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN), Hong Kong, 2011, pp. 245-256, DOI: 10.1109/DSN.2011.5958223
[13] P. Hunt, M. Konar, F. Junqueira, B. Reed, ZooKeeper: wait-free coordination for internet-scale systems, USENIXATC'10: Proceedings of the 2010 USENIX conference on USENIX annual technical conference, June 2010, Pages 11
[14] H. Yusuf1, I Surjandari. (2020). Comparison of Performance Between Kafka and Raft as Ordering Service Nodes Implementation in Hyperledger Fabric. International Journal of Advanced Science and Technology, 29(7s), 3549-3554. Retrieved from http://sersc.org/journals/index.php/IJAST/article/view/17652
[15] H. Yusuf, I Surjandari, Comparison of Performance Between Kafka and Raft as Ordering Service Nodes Implementation in Hyperledger Fabric, International Journal of Advanced Science and Technology Vol. 29, No. 7s, (2020), pp. 3549-3554, ISSN: 2005-4238 IJAST
[16] C. Wang, X. Chu, Performance Characterization and Bottleneck Analysis of Hyperledger Fabric, arXiv:2008.05946v1 [cs.DC]
[17] C McCabe, Kafka improvements proposals (having status accepted), 2020, https://cwiki.apache.org/confluence/display/KAFKA/KIP500

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu St., 400084 Cluj-Napoca, Romania

Email address: mpetrescu@cs.ubbcluj.ro

Montran Corporation, Romania

Email address: rpetrescu@montran.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 2, 2020
DOI: 10.24193/subbi.2020.2.06

DEFECT PREDICTION-BASED TEST CASE PRIORITIZATION

CRISTINA MARIA TIUTIN, MARC-TITUS TRIFAN, AND ANDREEA VESCAN

Abstract. Changes in the software necessitate confirmation testing and regression testing to be applied since new errors may be introduced with the modification. Test case prioritization is one method that could be applied to optimize which test cases should be executed first, involving how to schedule them in a certain order that detect faults as soon as possible.

The main aim of our paper is to propose a test case prioritization technique by considering defect prediction as a criteria for prioritization in addition to the standard approach which considers the number of discovered faults. We have performed several experiments, considering only faults and the defect prediction values for each class. We compare our approach with random test case execution (for a theoretical example) and with the fault-based approach (for the Mockito project). The results are encouraging, for several class changes we obtained better results with our proposed hybrid approach.

1. Introduction

In order to establish if a delivered software is reliable, appropriate verification practices have to be performed. A valuable and proper technique in this sense is provided by testing. In spite of its significant advantages, testing is, most of the time, a particularly expensive and demanding activity. The growth in software complexity is reflected in an exponential manner towards the cost of testing. As software evolves, the number of tests needed to preserve correct functionality increases as well, leading to a proportionate extension in the time

Received by the editors: 22 May 2020.
2010 *Mathematics Subject Classification.* 68M15,6804,68N30.
1998 *CR Categories and Descriptors.* D.2.8 **[Software engineering]**: Metrics – *Complexity measures* C.4 **[Performance of systems]** – *Performance attributes* D.2.4 **[Software engineering]**: Software/Program Verification – *Reliability* D.2.5 **Software engineering**: Testing and Debugging – *Testing tools* .
Key words and phrases. Test Case Prioritization, Regression Testing, Defect Prediction, Average Percentage of Faults Detected (APFD).

taken to execute the test suite. Thus, a solution which manages to lower the cost, while also preserving the verification quality is required.

The source code quality is a characteristic of any software project. The constant variation of this metric should be kept within certain bounds to ensure the proper functionality of an application, while also not neglecting its general cost. The quality of the code involves taking into consideration many factors, like the number of programmers writing the code or the change frequency as the entire bug-fixing process is performed. Increased consideration must be given to this process, as a module in which a found bug is fixed may be prone to future failures. Considering the overall quality of the source code (which will be later reflected in the quality of the product), a specific amount of attention should be involved in the bug fixes. One of the methods that help with this aspect is regression testing, through the various existing approaches.

Regression Testing (RT) [7] is "the process of validating modified software to detect whether new errors have been introduced into previously tested code and to provide confidence that modifications are correct". Similar other definitions may be found in [4] as "the retesting of the software that occurs when changes are made to ensure that the new version of the software has retained the capabilities of the old version and that no new defects have been introduced due to the changes", in [10] as "performing testing after making a functional improvement or repair to the program", and in [20] as "rechecking test cases passed by previous production versions".

Regression testing [1] is an essential part of any viable software development process and in practice is often incorporated into a continuous integration service. It is well known that if the regression tests do not finish in a timely manner, the development process is disrupted.

The challenging problem regarding regression testing refers to the fact that at a given point during the development, the test suites are so large that running all the test cases would take too much time. There are many approaches that investigate the regression testing problem, from test suite minimization to test case selection and test case prioritization. A review on various RT techniques is presented in paper [3]. A more detailed view about subtypes of RT is presented in [14].

The Test Case Prioritization (TCP) [3] technique helps to increase the rate of fault detection. It also increases in practice the effectiveness of test suites. To evaluate the regression-based test suite prioritization, the ordering is in general measured using the APFD (Average Percentage Faults Detected) metric [12]. More information regarding APFD metric is provided in Section 2.2.

Test Case Prioritization has been widely researched as a strategy for reducing the time needed to discover regressions in software. While many different

approaches have been developed and evaluated, previous experiments have focused on faults or code coverage used as prioritization techniques, and only few approaches [11] considered the defect prediction probability of a class. The probability associated to a class may determine the likelihood of it to contain software bugs. This metric could be used as a test case prioritization technique. We propose an investigation in this direction, i.e. test case prioritisation based on defect prediction.

The aim of this paper is to propose and investigate an algorithm that considers both the faults and the defect prediction probability as criteria for prioritization. We employ the use of defect prediction probability of a class as described in [11] where the aim was to study how to configure the Schwa tool [5] to maximize the likelihood of an accurate prediction. More information regarding the computation of the defect prediction probability of a class is provided in Section 2.2.

The reminder of the paper is organized as follows: Section 2 presents related work regarding the test case prioritization and also referring to defect prediction, Section 3 describes our approach regarding Test Case Prioritization that considers also the defect prediction probability. The experiments in Section 4 revealed that our approach finds better solutions for several class changes, and the last section outlines the concluding remarks and future work.

2. Background on Test Case Prioritization and Defect Prediction

Regression testing [7] is an important process that mostly every software systems will go through multiple time during its development and maintenance process. Being highly time consuming, performance improvements are a must, thus several studies are currently proposed to minimize the cost of the process and maximize its efficiency.

2.1. Definition of Test Case Prioritization.

Definition for TCP is given by Graves [7] in what follows.

Definition 1. ***Test case prioritization*** [7]***:*** *Given a test suite, T, the set of permutations of T, PT; a function from PT to real numbers, f.*

Problem: to find $T \in PT$ such that

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')] \tag{1}$$

The function f assigns a real value to a permutation of T according to the test adequacy of the particular permutation.

The ideal order would be the one that reveals faults soonest, the rate of which can be expressed in APFD (Average Percentage Faults Detected) metric

(described in the next subsection). The latest regression testing approaches use various code coverage criteria since fault detection is not known in advance. Several approaches are scrutinized by Rothermel et al. [15] using various coverage measurements, showing that coverage prioritisation can improve the rate of fault detection [19].

In what follows we present a short overview of various test case prioritization techniques and also studies related to defect prediction.

2.2. **Test case prioritization approaches.** This section details the APFD and defect probability metrics, alsor presenting various existing approaches for test case prioritization.

Average Percentage Faults Detected

APFD measures *the effectiveness of TCP as the rate of fault detection* achieved by the produced ordering of test cases [16]. APFD relates with earlier fault detection abilities.

Definition 2. *Average Percentage Faults Detected (APFD)* [12] *is defined as follows:*

$$APFD = 1 - \frac{Tf1 + Tf2 + ... + Tfm}{mn} + \frac{1}{2n} \tag{2}$$

Where n be the no. of test cases and m be the no. of faults. (Tf1,...,Tfm) are the position of first test T that exposes the fault.

Paterson et. al [11] proposes a test case prioritisation technique considering defect prediction, a strategy which analyses code features in order to predict the likelihood that a file or function inside a software system is faulty. The paper introduces a test case prioritisation approach, namely G-clef, that uses bug prediction data to reorder a test suite in such a way that it concentrates first on the classes that are prone to include faults. The paper not only presents this new test case prioritization strategy, but also compares the approach with other nine existing approaches using an empirical study on real faults.

Defect prediction probability of a class

Schwa [5] uses a ranked-based technique, Time-Weighted Risk (TWR) to estimate how reliable a Java class is, thus the function has its maximum value when a component was changed recently:

$$twr(t_i) = \frac{1}{1 + e^{-12t_i + w}} \tag{3}$$

Schwa [5] estimates the likelihood that a Java class c contains a bug using Equation 4, in which each of the three factors (i.e., revisions, authors, and fixes) is calculated and modified by a weight, where the sum of all weights

must be equal to 1. For each component the score is computed as provided in Equation 4. Intuitively, a Java class with higher defect value is less reliable, thus more likely to contain a bug, than those classes with low defect value.

$$\begin{aligned} score = revisions * revisions_{weight} \\ + fixes * fixes_{weight} \\ + authors * authors_{weight} \end{aligned} \tag{4}$$

Test case prioritisation approaches

Test Case Prioritisation (TCP) arranges test cases into an optimal order so that a specific criteria is met as early as possible. The work of [13] presents a black-box strategy called REMAP that incorporates three fundamental components: a rule miner, a static prioritiser, and a dynamic executor and prioritiser. The relations between test cases are defined using fails and pass rules being mined from the historical execution data. Multi-objective search is applied to statically prioritise test cases considering two objectives: fault detection capability and test case reliance score. The test case order is dynamically updated using the results of the test case execution and the fail and pass rules.

2.3. **Defect prediction approaches.** This section contains various existing approaches related to defect prediction approaches.

When discussing a defect prediction solution, one approach taken into consideration is a multi-objective approach [2], the two conflicting objectives being the cost and effectiveness. The approach allows software engineers to choose between various predictors: predictors that identify a high number of defect-prone artifacts, predictors requiring a lower cost, and predictors achieving a cost-effectiveness compromise.

In paper [9], the search problem was enhanced, such that the defect prone classes are predicted using the Object-Oriented metrics design suite instead of static code metrics. An extensive comparison of eighteen machine learning techniques in the context of defect prediction was performed. Six releases of widely used Android application package were used.

A qualitative and quantitative study regarding defect prediction was done [17] to investigate what practitioners consider and expect in contrast to research findings. The study that was done through interviews and questionnaires. The results revealed that most respondents are willing to adopt defect prediction techniques, but there is a discrepancy between practitioners' perceptions and supported research evidence regarding defect density distribution. Also, the most preferred level of granularity of defect prediction by practitioners is at the feature level.

However, as mentioned before, there exists a gap regarding test case prioritization and defect prediction probability of classes. The approach we propose in this work addresses an empirical attempt of the method to consider defect prediction probability of a class as well.

3. Our Defect Prediction - based Test Case Prioritization

Our approach investigates the test case prioritization problem considering various criteria (faults and defect prediction probability) using a Greedy-based approach in order to select the test case ordering. We compare our approach that uses the defect probability with two other approaches: one approach that only considers the faults found by the test cases and the other that is a random execution of the test suite.

As we mentioned before, our approach is based on greedy strategy. We have tested 3 versions of greedy algorithm in order to obtain the ordering which is expected to retrieve the best results, as graphically depicted in Figure 1: a random prioritization, a faults-based prioritization order, and also an approach based on defect prediction.

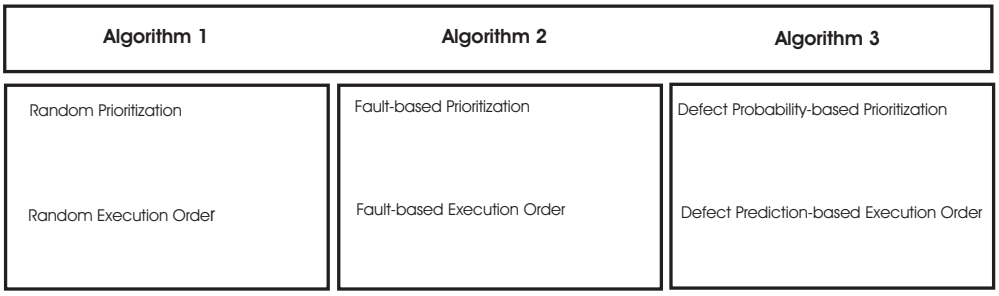


Figure 1. Overview of the three investigated Test Case Prioritization approaches.

We compare our approach with the basic Random Prioritization approach that is described in Algorithm 1. The approach just randomly orders the test cases that are relevant for the changed c class and afterwards adds the remaining test cases from the test suite. We performed our expriments considering that each class c of the project p is changed.

The second approach in our investigation uses the number of faults discovered by each test case that is relevant for the changed class c. The description of this approach is provided in Algorithm 2.

Random Prioritization Algorithm
Data:
list of all classes in a project with their corresponding tests;
list of all tests with corresponding number of bugs discovered for each test.
Result:
prioritized list of tests (and APFD) for each class in the project.
foreach *class c in project p* **do**
    get the list of tests relevant for c;
    randomly order the tests;
    add to the resulted list all the tests that are not relevant for the class;
    compute APFD for the obtained prioritized list of tests;
end

Algorithm 1: Random Prioritization Algorithm

Faults-Based Prioritization Algorithm
Data:
list of all classes in a project with their corresponding tests;
list of all tests with corresponding number of bugs discovered for each test.
Result:
prioritized list of tests (and APFD) for each class in the project.
foreach *class c in project p* **do**
    get the list of tests relevant for the class c;
    sort list descending based on the number of bugs discovered by each test;
    add randomly to the resulted list all the tests that are not relevant for the class;
    compute APFD for the obtained prioritized list of tests;
end

Algorithm 2: Faults-Based Prioritization Algorithm

Our new proposed approach uses the defect prediction probability for each class. The tests are order based on the maximum defect probability among all test cases that are relevant to the changed class c. The description of this approach is provided in Algorithm 3.

Defect-Prediction Prioritization Algorithm
Data:
list of all classes in a project with their corresponding tests;
list of all tests with corresponding number of bugs discovered for each test.
Result:
prioritized list of tests (and APFD) for each class in the project.
foreach *class c in project p* **do**
  get the list of tests relevant for the class;
  foreach *test t in tests for class c* **do**
    get the list of all classes tested by the test t;
    access the maximum defect probability from all the classes;
  end
  sort list descending based on the maximum defect probability;
  add randomly to the resulted list all the tests that are not relevant for the class;
  compute APFD for the obtained prioritized list of tests;
end

Algorithm 3: Defect-Prediction Prioritization Algorithm

We will exemplify our approach by using a theoretical example provided in Table 1 and Table 2.

	t_1	t_2	t_3	t_4	t_5	t_6	defectProb
c_1	1	0	0	0	0	0	0.25
c_2	0	0	0	1	0	0	0.65
c_3	1	0	0	1	0	0	0.70
c_4	0	0	0	0	1	0	0.55
c_5	0	0	0	0	0	1	0.45
c_6	1	0	0	0	0	0	0.85
c_7	1	0	0	0	1	1	0.80
c_8	0	0	0	0	0	0	0
c_9	1	1	1	0	0	1	0.82
c_{10}	0	0	0	0	1	0	0.72

TABLE 1. Class to tests cases matrix

	f_1	f_2	f_3	f_4	f_5
t_1	1	0	0	0	0
t_2	0	0	0	1	0
t_3	0	0	0	0	0
t_4	1	0	0	0	1
t_5	0	1	0	0	0
t_6	0	0	1	0	0

TABLE 2. Test cases to bugs matrix

We consider the provided order of the test cases as

$$\text{TC}=[t_1, t_2, t_3, t_4, t_5, t_6].$$

As we mention earlier, test case prioritization is applied in the context of regression testing, thus when a change is done in the source code due to the fixing of a fault. In that follows next, we present each approach using this theoretical case study and considering that the changed class is c_3.

The random algorithm starts from an existing order of the tests (t_1, t_2, ..., t_n) and after generates a new random execution order of the test suite.

First, the tests corresponding for class c_3 are selected. Those tests are (t_1, t_4). The algorithm randomly chooses an order for the list and then appends the rest of the tests (that do not test the class c_3). A possible order of the tests may be TC3=[t_1, t_4, t_5, t_6, t_2, t_3].

The most used criteria regarding the test case prioritization is the one regarding the faults the test cases discover, thus ordering the test cases to be executed using the discovered number of faults. For the theoretical example this ordering is: C3=[t_4, t_1, t_5, t_6, t_2, t_3].

The tests t_1 and t_4 are ordered descending based on the number of bugs discovered. Test t_1 discovers 1 bug, while t_4 discovers 2 bugs, such that the final order of the tests may be TC3=[t_4, t_1, t_5, t_6, t_2, t_3], or any random order between the third and the last test.

Our proposed approach considers also the defect probability defined by Peterson et. all in paper [11].

The probability is defined as the likelihood that a Java class contains a bug. This estimation is computed based on three weights, corresponding to the importance given to revisions, fixes and authors, applied to the Time-Weighted Risk of a class, a metric which uses different features to estimate how reliable a Java class is. The sum of the weights must be equal to 1. For the proposed approach, the weights were defined as 0.25 for revisions, 0.5 for fixes and 0.25 for authors. The computation uses an upper limit for the number of commits taken into consideration when estimating the defect probability. Two different values were chosen as an upper limit, 50 and 1000 commits respectively.

Our defect prediction-based prioritization approach is performed in three major steps: the first one selects first the test cases that are related to the class that was changed and order them based on the number of discovered faults, the second one orders the remaining test cases from this set (if there are test cases that did not discover any faults) based on the defect probability value, and the last step adds randomly the test cases that did not take part of the changed class, thus completing the test suite with all the test cases.

Using the defect probability prioritization algorithm, the ordering of the tests t_1 and t_4 is determined by the maximum defect probability for the classes that are tested for each test. For test t_1, the classes verified by it are c_1, c_3, c_6, c_7 ,c_9, the maximum probability being 0.85, for c_6. In a similar matter,

for test t_4 the verified classes are c_2 and c_3, with a maximum probability of 0.70, corresponding to c_3. Based on this information, the test order will be t_4, followed by t_1. A possible test ordering for class c_3 is TC3=[t_4, t_1, t_5, t_6, t_2, t_3] (or the last 4 tests in any order).

4. Experiments and results

Choosing the method for our research investigation was based on the book of Yin [18] that supported us in identifying the case study method and revealed how to do the research design.

Generalization from a case study to theory is an important issue, Yin [18] stating that the analytic generalization should be used for case studies: multiple cases resemble multiple experiments, thus the mode of generalization being analytic. Replication [18] may be claimed if two or more cases support the same theory: some replications seek to duplicate the exact conditions of the original experiment, others change some experimental conditions.

4.1. **Experiments Design.** The replication strategy that we used considered various number of classes, test cases and faults, those numbers being changed from one experiment to the others. The first experiment considers the theoretical example with a small number of classes, test cases and faults. The next two experiments are based on a open source project and even if the same number of classes, test cases and faults are used, they have different number of classes with the defect probabilities greater than 0 as described next.

Experiment 1: 10 classes, 6 test cases, 5 faults.

Experiment 2: 365 classes, 116 test cases, 38 faults, 199 classes with defect probabilities > 0 (considering 50 commits).

Experiment 3: 365 classes, 116 test cases, 38 faults, 336 classes with defect probabilities >0 (considering 1000 commits).

It is worth mentioning that we performed the above experiments considering that each class was changed, thus applying test case prioritization for each scenario.

4.2. **Case studies used.** The experiments are based on a theoretical project and on an open project, Mockito from the Defect4J database [8]. We have used the defect probability defined by Peterson et. all in paper [11] as we mentioned earlier in Section 3.

The Defect4J database [8] contains a set of software projects that contain reproducible bugs. The fact that the projects were originally obtained from different version control systems allows the possibility of collecting information regarding the faults of the program from versioning perspectives. Each project

has associated a list of faults with the corresponding test that discover a particular fault. In addition, a list of classes that are verified during the execution of a particular test is also kept.

Those files represented the base for the data pre-processing. Two comma separated value files were created, representing different matrices useful for further computations. One of the matrices represented a binary bug-to-tests correspondence, where 1 denotes that a bug is discovered by a certain test. The form of a row is class name and k values of 1 or 0, where k is the number of tests. The second matrix represents the tests relevant for a particular class. A row-structure contains the class name and k values of 1 or 0, where k is the number of tests, where 1 represents if a particular test is relevant for the given class.

Another tool that was used for computing the defect probability for the classes of a certain project is Schwa [6]. The output contains a json file with the defect probability for each class, while also mentioning a defect probability value for each method that is part of a class. The information was processed in a csv file, which is a mapping between the class names and the defect probabilities obtained. Two separate files were obtained, one by taking into account the last 50 commits of the Mockito project, obtained from the Defect4J library, and other file made by analyzing the last 1000 commits of the same project.

The effectiveness of the proposed prioritization technique, thus the ordering of the test cases is assessed using the rate of faults detected using the APFD metric as described in the above sections.

4.3. **Experiment 1.** The first case study considers the theoretical example that we provided in Section 3. The case study contains: 5 classes, 7 test cases and 4 faults, along with the defect probabilities being provided. The class that was modified is c_3.

We consider the provided order of the test cases as

$$\text{TC}=[t_1, t_2, t_3, t_4, t_5, t_6].$$

The test cases that are involved in the c_3 class that is changed are: t1 and t4.

The execution of the random algorithm found the following result: [t1, t4, t3, t2, t6, t5]. The obtained APFD values is: 0.48.

The execution of the algorithm that considers only the number of discovered faults by the test cases found the following result: [t4, t1],[t5, t6, t2, t3]. The APFD value is: 0.62.

The execution of our proposed algorithm found the following result: [t1, t4],[t5, t6, t2, t3]. The APFD value is: 0.58.

For this example, the proposed algorithm does find better solution than the random approach (APFD is greater, $0.58 > 0.48$), but did not find better solution than the faults-based approach ($0.58 > 0.62$).

As mentioned earlier, we have also executed the algorithm considering that each class is changed, that performing test case prioritization for each class modification. The results in Figure 2 still do not find better solution for our approach. We executed the algorithm such that each class in the project is changed thus being required to apply test case prioritization in each scenario.

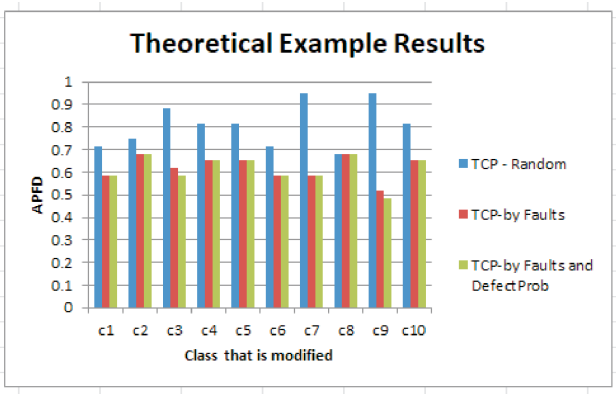


FIGURE 2. Theoretical example results

4.4. **Experiment 2.** Our next experiment considers the Mockito project from the Defect4J dataset [8].

The Mockito case study contains: 365 classes, 116 test cases and 38 faults, along with the defect probabilities being provided. The defect probabilities were computed using the last 50 available commits, which resulted in values for 190 classes. The rest of the classes had associated a defect probability of 0.

For example, for class "org.mockito.internal.invocation.serializablemockitomethod" the APFD values found for each approach are: TCP-Random APFD=0.72, for TCP-byFaults APFD=0.72, and for TCP-byFaultsAndDefectProb APFD=0.72.

Comparing our approach with the byFaults approach we obtained the results in Figure 3: for 26 classes the APFD values are higher. If we consider APFD highest or equal then for 132 number of classes we obtained better or equal APFD results.

4.5. **Experiment 3.** The last conducted experiment considered the same Mockito case study with different defect probabilities computed.

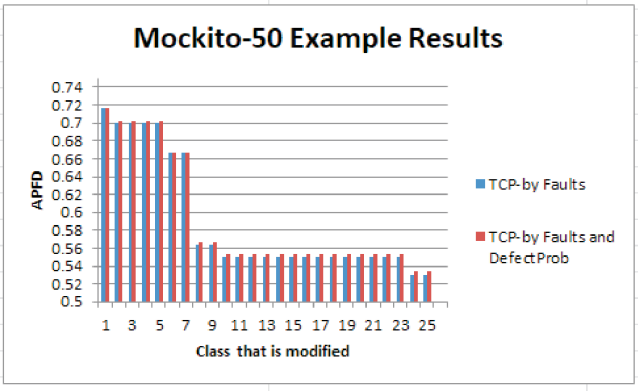


FIGURE 3. Mockito-50 Results: For out of 190 classes, we obtained better or equal APFD values for 132 classes, and obtained higher APFD values for 26 classes.

The Mockito case study contains: 365 classes, 116 test cases and 38 faults, along with the defect probabilities being provided. The defect probabilities were computed using the last 1000 available commits, which resulted in values for 336 classes. The rest of the classes had associated a defect probability of 0.

The results for this experiments revealed that for 114 classes we obtained higher or equal APFD when comparing TCP-byFaults approach with the TCP-byFaultsAndByDefectPred. Figure 4 contains these results.

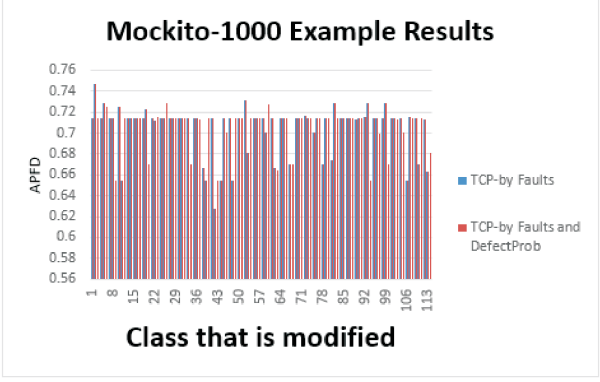


FIGURE 4. Mockito-1000 Results: For out of 336 classes, we obtained better or equal APFD values for 114 classes.

4.6. **Discussions.** Following the replication approach to multiple experiments [18], each individual experiment was finalized by an individual report (conclusion) that will be next considered to be part of a summary report, i.e. a cross-case conclusions.

In our case the results obtained for each experiment are reported in the above sections and in this section conclusions about the potential used of defect prediction probabilities for the test case prioritization problem are drawn: augmenting the standard test case prioritization criterion, i.e. number of faults discovered by the test cases, with the defect prediction probability of each class may lead to better results regarding which test cases should be first executed in the context of regression testing.

Our approach considered only one of the classes to be modified, thus computing the regression test suite only for this one modification. Future work will tackle these multiple changes in the classes.

5. Conclusions

Regression testing, with all existing strategies plays an important role in identifying and fixing faults after software changes are performed. Test case prioritization is one of the strategies that could be applied and that can provide important information about the system under test regarding the best test cases that may identify existing faults.

Our hybrid test case prioritization approach considers not only the number of faults discovered by the test cases but also the defect probability of each class.

The design of our experiments considered various combinations of number of classes, test cases, faults and classes with defect probability not zero. Our hybrid approach is compared with two other approaches: a random based approach and a faults-based approach. The results are encouraging, for several class changes we obtained better results with our proposed hybrid approach.

References

[1] Paul Ammann and Jeff Offutt. *Introduction to Software Testing,2nd edition.* Cambridge University Press, 2016.

[2] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Multi-objective cross-project defect prediction. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 252–261, 2013.

[3] Anjali Choudhary and Tarun Dalal. A review on regression testing techniques. *IJ of Emerging Trends and Technology in Computer Science*, 4(3):56–59, 2015.

[4] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing.* Springer-Verlag New York, Inc., 2002.

[5] A. Freitas. Software repository mining analytics to estimate software component reliability, Faculdade de Engenharia da Universidade do Porto, Master's thesis, 2015.

[6] Andre Freitas. *Software Repository Mining Analytics to Estimate Software Component Reliability*. PhD thesis, Faculdade de engenharia da Universidade do Porto, 6 2015.

[7] T. L. Graves, M. J. Harrold, J. Kim, A. Porters, and G. Rothermel. An empirical study of regression test selection techniques. In *Proceedings of the 20th International Conference on Software Engineering*, pages 188–197, 1998.

[8] René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, page 437–440, New York, NY, USA, 2014. Association for Computing Machinery.

[9] Ruchika Malhotra and Rajeev Raje. An empirical comparison of machine learning techniques for software defect prediction. In *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, BICT '14, page 320–327, Brussels, BEL, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[10] Glenford J. Myers and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[11] D. Paterson, J. Campos, R. Abreu, G. M. Kapfhammer, G. Fraser, and P. McMinn. An empirical study on the use of defect prediction for test case prioritization. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 346–357, 2019.

[12] R Pradeepa and K VimalDevi. Effectiveness of test case prioritization using apfd metric: Survey. In *International Conference on Research Trends in Computer Technologies (ICRTCT—2013). Proceedings published in International Journal of Computer Applications (IJCA), ISSN: 0975–8887*, pages 1–4, 2013.

[13] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen. Remap: Using rule mining and multi-objective search for dynamic test case prioritization. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 46–57, 2018.

[14] Mohammad Rava and Wan M.N. Wan-Kadir. A review on prioritization techniques in regression testing. *International Journal of Software Engineering and Its Applications*, 01(1):221–232, 2016.

[15] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: An empirical study. In *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*, pages 179–188. IEEE, 1999.

[16] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: an empirical study. In *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, pages 179–188, 1999.

[17] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang. Perceptions, expectations, and challenges in defect prediction. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[18] Robert K. Yin. *Case Study Research: Design and Methods (Applied Social Research Methods)*. Sage Publications, fourth edition. edition, 2008.

[19] Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ISSTA '07, page 140–150, New York, NY, USA, 2007. Association for Computing Machinery.

[20] Michal Young and Mauro Pezze. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2005.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu St., 400084 Cluj-Napoca, Romania

Email address: {tcie2430, tmie2434}@scs.ubbcluj.ro, avescan@cs.ubbcluj.ro