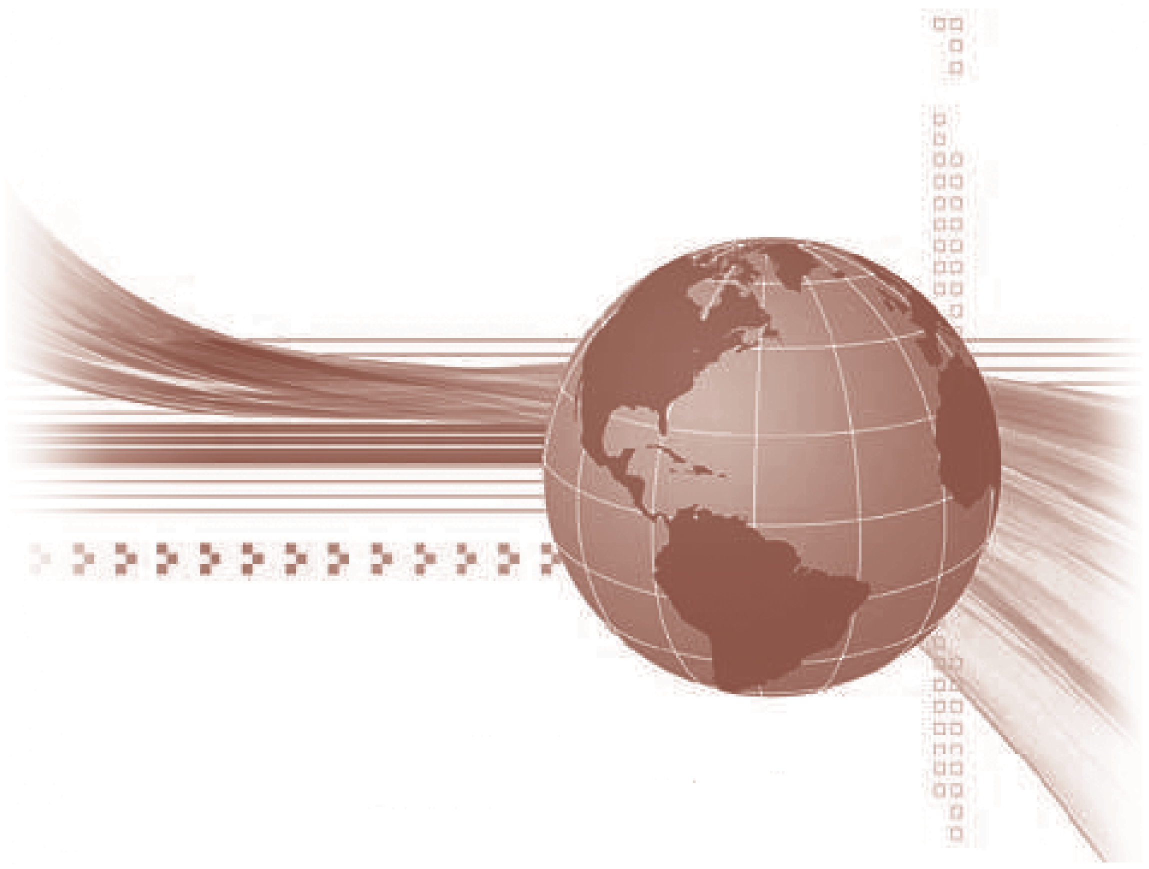
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

1/2021

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI INFORMATICA

No. 1/2021

January - June

EDITORIAL BOARD

YEAR ....VOLUME 66 (LXVI) 2021
MONTH JUNE
ISSUE 1

STUDIA UNIVERSITATIS BABEȘ-BOLYAI INFORMATICA

1

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

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.01

AN ANALYSIS ON VERY DEEP CONVOLUTIONAL NEURAL NETWORKS: PROBLEMS AND SOLUTIONS

TIDOR-VLAD PRICOPE

Abstract. Neural Networks have become a powerful tool in computer vision because of the recent breakthroughs in computation time and model architecture. Very deep models allow for better deciphering of the hidden patterns in the data; however, training them successfully is not a trivial problem, because of the notorious vanishing/exploding gradient problem. We illustrate this problem on VGG models, with 8 and 38 hidden layers, on the CIFAR100 image dataset, where we visualize how the gradients evolve during training. We explore known solutions to this problem like Batch Normalization (BatchNorm) or Residual Networks (ResNets), explaining the theory behind them. Our experiments show that the deeper model suffers from the vanishing gradient problem, but BatchNorm and ResNets do solve it. The employed solutions slighly improve the performance of shallower models as well, yet, the fixed deeper models outperform them.

1. Introduction

We have witnessed a lot of breakthroughs in deep learning lately [15] and all of them had a certain thing in common: very large and deep neural networks. The network depth has played probably the most important role in these successes, just over a span of a few years, the top-5 image classification accuracy over the ImageNet dataset has increased from **84%** [12] to **95%** [20], [16] using deeper networks with rather small receptive fields [2]. There seems to be a general rule that deeper is better and other results in this area have also underscored the superiority of deeper networks [25] in terms of accuracy and/or performance.

However, to achieve the advancements we have today, challenging problems had to be solved. There is a fundamental problem that very deep CNNs (Convolutional Neural Networks) suffer from. It was showed [10] that training

Received by the editors: 26 January 2021.
2010 *Mathematics Subject Classification.* 68T45 .
1998 *CR Categories and Descriptors.* I.2.1 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*.
Key words and phrases. Deep Learning, Neural Network, Image Classification, Deep Convolutional Neural Network, Vanishing Gradient Problem, VGG.

becomes more difficult as we increase the number of layers of a NN (Neural Network), stacking many non-linear transformations typically results in poor propagation of activations and gradients [19]. This is caused by the well-known problem of **vanishing/exploding gradients** [7]. With a big model, as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitively small (or infinitely large) and a meaningful signal won't reach the input layers causing the network not to learn anything even after the first iterations.

In this paper, we are going to visualize and explore this problem, analyze and test proposed solutions like **BatchNorm** [10], Resnets [6] and DenseNets [9]. We experiment on VGG (Visual Geometry Group) architectures [16] which are based on convolutional layers and are still an inspiration for top models these days. The **motivation behind this work** is the fact that current and previous state-of-the-art technology in computer Vision AI does heavily rely on a very deep convolutional architecture. Therefore, it is **important** to know **how to detect problems** and how to **successfully fix them** when using such tools. We will confirm one of the statements that were thought about the VGG networks - going deeper without any change whatsoever is unacceptable, visualizing the gradients during training. We propose some intuition and a mathematical underpinning of the problem that causes this phenomenon and explore solutions.

Our main contribution is a throughout evaluation of VGG networks of increasing depth using different stabilization techniques on the CIFAR100 image dataset [11]. We show that a plain (traditional) VGG network with 7 convolutional layers outperforms a much deeper network that uses 37 convolutions on a same setup. We prove that this is caused by the vanishing gradient problem (analyzing the gradients with respect to the model parameters) and we fix it using BatchNorm and Resnets showing that deeper is better if proper techniques are used to stabilize the learning of such models.

For the purpose of this research, we have used one of the most powerful GPU machines openely available to public as of today: the Nvidia Tesla V100, which allowed for 60% decrease in training time compared to other solid GPU workstations like Tesla T4 or K80.

2. Identifying problems of a deep CNN

As a baseline model we have a VGG network with 7 convolutional layers and 1 flatten layer. After training for 100 epochs, this model gets a train accuracy of around 54% and a test accuracy of 49%. The learning stage of this model is healthy enough, the accuracy does not decrease after a certain point and the generalization gap analyzing the loss is not that big. The gradient flow -

mean absolute value of gradients with respect to the model parameters can be seen in Figure 1. However, these are not satisfactory results, good models on this dataset achieve consistently over 70% accuracy [9].

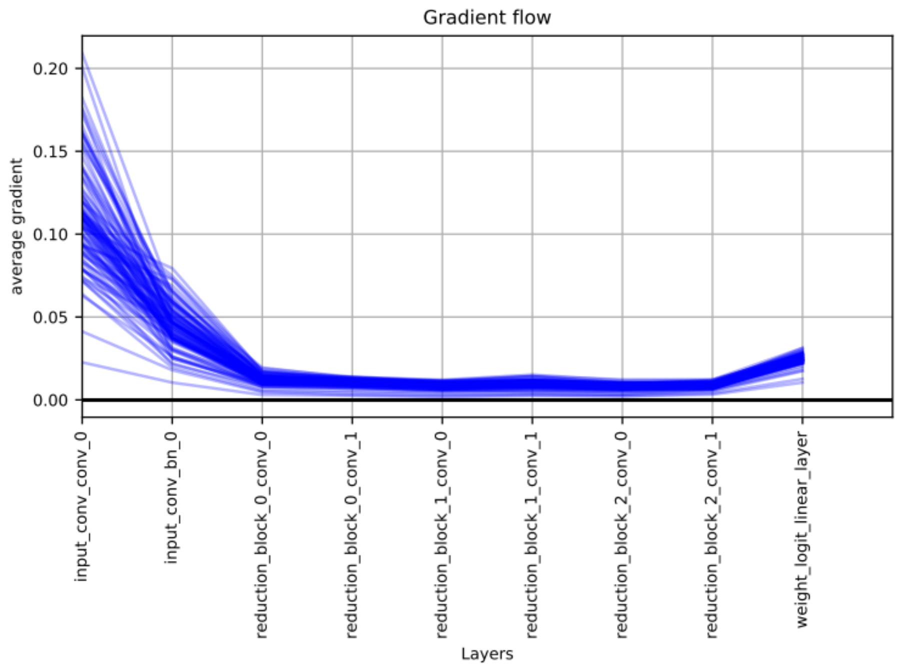


FIGURE 1. Gradient Flow in each layer for the healthy VGG 08 network.

Therefore, we tried repeating convolutional blocks over and over until we ended up with a VGG neural network with 37 convolutional layers and 1 flatten layer. Unsurprisingly, training this network, in its current form, did not yield great results, the test and train accuracy remained steady at 1% during the whole training stage and the loss did not decrease almost at all. It seems that the more shallow architecture beat the deeper one in this experiment. In 1989, Cybenko proved [3] that a network with a large enough single hidden layer of sigmoid units can approximate any decision boundary. Empirical work, however, suggests that it can be difficult to train shallow nets to be as accurate as deep nets. Moreover, for vision tasks, multiple studies suggest that deeper models are **preferred** under a **parameter budget** [4], [19], [16].

So why is it not the case that we get better performance with higher number of hidden units?

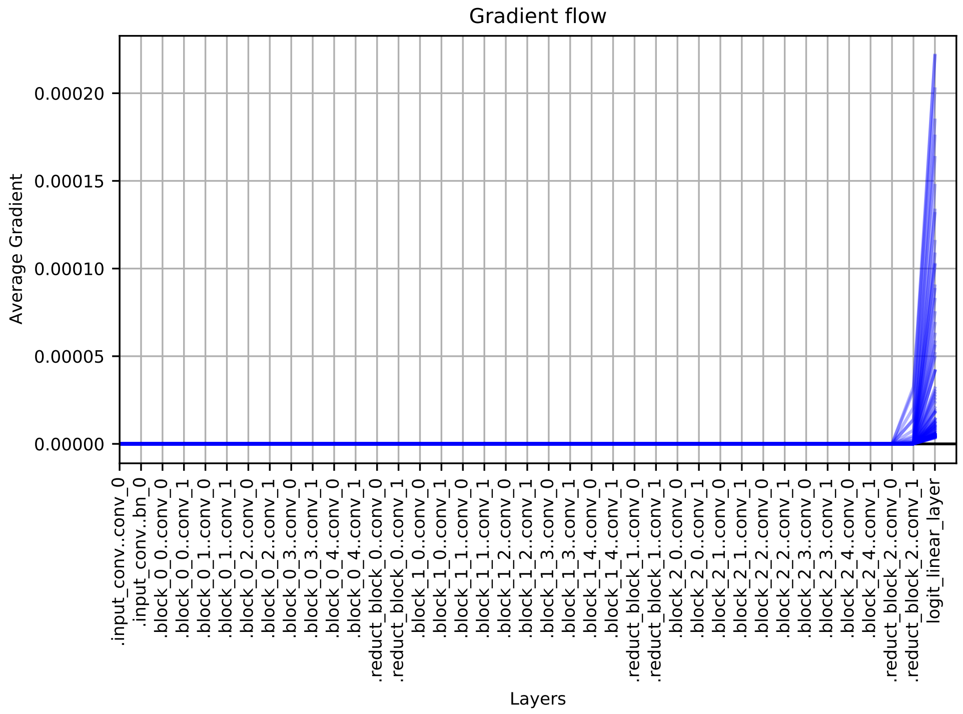


FIGURE 2. Gradients vanishing when training a VGG model with 37 convolutional layers (VGG 38 network). Simply stacking layers does not work.

Well, increasing network depth does not work by simply stacking layers together. Very deep networks are hard to train because of the vanishing gradient problem (Figure 2). An **intuition** for that happening is that, when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero after several applications of the **chain rule**, so gradients aren't really back-propagated sufficiently to the initial layers of the network. This can be clearly seen in the **Figure 2** that shows the mean absolute value of the gradients at each epoch. The gradients quickly turn very close to 0 after just 2 layers during backpropagation from output layer to input layer.

This is just an intuition, but neural networks haven't been regarded as uninterpretable black-boxes for no reason, can we somehow explain this phenomenon mathematically?

In a way, yes. In very deep architectures, the variance of the data changes at each activation and the idea that earlier layers influence later layers in complex ways is not new. The problem is to understand why and how these high order interactions between layers are an issue for learning.

Suppose that we are minimizing a loss function $f(w)$ using gradient descent, where w are the weights. We consider what happens when we take a step in the direction of the gradients from the current weights w_0 . Of course, we don't know the form of function f yet, however, recall the **Weierstrass Approximation Theorem** in R^n which states that every real-valued continuous function in a closed n dimensional subspace can be uniformly approximated as closely as desired by a **polynomial function**. Note that this does not contradict our context here, as it is generally assumed NNs do provide differentiable, well-behaved functions (as gradients are backpropagated through the layers), so it is a reasonable assumption to consider f continuous (as a consequence). To approximate f, usually a second order **Taylor** polynomial expansion is taken (around the current weights w_0): $f(w) \approx f(w_0) + (w - w_0)^T g + \frac{1}{2}(w - w_0)^T H(w - w_0)$ where g and H are the gradient and Hessian matrix of $f(w)$ at w_0 . When we take a step in the direction of the gradient with size ϵ , the loss function becomes :

$$f(w_0 - \epsilon g) \approx f(w_0) - \epsilon g^T g + \tfrac{1}{2}\epsilon^2 g^T H g \quad (*)$$

This is actually a well known formula in convex optimization as it was used in old papers that were not even Deep Learning related [21]. Notice the third term on the right-hand side of the equation: $\frac{1}{2}\epsilon^2 g^T H g$. If this term was $\mathbf{0}$, the loss function would strictly decrease. This happens when the model has no second-order terms - i.e. when it is a strictly linear model. On the other hand, if this term was sufficiently large, it may exceed the absolute value of $\epsilon g^T g$ so the loss might actually increase. This happens when the **second-order** effects outweigh the **first-order** effects. It is regarded that the last term (the one that contains the Hessian and the gradient) represents the effect of the curvature of the loss function [14]. If the **curvature is small**, the **gradient is mostly constant**, meaning we can take a large step-size ϵ and decrease the loss. On the other hand, when the curvature is large, the gradient changes quickly, meaning a large step-size poses a risk of increasing the loss. In the worst case, the gradient is the eigenvector of H with the largest eigenvalue.

The mathematical background presented above was needed as solutions to the vanishing gradient problem do refer to this problem of **conditioning**, to be more precise, the *ill-conditioning* of the Hessian matrix. The only way to ensure that the curvature does not cause the loss to increase is by decreasing the step-size ϵ -making it extremely small. Using a very small learning rate (lr) with **VGG 38** is just not practical, though. Of course, we analyzed what happens only for second-order effects, but this gives tremendous insight into the behavior of deeper neural networks. We are confident that this translates to higher order effects caused by very deep NN architectures, that need higher order **Taylor** series for a good approximation, where there are third, fourth, and even higher-degree effects between the weights. This means that gradient updates can be even more unpredictable because the higher order interactions complicate the gradient update, and the only way to ensure that these effects do not adversely affect the loss is to make the step-size extremely small, **or to incorporate techniques that allow higher learning rates to be used**.

3. Background Literature

3.1. **Batch Normalization.**

Batch normalization (BN) [10] is a technique to normalize activations in intermediate layers of deep neural networks. BN has become a staple in state-of-the-art models because of its tendency to speed up training and improve performance. The main idea is to normalize the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. It is empirically proved that this solves the **vanishing gradient** problem in very deep CNNs possibly due to more controlled activations and well-behaved gradient updates.

To our understanding, the **motivation** comes from the fact that we always knew input normalization is needed for a healthy learning; if the input layer is benefiting from it, why not do the same for the values in the hidden layers, as the distribution of each layer's inputs changes all the time during training? We normalize the input layer by **adjusting** and **scaling** the activations. This way, it reduces the amount by what the hidden unit values **shift around**. The authors refer to this phenomenon as **internal covariate shift**.

However, after this shift/scale of activation outputs by some randomly initialized parameters, the weights in the next layer are no longer optimal. To address this problem, the authors introduced, for each activation, **a pair of trainable parameters** γ, β, which scale and shift the normalized value: $y = \gamma x + \beta$. BN lets the gradient descent do the denormalization by changing only these **two weights** (γ, β) for each activation, instead of losing the stability of the network by changing **all the weights**.

It comes as a consequence that BN allows each layer of a network to learn by itself a little bit more independently of other layers, but, intuitively, **why does that help?** Recall formula $(*)$. With BN, the mean and variance of the activations of each layer are independent by the values themselves, **they are not decided by complex interactions between multiple layers, but rather by two simple parameters**. This means that the **magnitude**

of the higher order interactions are likely to be **suppressed**, allowing **larger learning rates** to be used.

Among other **advantages** of BN, it helps bypass local minima and makes the training more resilient to weights initialization and the authors show that it is invariant to parameter scale. It is a form of **regularization**, the networks with BN usually do not require Dropout [17]. BN also enabled the training of deep neural networks with sigmoid activations that were previously deemed too difficult to train due to the vanishing gradient problem.

Nevertheless, as any method in machine learning, there are some **limitations**. Convergence is necessary for generalizing well, but if a network converges without normalization, BN does not add further improvement in generalization [1]. It was also showed that BN strongly depends on how the batches are constructed during training, and it may not converge to a desired solution if the statistics on the batch are not close to the statistics over the whole dataset. Moreover, it was shown [13] that BN fails/overfits when the mini-batch size is 1 and are in general, very sensitive to the mini-batch size.

3.2. Residual Neural Networks.

Residual Neural Networks (Resnets) [6] are a family of neural networks with a specific common trait: they use skip connections in their architecture to fit the input from the previous layer to the next layer without any modification of the input. Resnets solve the vanishing gradient problem by letting the gradients flow directly through the skip connections backwards from later layers to initial filters. Other problems that these NNs solve is the **shattered gradients** problem in which we get gradients that are not correlated within samples in any way.

The **motivation** behind the authors' work is the fact that adding multiple layers to an already defined NN architecture shouldn't come at any performance cost if the layers that we add are identity mappings - that don't do anything. It should be easy for a NN (which is a good function approximator) to learn the identity map $f(x) = x$. The authors also took inspiration from other sources as Resnet was not the first one to use skip connections. **LSTMs** [8] have a similar mechanism with their parametrized forget gate that controls how much information will flow to the next time step and there is also **Highway Networks** [18] which actually contain Resnets in their solution space and yet they perform no better than them.

It is said that the problem of training very deep CNN models has largely been overcome via carefully constructed initializations and BN, however, architectures incorporating skip-connections such as highway and resnets perform much better than standard feedforward architectures despite BN. **But why**

wasn't BN enough to train very deep models, what is it that these deep residual models do better? In short, when training deep networks there comes a point where an increase in depth causes accuracy to saturate, then degrade rapidly - the **degradation problem** caused by **shattered gradients**. Shattered gradients resemble white noise and cancel each other out, making training more difficult. Shallow networks have unshattered gradients. However, for deeper networks, training them with batch norm leads to shattered gradients, while training them without it leads to the vanishing gradient problem. ResNets help ameliorate both problems, one of the arguments is that they resemble an ensemble of shallow networks.

The authors tested a 152-layered NN for ImageNet classification. It is really impressive that this was **8x** bigger than **VGG** nets, but it does require less computation according to the no of Flops.

Limitations of the Resnet concern a mathematical underpinning of the empirical research. Moreover, a study [22] found out that Resnet and variants of Resnet extremely vulnerable to adversarial examples (or attacks) [5] , which are input examples slightly perturbed with an intention to fool the network to make a wrong classification.

3.3. **Densely connected neural networks.**

Densely connected neural networks (Densenets) [9] extend on the idea of shortcut connections present in Resnets, connecting all the layers directly with each other. In this novel architecture, the input of each layer consists of the feature maps of all earlier layer, and its output is passed to each subsequent layer. The feature maps are aggregated with **depth-concatenation** and not with **summation** using identity mappings like Resnets. These connections form a dense circuit of pathways that allow better gradient-flow, thereby solving the vanishing gradient problem.

A key insight in this architecture is that each layer has direct access to the gradients of the loss function and the original input signal, the model requires fewer layers, as there is no need to learn redundant feature maps, allowing the *collective knowledge* to be **reused** - feature reuse, making the network highly **parameter-efficient**. Fewer and narrower layers means that the model has fewer parameters to learn, making them easier to train. The authors also talk about the importance of variation in input of layers as a result of concatenated feature maps, which prevents the model from over-fitting the training data which makes sense.

The full architecture proposed in the paper makes use of dense blocks and transition blocks. The dense blocks, as we mentioned before, are composed of interconnected dense layers (that here are 1x1 conv + 3x3 conv). A term that is frequently brought up in the paper is the growth rate k that dictates how many channel features are concatenated and fed as input to the next dense layer. Transition blocks are used between dense blocks and use Convs and average pooling for dimensionality reduction.

Densenet models without hyper-parameter tuning are **compared** to **Resnet** with optimal hyper-parameters over the ImageNet dataset and it turns out that the Densenet model has a **significantly lower validation error** than the ResNet model with the **same number of parameters**. Moreover, another experiment showed that a Densenet model with **20M** parameters model yields similar validation error as a 101-layer ResNet with more than **40M** parameters.

Therefore, it seems that Densenet is a **clear improvement** over the previous state-of-the-art Resnet, granted the two architectures have the main concept of **skipping layers in common**, the execution being different. In relationship with **Batch Normalization**, both of them do use it which shows how important this technique still is.

4. Solution Overview

In order to solve the vanishing gradient problem, we have chosen **Batch Normalization** as a first-hand solution. The motivation behind this is the fact that all the solutions from the literature review section had this technique in common, so it comes as natural to apply it.

Implementing BatchNorm is like applying pre-processing but for hidden layers. The idea is to normalize the output coming from a previous hidden layer (likely Conv), restricting the amount by what a hidden unit value can shift around. An idea that we have brought up in the literature review as well is the reduction of the internal covariate shift. **Covariance shift** is directly linked to the different distributions that can appear in the data: if it changes between training data (for example, we train the model on greyscale images) and test data (we test it on RGB images), our algorithm would be, of course, pretty poor; BN tries to solve that.

The algorithm can be seen below, note the two model parameters introduced by BN (γ, β) that help the optimizer undo the normalization if it's a way for it to minimize the loss function. We add these two trainable parameters to each layer, so the normalized output (that has 0 mean and 1 standard deviation) is multiplied by a *standard deviation* parameter γ and add a *mean* parameter β. In practice, restricting the activations of each layer to be strictly 0 mean and unit variance can limit the expressive power of the network. Therefore, in practice, batch normalization allows the network to learn parameters γ and β that can convert the mean and variance to any value that the network desires.

Algorithm 1 Batch Normalization

```
**Input:** Values of X over a mini-batch after a Conv Layer: $x_i, i \in 1, 2, ..., n$.
Parameters to be learnt: $\gamma, \beta$.
$\mu \leftarrow \frac{1}{n} \sum_i^n x_i$ //mini-batch mean
$\sigma^2 \leftarrow \frac{1}{n} \sum_i^n (x_i - \mu)^2$ //mini-batch variance
$\hat{x_i} \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \forall i \in 1, 2, ..., n$ // normalization
$y_i \leftarrow \gamma \hat{x_i} + \beta, \forall i \in 1, 2, ..., n$ // scale and shift
**Output:** $y$
```

It comes as a natural question, though, if we should apply BN before and after activations. We have researched this issue quite thoroughly and there is not a clear definite answer to it. Although the proposed approach in the original paper used BN before activations, many empirical experiments have been conducted by the community [23] and great results are showed applying BN after ReLu activations. However, **our understanding is that BN helps more by reducing the high-order realationships between parameters of different layers than reducing the covariate shift, therefore the order might not really matter**. We will apply BN before activation and possible pooling layer as described by the authors of the original paper.

It is said that BN also **regularizes** the model. The **intuition** for this is that BN adds extra sources of noise so that every layer has to learn to be robust to account for the variations in its inputs: because the data points are randomly chosen to form a minibatch, the standard deviation randomly fluctuates and BN multiplies each hidden unit by such randomly fluctuating standard deviations and also subtracts the randomly fluctuating means of the minibatch data points.

BatchNorm will likely solve the problem, however, to get even more improvement in the performance of the model, we also chose to implement **Residual Blocks**.

Implementing **Resnet** is straight-forward: to construct a **skip connection** over a layer that applies transformation F to an input vector x, we modify the output of the whole block to another map $H(x) = F(x) + x$.

The idea is that **even if** there is **vanishing gradient** for the weight layers, we always still have the identity x to transfer back to earlier layers. The weight layers have to learn this kind of residual mapping: $F(x) = H(x) - x$. Intuitively, if we bypass the input to the first layer of the model to be the output of the last layer of the model, the network should be able to predict whatever function it was learning before with the input added to it.

However, this does **not** work if $F(x)$ changes the dimensions of x, so we need to be careful when implementing it. We have to find a **linear projection** $g = Wx$ such that we preserve the features in x but we reduce its dimensionality. This is addressed in the original paper: if we have to increase the dimensions of x to match $F(x)$, then **padding** is recommended as it does not add any more model parameters and is quite efficient. If we need to reduce it, we can apply any **pooling** transformation or, a **1x1 convolution** with an appropriate stride - this is what the authors use in their experiments. We can view the pooling reduction as a direct scaling without adding extra parameters, however, the **1x1 convolution** approach would work better in theory because, intuitively, this is like a *learnt* scaling.

5. Experiments

We base our experiments using the **CIFAR100** dataset which contains **60k**, 32x32 colored natural images. For all our experiments, we train on **100** epochs, having **47.5k** of the images as training data, **2.5k** as validation data and the rest (**10k**) as test data. Note that at each experiment we shuffle the samples and apply some basic data augmentation: random crop, horizontal flip and gaussian noise on all 3 RGB channels having the *mean* $(0.4914, 0.4822, 0.4465)$ and the *std* $(0.2023, 0.1994, 0.2010)$.

The architectures we are going to use are quite similar at a base level. We have **convolutional processing blocks** that are repeated in the network. Such a block is a cascade of **2** convolutional layers, each followed by a **Leaky ReLu** activation function. We also have 1 to 3 **reduction blocks** that are used to downgrade the units in terms of width and height through pooling. Each such block is a cascade of 2 convolutional layers with an average pooling layer in the middle. All these consecutive blocks and with a flatten and a softmax layer. Denote **VGG 08** being such a NN with 7 convolutional layers + 1 flatten layer and **VGG 38** a NN with 37 convolutional layers + 1 flatten layer.

The motivation behind using leaky ReLu is that it's more unlikely to suffer from vanishing gradients than other non-linear activation functions (sigmoid/tanh). Plus, we use the leaky version because we want to better account for the negative values that come through the layers. Average pooling was used in image classification by previous state-of-the-art models like **Densenets**. By default, we use **Adam** with **lr=0.001** and a batch-size of **100**.

The first experiment is to test the effectiveness of **BatchNorm** to solve the **vanishing gradient** problem. To do that, we have applied BN directly after every convolutional layer and let the **VGG 38** train with the same hyperparameters as before. The mean absolute values of the gradients at each epoch

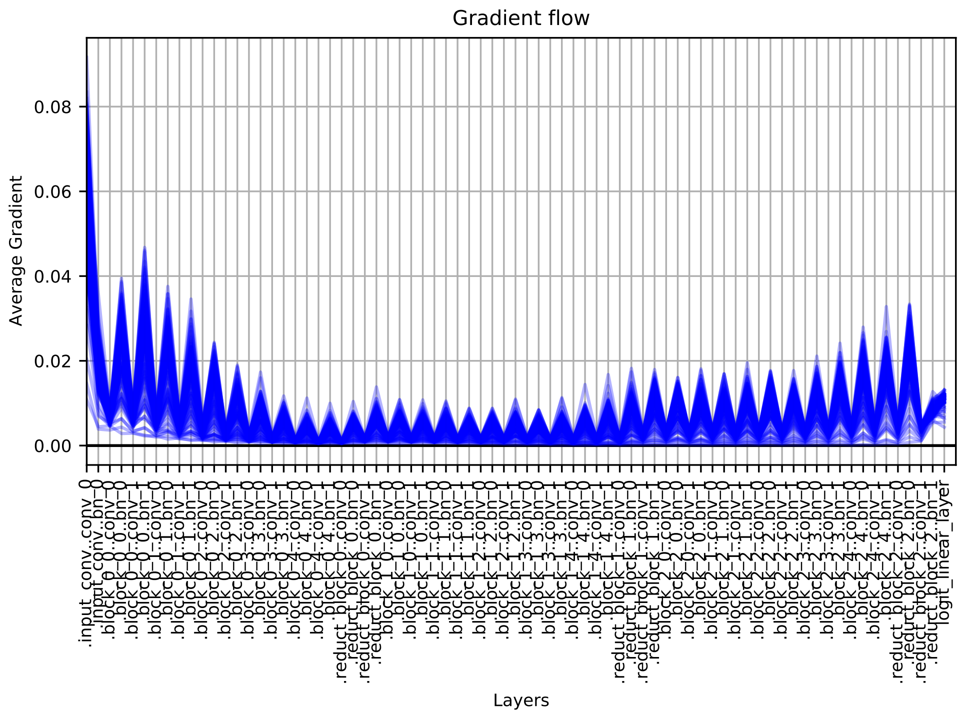


FIGURE 3. Mean abosolute values of gradients w.r.t model parameters at each epoch for VGG 38 with BN.

are displayed in **Figure 3**. We have also applied BN to the **VGG 08** model to see if it improves the results even for more shallow networks and it did (about 4%). All these results with the accuracy can be seen in **Table 1**. Note that for all the experiments, we have conducted tests on different seeds (for the initial random weights initialization) to check the robustness of the results and solidy our claims. We used 5 different seeds (0, 100, 550, 1000, 40000) and we recorded a standard deviation of the accuracy results of 0.499.

It is clear (Figure 3) that BN **solves** the main problem we are dealing with in this paper. However, the performance at the moment of **VGG 38** does not justify its complexity as **VGG 08** still has **similar performance with less number of model parameters**. That's why we now investigate if we can improve the performance with an intuitive change in hyper-parameters.

As we noted multiple times in this paper, BN allows training with higher learning rates, so this is the first thing we are going to try out. We try

0.01 and 0.1, however, the results are not that successful (Table 1), **0.001** still seems to be the appropriate learning rate to be used. We have chosen these learning rates because successful models from the literature that use BN typically choose higher learning rates [6] [9], however, they do use a **decaying factor**, so that might explain why it doesn't work that well in our case.

We have also tried different batch sizes, as we have stated in the literature review that this is a factor that can influence BN in a big way. We choose to experiment with **256** as it was used in other studies [6]. We have also tried **512** because it is more time efficient to use large batch sizes and that also approximates the **gradient** of the whole dataset a little bit better. The computational time decreases by at least 41.66% when we change the batch size from **100** to **256** or **512**, which is impressive, that's why we move on from using **100** as batch size for the next experiments.

The original paper that introduces BatchNorm claims that it solves the training issues even with **sigmoid**-like activation functions (which are notorious for the vanishing gradient problem) and this seems to be universally accepted by the community. We tested that out to confirm it (Table 1, Figure 4).

Results so far with BN still can't yet justify the need of a deeper model (the test accuracies are similar between VGG 38 and VGG 08), so we conduct experiments with residual blocks added into the models. Firstly, we wanted to try out a default version with skip connections but without BN just to prove that residual blocks alone can solve the vanishing gradient problem and indeed, our experiment was successful (Table 1 - VGG 38 Resnet).

Model	*Batchsize*	*Lr*	*Weightdecay*	*Testacc*	*Trainacc*
VGG08 Baseline	*100*	*0.001*	*0*	*49.95%*	*55.24%*
VGG08 BN	100	0.001	0	53.89%	59.65%
VGG08 BN + Resnet	100	0.001	0	54.45%	60.74%
VGG38 baseline	100	0.001	0	1%	1%
VGG38 BN	100	0.001	0	46.78%	54.20%
VGG38 BN	100	0.01	0	44.14%	48.33%
VGG38 BN	100	0.1	0	22.46%	24.86%
VGG38 BN	256	0.001	0	47.22%	58.05%
VGG38 BN	512	0.001	0	46.49%	60.43%
VGG38 Sigmoid BN	512	0.001	0	26.88%	28.29%
VGG38 Resnet (only)	100	0.001	0	44.77%	52.20%
VGG38 BN + Resnet	512	0.001	0	58.84%	78.65%
VGG38 BN + Resnet	512	0.1	0.0001	30.22%	29.70%
VGG38 BN + Resnet	256	0.01	0.0006	57.67%	72.81%
VGG38 BN + Resnet	256	0.01	0.0001	58.25%	67.50%
VGG38 BN + Resnet	**256**	**0.001**	**0.0001**	**61.81** %	**85.21%**

TABLE 1. VGG08, VGG38 models - varying hyper-parameters. Statistical error: +/-0.49.

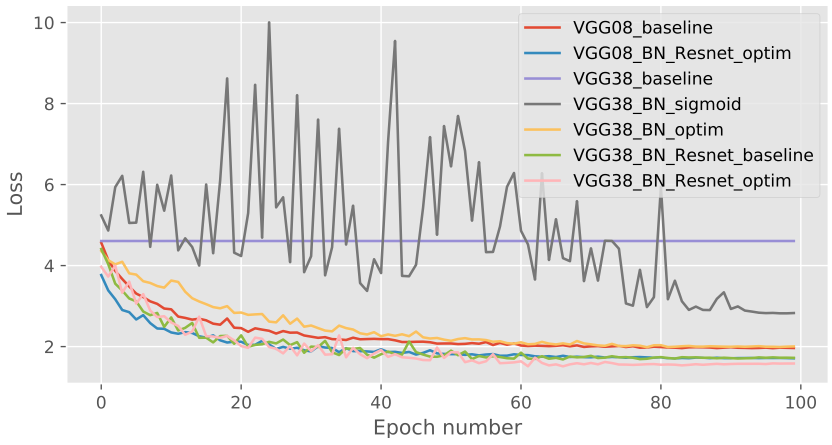


FIGURE 4. Validation loss during training for different models tested.

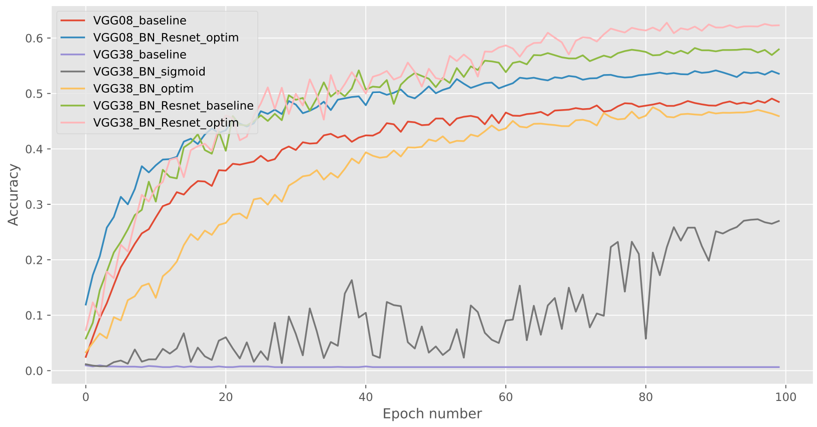


FIGURE 5. Validation accuracy during training for different models tested.

However, for the next experiments, we used **Resnet + BN** as it was recommended in the original paper [6]. We first tested it on VGG 08 to see if it adds any improvement to the previous version and indeed, the test accuracy is slightly higher. However, when training VGG 38 with BN and Residual Blocks, we got much higher results. Nevertheless, it was clear (Table 1) that these models were overfitting on the training data (huge generalization gap), so we tried adding L2 regularization - varying weight penalty, which also used in [6]. This has helped, the training evolution summaries can be observed in Figure 4 and 5.

6. Discussion

Looking at **Figure 3**, the gradients seem healthy as they do not come really close to **0** and the scale is big enough, BN solved the vanishing gradient problem. Note that we expect small gradients for this problem, however, the scale from **Figure 2** was in the $O(10^{-3})$ region and now it is in $O(10^{-1})$. This should mean that the network is indeed learning and the gradient flow provides meaningful signal backwards to the input layers and indeed, it achieves an accuracy of **46.78%**, similar to what **baseline VGG 08** got. Note that there is a **zig-zagg** phenomenon that can be observed, this also happens when training NNs with sigmoid activations (not zero-centered) - the data coming into a neuron is always positive, then the gradient on the weights (during backprop) will become either all be positive or all negative - this leads to zig-zagg dynamics in the gradient updates for the weights. We suspect a similar cause is in our case.

It seems **256** as batch size gives the best test accuracy, which is not surprising as other studies like [6] for image classification also use that.

Sigmoid as activation function for the hidden units lead to a **noisy** behaviour in training (fig. 4&5), however, the model does somewhat learn and achieves an accuracy of **26.88%** on test, which is not great but **proves the hypothesis that BN makes even deep networks with sigmoid work**.

Using **Residual Blocks** and **BN** for **VGG 08** lead to an about **1%** increase in acc, which is not really justifiable. However, when training **VGG 38** with the same architecture, we can begin to see why **deeper is better**, the best model achieving over 60% accuracy on test and 85% on train (**Table 1**), granted it overfits much harder than **VGG 08**. The architecture of this best model can be seen in **Figure 6**, note that we display the **convolutional processing** and **reduction blocks** that we mentioned in the previous section, there are such **5** processing blocks followed by **1** reduction block repeated **3** times in a **VGG 38** model.

Note that we have chosen to implement the linear projection to make the identity smaller as a **1x1 conv** with appropriate stride. We have done this because using max-pooling instead lead to less impressive results, moreover, this was the recommended way in [6] and we personally believe this adaptive *learnt scaling* is better than an absolute one.

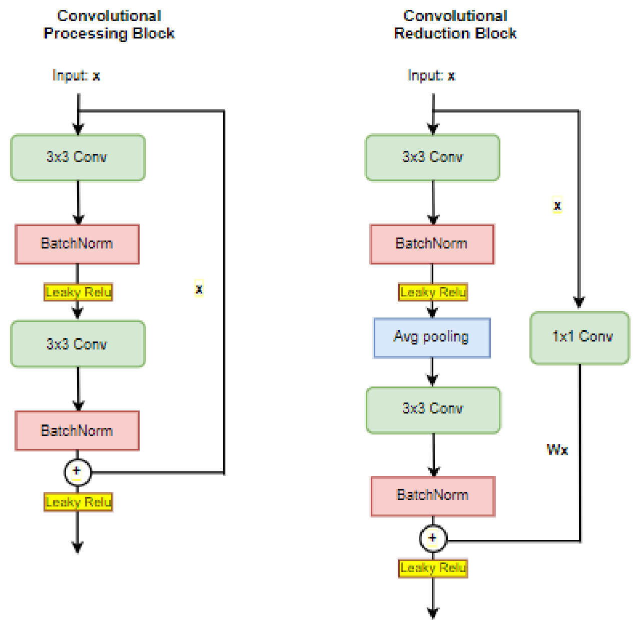


FIGURE 6. Building blocks for our best model VGG38 BN+Resnet

7. Conclusions and Further Work

Very deep CNNs are responsible for the recent *quantum leaps* in AI, not only in computer vision but also in Reinforcement Learning, for example, AlphaGo [15]. Training such models is a serious problem and different techniques need to be applied to get very good results. Simply stacking convolutional layers does not work for VGG models as we have seen in Figure 2. However, this can be solved by either BatchNorm or Resnets. Combining these two together is much better than using them separately. These methods can also improve more shallow networks (VGG 08); however, the very deep models completely outperform shallower methods in this case (over 7% increase), which is expected. However, there is still room for improvement regarding the hyper-parameters, furthermore, we use only 3x3 convolutions which can be very restrictive. Models similar to ours, like *Wide ResNet*, *ResNeXt*, [24] achieve much higher accuracy: 79.5%, 82.3%, but they also have much more **trainable parameters**, further experimentation on that can be beneficial.

References

[1] Bjorck, N., Gomes, C. P., Selman, B., and Weinberger, K. Q. Understanding batch normalization. In *Advances in Neural Information Processing Systems* (2018), pp. 7694–7705.

[2] Ciregan, D., Meier, U., and Schmidhuber, J. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition* (2012), IEEE, pp. 3642–3649.

[3] Cybenko, G. Mathematics of control. *Signals and Systems 2* (1989), 303.

[4] Eigen, D., Rolfe, J., Fergus, R., and LeCun, Y. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847* (2013).

[5] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[6] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[7] Hochreiter, S. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München 91*, 1 (1991).

[8] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[9] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 4700–4708.

[10] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (2015), F. R. Bach and D. M. Blei, Eds., vol. 37 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 448–456.

[11] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *University of Toronto* (2009).

[12] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*, 6 (2012), 84–90.

[13] Lian, X., and Liu, J. Revisit batch normalization: New understanding and refinement via composition optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics* (2019), pp. 3254–3263.

[14] Martens, J. Deep learning via hessian-free optimization. In *ICML* (2010), vol. 27, pp. 735–742.

[15] 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.

[16] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[17] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*, 1 (2014), 1929–1958.

[18] Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks. *CoRR abs/1505.00387* (2015).

[19] Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. In *Advances in neural information processing systems* (2015), pp. 2377–2385.

[20] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9.

[21] Thacker, W. C. The role of the hessian matrix in fitting models to measurements. *Journal of Geophysical Research: Oceans 94*, C5 (1989), 6177–6196.

[22] Wu, D., Wang, Y., Xia, S.-T., Bailey, J., and Ma, X. Skip connections matter: On the transferability of adversarial examples generated with resnets. unpublished, 2020.

[23] XalosXandrez. Batch normalization before or after relu? https://www.reddit.com/r/MachineLearning/comments/67gonq/dbatchnormalizationbeforeorafterrelu/. Published: 2017-04-25.

[24] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 1492–1500.

[25] Yu, D., Seltzer, M. L., Li, J., Huang, J.-T., and Seide, F. Feature learning in deep neural networks-studies on speech recognition tasks. *arXiv preprint arXiv:1301.3605* (2013).

The University of Edinburgh, School of Informatics, 10 Crichton St, Newington, Edinburgh EH8 9AB, United Kingdom
Email address: `T.V.Pricope@sms.ed.ac.uk`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.02

PERFORMANCE BENCHMARKING FOR NOSQL DATABASE MANAGEMENT SYSTEMS

CAMELIA-FLORINA ANDOR

Abstract. NoSQL database management systems are very diverse and are known to evolve very fast. With so many NoSQL database options available nowadays, it is getting harder to make the right choice for certain use cases. Also, even for a given NoSQL database management system, performance may vary significantly between versions. Database performance benchmarking shows the actual performance for different scenarios on different hardware configurations in a straightforward and precise manner. This paper presents a NoSQL database performance study in which two of the most popular NoSQL database management systems (MongoDB and Apache Cassandra) are compared, and the analyzed metric is throughput. Results show that Apache Cassandra outperformes MongoDB in an update heavy scenario only when the number of operations is high. Also, for a read intensive scenario, Apache Cassandra outperformes MongoDB only when both number of operations and degree of parallelism are high.

1. Introduction

Big data came along with big challenges regarding how to store, manage and distribute a huge quantity of data, generated in short time and from diverse sources. NoSQL databases were the response to these challenges, specialized in solving specific big data problems. NoSQL database management systems are diverse and it is harder to choose the best fit for specific use cases than it is in the case of relational database management systems. Of course, relational database management systems present differences from one product to another, but those differences are less significant than the differences between NoSQL database management systems. Relational database management systems are based on the relational model, and the query language used is SQL,

Received by the editors: 18 April 2021.

2010 *Mathematics Subject Classification.* 68P15, 68P99.

1998 *CR Categories and Descriptors.* H.2.1 [**Database Management**]: Logical design – *Data models*; H.2.4 [**Database Management**]: Systems – *Distributed databases, Parallel databases*.

Key words and phrases. NoSQL database, performance benchmarking, MongoDB, Cassandra.

but NoSQL database management systems do not share the same data model or query language. It's quite common to see a different query language for each NoSQL implementation, and a specific data model, usually other than relational. Figuring out which NoSQL database management system fits best your use case is far more difficult than it seems at first, and it requires a thorough study of several NoSQL technical documentations and fine tuning. The hardware configuration is also important, and performance benchmarking is a good solution in this case. As NoSQL database management systems have a fast evolution, observing how their performance evolves between versions can offer meaningful knowledge.

This paper presents a performance benchmarking study which involves two of the most popular NoSQL database management systems, MongoDB (version 4.4.2) and Apache Cassandra (version 3.11.9). The benchmarking experiments were performed with YCSB, a free and open source benchmarking framework, which was also used to generate the data sets involved in the experiments.

2. Background

2.1. NoSQL Data Models.

2.1. NoSQL Data Models. The NoSQL data models considered for this case study are column-family and document, which are two of the four main NoSQL data models. The remaining NoSQL data models are key-value and graph.

The key-value model is the least complex model, and NoSQL database management systems that use it have a very limited query language, but very fast operations. Both column-family and document model derive from the key-value model.

The graph model is the most complex NoSQL data model, and while it's a good fit for highly interconnected data, it has some drawbacks regarding horizontal scalability. NoSQL database management systems that use the graph data model have expressive query languages and constant read performance.

The document data model has much more in common with the column-family data model than it has with the other two main NoSQL data models. Both document and column-family data models support high availability, horizontal scalability, flexible schema and reasonable expressive query languages. Yet document NoSQL database management systems tend to offer more schema flexibility and richer query languages than column-family NoSQL database management systems. Also, column-family NoSQL database management systems tend to support faster write operations, even at scale.

2.2. NoSQL database management systems. MongoDB[8] and Cassandra[2] are two open source NoSQL database management systems. MongoDB is based on the document data model and Cassandra is based on the column-family data model. MongoDB has a flexible schema, that can be easily modified, as the application's requirements evolve. It is more difficult to adapt the database schema in Cassandra, where data modeling is query driven (the application's queries must be known from the start). When designing database schema in Cassandra, the structure of the tables must optimize the application's queries. It's not uncommon to have several versions of a table, with minor structure changes in order to optimize different queries on the same data (data duplication is common in NoSQL databases). From the query language perspective, MongoDB is by far superior to Cassandra. While Cassandra has a query language somehow similar to SQL (but far more limited), MongoDB's query language is JavaScript based, rich and expressive. Also, MongoDB has support for many types of secondary indexes (text, geospatial, hidden, etc.), that are not available in Cassandra. High availability and horizontal scalability are well supported in MongoDB and Cassandra, but the distribution models are different.

2.3. **NoSQL performance benchmarking.** Performance benchmarking is quite handy when working with NoSQL databases. There are benchmarking tools that can be used only for a specific database management system (DBMS), like *cassandra-stress*[11] for Cassandra or *cbc-pillowfight*[10] for Couchbase. These types of tools are useful to test a given NoSQL DBMS in certain scenarios, but they don't help much when a comparison between several NoSQL DBMSs is the goal of the benchmarking experiment. For a fair comparison between several NoSQL DBMSs, a benchmarking tool which has support for all options considered is necessary. Unfortunately, there are not many benchmarking tools of this kind available in the open source section. *YCSB*[4] is an open source benchmarking framework aimed at cloud systems and NoSQL DBMSs. YCSB is a popular benchmarking framework, relatively easy to understand and use. It supports many NoSQL DBMSs and can be used on both Windows and Linux operating systems. Also, YCSB can be used to generate both the data set involved in testing and the database requests according to the chosen workload type. YCSB was also used by *MongoDB Inc.* for performance testing of MongoDB, see [12]. Other organizations and researchers used YCSB as well, for benchmarking NoSQL DBMSs. All NoSQL benchmarking experiments presented in [3], [5], [6] and [7] used YCSB as benchmarking tool. Other NoSQL benchmarking tools emerge, like NoSQLBench[9] (used by DataStax), but are still in early stages of development.

3. Case study

The case study presented in [1] analyzes the database performance metric called *throughput*, measured in number of operations/second. Other important and useful database performance metrics are *latency* (measured in number of microseconds/operation) and *total runtime* (the time necessary to run a certain number of database operations). The case study presented in this paper also analyzes the *throughput*. As NoSQL DBMSs evolve fast and change a lot from one version to another, it is important to see how those changes affect performance.

3.1. **Experimental setting.** I reproduced the experimental study presented in [1] using newer versions of database management systems and operating system without changing the hardware configuration. That experimental study involved three physical servers with the same hardware configuration. Windows 7 Professional 64-bit was the operating system installed on all servers. YCSB version 0.12.0, MongoDB version 3.4.4 and Apache Cassandra 3.11.0 were each installed on its dedicated server. I followed the same benchmarking methodology and I performed all benchmarking tests in the same order and under the same conditions as those performed in the experimental study presented in [1]. The YCSB, MongoDB and Cassandra versions used in my experimental study were newer. MongoDB version 4.4.2 was installed with default settings and the default storage engine, Wired Tiger. Apache Cassandra version 3.11.9 was installed with default settings and the settings necessary to avoid write timeouts:

- `counter_write_request_timeout_in_ms` set to 100000
- `write_request_timeout_in_ms` set to 100000
- `read_request_timeout_in_ms` set to 50000
- `range_request_timeout_in_ms` set to 100000.

YCSB version 0.17.0 was used to generate the data set and the database requests involved in tests.
Each application involved (Cassandra, MongoDB and YCSB) ran on its own server. The server configuration is as follows:

- OS: Windows 10 Professional 64-bit
- RAM: 16 GB
- CPU: Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 4 cores, 8 logical processors
- HDD: 500 GB.

I used the *YCSB client* to generate a data set having the same size and schema as the one used in the experimental study I reproduced (4 million records, each record made of 10 fields, each field contains a 100 byte string value that was randomly generated). The same predefined YCSB workloads, Workload A (50% read operations, 50% update operations) and Workload B (95% read operations, 5% update operations) were involved, and the asynchronous version of Java Driver was used for both DBMSs. When a benchmarking test is run using YCSB, the workload type, the total number of operations to be executed and the number of client threads must be specified. After the test run, YCSB outputs a file that contains the measured results. For each *workload* considered (Workload A and Workload B), the *number of operations* parameter was set to 1000, 10000, 100000 and 1000000. For each workload and number of operations considered, the *number of client threads* parameter was set to 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512. Each test having a certain combination of values for DBMS, workload, number of operations and number of client threads was repeated three times. I will consider a set of tests all tests run for a combination of DBMS, workload and number of operations. Before and after the execution of each set of tests, database server information was captured. The database server was restarted before the execution of every set of tests. When all tests for the first workload were executed, the data set was deleted and a new data set with the same characteristics (schema and number of records) corresponding to the second workload was generated and loaded into the database.

3.2. **Results.** Each test was repeated three times for every combination of DBMS, workload, number of operations and number of client threads. As a consequence, a throughput average was computed for every combination of DBMS, workload, number of operations and number of client threads. This throughput average was used to create the following charts. A comparison between Cassandra and MongoDB for each combination of number of operations and workload is displayed in the first eight charts (Figures 1 to 8).
In case of Workload A (50% update operations, 50% read operations), Figures 1 and 2 show that MongoDB outperforms Cassandra by far, when the number of operations is relatively small (1000, 10000). When the number of operations is set to 100000, Cassandra's performance is almost as good as MongoDB's, as shown in Figure 3. However, Figure 4 shows that when the number of operations is set to 1000000, Cassandra outperforms MongoDB by far when the number of client threads is greater than or equal to 64.
In case of Workload B (5% update operations, 95% read operations), Figures 5, 6 and 7 show that MongoDB outperforms Cassandra by far when the number of operations is set to 1000, 10000 and 100000 operations. Figure 8 shows that Cassandra outperforms MongoDB only when the number of operations is set to 1000000 and the number of client threads is greater than or equal to 64.

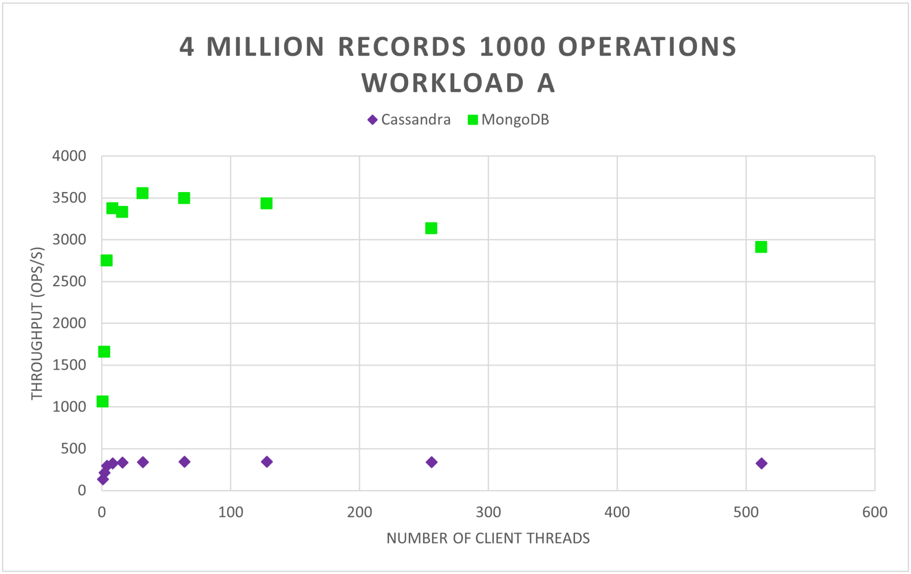


FIGURE 1. 4 Million Records Workload A 1000 Operations - Throughput

When compared to the results presented in [1], it can be observed that in the case of Workload A and number of operations set to 1000000, Cassandra outperformed MongoDB when the number of client threads was greater than or equal to 32, but the throughput does not exceed 45000 operations/second. My experimental study shows that, for the same scenario, while Cassandra outperfomes MongoDB only when the number of client threads is greater than or equal to 64, the throughput is significantly higher, with the maximum value around 76000 operations/second.
In the case of Workload B (5% update operations, 95% read operations), the results presented in [1] show that MongoDB outperformed Cassandra in all scenarios, and achieved high throughput values (between 68000 and 80000 operations/second) when the number of operations was high (100000 and 1000000 operations). My experimental study shows that, in case of Workload B and number of operations set to 1000000, Cassandra outperformes MongoDB when the number of client threads is greater than or equal to 64. Also, Cassandra presents a maximum throughput value around 62000 operations/second. Compared to MongoDB's throughput values presented in [1], MongoDB's throughput values observed in my study do not exceed 40000 operations/second.

The last four charts (Figures 9 to 12) display the individual performance of each DBMS, for each workload. Figure 9 presents the evolution of throughput on Workload A for Cassandra. When the number of operations is low

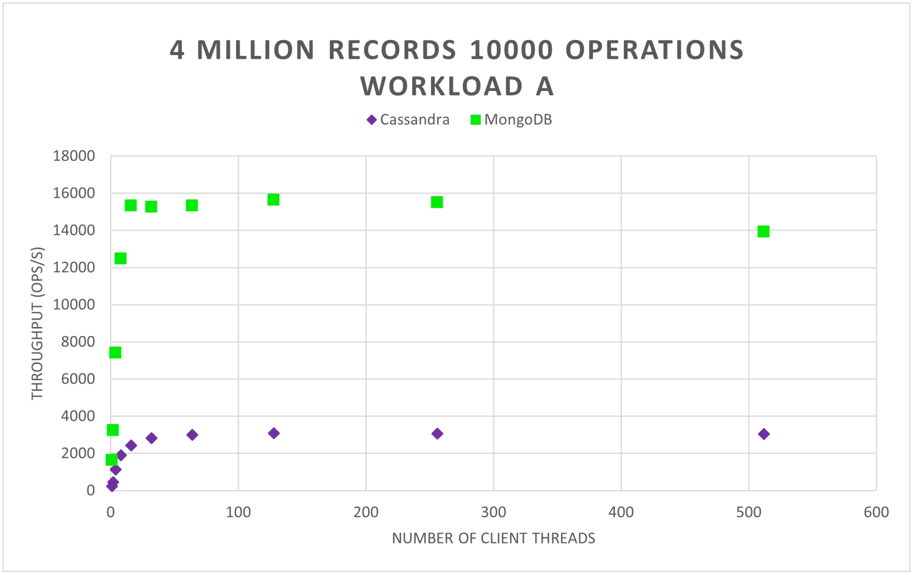


Figure 2. 4 Million Records Workload A 10000 Operations - Throughput

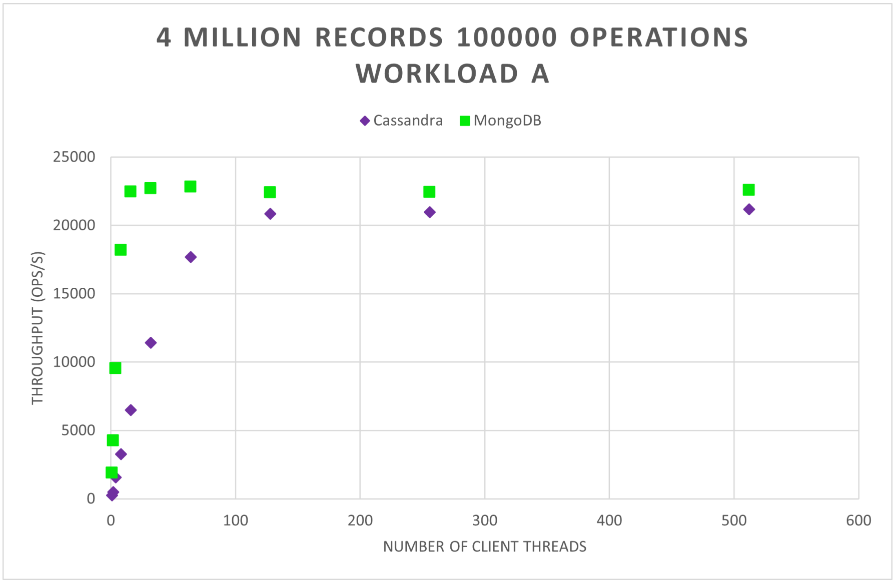


Figure 3. 4 Million Records Workload A 100000 Operations - Throughput

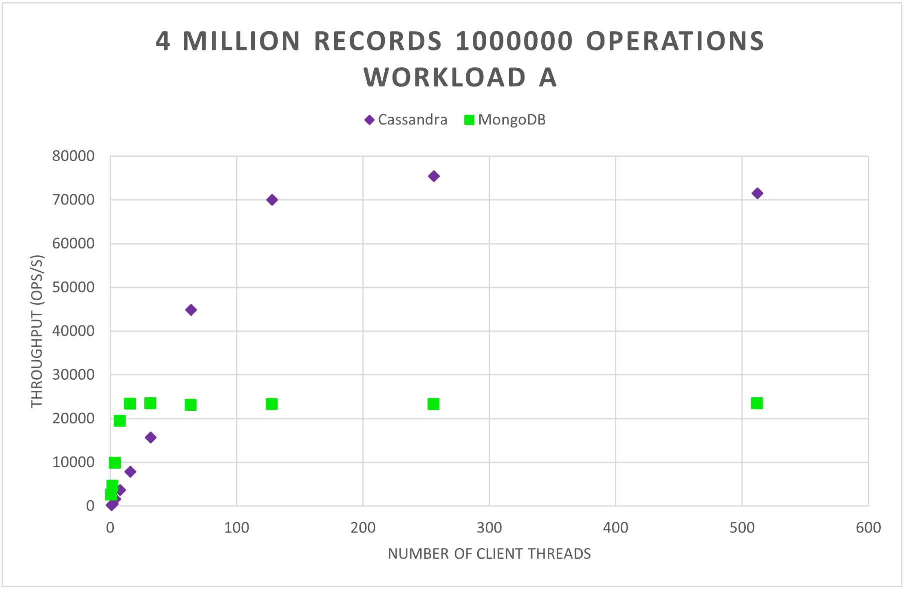


FIGURE 4. 4 Million Records Workload A 1000000 Operations - Throughput

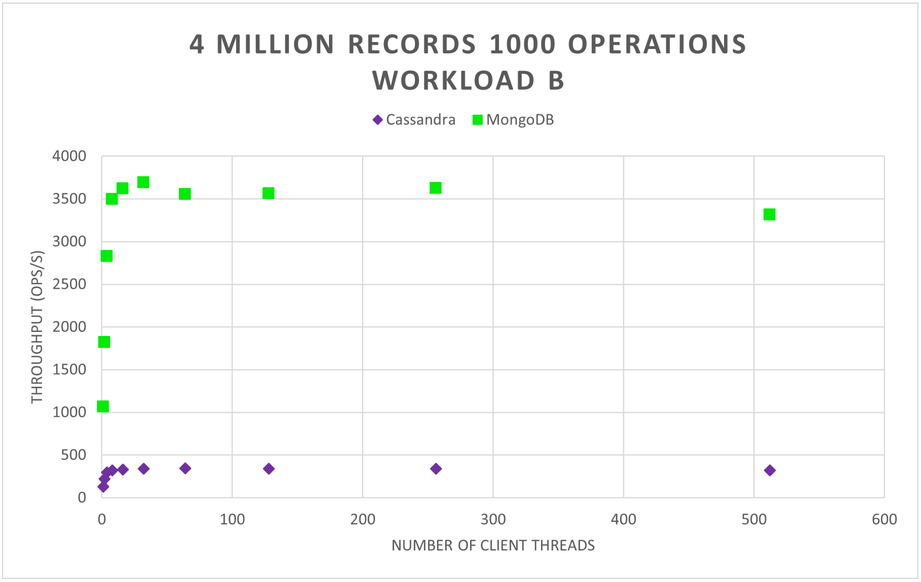


FIGURE 5. 4 Million Records Workload B 1000 Operations - Throughput

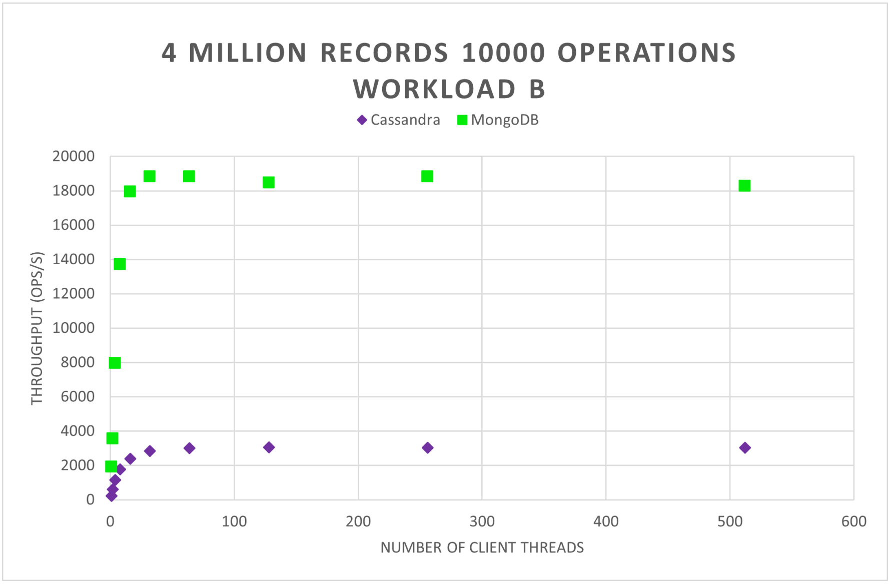


Figure 6. 4 Million Records Workload B 10000 Operations - Throughput

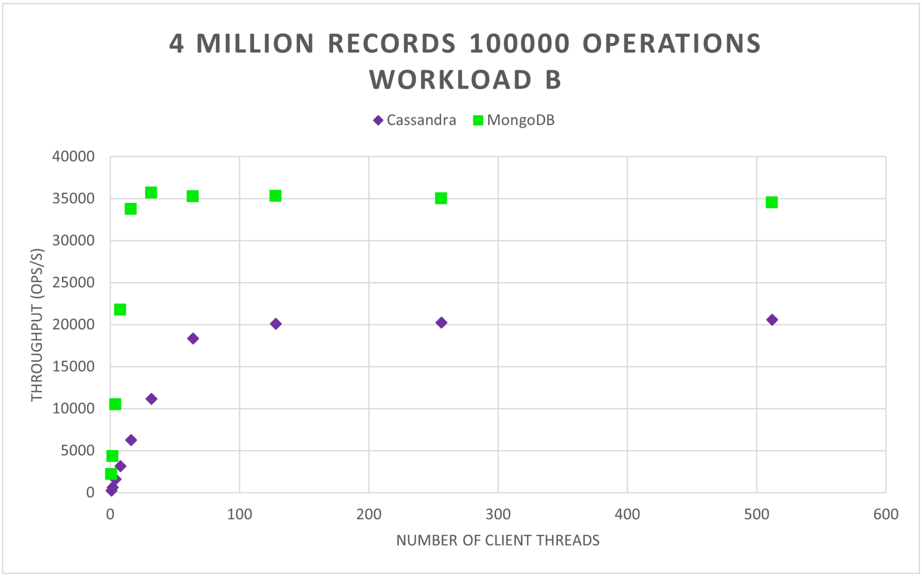


Figure 7. 4 Million Records Workload B 100000 Operations - Throughput

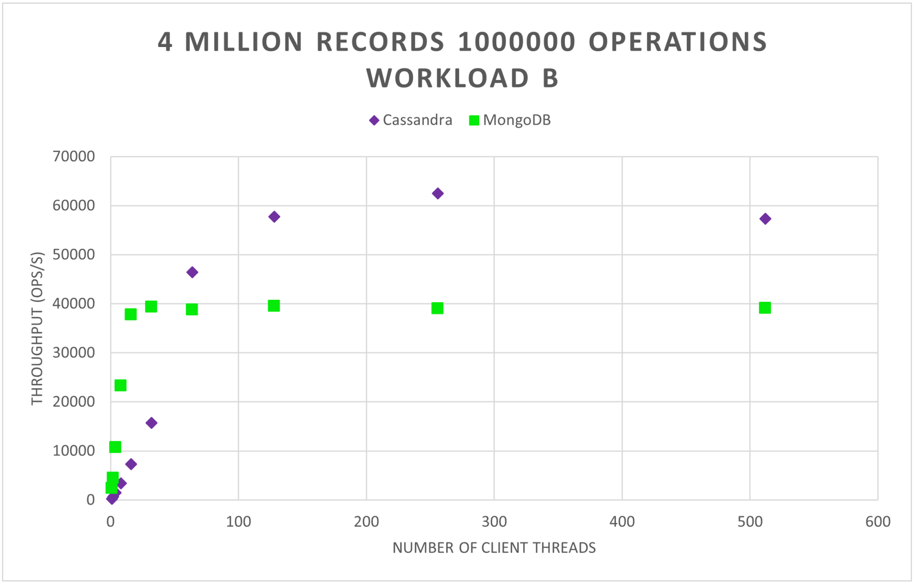


Figure 8. 4 Million Records Workload B 1000000 Operations - Throughput

(1000, 10000 operations), Cassandra's maximum throughput value does not exceed 3100 operations/second. When the number of operations grows at 100000, we can observe a significant increase of throughput (up to 22000 operations/second), especially as the number of client threads grows. The highest throughput values (up to 76000 operations/second) can be observed when the number of operations is set to 1000000, with a slightly decrease when the number of threads is set to 512.

In Figure 10, the evolution of throughput on Workload A for MongoDB is displayed. As the number of operations increases, the MongoDB throughput values increase as well, but remain almost constant when the number of operations is greater than or equal to 100000. The maximum throughput value does not exceed 23500 operations/second.

Figure 11 presents the evolution of throughput on Workload B for Cassandra. Cassandra's throughput patterns observed for Workload A are preserved, but the throughput values observed when the number of operations is set to 1000000 are lower and do not exceed 63000 operations/second.

Figure 12 displays the evolution of throughput on Workload B for MongoDB. The throughput remains the same as observed for Workload A, when the number of operations is set to 1000, but increases when the number of operations is greater than or equal to 10000. When the number of operations is

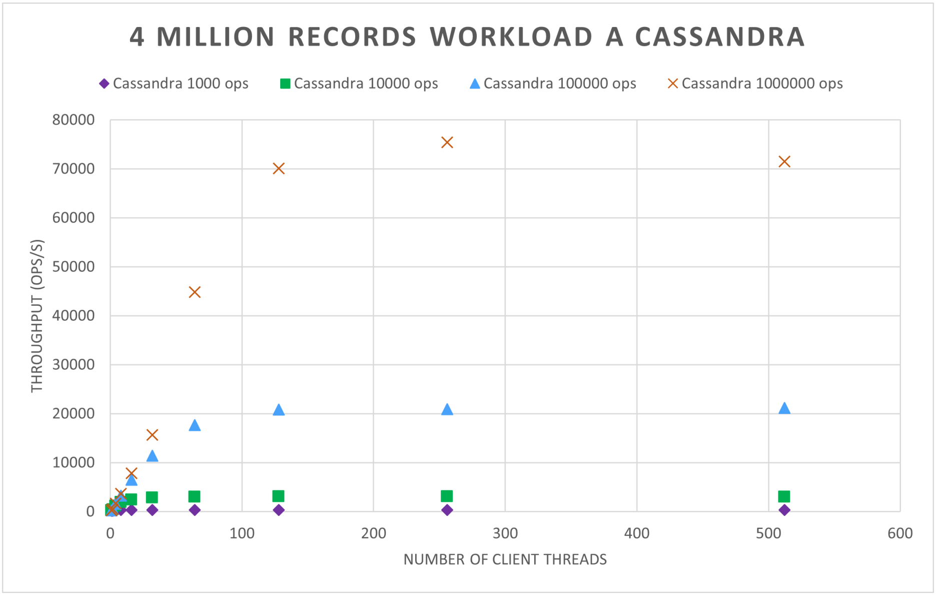


Figure 9. 4 Million Records Workload A Cassandra - Throughput

set to 100000, throughput values rise significantly (compared to those observed for number of operations set to 10000). Throughput values observed when the number of operations is increased at 1000000 are slightly higher than those observed for number of operations set to 100000, but do not exceed 45000 operations/second.

Cassandra version 3.11.9 presents great throughput improvements for both workloads, especially when the number of operations and number of client threads are high. MongoDB version 4.4.2 presents significantly lower throughput values for Workload B, a read intensive workload.

4. Conclusions and future work

NoSQL database management systems have a fast evolution, with significant changes between versions. Database performance benchmarking offers a good overview of how these changes impact application workloads. The experimental study presented in this paper reveals that the newer Cassandra version has important throughput improvements, especially when the number of operations and degree of parallelism are high, for both read and update operations. This is significant, as Cassandra is generally known to offer fast write operations, but not as fast read operations. Also, the newer MongoDB version presents decreased throughput values when the number of operations

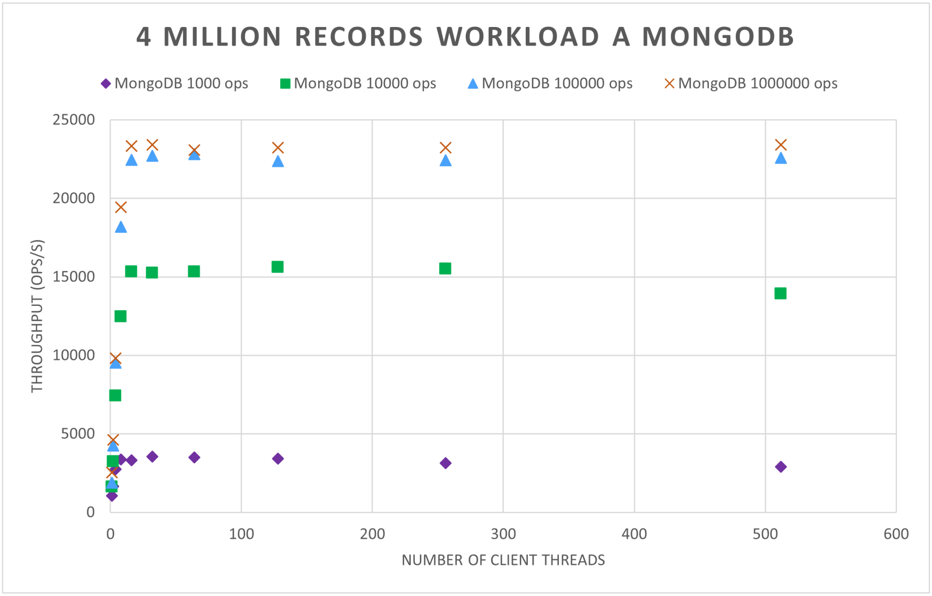


FIGURE 10. 4 Million Records Workload A MongoDB - Throughput

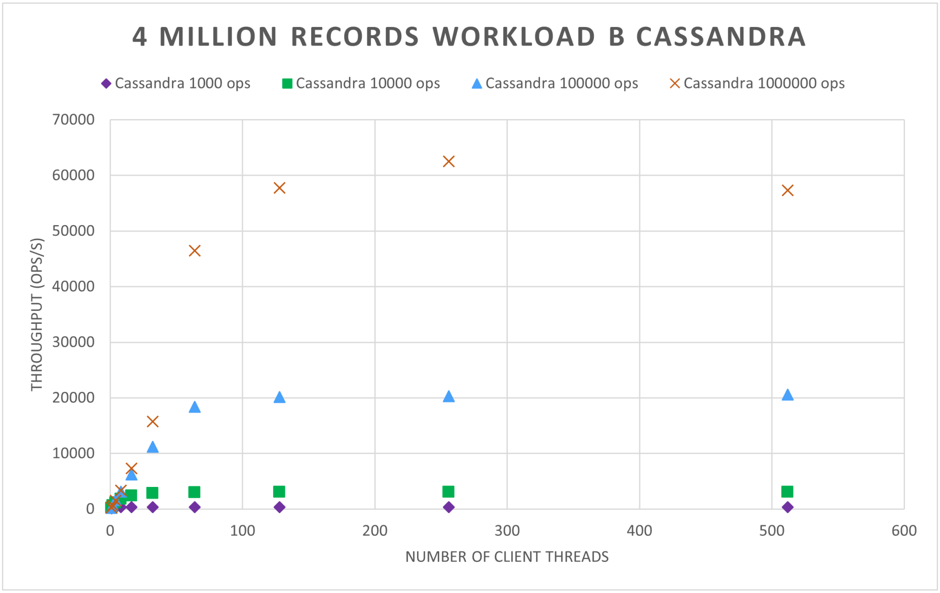


FIGURE 11. 4 Million Records Workload B Cassandra - Throughput

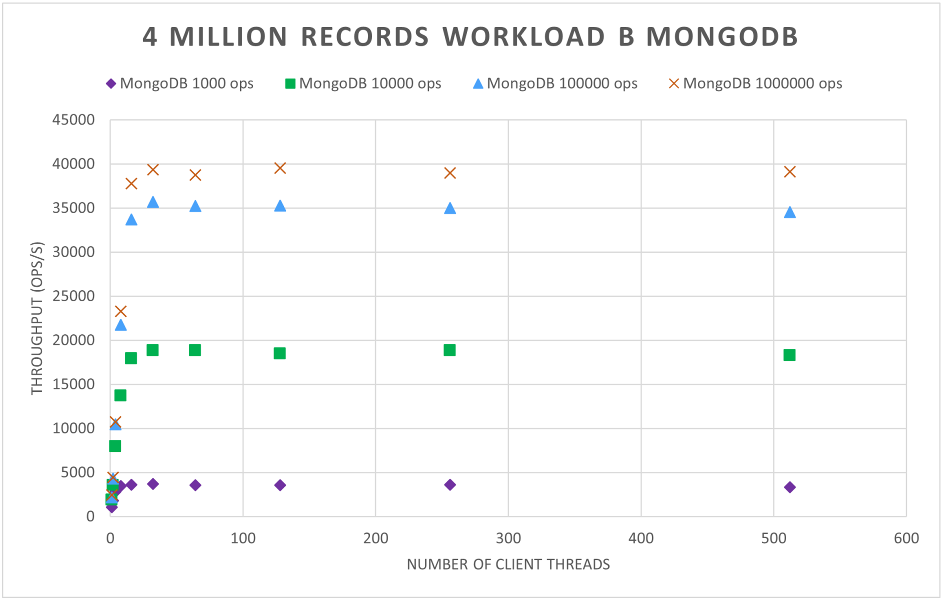


FIGURE 12. 4 Million Records Workload B MongoDB - Throughput

and degree of parallelism are high in case of Workload B (95% read operations, 5% update operations). In this case, we can observe that the newer MongoDB version does not handle a high number of operations with a high degree of parallelism as well as the newer version of Cassandra does. Also, surprisingly, the older MongoDB version (3.4.4) performed better than the newer version (4.4.2) in case of Workload B, under the same experimental conditions. Generally, one would expect performance improvements from newer versions, but it is not always the case. This further shows that monitoring the evolution of performance between versions is important and worth doing.

In the future, I intend to replicate other database performance experimental studies and to analyze other database performance metrics as well. I plan to focus on analyzing the *latency* performance metric, which comes in several variants: average latency, maximum latency, minimum latency, 95th percentile latency and 99th percentile latency. As latency reveals how much time is needed to execute a database operation, it certainly is a performance metric worth analyzing for all application use cases that require very short response times.

Acknowledgments

Parts of this work were supported through the MADECIP project *Disaster Management Research Infrastructure Based on HPC*. This project was granted to Babeş-Bolyai University, its funding being provided by the Sectoral Operational Programme *Increase of Economic Competitiveness*, Priority Axis 2, co-financed by the European Union through the European Regional Development Fund *Investments in Your Future* (POSCEE COD SMIS CSNR 488061862).

References

[1] C.-F. Andor and B. Pârv. NoSQL Database Performance Benchmarking - A Case Study. *Studia Informatica*, LXIII(1):80–93, 2018.

[2] Apache Cassandra. `http://cassandra.apache.org/`. Accessed: 2021-02-14.

[3] Performance Analysis: Benchmarking a NoSQL Database on Bare-Metal and Virtualized Public Cloud - Aerospike NoSQL Database on Internap Bare Metal, Amazon EC2 and Rackspace Cloud. `http://pages.aerospike.com/rs/229-XUE-318/images/Internap_CloudSpectatorAerospike.pdf`. Accessed: 2021-03-24.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[5] Fixstars. GridDB and Cassandra Performance and Scalability. A YCSB Performance Comparison on Microsoft Azure. Technical report, Fixstars Solutions, 2016.

[6] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla. Performance Evaluation of NoSQL Databases. *EPEW 2014: Computer Performance Engineering, Lecture Notes in Computer Science*, 8721:16–29, 2014.

[7] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser. Performance Evaluation of NoSQL Databases: A Case Study. *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, pages 5–10, 2015.

[8] MongoDB. `https://www.mongodb.com/`. Accessed: 2021-02-14.

[9] NoSQLBench. `https://github.com/nosqlbench/nosqlbench`. Accessed: 2021-03-24.

[10] Stress Test for Couchbase Client and Cluster. `https://docs.couchbase.com/sdk-api/couchbase-c-client/md_doc_cbc-pillowfight.html`. Accessed: 2021-03-21.

[11] The cassandra-stress tool. `https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/tools/toolsCStress.html`. Accessed: 2021-03-21.

[12] YCSB MongoDB Performance Testing. `https://www.mongodb.com/blog/post/performance-testing-mongodb-30-part-1-throughput-improvements-measured-ycsb`. Accessed: 2021-03-24.

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
Email address: `camelia.andor@ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.03

CONSTRUCTING UNROOTED PHYLOGENETIC TREES WITH REINFORCEMENT LEARNING

PANNA LIPTÁK AND ATTILA KISS

Abstract. With the development of sequencing technologies, more and more amounts of sequence data are available. This poses additional challenges, such as processing them is usually a complex and time-consuming computational task. During the construction of phylogenetic trees, the relationship between the sequences is examined, and an attempt is made to represent the evolutionary relationship. There are several algorithms for this problem, but with the development of computer science, the question arises as to whether new technologies can be exploited in these areas of computational biology.

In the following publication, we investigate whether the reinforced learning model of machine learning can generate accurate phylogenetic trees based on the distance matrix.

1. Introduction

In phylogenetics, the evolutionary relationships among biological entities are examined. These entities can mean species, individuals but also genes. This paper will focus on the relationship between genes. Trying to identify the inheritance and mutation processes is an important challenge. It can help biologists to refine their understanding of how evolution works and by that further develop the models of evolution or a current example of its usefulness: defining relationships can help to see the geographical distribution of certain subspecies/mutations. A phylogenetic tree is a branching diagram that represents the lineage relationships between genes. It reflects how the examined samples evolved from a common ancestor which is in the root of the tree. Each internal node splits apart a single group into two descendant groups. The genes of interest can be found in the leaves of the tree. It is possible to create an unrooted tree; in this case, the ancestral root is not defined, only

Received by the editors: 24 April 2021.

2010 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* code [**Artificial Intelligence**]: Applications and Expert Systems – *Medicine and science*.

Key words and phrases. Bioinformatics, Reinforcement Learning, Machine Learning Algorithms.

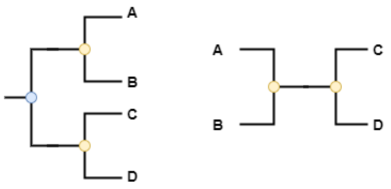


FIGURE 1. Example of a rooted (left) and an unrooted (right) phylogenetic tree. The leaves represent the taxon (A, B, C, D) and the yellow points are the internal nodes. For example, the most recent common ancestor of A and B is at their branch point. The blue point represents the root of the tree, which on the left example is the most recent common ancestor of A, B and C, D nodes.

the relatedness of the leaf nodes. In this paper, we will concentrate on these unrooted trees.

There are many mathematical and algorithmic approaches to construct the tree of given genes. For distance-based algorithms - like UPGMA and Neighbor join – first we need to perform a multiple sequence alignment to compute pairwise distances. This data is stored in the distance matrix. In these approaches, we try to generate a tree where sequences with shorter pairwise distances are closer to each other.

Another type of algorithm is the character-based approach. The Maximum Parsimony method implies an implicit model of evolution. It tries to find a tree with a minimal number of evolutionary steps required to explain the input data. The Maximum Likelihood method uses probability calculations based on a given model of evolution. It considers all possible trees and therefore it is computationally intense, more precisely it is an NP-hard problem [2].

We will discuss the Neighbor join (NJ) [16] algorithm in more detail in the Related Works section. We investigate how an unrooted tree that was constructed based on the distance matrix could be constructed by a reinforcement learning model. Even though NJ is considered a fast algorithm (there are heuristic-accelerated versions, see Related Works), implementing the original method leads to an algorithm of $O(n^3)$ time complexity [20], which is not ideal for large data sets.

Reinforcement learning (RL) is one of the three paradigms of machine learning [22]. It is life-like in the sense that learning is based on experiences. Given an environment and an agent within, the agents goal is to find a series of actions with the maximum reward. The agent receives a reward after every action/step it makes, and the purpose is to learn what decision to make at each state to earn the highest reward at the end of the episode, that is, to determine an optimal policy.

In this paper, we examine how the RL agent can construct an accurate phylogenetic tree by making decisions in the environment described in the Main contribution section. During the training phase, the agent tries to find the most suitable policy which would be used on the test data set to determine the accuracy of the algorithm. Our experiments have shown that with this approach, it is possible to create sufficiently accurate unrooted phylogenetic trees based on the distance matrices.

2. Related Works

The Neighbor join algorithm was first introduced by Saitou and Nei [16] in 1987. The main principle is to minimize the total branch length at each stage; therefore, it is a greedy algorithm. For choosing the two taxa to merge, each step a Q matrix has to be calculated, shown in Formula 1.

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n}(d(i,k)) - \sum_{k=1}^{n}(d(k,j)) \quad (\forall i,j : 1 \leq i < j \leq n) \tag{1}$$

The two sequences with the smallest Q value will be joined. After that, we have to recalculate each taxon's distance to the new u node. In 1988, Studier and Keppler [20] published an improved version of NJ, correcting the way of inferring the distances, as shown in Formula 2 (considering i, j is joined and k is every other taxon). In the RL approach, we eliminate the Q matrix and let the RL agent learn a policy to decide in each state which two taxa to choose for the next join.

$$d(u,k) = \frac{1}{2}[d(i,k) + d(k,j) - d(i,j)] \tag{2}$$

As mentioned before, this algorithm has $O(n^3)$ time complexity [20], which is not ideal for large data sets. There are several works in the literature with heuristic-accelerated versions. In this section, we discuss some of these approaches.

QuickJoin [10] introduces an algorithm with $\Theta(n^2)$ complexity, although the worst-case remains $O(n^3)$. It uses a quad-tree to find the lower bounds of $Q(i,j)$ values, therefore there is no need to calculate the whole Q matrix: the algorithm can skip when the lower bound is higher than one of the known $Q(i,j)$ value. This prunes the search for the minimal $Q(i,j)$ value. For building the quad-tree they use a linear function, which only depends on $d(i,j)$.

RapidNJ [18] uses the observation that in the formula of the Q matrix (shown in Formula 1), the sum is constant in the context of row i. Therefore, it can be used as an upper bound for each row in Q, reducing the search space. This approach has a worst-case $O(n^3)$ complexity, but in the paper, they showed that in practice it has a better performance. NINJA [24] is based on the same idea: dramatically reducing the viewed candidates at each step, but it improves the results of RapidNJ, while still offering $O(n^3)$ worst-case time complexity.

There are two other methods worth mentioning: Relaxed neighbor join [4] and Fast neighbor join [3] to improve the speed by choosing the taxon to join from a subset. In the relaxed version, a transformed distance is calculated for the sequences and two taxa are joined if they are they are the minimum transformed distance of each other. Fast neighbor join offers $O(n^2)$ time complexity, using the visible set as the candidate set for choosing two taxa to join. These two methods are proven fast, but at the cost of the phylogenetic tree they construct provides only an approximate solution if the pairwise distances are not nearly-additive.

These works are just some of the more important milestones, but it also shows how important the improvement of the time complexity of the NJ is in phylogenetics. With the development of artificial intelligence in recent years, there has been a tendency to take advantage of the opportunities offered by machine learning in other fields as well, such as phylogenetics. We would like to present some of these approaches.

Works using machine learning for phylogenetic tree construction already exists. In [1], they introduced an approach to the case where the distance matrix is incomplete. By using deep architectures, they could eliminate the need for a molecular clock assumption, representing a real-world occurrence of the problem.

Multiple sequence alignment is also a challenging problem of bioinformatics. In [11] a reinforcement learning-based approach was introduced. They found that the RL approach outperformed (in most cases) other methods, whilst decreasing the computational time. The training process used the Q-learning algorithm. Another solution for this problem uses a deep RL algorithm and a long short-term memory network, introduced in [8]. Their experiments show, that this version not only outperforms canonical multiple sequence aligner tools but other RL approaches too.

In [23] a convolutional network was used to infer the topology of an unrooted tree by classification. They experimented on simulated data sets and the results showed that this model has great potential. It was not only faster than other methods but it was highly accurate and the accuracy of the classification could be further improved by the number of training data sets. The limitation of this approach is the number of sequences, it was constructed to classify the topology of four sequences (quartet topologies).

An example of a more specific use-case of phylogenetic trees was presented in [9]: an inverse reinforcement learning approach to model a cancer cell's genetic evolutionary process. In this method, the optimal policy and the reward function were reverse engineered to reach interpretable biological conclusions.

In this section, the variants and complexity of NJ algorithms were presented, as well as examples of how artificial intelligence has been used in bioinformatics. The question arose as to whether the NJ problem could be solved by reinforcement learning if we consider each joining operation as a step that also influences the possible future operations. Thus, it is up to the RL agent to decide which two taxon's merging will lead to an optimal solution. At the time of writing, to the knowledge of the authors, no attempt has been made to use the RL method to determine the topology of phylogenetic trees.

3. Main contribution

In this section, we introduce the proposed model of reinforcement learning to solve the problem of constructing unrooted phylogenetic trees based on distance matrices.

The scene of learning is the environment where the agent performs actions that cause the state of the environment to change and the next step takes place in this new state. If the agent reaches the predefined stop condition, the episode ends and the process starts all over again. The agent's knowledge of the environment is called observation and along these, it tries to find connections between the decisions and the reward it receives at the end of each episode. The goal of the model is to learn a policy, to be able to make good (rewarding) decisions in any given state of the environment.

3.1. **Environment.**

The input of the proposed model is an $(n+1) \times (n+1)$ $(n > 3)$ dimensional matrix. The first row and column contain the sequence label, which will symbolize the particular taxa in the completed phylogenetic tree. Omitting the elements containing the labels, we get an $n \times n$ dimensional matrix, which is the distance matrix of the sequences.

A distance matrix is symmetric and the diagonal only contains zeros, therefore it is enough to store the upper triangular of it as a vector for the model. Since labels must also be stored for the tree, we create two vectors from every distance matrix. The *state* space $\mathcal{S} = \{s_0, s_1, ..., s_{\frac{n*(n-1)}{2}}\}$ contains the distance values, and the *labels* indicate which two sequence labels belong to that distance. The length of $\mathcal{S}$ is the size of the upper triangular matrix, which is

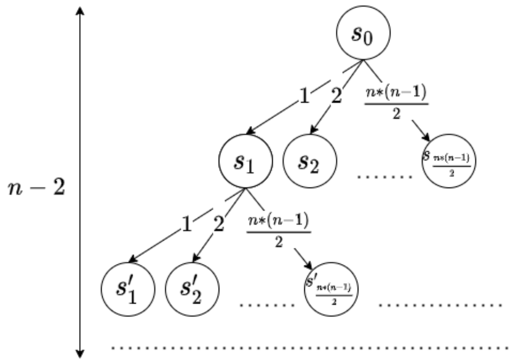


FIGURE 2. Schematic model of the $\mathcal{S}$ state space

$\frac{n*(n-1)}{2}$. If we assume that the i. sequence label is *"i"* ($\forall i : 1 \leq i \leq n$) then the initialization of the state vector should follow Formula (3).

$$p = \sum_{k=1}^{i-2}(n-k) + j - i \quad (\forall i, j : 1 \leq i < j \leq n) \quad s_p = d(i, j) \tag{3}$$

The *action* space $\mathcal{A}$ is defined as choosing a state from $\mathcal{S}$. More specifically, $\mathcal{A} = \{a_1, ..., a_{n-2}\}$ where $1 \leq a_r \leq \frac{n*(n-1)}{2}(\forall 1 \leq r \leq n-2)$. Every action represents a merge of two sequences in the phylogenetic tree, therefore the length of the *action* space is $n-2$: given n sequences we define a step as arbitrary choosing two sequences to merge, after 1 step there are $n-1$ sequences left, after $n-2$ steps there are $n-(n-2) = 2$ left, at this point we do not need to take more steps, because the only choice we have is merging the last two taxon which results in a phylogenetic tree.

After every step, the environment has to be updated according to the chosen action. This consists of updating the distance values, which are done according to the NJ algorithm, and of modifying the labels. A new distance must be calculated for every position where at least one of the corresponding sequences contains a taxon that was chosen for being merged. For example, if in a given step *"i"* and *"j"* are the two taxon that will be merged to a common ancestor, then we have to update every distance that contains either *"i"* or *"j"* and in the corresponding label change *"i"* or *"j"* to *"ij"*. As a result, the constructed tree will contain *(i,j)* (according to the Newick format [13]). Note that after the first step, a merging can consist of an already existing subtree. For this model, branch lengths are not calculated only the topology of the tree but future work will consider extending the model.

As Figure 2 shows, the environment itself is a tree graph with a depth of $n-2$. The tree contains invalid states: if the algorithm starts with n taxon then after the first step there are $n-1$ taxon left and the number of possible taxa pairs for the next step is $\frac{(n-1)(n-2)}{2}$; thus, in this episode we define the following states $s'_{\frac{(n-1)*(n-2)}{2}+1}, s'_{\frac{(n-1)*(n-2)}{2}+2}, ..., s'_{\frac{n*(n-1)}{2}}$ as invalid states.

The RL agent starts in the s_0 state, in each step it chooses a number between 1 and $\frac{n(n-1)}{2}$ and moves to the corresponding state. If the state is valid, then the two nodes can be merged and the affected distances have to be recalculated, hence the change in the state's notation system (s'). If the chosen state was invalid then in this episode the agent failed to construct a phylogenetic tree and the episode ends. Figure 2 is an example where the agent's first choice is 1 and the other not chosen nodes are not further explored in this episode.

3.2. **Reward function.** If the agent chooses an action that corresponds to an invalid state then the episode ends with -1 as the reward. If the RL agent reaches a leaf of the tree with a valid state then the episode also ends and the reward will be proportional to the symmetric difference [15] between the RL constructed tree and the phylogenetic tree calculated by the aforementioned NJ algorithm. Let π be the sequence of decisions made by the agent, where $\pi = (\pi_0, .., \pi_k)$ and $\forall k \in \{1, ..., n-2\}, \pi_k$ is an element of the state space. Furthermore, we define symdiff(t_1, t_2) as the symmetric difference between phylogenetic trees t_1 and t_2. In this case, the *reward* function can be defined as shown in Formula (4) where $tree_c$ is the tree constructed by the RL agent and $tree_{NJ}$ was calculated by the original NJ algorithm.

$$r(\pi_k|\pi_{k-1}, ..., \pi_1) = \begin{cases} -1 & , \pi_k \text{ is an invalid state} \\ \frac{(3n-6)-\text{symdiff}(tree_c, tree_{NJ})}{(3n-6)} * 10 & , k = n-2 \\ 0 & , \text{otherwise} \end{cases} \tag{4}$$

In Formula (4), $3n-6$ is the maximum symmetric difference between two trees with n nodes based on Robinson-Foulds (1981) [15]. With the given function, we transform the symmetric difference, which is in $[0, 3n-6]$ - where 0 means the two trees are identical - to $[0, 10]$ where 10 means that they are identical. This is necessary because the agent tries to maximize the reward, and the goal is to construct phylogenetic trees similar to the tree that was produced by the NJ algorithm. The third case of a step is when the agent chose a valid state but it was not a leaf node. In this case, the model rewards this step with 0, because in this state it cannot determine whether this step was optimal for the phylogenetic tree that will be ideally constructed at the end of the episode.

Algorithm 1: One step in the RL environment

```
Input: action
if episodeEnded then
    $resetEnvironment()$;
end
if state[action] == -1 then
    $episodeEnded = True$;
    $terminate : state, reward = -1$;
else
    $firstNode$ = get first node from labels[action];
    $secondNode$ = get second node from labels[action];
    $newNode = firsNode + secondNode$;
    $newLabels = []$;
    $newDistances = []$;
    for label in labels do
        if label contains firstNode then
            $distFirst = state$[index of label];
            get otherNode from label;
            for otherLabel in labels do
                if otherLabel contains secondNode and otherNode then
                    $distSecond = state$[index of otheLabel];
                end
            end
            $newDistance = \frac{1}{2} * (distFirst + distSecond - state[action])$;
            add newNode + otherNode to newLabels;
            add newDistance to newDistances;
        end
    end
    delete affected states and labels;
    add newLabels to labels and newDistances to state;
    add $-1$ to state and empty label to labels for every invalid state;
    $addNewNodeToNewickTree(firstNode, secondNode)$;
    if there is only one valid state left then
        $addNewNodeToNewickTree$(remaining nodes);
        $constructedTree = treePieces[0]$;
        $episodeEnded = True$;
    end
    if episodeEnded then
        $diff = symmetricDifference(goalTree, constructedTree)$;
        $terminate : state, reward = ((3 * n - 6) - diff)/(3 * n - 6) * 10$;
    else
        $transition : state, reward = 0, discount$;
    end
end
```

Algorithm 1 presents the pseudo-code of a step in the proposed RL environment. The input is the chosen action and at the end of the step the environment either transitions into a new state (see line 40 of Algorithm 1) or the episode ends because of an invalid state (see lines 4-6 of Algorithm 1) or because it finished the construction of the phylogenetic tree (see lines 31-34 of Algorithm 1).

3.3. **Policy.** The policy is the strategy that the agent uses to reach its goal in a given environment. It is the function of the current state of the environment. In this model, the RL agent starts exploring the states with a random policy and the goal is to find the optimal policy by the end of the training phase. Updating the policy by determining the optimum next action is based on the Q values, the values of specific actions. The Q value is the function of the current state s and the action a made in that state. As shown in Formula (5), the Q value consists of the immediate reward received by taking action a in state s and the maximum reward that could be earned in the following state (s'), multiplied by the discount factor (γ). It is a recursive equation as s' will depend on s'' and so on. In an optimal policy, the agent chooses an action with the maximum Q value. The equation for updating the Q values is shown in Formula (6). For calculating the new Q' value for a given state and action the formula contains the learning rate α which determines to what extent should new information overwrite old Q values: set to 0 means only the old information is taken to account, and 1 means only the most recent information.

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a) \tag{5}$$

$$Q'(s_t,a_t) = Q(s_t,a_t) + \alpha * (r(s_t,a_t) + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)) \tag{6}$$

Calculating all the Q values for a given state is complex, both computationally and in many cases storage-wise. In these cases, deep Q learning [12] could be a solution, because it uses a neural network to estimate the Q values for each state-action pair and therefore approximates the optimal Q function. The input of the deep Q network is a state and the outputs are estimated Q values for each possible action from the given state. The proposed model used a deep Q network with 100 layers supplemented with a replay buffer of a maximum size of 10000. A replay buffer consists of the previous step's transition data and the deep Q network samples a small batch of transitions for its calculations because it is a more stable approach to use uncorrelated samples than using only the latest transitions.

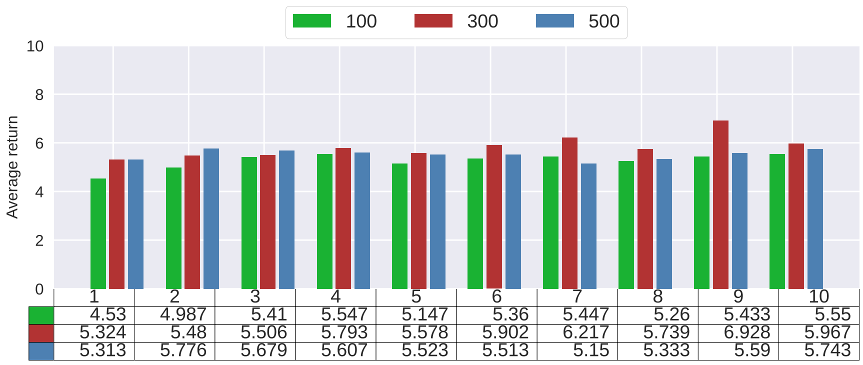
4. Experimental Results

For the experiments, we simulated data sets with different tree topologies using Rose [19] which is a tool that implements the probabilistic model of evolution. As an input of this algorithm, we created several trees in Newick [13] format and set 50 as the average length of the sequences. The trees can be divided into three categories according to their topology: balanced, pectinate, and random. The output of Rose is the generated alignment. From the alignments, we calculate the distance matrices with Emboss's [14] *distmat* module. The reinforcement learning algorithm can be parameterized to work with any constant number of sequences and the training has to run on a data set that contains alignments with the same number of sequences. To create the tree we wanted to resemble the one we constructed, we used the DendroPy library's [21] NJ algorithm and also to calculate the symmetric difference. We implemented the proposed approach in Python using Tensorflow's reinforcement learning library, TF-Agents [6].

We trained the proposed RL algorithm on data sets of different sizes: 100, 300, and 500 distance matrices, each containing 3 types of topologies. 25% – 25% of the distance matrices was based on pectinate and balanced trees and the rest 50% was based on random trees. For these experiments, each data set had 6 sequences, and for each type of topology, we assigned randomized branch lengths to have a diverse training set.

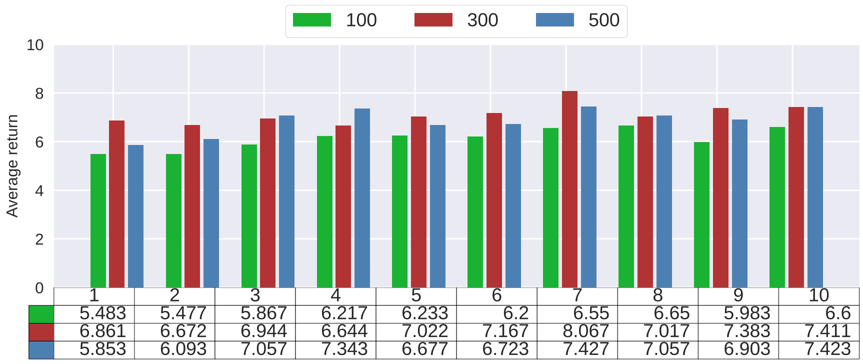
Figure 3 shows how the average return changes as the number of training sessions increases. One training session means going through the training data set and trying to construct the phylogenetic tree for the given distance matrix and repeating on each one 10 times to encourage exploitation. Every test case had the same parameters except the discount factor. Each time the average return was calculated using the trained policy on a different evaluation environment which also contained the training data set, thereby we can determine whether the policy became more accurate or not on the training data set.

We designed this experiment to determine the discount factor which is an important training parameter. The discount factor is between 0 and 1 and it means whether the agent should prioritize immediate rewards (0) or prefer potential future rewards (higher values) the agent expects to receive. In the experiment shown in Figure 3a, we used 0.05 as the discount factor and the diagrams show what we would expect: in this environment, immediate rewards have smaller significance than future rewards. Higher discount factor values showed more suitable tendencies. In this algorithm, the RL agent has to focus on maximizing what he receives at the end of the episode but also consider immediate rewards due to avoiding invalid states. The results of the experiments



	1	2	3	4	5	6	7	8	9	10
🟩	4.53	4.987	5.41	5.547	5.147	5.36	5.447	5.26	5.433	5.55
🟥	5.324	5.48	5.506	5.793	5.578	5.902	6.217	5.739	6.928	5.967
🟦	5.313	5.776	5.679	5.607	5.523	5.513	5.15	5.333	5.59	5.743

(A) 0.05 discount factor



	1	2	3	4	5	6	7	8	9	10
🟩	5.483	5.477	5.867	6.217	6.233	6.2	6.55	6.65	5.983	6.6
🟥	6.861	6.672	6.944	6.644	7.022	7.167	8.067	7.017	7.383	7.411
🟦	5.853	6.093	7.057	7.343	6.677	6.723	7.427	7.057	6.903	7.423

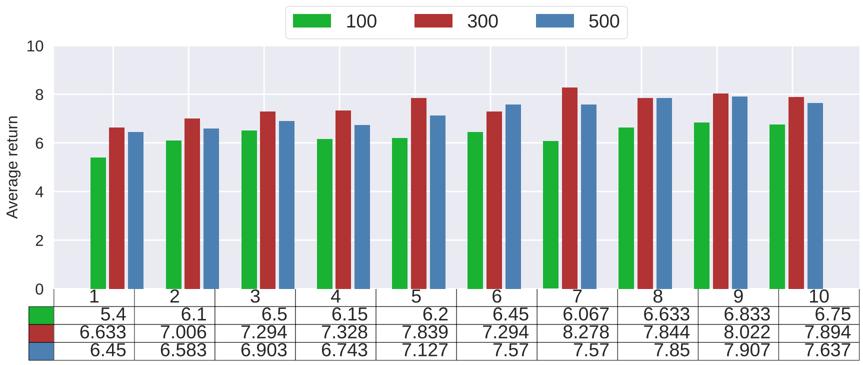
(B) 0.5 discount factor

FIGURE 3. Average return of the reward function at the end of each session with different discount factor parameters

that were performed to determine the discount factor are shown in Figure 3, and amongst them, the most promising test case is shown in Figure 3d.

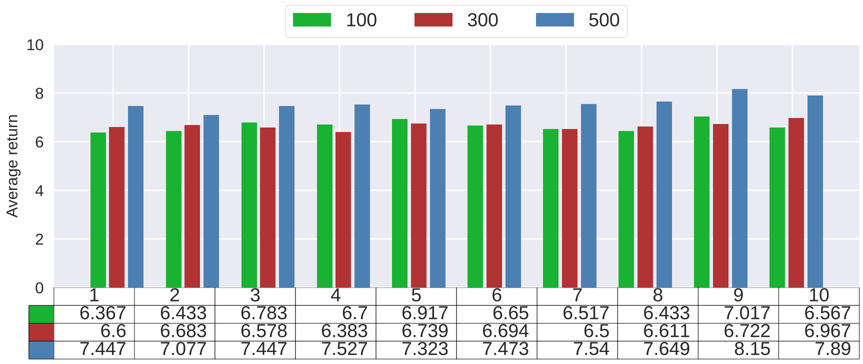
Based on these results, for the upcoming experiments, we used 0.95 as the discount factor.

For determining the accuracy, we experimented with different topology ratios in the training data set and examined how these differences affected the outcome. In the experiment shown in Figure 4, we trained every model during 10 sessions (e.g. for a data set containing 500 distance matrices this means 50000 episodes). According to the previous experiment, the larger data sets could reach better results after several episodes of training, therefore we only examined the cases where the training sets contained 300 and 500 distance



	1	2	3	4	5	6	7	8	9	10
🟩	5.4	6.1	6.5	6.15	6.2	6.45	6.067	6.633	6.833	6.75
🟥	6.633	7.006	7.294	7.328	7.839	7.294	8.278	7.844	8.022	7.894
🟦	6.45	6.583	6.903	6.743	7.127	7.57	7.57	7.85	7.907	7.637

(c) 0.75 discount factor

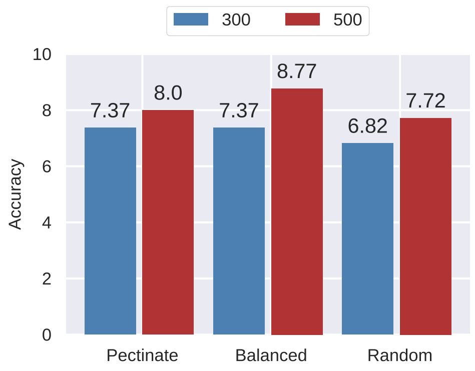


	1	2	3	4	5	6	7	8	9	10
🟩	6.367	6.433	6.783	6.7	6.917	6.65	6.517	6.433	7.017	6.567
🟥	6.6	6.683	6.578	6.383	6.739	6.694	6.5	6.611	6.722	6.967
🟦	7.447	7.077	7.447	7.527	7.323	7.473	7.54	7.649	8.15	7.89

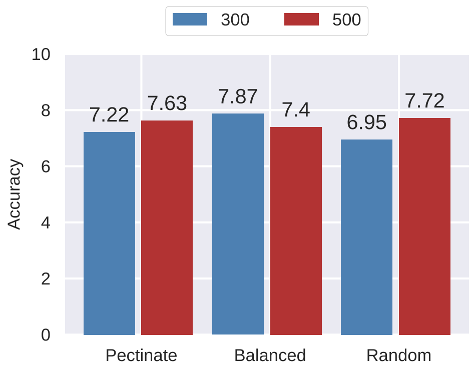
(d) 0.95 discount factor

Figure 3. Average return of the reward function at the end of each session (cont.)

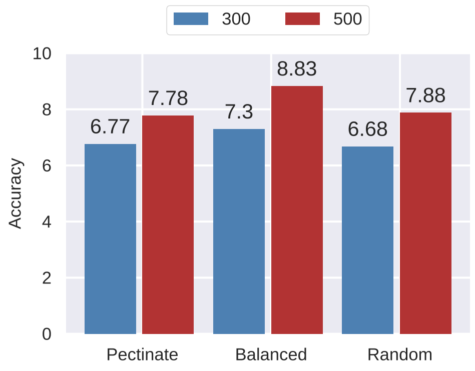
matrices. For this experiment, first, we had to create a test data set in the same way as the training data set and separate the trees by their topology, so we can evaluate the differences between them. For every topology, the test data set on which we evaluated the accuracy, contained 100 distance matrices. The result of the training is a policy by which the RL agent makes decisions about the next step in the algorithm. Using this policy, we constructed trees from the test data set and used the end reward as the accuracy (as mentioned before if the algorithm constructed a tree, then the end reward is between 0 and 10 where 10 means the tree is identical to the goal tree). In this evaluation phase, we still used the DendroPy library's NJ algorithm to determine the symmetric difference.



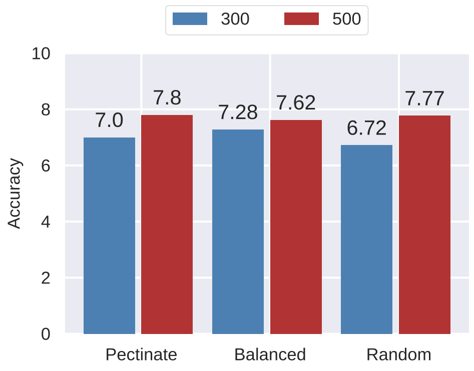
(A) 40% - 10% - 50% ratios



(B) 30% - 10% - 60% ratios



(C) 40% - 20% - 40% ratios



(D) 30% - 20% - 50% ratios

FIGURE 4. Accuracy on test data set with different topology ratios (balanced - pectinate - random) in the training set

Figure 4 shows how the different topology ratios in the training data set affected the accuracy of the model. In the cases of the smaller data set, the model could reconstruct a balanced tree with the highest accuracy, even if it was mostly trained on trees with other topologies. Therefore the training data set could contain balanced trees in a small ratio (10%) without compromising the accuracy of this topology. The overall highest accuracy was detected if the policy was trained on the data set containing 500 distance matrices and the proportion of trees was as follows: 40% pectinate, 10% balanced and 50% random. Upon these results we examined whether a longer training session could improve the policy's accuracy.

Table 1. Accuracy results of the trained policies on different topologies. Each policy was trained on the data set containing 500 distance matrices: 10% pectinate, 40% balanced and 50% random trees. The columns associated with the topologies contain the average return of the model

Episodes	Pectinate	Balanced	Random	Average accuracy
50 000	8.0	8.77	7.72	81.63%
100 000	8.48	8.55	8.28	84.36%
150 000	8.15	8.53	8.02	82.33%

Table 1 shows how the accuracy increases as we double the number of episodes in the training session in the case when the training data set contained 500 distance matrices. With 100000 episodes of training the average accuracy was 84.36%, but the further increase of the episodes resulted in the decrease of the accuracy, possibly due to overfitting, which means the model became too specific to the training data. For the model to work on different data as well, the training has to stop when the prediction has sufficient accuracy but it does not yet become too precise to the data it was trained on, because that would impact the generalization negatively. Our experiments showed that the model on the aforementioned training data set was the most accurate when it was trained for 100000 episodes.

Both the training and test data set were created with Rose [19] using the same parameters and therefore having the same model of evolution. It is expected that this evolutionary model affects the accuracy of the algorithm. To verify it we tested the model on data with a different model of evolution. We selected a 6-element subset of the Sarich data set [17] [5] and constructed the phylogenetic tree with the trained model. The result showed 50% accuracy, which means that indeed the RL model approximates the model of evolution of the training data set. To have a more generalized model further experiments have to be performed by expanding the training data set with different types of evolution models.

5. Conclusions and Future Work

In this study, we examined how a phylogenetic tree constructed by NJ could be recreated with a type of artificial intelligence, the reinforced learning model. The essence of the proposed model is that the RL agent moves in a tree graph where each descendant indicates which two individuals are joined in the constructed tree. At the end of the episode, the agent is rewarded for how well he managed to recreate a tree similar to the one constructed by the traditional NJ algorithm. The model uses a deep Q network to find the optimal policy.

Our experiments showed that the proposed model has the possibility to produce an accurate phylogenetic tree based on the distance matrix, and this accuracy could be further improved by refining the training data set. Although the model has great potential for constructing accurate phylogenetic trees, it has its limitation. For each number of sequences that have to be joined, a unique RL model has to be trained because a model only works for a constant number of sequences that the training session's data sets contained.

This model outlines a generalized approach to the problem and in this form is a model illustrating artificial intelligence rather than an algorithm for producing real phylogenetic trees. However, we believe that it could serve as a useful basis on which to build a solution to the real problem.

In the future, we want to supplement the model with a long short-term memory network [7]. It may be worthwhile to implement this addition to the deep Q learning phase because it has the advantage of being able to memorize long/hidden dependencies and thus make more accurate decisions.

Besides, further experiments could be performed to improve the presented results by expanding the training data set ($10 - 100$ times and with different models of evolution) and by increasing the number of episodes in the training phase accordingly.

The presented results, although promising, were still generated on simulated data. For the RL approach to provide a solution in real cases, the model should be supplemented to work with incomplete or inaccurate data structures, or even starting one step further, where the sequences serve as input, thus providing more information, not just a statistic representation about the sequences. Also, when calculating trees, it would be worthwhile to calculate the length of the branches as well, not just the topology. In this case, the comparison of trees in the reward function would not be done according to the symmetric difference, but according to the Robinson-Foulds algorithm [15]. Furthermore, the case where a rooted phylogenetic tree has to be constructed could be examined.

Our proposed model is an example of how artificial intelligence and deep learning can be applied in bioinformatics, where many computationally complex problems possibly could be solved more effectively by the application of these technologies.

6. Acknowledgements

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

References

[1] Bhattacharjee, A., and Bayzid, M. S. Machine learning based imputation techniques for estimating phylogenetic trees from incomplete distance matrices. *BMC genomics 21*, 1 (2020), 1–14.

[2] Chor, B., and Tuller, T. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics 21*, suppl_1 (2005), i97–i106.

[3] Elias, I., and Lagergren, J. Fast neighbor joining. In *International Colloquium on Automata, Languages, and Programming* (2005), Springer, pp. 1263–1274.

[4] Evans, J., Sheneman, L., and Foster, J. Relaxed neighbor joining: a fast distance-based phylogenetic tree construction method. *Journal of molecular evolution 62*, 6 (2006), 785–792.

[5] Felsenstein, J., and Felenstein, J. *Inferring phylogenies*, vol. 2. Sinauer associates Sunderland, MA, 2004.

[6] Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018. [Online; accessed 06-April-2021].

[7] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[8] Jafari, R., Javidi, M. M., and Rafsanjani, M. K. Using deep reinforcement learning approach for solving the multiple sequence alignment problem. *SN Applied Sciences 1*, 6 (2019), 1–12.

[9] Kalantari, J., Nelson, H., and Chia, N. The unreasonable effectiveness of inverse reinforcement learning in advancing cancer research. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 437–445.

[10] Mailund, T., and Pedersen, C. N. Quickjoin—fast neighbour-joining tree reconstruction. *Bioinformatics 20*, 17 (2004), 3261–3262.

[11] Mircea, I.-G., Bocicor, I., and Czibula, G. A reinforcement learning based approach to multiple sequence alignment. In *International Workshop Soft Computing Applications* (2016), Springer, pp. 54–70.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.

[13] Olsen, G. The "newick's 8: 45" tree format standard. *World-Wide-Web Reference. http://evolution.genetics.washington.edu/phylip/newick_doc.html* (1990).

[14] Rice, P., Longden, I., and Bleasby, A. Emboss: the european molecular biology open software suite. *Trends in genetics 16*, 6 (2000), 276–277.

[15] Robinson, D. F., and Foulds, L. R. Comparison of phylogenetic trees. *Mathematical biosciences 53*, 1-2 (1981), 131–147.

[16] Saitou, N., and Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution 4*, 4 (1987), 406–425.

[17] Sarich, V. M. Pinniped phylogeny. *Systematic Zoology 18*, 4 (1969), 416–422.

[18] Simonsen, M., Mailund, T., and Pedersen, C. N. Rapid neighbour-joining. In *International Workshop on Algorithms in Bioinformatics* (2008), Springer, pp. 113–122.

[19] Stoye, J., Evers, D., and Meyer, F. Rose: generating sequence families. *Bioinformatics (Oxford, England) 14*, 2 (1998), 157–163.

[20] Studier, J. A., and Keppler, K. J. A note on the neighbor-joining algorithm of saitou and nei. *Molecular biology and evolution 5*, 6 (1988), 729–731.

[21] Sukumaran, J., and Holder, M. T. Dendropy: a python library for phylogenetic computing. *Bioinformatics 26*, 12 (2010), 1569–1571.

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

[23] Suvorov, A., Hochuli, J., and Schrider, D. R. Accurate inference of tree topologies from multiple sequence alignments using deep learning. *Systematic biology 69*, 2 (2020), 221–233.

[24] Wheeler, T. J. Large-scale neighbor-joining with ninja. In *International Workshop on Algorithms in Bioinformatics* (2009), Springer, pp. 375–389.

ELTE Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary
Email address: `ie33ou@inf.elte.hu, kiss@inf.elte.hu`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.04

DETECTING THE MOST IMPORTANT CLASSES FROM SOFTWARE SYSTEMS WITH SELF ORGANIZING MAPS

ELENA-MANUELA MANOLE

Abstract. Self Organizing Maps (SOM) are unsupervised neural networks suited for visualisation purposes and clustering analysis. This study uses SOM to solve a software engineering problem: detecting the most important (key) classes from software projects. Key classes are meant to link the most valuable concepts of a software system and in general these are found in the solution documentation. UML models created in the design phase become deprecated in time and tend to be a source of confusion for large legacy software. Therefore, developers try to reconstruct class diagrams from the source code using reverse engineering. However, the resulting diagram is often very cluttered and difficult to understand. There is an interest for automatic tools for building concise class diagrams, but the machine learning possibilities are not fully explored at the moment. This paper proposes two possible algorithms to transform SOM in a classification algorithm to solve this task, which involves separating the important classes - that should be on the diagrams - from the others, less important ones. Moreover, SOM is a reliable visualization tool which able to provide an insight about the structure of the analysed projects.

Introduction

Nowadays, many software engineering problems are tackled by means of computational intelligence. The idea is to reformulate some difficult, repetitive, expensive, or even boring software engineering activities as search problems that can benefit from the advantages of Artificial Intelligence. Various

Received by the editors: 24 May 2021.

2010 *Mathematics Subject Classification.* 68T05, 68T20.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*; D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided software engineering (CASE).*

Key words and phrases. self organizing maps, artificial neural networks, machine learning, classification algorithms, search based software engineering, software visualisation, key class detection, condensing class diagrams .

problems can be approached: defect detection [2] [21], cost estimation [13], requirements analysis [8], software maintenance [9], test oracles [28], maintaining legacy systems [29].

The focus of this paper is a less popular, yet challenging software engineering problem: detecting the key classes from a software system for automatic diagram construction. From a machine learning perspective, we have a data set containing OOP classes and the goal is to classify them in groups of relevant and non-relevant ones. The relevant (key) classes are meant to be represented on the UML diagrams while the classes with lowest scores are ignored.

Self Organizing Maps (SOM) have been previously used for software analysis and visualization but was never applied to the class detection problem before. The paper is structured as follows: the first chapter presents the basics of Self Organizing Maps. In the second section we review some related articles of the domain. Then, the key detection problem and its applications are highlighted. The fourth chapter deals with our methodology for resolving this problem, where we describe the steps of our approach, beginning with data representation and visualization and continuing with SOM algorithm and its transformation in a hybrid method for classification purposes. The experimental setting and the performance results are presented in the fifth chapter, followed by the concluding remarks section.

1. Self Organizing Maps

The SOM algorithm was developed in the 1980's by Professor Teuvo Kohonen [11]. SOM are presented as special unsupervised neural networks. Their main purpose is to create a low dimensional representation of the input space of the training data. This representation forms a map of neurons where they compete and organize themselves in such a way that the topological properties of the input space is preserved. This means that if two instances from the input space are close to each other, they will be near each other in the map, too. Only one neuron is activated at any time (the winning neuron), because of the competition process which induce inhibitory connections.

The SOM algorithm is inspired from the neural cortex of the brain. Different sensors (motor, visual, auditory) are mapped in certain areas of the cortex. They form a map (topographic map), where each piece of input information is stored in its neighbourhood and where neurons responsible with closely related pieces of information is kept close together so that they can communicate fast via short synapses. The location of a neuron in a SOM corresponds to a particular instance from the input space [11].

The architecture of SOM is fairly simple. The Kohonen map has a feed-forward structure with an input layer and a computational layer with neurons arranged in a matrix. Each neuron is connected to all nodes from the input layer. The number of input units is equal to the number of dimensions from the input space. In the training phase, the map is built using the input data. Then, a new input instance can be automatically classified using the mapping phase, by matching it to its nearest neuron.

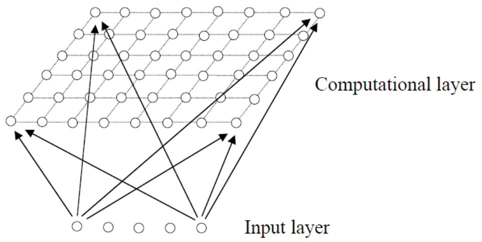


FIGURE 1. SOM architecture

The main components of any self organizing map are:

- **Initialization phase**: The connection weights of each neuron j from the computational layer, $w_j = (w_{j1}, w_{j2}, ..., w_{jD})$ are initialized at random.
- **Competition**: For each input instance, the distance between the input vector: $x = (x_1, x_2, ..., x_D)$ and the weight vector: $w_j = (w_{j1}, w_{j2}, ..., w_{jD})$ is computed as: $d_j^2(x) = \sum_{i=1}^{D}(x_i - w_{ji})^2$, if we use the Euclidean distance. Other distances can be used as well. The neuron with the smallest distance from the input instance becomes the winning neuron or the Best Matching Unit (BMU).
- **Cooperation**: Now, the winning neuron influences the other neighboring neurons, by signaling them. The neurons which are closer to the BMU are excited more than the neurons which are far away. For this, we define a topological neighborhood function, which decays with distance: $T_{j,I(x)} = exp(-S_{j,I(x)}^2/2\sigma^2)$, where S_{ij} is the distance between neurons i and j and $I(x)$ is the index of the winning neuron. σ is a time dependence, for example the exponential decay: $\sigma(t) = \sigma_0 exp(-t/\tau_\sigma)$
- **Adaption**: In this phase, the weights of the BMU and its neighbors are updated. The excited neurons are moved closer to the data point, by adjusting the associated connection weights. The BMU is modified by a greater amount than the neighboring neurons. The update rule is: $\Delta w_{ji} = \eta(t) \times T_{j,I(x)}(t) \times (x_i - w_{ji})$, where η is the learning rate, defined as: $\eta(t) = \eta_0 exp(-t/\tau_\eta)$.

The steps from above are repeated for a predefined number of iterations or until there are not significant changes in the map organization.

As stated before, SOM is usually used for visualisation purposes. An idea is to simply draw the matrix of neurons according to their positions from the map. Each neuron is equivalent with a reduced representation of an input instance (or of a group of instances). Another common technique is to represent a matrix of distances (called U-Matrix) between the weight vectors of adjacent neurons. The matrix is coloured with values of different intensities. Lighter colour between two neurons denotes a small distance, while a darker colour denotes a large distance. Figure 2 is an example of a U-matrix. The black dots denote the neurons.

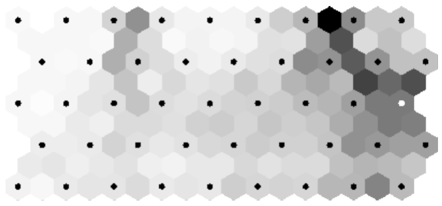


FIGURE 2. SOM U-Matrix

The U-matrix is useful especially for cluster analysis. The dark areas in the map are interpreted as boundaries between the clusters of the underlying data.

The neurons from the map are activated with various frequencies. Some neurons are activated multiple times, others are activated just once, others are never activated. To see the overall distribution of activated neurons, activation frequency map can be used.

2. RELATED WORK

In this section we give a brief overview of SOM models used for software engineering problems.

Czibula et al. [17] [3] discuss about software restructuring and refactoring via clustering with SOM. In their experiments, SOM was able to distinguish clusters and proved to be effective for visualization purposes. Even though the U-matrix can be used to identify the boundaries between clusters, it is argued that its interpretation is a matter of subjectivity. To overcome the need of a reliable clustering tool, their key-idea was to apply a hierarchical clustering algorithm on the trained units.

SOM is also effective for defect prediction, as proposed in [16]. The goal is to detect two clusters, one containing non-defective entities and another one containing defects. An analysis is performed by observing the U-matrix and the boundaries of high distances. The results showed a good performance of SOM in terms of the Area Under The Receiver Operator Characteristic Curve (AUC) and quantization error.

Another very interesting approach for default prediction is studying a hybrid SOM in a semi-supervised manner described by Abaei [1]. The technique is suited when we face a limited amount of defect data. The algorithm has several phases. Firstly, SOM is trained and an initial clustering is performed. Then, the neurons from the map that are not activated are removed from the system. The weights of the remaining (activated) neurons are assigned to labels, based on some thresholds given by certain software measurements. These weights are fed to a neural network for further training. It has been observed that the neurons of SOM tend to resemble the input data as they move towards it. Using some thresholds to stop the neurons from approaching too close to the input patterns is a justified heuristic.

In terms of visualization and analysis of software engineering data, the research conducted by MacDonell [14] presents a range of situations in which one would benefit from SOM. One application involves clustering software artefacts in groups with low, medium and high defect counts. The clusters can be interogated for statistics purposes. Another useful application is building component maps, depicting the distribution of one software metric per map. Groups with similar values for the analyzed metric are detected (similar number of attributes per class, the depth in the inheritance tree, the number of child classes, the number of lines per code). The visualization with SOM reveals useful information about the distribution of artefacts. Pedrycz et al. [23] analyses the Linguist open source Java project in terms of software metrics. Besides, in [22] they identified relationships between classes, for example, one cluster will contain classes having a specific keyword, or specific type (helper classes, error handling classes, interfaces, etc.). Some patterns between the software metrics were identified as well.

3. Key class detection and condensing class diagrams

The problem that we study can be found in the literature with various names: detecting class importance, condensing reversed engineering class diagrams or key class detection. We dedicate this separate chapter to review the existing approaches of the domain.

Detecting key classes using an automatic approach can be an interesting research topic and has several applications: condensing reversed-engineered class diagrams [19] [27], helping program comprehension, analyzing design evolution [7], prediction of future changes [31], recommending potentially relevant files that the developer should view (easy navigation) [25] and others.

Although several existing Computer Aided Software Engineering tools can be configured to remove several properties in a class diagrams, they are unable to automatically identify classes that are less important [19]. Some experimental research showed that developers experience more difficulties in finding the information they need in reverse engineered diagrams and also find the level of detail in "forward" designed ones more appropriate [5].

What makes a class important? There is a research debate about what are the attributes which determine the relevance of a class. It is claimed that an important class represents a key concept that is usually found in the documentation and that has a higher degree of control within the application. This control can be measured, for example, by identifying the tightly coupled classes. The software can be also seen as a network, where classes are vertices and the dependencies between them are edges. A class which is used by many other classes is a good candidate to be an important one, representing a fundamental information or business model. Also, a class which is using other important classes may be an important one as well.

The topic can also be treated subjectively because different groups of developers could define sets of different sizes with key classes. The key class should respect the level of detail that the developer wants to deal with.

Another issue we may face when implementing an automatic key detection system is the imbalanced data. In a real world system, only a few classes are used to document the architectural design. For instance, the developers of Tomcat 5.5 thought that only six classes would be enough to represent the important concepts of the system. Nevertheless, there are many other classes that can be included, if a developer wants to see a more detailed view.

There are a few automatic approaches in the literature for this problem. An interesting article by Șora [18] describes the importance of a class by the amount and types of interactions it has with other classes. The approach is based on an graph ranking algorithm based on Page Rank in order to model the dependencies of the system. Vale and Maia [4] use Trace Extractor which is a tool that saves files with invocation trees for each triggered concurrent thread in the studied program. These trees are used to compute the importance of the involved classes. The algorithm has a tree compression phase to remove identical parts, then a phase which classifies the subtrees as relevant and non relevant. This was done with a Naive Bayes classifier. The final step is to identify the key classes from the subtrees by considering some of the roots as the important classes (or the subtree can further split and assessed).

Osman [19] and Thung [27] solved the problem of condensing reversed engineering class diagrams by identifying only the important classes. They employed supervised methods, among which random forest was the best performing one. Later, they proposed an optimistic classification strategy for dealing with data points with unknown label. To our knowledge, they did not study SOM, so in this research we guide our experiments towards this unexplored direction. In Section 4 we use their research as reference for the approaches documented in the present article.

We found one unsupervised approach with K-means [30] and ensemble learning using the same data sets and we mention its performance as well.

4. Methodology

This chapter presents our approach of SOM for classification, which is quite unusual for this type of machine learning model. The idea is to separate the key classes from the less important ones via a modified SOM. We propose two types of algorithms: a majority voting technique on the trained map units and hybrid approach combining SOM with a classic neural networks.

4.1. **Input data.** For this research we employ the same data sets and preprocessing strategy from Osman [19] and Thung [27]. They prepared and published nine data sets, each representing a different open-source software project. Obviously, the instances are in fact OOP classes represented as numeric feature vectors. The label is denoted by the last column of the data set, called "In Design", which tells whether the current instance appears or not on the design UML diagram of the project. The nine projects are described in Table 1.

The features that characterize the instances are in fact the values for different software metrics associated with the underlying class. Generally speaking, software measures refer to quantifiable and scalar descriptions of properties of software artefacts [22].

Osman and Thung focused their work on finding metrics serving the goal to distinguish the classes that represent important concepts for the UML diagram.

In one of their earliest studies [19], they chose a set of 11 metrics, called "design metrics", which are well known measures that can be found in any software engineering article: the number of attributes, the number of operations, getters and setters, or various types of dependencies with other classes and coupling measures. The authors conducted a survey [20] about class diagram comprehension which revealed that the metrics related to size and coupling are preferred among developers. The second argument was that coupling is an important structural element in object oriented systems.

Project	Description	Total Classes	Classes In Design Diagram
ArgoUML	UML diagramming application	903	44
JGAP	A framework for performing genetic algorithms and genetic programming	171	18
JPMC	A collection of automated intelligent agents in financial sector	121	24
JavaClient	A framework for developing robotics applications	214	57
Mars	An application for creating simulation of possible human settlement on Mars	840	29
Maze	An application for solving maze puzzles	59	28
Neuroph	A framework for developing neural network architectures	161	24
Wro4J	An application for optimizing web resources	87	11
xUML	A software for producing executable and testable systems from specified data models and associated state machines	84	37

TABLE 1. The datasets [27]

Later, they addressed a new question: *How can we represent the relationships between classes from a new perspective in order to assess class importance?* This led to the concept of network metrics, which greatly improved the accuracy of the existing models [27]. The software project is seen as a network in which the nodes represent the classes and the edges denote the relationships between them (aggregation, composition, generalization and dependency). Based on this graph, a series of 10 metrics were computed. Among them, we mention: Barycenter Centrality- sum of shortest distances of node v to all other nodes, Betweenness Centrality - number of shortest paths between all possible pairs of other nodes that go through node v, Closeness Centrality - the mean shortest distance of node v to all the other nodes, Page Rank- suggesting that nodes with more incoming links are more important. Besides these, some custom metrics based on the partial known knowledge are defined: the proportion of known important classes among the neighbours of a class, the shortest distance to known important classes, neighbour existence, etc. The need for these last metrics is that in real scenarios, the amount of labeled data is limited.

The full description of the data sets and the feature extraction can be found in [19] and [27], together with download information.

4.2. **Data Visualization.** Before presenting the classification algorithms, we take advantage of the visualization capabilities of SOM to investigate the data distribution and if any clusters can be already observed.

Figure 3 presents the maps with the activated neurons for two of the datasets: JavaClient and xUML. Each neuron that was activated at least once, was the BMU (Best Matching Unit) for at least one data point, which means that the neuron can be seen as the reduced representation of that data point.

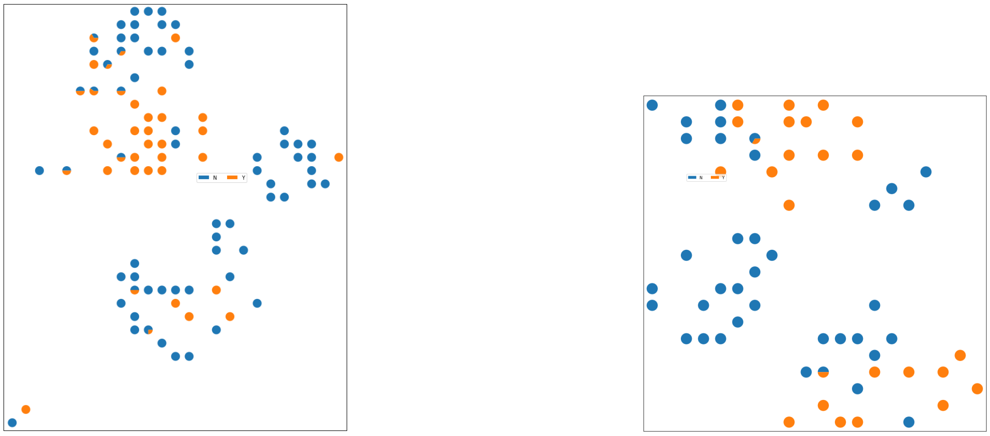


FIGURE 3. Map of activated neurons for JavaClient(left) and XUML(right)

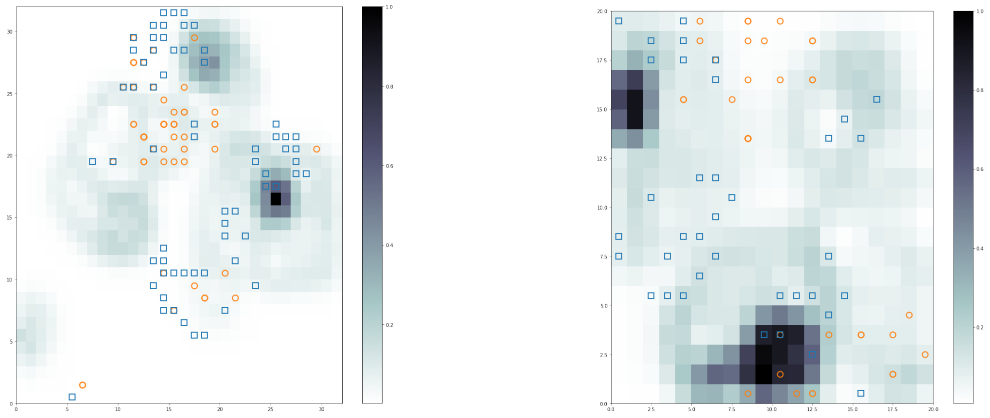


FIGURE 4. U-Matrix for JavaClient(left) and XUML(right)

We illustrate the trained map with the neurons coloured as pie charts, denoting the proportion of positive instances (key classes) and negative instances (non-important classes). The positive instances are marked with orange, while the negative ones are blue. We observe that positive ones are grouped together, with a few exceptions. Also, the neurons which are closer to the "centre" of the cluster are homogeneous, while the ones closer to the boundaries between the clusters are more mixed: they contain both positive and negative instances.

The U-matrix (Figure 4) can also provide insightful information. With light blue, small distances between neighbouring neurons are represented, while the dark blue stands for large distances between the neurons. The scale on the right side of the map depicts the distances. We label again the associated data points for each neuron. The orange circles represent the positive instances, the blue squares are the negative ones. Again, we observe the neurons with overlapping symbols - which map both positive and negative instances are closer to the cluster boundaries (dark regions).

Overall, for visualisation purposes and dimensionality reduction, we are satisfied with what SOM achieved. The data is separated in groups, but we would like to know exactly which information is encoded in the neurons. Labelling every neuron with the name of the classes they represent may provide some insight. In Figure 5 and Figure 6 we discover the following: each neuron (or neighbourhood of neurons) tends to map classes with the same kind of responsibility or belonging to the same package. Thus, it would be interesting to study other types of software metrics specific to the key class detection problem and assess if they improve the partitioning. We are not discouraged by these findings because the partitioning by design diagram classes is still a satisfactory one, as it can be observed on the coloured maps.

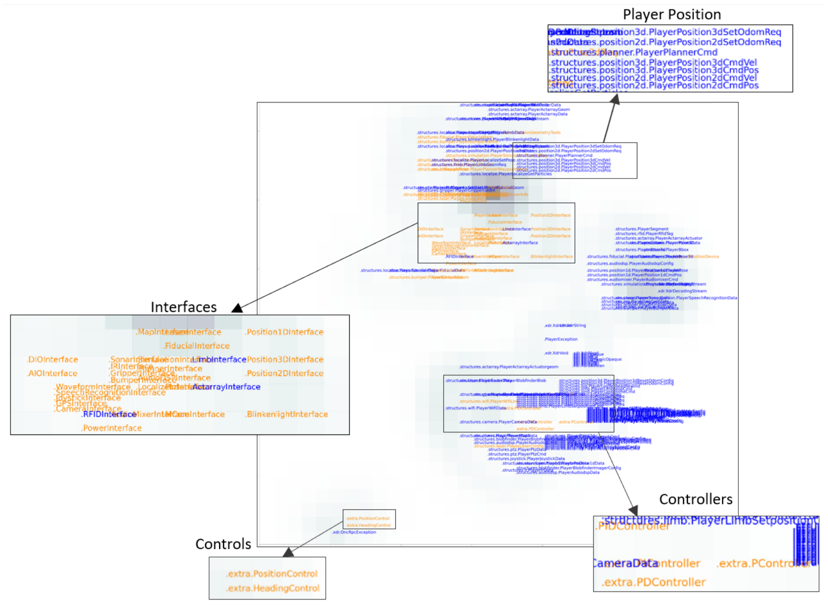


FIGURE 5. Insight of JavaClient Project with SOM

We present a "zoomed in" map representation where we mark some of the groups that we found: interfaces, controllers, factories, etc. JavaClient is an application in robotics. We observe on the map groups of classes responsible

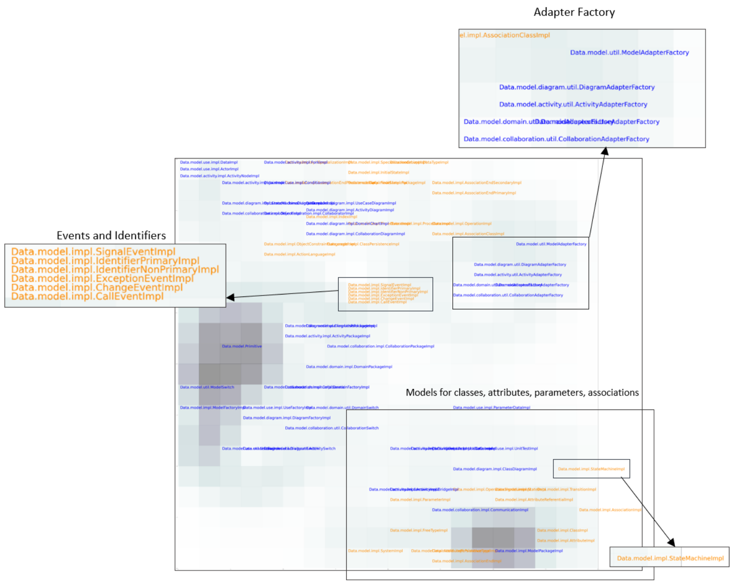


FIGURE 6. Insight of XUML Project with SOM

for the player position and controls. XUML is an application for building executable systems from data models and state machines, so in the corresponding map we find components associated to notions such as classes, attributes, parameters, associations, or events. For this project, the map units seem better separated by the "In Design" label.

4.3. **Classification with SOM - A Majority Voting Approach.** As discussed in previous chapters, we consider the classes that are on the design diagram as instances from the positive class, and all the others are in the negative class. The current section explains a probabilistic classification algorithm for the trained map units. An unseen data point is assigned to a class by using a type of majority voting. In the literature, other such techniques have been successfully applied for SOM [26], [12].

The algorithm from our own approach has the following steps:

Training the SOM

The first step is nothing else than building the map for the data set. A classical SOM is trained, similar to the one from Section 4.2. The result is a map of trained neurons, which have learned to represent the input data. On the map we find neurons which were activated at least once: they are the best matching unit for at least one input data point. There are also neurons that were never activated, these ones will be ignored in the next steps and we will call them dead neurons (similar to the approach presented in [1]. Dead neurons help their neighbours perform better in training process; however, they are no longer needed after the SOM algorithm is finished.

Build statistics for the trained neurons

Each neuron from the trained map is now analysed. For every unit j, we compute the probability of positive and negative instances that have as best matching unit the neuron j. This is similar to the pie chart mapping from section 4.2. The formulas are the simple probability computations, defined as follows:

$$P_{pos(j)} = \frac{number\_of\_positive\_instances}{total\_number\_of\_mapped\_instances}$$

$$P_{neg(j)} = \frac{number\_of\_negative\_instances}{total\_number\_of\_mapped\_instances} = 1 - P_{pos(j)}$$

Classification

When an unseen instance is going to be classified, it is presented to the trained SOM, which will try to compute the best matching unit for it. There are two cases:

1. The BMU is an activated neuron from the training phase. This means that the percent of positive and negative instances mapped so far by the neuron is known, and we have a good chance that the new instance will belong to the majority class, too.

2. The BMU was not activated in the training phase. This means that no information is available to be able to distinguish if the instance is positive or negative. In this case, we determine the top n closest neurons that were activated and compute the probabilities among them. The reason why we do not choose the closest neighbour is that one single BMU may not reflect the entire neighbourhood. For example, the closest neuron may show a probability of 100% negative, but the rest of the neighbouring neurons may encode only positive instances.

To be observed that this approach remains unsupervised. In the training phase, no information about the actual labels is used when the unit weights are updated. The probabilities are computed after the training has finished and no weights are updated for neurons which are mixed.

4.4. **Classification with SOM - Adding supervised layers.** The second technique is a hybrid approach that combines the SOM with a few traditional neural network layers. The technique is inspired and adapted from [24], in which the authors add one additional layer to the SOM.

For this study, two more layers are added to the classical SOM, which will be responsible with the classification. Some existing approaches use the label information during the SOM training to update the weights, or augment the input vectors with the label information [10], [15]. The present approach is different because it keeps the SOM independent of the label information. As a result, only the additional layers will update their neurons' weights to learn the classification. The SOM organization remains intact. The reason is that we want to preserve the unsupervised and competitive characteristics of the SOM learning step.

The architecture of the proposed system has two components.

SOM component

The first component is the SOM which is trained in the classical unsupervised manner. After the SOM units are organized, we use forward connections to bind them to the additional supervised layers.

In order to be fed forward, the SOM units need to use an activation function. For this, we use the Gaussian similarity, so the activation of the unit j of the SOM map has the following formula:

$$a_j = e^{\frac{-d_j^2(x)}{2\sigma^2}}$$

where $d_j^2(x)$ is the distance between the unit j and the input data x and σ is the width of the Gaussian.

Additional layers

The second component is responsible for learning to perform classification, and consists of the two additional neuron layers.

One of the layers is fully connected to the SOM units and is using ReLU activation. The output of a neuron l from this layer is:

$$o_l = ReLU(\sum_j w_{jl} \times a_j + bias_l)$$

where w_{jl} is the connection weight between the neuron l and SOM unit j and ReLU is the Rectified Linear Unit activation function computed as:

$$ReLU(x) = max(0, x)$$

The other layer is the output layer, which has a single neuron connected to the previous ReLU units. This neuron uses sigmoid activation to output the probability of the positive class. The sigmoid function is a common choice for binary classification and has the following formula:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

To train the two layers, cross entropy loss function is used with gradient descent. The cross entropy cost is popular among state of the art models. It is preferred in [24] and [6] as it leads to faster convergence and to better local optimum than the squared error function. The cross entropy formula for the binary classification is:

$$C(y, o) = -ylog(o) + (1 - y)log(1 - o)],$$

where y denotes the true label and o is the prediction.

4.5. **A comparison between the two approaches.**

The common part of the two proposed algorithm consists of the unsupervised SOM component. Both strategies are meant to transform SOM for classification tasks.

The majority voting strategy operates on the trained units, by associating simple probabilities to each unit based on the labels of the instances mapped to each neuron. The classification is performed using these probabilities.

The second strategy is in fact a hybrid algorithm, combining two ML models: SOM and neural networks. The classification step is more complex than the aforementioned technique, because it requires learning. The unsupervised SOM units are fed to the neural network which will perform supervised training.

The two architectures are presented side by side in Figure 7.

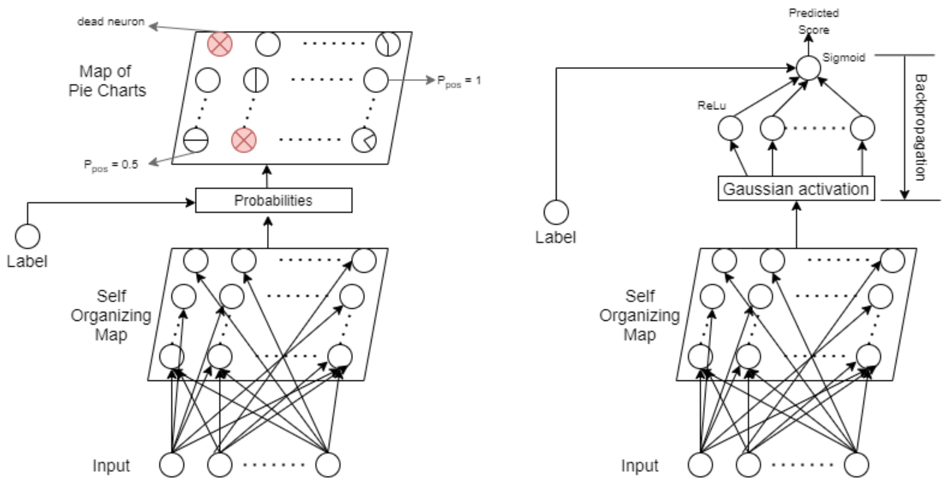


FIGURE 7.
Comparison between the proposed models: SOM with majority voting (left) and SOM with additional layers (right)

5. Experimental Settings and Performance Results

In the following, the two proposed classification algorithms are evaluated for the key class detection problem. Besides, we explain the experimental settings involved in our study.

5.1. **Experimental Settings.** To implement the SOM, we used the *MiniSom* Python library. For the majority voting algorithm, we extended *MiniSom* with custom methods where necessary. For the additional neural layers we used the *Keras* library. The connections between the SOM and the rest of the layers have been implemented from scratch.

For the SOM component we used a traditional rectangular map topology. We performed some tests with a hexagonal topology which brought no significant improvement, therefore we decided not to include it.

Regarding the algorithm based on majority voting, our experiments showed that a map of 5x5 or 7x7 neurons is sufficient for the smaller data sets. For the larger ones we discovered that we need a map with the size around 20x20. The number of neighbours n used to classify unseen instances mapped to a dead BMU was set to 5 by default.

In the second model, the number of neurons on the additional layer connected to the SOM was set to 2/3 of the size of the SOM map. In addition, a dropout strategy from *Keras* is used on this layer.

Because the data sets are imbalanced, leave one out validation is the chosen strategy. In this way, the training set contains as many positive instances as possible. Moreover, each and every data instance is used once for testing. For the supervised component, a strategy for initializing the bias of the output layer was used:

$$b = log(positive\_instances/negative\_instances)$$

To help the training even more, the classes are weighted, telling the model to pay more attention to the positive class, which has a greater weight :

$$weight_0 = (1/negative\_instances) * (total\_instances/2)$$
$$weight_1 = (1/positive\_instances) * (total\_instances/2)$$

The performance will be assessed in terms of the AUC. There are two motivations for using this measure. Firstly, AUC is preferred for highly imbalanced data because it does not favour models that predict the majority label for all data points. Secondly, this metric was also used by Osman and Thung [19] [27], so reporting the same performance measure will help comparing the models.

However, some scientists claim that AUC should not be used when comparing one model to another and should only be used to determine if a ML model is better than random guessing. In absence of any other performance measures for the approach by Thung et al [27], we can only rely on the AUC score. Their experiment is performed in *Weka*, and computes the AUC with the help of a function from the same library.

To evaluate SOM for our novel approach, the *scikit-learn* library for performance metrics was used. For the AUC we applied the *roc_auc_score* function which determines the AUC based on the prediction scores. The implementation uses Riemann integrals for approximating the area under the ROC curve. The thresholds on the curve are also computed automatically by this function. Besides, we also calculated the precision and recall scores. To be noted that the AUC relies on the class probabilities, while for the precision and recall, the predictions are transformed in discrete labels. A default probability threshold of 0.5 was used (the predictions with probabilities greater than 0.5 are considered positive). In a real application, the developers should adjust this threshold, based on the level of detail they prefer for the generated diagram.

5.2. **Results Analysis.** Osman [19] and Thung [27] conducted various research in the domain, with performance ranging from 0.60 to 0.90 (AUC), but we compare our results with their best performance on each data set. Moreover, we also mention the results obtained by the K-means approach employed by Yang et al. [30].

The final performance results are presented in Table 2 and Table 3. Our methods are compared to the random forest optimistic classifier from Thung et al [27] (Baseline 1) and to the K-means approach with ensembles [30] (Baseline 2). Despite the fact that the average AUC is slightly lower than the Baseline 1, SOM achieved better results on some of the datasets. Particularly for xUML project, an outstanding 0.95 AUC was obtained. Compared to the K-means approach, our results are significantly better. However, their study has advantages as well, employing a strategy which requires only a small amount of labelled data.

In general, our approach with additional layers is better than the majority voting one. The surprise was for the Wro4J project which was very difficult for the neural network, but outperformed the literature with the majority voting strategy.

The greatest challenge remains the highly imbalanced data sets. Also, the precision of the neural network is quite low, while the recall is significantly higher. This means that most of relevant classes were successfully retrieved, but the model tends to consider many false positives as well. This could be slightly adjusted by tuning the classification probability threshold.

Project	Baseline 1 [27]	Baseline 2 [30]	SOM & maj voting	SOM & supervised layers
ArgoUML	0.757	0.658	0.71	0.73
JGAP	0.797	0.835	0.67	0.77
JPMC	0.813	0.553	0.76	0.78
JavaClient	0.862	0.774	0.88	0.89
Mars	0.845	0.776	0.76	0.83
Maze	0.767	0.584	0.65	0.61
Neuroph	0.915	0.894	0.88	0.88
Wro4J	0.763	0.680	0.79	0.74
xUML	0.905	0.814	0.94	0.95
Average	0.825	0.730	0.78	0.80

Table 2. AUC achieved by our SOM-based techniques compared to other existing models

Project	SOM & maj voting		SOM & supervised layers	
	Precision	Recall	Precision	Recall
ArgoUML	0.42	0.25	0.14	0.64
JGAP	0.36	0.28	0.23	0.72
JPMC	0.54	0.54	0.52	0.50
JavaClient	0.71	0.81	0.70	0.80
Mars	0.23	0.21	0.13	0.65
Maze	0.74	0.52	0.54	0.78
Neuroph	0.50	0.58	0.47	0.63
Wro4J	0.55	0.55	0.33	0.73
xUML	0.78	0.84	0.92	0.87
Average	0.54	0.51	0.44	0.70

Table 3. Precision And Recall obtained with SOM

6. Concluding Remarks and Further Improvements

This study has proved that SOM is a good challenger for the common classification algorithms. We conclude that it is not only a versatile tool for learning and visualization, but also extensible for different ML tasks.

In general, the software engineering tasks are challenging to tackle from a machine learning perspective, and the problem presented in this research is not an exception. With the help of the Self Organizing Maps, the distribution of the data can be assessed, as well as the relevance of the features that were used to represent it. In the future, it would be interesting to research for other software metrics that would better reflect the degree of importance of a class within a OOP system, but any study related to software measures is a research topic on its own.

The SOM algorithm applied on the important classes detection problem was able to converge quite fast to an acceptable map representation. Also, the majority voting strategy achieved a satisfactory performance for classification, despite its simplicity. The additional neural network did not need many layers and neurons, as we found that one hidden layer (besides SOM map) and a single neuron for the output is enough to learn the classifier. By adding a few more layers, the model is prone to achieve even better results if properly tuned.

Other classification strategies based on SOM are worth exploring. There is the possibility to transform SOM in a completely supervised algorithm, by adjusting the map weights based on the label information and loss function, but for this article the choice was to keep the map organization unchanged.

On a practical note, the purpose of search-based software engineering is to find automatic solutions for everyday tasks. Therefore, a possible application would be to develop an IDEE plugin which integrates the ML model to generate automatic condensed diagrams. In this way, the developers can have a quick initial summary, so that the time spent understanding and navigating through a new project is reduced.

References

[1] Abaei, G., Selamat, A., and Fujita, H. An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. *Know.-Based Syst. 74*, 1 (Jan. 2015), 28–39.

[2] Ceylan, E., Kutlubay, F. O., and Bener, A. B. Software defect identification using machine learning techniques. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)* (2006), pp. 240–247.

[3] Czibula, G., and Czibula, I. Unsupervised restructuring of object-oriented software systems using self-organizing feature maps. *International Journal of Innovative Computing, Information and Control 8* (03 2012).

[4] do Nascimento Vale, L., and de Almeida Maia, M. Key classes in object-oriented systems: Detection and assessment. *International Journal of Software Engineering and Knowledge Engineering 29*, 10 (2019), 1439–1463.

[5] Fernández-Sáez, A. M., Genero, M., Chaudron, M. R., Caivano, D., and Ramos, I. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: A family of experiments. *Information and Software Technology 57* (2015), 644 – 663.

[6] Golik, P., Doetsch, P., and Ney, H. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH* (2013).

[7] Hammad, M., Collard, M. L., and Maletic, J. I. Measuring class importance in the context of design evolution. *2010 IEEE 18th International Conference on Program Comprehension* (2010), 148–151.

[8] Iqbal, T., Elahidoost, P., and Lucio, L. A bird's eye view on requirements engineering and machine learning. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)* (12 2018).

[9] Jha, S., Kumar, R., Son, L., Priyadarshini, I., Sharma, R., Long, H., and Abdel-Basset, M. Deep learning approach for software maintainability metrics prediction. *IEEE Access PP* (04 2019), 1–1.

[10] Kohonen, T. The 'neural' phonetic typewriter. *Computer 21*, 3 (1988), 11–22.

[11] Kohonen, T. The self-organizing map. *Proceedings of the IEEE 78*, 9 (1990), 1464–1480.

[12] Lau, K., Yin, H., and Hubbard, S. Kernel self-organising maps for classification. *Neurocomputing 69* (10 2006), 2033–2040.

[13] Lin, J.-C. Automatically estimating software effort and cost using computing intelligence technique. *Computer Science & Information Technology 2* (10 2012), 381–392.

[14] MacDonell, S. G. Visualization and analysis of software engineering data using self-organizing maps. In *2005 International Symposium on Empirical Software Engineering, 2005.* (2005), pp. 10 pp.–.

[15] Mattos, C., and Barreto, G. Artie and muscle models: building ensemble classifiers from fuzzy art and som networks. *Neural Computing and Applications 22* (01 2013), 49–61.

[16] Onet-Marian, Z., Czibula, I., Czibula, G., and Sotoc, S. Software defect detection using self-organizing maps. *Studia Universitatis Babeș-Bolyai Informatica LX* (01 2015), 55–69.

[17] Onet-Marian, Z., Czibula, I.-G., and Czibula, G. A hierarchical clustering-based approach for software restructuring at the package level. *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)* (09 2017), 239–246.

[18] Șora, I. Helping progran comprehension of large software systems by identifying their most important classes. In *ENASE* (Department of Computer and Software Engineering University Politehnica of Timisoara, Romania, 2015).

[19] Osman, M. H., Chaudron, M. R. V., and v. d. Putten, P. An analysis of machine learning algorithms for condensing reverse engineered class diagrams. *2013 IEEE International Conference on Software Maintenance* (2013), 140–149.

[20] Osman, M. H., Zadelhoff, A., and Chaudron, M. UML class diagram simplification: A survey for improving reverse engineered class diagram comprehension. *Empirical Studies on the Effects of Modeling in Software Development* (01 2013), 291–296.

[21] Paramshetti, P., and Phalke, D. Survey on software defect prediction using machine learning techniques. *International Journal of Science and Research (IJSR) 3*, 12 (2014).

[22] Pedrycz, W., Succi, G., Musilek, P., and Bai, X. Using self-organizing maps to analyze object-oriented software measures. *Journal of Systems and Software 59* (10 2001), 65–82.

[23] Pedrycz, W., Succi, G., Reformat, M., Musilek, P., and Bai, X. Self organizing maps as a tool for software analysis. *Canadian Conference on Electrical and Computer Engineering 1* (02 2001), 93 – 97 vol.1.

[24] Platon, L., Zehraoui, F., and Tahi, F. Self-organizing maps with supervised layer. *12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)* (06 2017), 1–8.

[25] Singer, J., Elves, R., and Storey, M.-A. Navtracks: supporting navigation in software maintenance. *IEEE International Conference on Software Maintenance, ICSM 2005* (10 2005), 325 – 334.

[26] Sousa, R., Rocha Neto, A., Cardoso, J., and Barreto, G. Robust classification with reject option using the self-organizing map. *Neural Computing and Applications 26* (01 2015).

[27] Thung, F., Lo, D., Osman, M. H., and Chaudron, M. R. V. Condensing class diagrams by analyzing design and network metrics using optimistic classification. *Proceedings of the 22nd International Conference on Program Comprehension* (2014), 110–121.

[28] Vanmali, M., Last, M., and Kandel, A. Using a neural network in the software testing process. *Int. J. Intell. Syst. 17* (01 2002), 45–62.

[29] Wiggerts, T. Using clustering algorithms in legacy systems remodularization. In *Proceedings of the Fourth Working Conference on Reverse Engineering* (1997), pp. 33–43.

[30] Yang, X., Lo, D., Xia, X., and Sun, J. Condensing class diagrams with minimal manual labeling cost. In *Proceedings - 2016 IEEE 40th Annual Computer Software and Applications Conference, COMPSAC 2016* (United States of America, 2016), vol. 1, IEEE, Institute of Electrical and Electronics Engineers, pp. 22–31. International Computer Software and Applications Conference 2016, COMPSAC 2016 ; Conference date: 10-06-2016 Through 14-06-2016.

[31] Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A. Mining version histories to guide software changes. *Proceedings of the 26th International Conference on Software Engineering* (2004), 563–572.

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

Email address: `meic2001@scs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.05

RETINAL BLOOD VESSEL SEGMENTATION ON STYLE-AUGMENTED IMAGES

MARCELL DÁVID TÓTH AND ATTILA KISS

Abstract. The average human lifespan increased dramatically in the second half of 20th century. It was mainly due to technological improvements, which were driven by the continuous war preparations, and while humans have got another 20 years to live, unfortunately there are some sad side effects added to the elderly life. Various diseases can attack the eye, our major organ responsible for receiving information, therefore many researches were devoted to examine these diseases, their early signs, and how could they be stopped. From the start of 21th century, methods aided by computer were more and more involved in these processes, up to the current trend of using Convolutional Neural Networks (CNNs). While supervised methods, CNNs do achieve accuracy which can be compared to a skilled ophtalmologist, they require a tremendous amount of labeled data which is sparse in medical fields because the amount of time and resources needed to create them. One natural solution is to augment the data present, that is, copying the distribution while adding a small variety, like coloring an image differently. That is, what our paper aims to explore, whether a texturing algorithm, the Neural Style Transfery can be used to make a data set richer, and therefore helping a classifier CNN to achieve better results.

1. Introduction

The eye is one of the most important parts of the human body as the majority amount of information gained by perceiving the world around us. Sadly, there are numerous diseases could threaten this organ, and many of them do not result in immediate loss of eyesight, but damage the tissues and other parts of the eye over a long period of time. These diseases are more common among the elderly, and as average lifespan increased in the 20th century, they

Received by the editors: 4 December 2020.

2010 *Mathematics Subject Classification.* 68U10, 68T01, 92B20.

1998 *CR Categories and Descriptors.* I.4.6 [**I.2 Image Processing and Computer Vision**]: Segmentation – *Region, partitioning*; I.5.1 [**Pattern Recognition**]: Models – *Neural Nets*.

Key words and phrases. Medical Image Processing, Retinal Blood Vessel Segmentation, Neural Style Transfer.

became more relevant to combat. Such disease is *diabetic retinopathy* (DR), which are the leading cause of blindness in developed countries, according to the World Health Organization (WHO).

On the bright side, there are number of indicators of DR, which can suggest early treatments, that slow down or even halt vision loss. These signs can detected during *dilated fundus examination* by screening the retina with a fundus camera, gaining information about the retina, blood vessels, etc.

This is done with *fluorescein angiography*: a small amount of sodium fluorescent dye is given to the patient orally or via injection. Then the retina gets illuminated with blue light, and green light reflected back by the dye reveals the blood vessels. Although the method is known to be reliable and accurate, it can cause side effects, such as nausea or anaphylaxis.

To offer an alternative solution which is more comfortable for the patients, researches began in search of *blood vessel segmentation methods* on retinal images. The task is to create a classifier algorithm, which, upon seeing a fundus image, accurately labels each pixel as vessel or non-vessel. To be reliable, such algorithms require a huge amount of training examples with ground truth labelling, as the classifier's parameters need to be set. Unfortunately, segmented fundus images are expensive and time consuming to produce, because the process to create one involves trained ophthalmologists.

The area of data augmentation deals with the *problem of lacking data* in quantity. A natural solution is to create new images using certain transformations on the original ones: color enhancing, whitening, adding noise, etc. These transformations preserve spatial attributes, therefore the new image has the same segmentation as its origin. Also, there are some other techniques, such as cropping, rotation or flipping the images, executing the same on the segmented image.

After this introduction, our paper will go as follows: in the next two sections, we will give a short description of the already known supervised methods for segmenting retinal images, and also, we will introduce Neural Style Transfer (NST) algorithm, emphasizing it as a data augmentation technique. In section 4, we will go through our experiment of using NST to create fundus images, the results and conclusions will be presented in section 5 afterwards.

2. Related works

2.1. **Segmentation methods.** The area of retinal blood vessel segmentation is rich in researches, from the late 80's until today. The very first methods were *rule-based*, which means that certain areas were marked as vessel, if pixels in it had a common property, followed a shape, etc. Later on *superwised methods* took a major role, with trained classifiers achieving accuracy over 90% and more.

Such one rule based method was *matched filtering*, dating back to, where researchers discovered that the cross section of a blood vessel follows the shape of a Gauss-curve, speaking of grayscale pixel intensity value. While being easy to interpret, this method achieved good results in 1989 [2], and was improved later, see papers [1, 19]. An other rule based method features *Multi-Scale Line Detection*, which builds upon an even more simple observation: blood vessels are made of linear segments piecewise [12].

With *Neural Networks* (NN) became popular in the last decade, more and more works were devoted to explore their abilities in the topic of discussion. Marín et al. [11] used a small, 3-layer deep NN feeded with 7-dimensional feature vectors to measure the probability of a pixel being blood vessel in the image. The 7 features consist the local average, minimal, maximal and center intensities, as well as variance and Hu-moments.

We follow the work of [18], details explained along our paper. For a thorough review, look up [4].

2.2. **Neural Style Transfer.** While seemingly confusing to mention here, NST has its relevance in our topic. The idea to create artistic images with the aid of a computer is not new, but the first truly successful attempt was only in 2016 by Gatys et al. [5, 6]. Since then, more than a hundred research papers were dedicated to explore the capabilities of the method. For a thorough review until the near end of 2018, look up paper [8].

The goal of NST is to create an image, given a content image and a style image, with the restriction that the result has to have the similar semantic information (what we actually see on the image) as the content and also similar textures (colors, shapes, etc.) as the style. The original NST, that we used, is an image optimisation technique, which means that the algorithm starts from an initial white-noise or the content image, and in each iteration small adjustments are done to match content and style respectively. Without delaying further, the original NST can be described the following way, picture [1] attached:

(1) The core is made of a pretrained deep neural network. Newer frameworks exploit the capabilities of the VGG19 because its success on image recognition tasks [17], that it can already classify images into a vast number of categories.

    As an image is passing through the network, the responses of the kernels are accumulated in *feature maps* for each layer. The deeper the layer we are currently examining, its feature maps contain the more and more complex information about the image. The user must

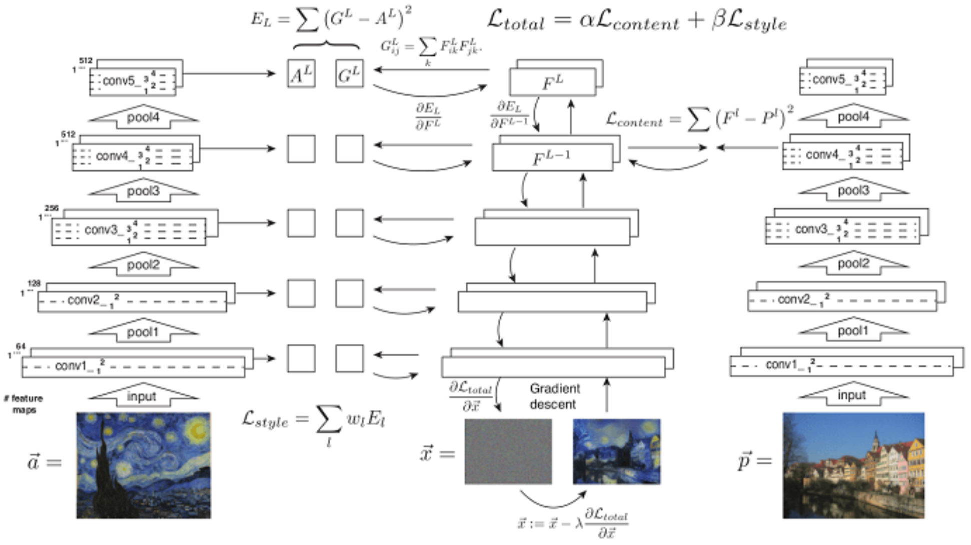

$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik}F^L_{jk}$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_3
conv4_3
conv3_3
conv2_2
conv1_2
pool4
pool3
pool2
pool1
input
512
256
128
64
# feature maps
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

FIGURE 1. An illustration of the Neural Style Transfer algorithm framework with the VGG16 network, originally used by Gatys et al.

choose a few layers to get these inner representations as outputs, with many guidelines had already been made to choose certain groups for perceptually pleasant results: one content representation layer is chosen in deeper sections, and multiple layers are chosen across the network for representing style.

(2) Let us denote the content representation feature maps by $P^1, \ldots, P^l$, with the same indexing on the result, let it be $F^1, \ldots, F^l$. The difference between actual contents is expressed as the L_2 distance of the feature maps, the sum of the element-wise difference squared. This is simply written here as:

$$\mathcal{L}_{content} = \sum_{i=1}^{l}(F^i - P^i)^2$$

In our work, we chose only the *block5_conv2* convolutional layer for content representation in the VGG19 network.

(3) Style matching is done in a slightly different way. For both style and result image, their *Gram-matrices* are calculated from features maps, denoted by $A^1, \ldots, A^l$ and $G^1, \ldots, G^l$. These matrices represent correlation between features on an arbitrary image, therefore the task

is to pull these correlations closer feature-wise:

$$\mathcal{L}_{style} = \sum_{i=1}^{l} (G^i - A^i)^2$$

It is worth noting that since its 2016, this way of representing style came through many refinements. Consider reading paper [10] for a better understanding on style matching, and papers [15, 7] for additional techniques like color-histogram matching and total variation loss. In our work, we chose the first convolutional layer from each block: *block1_conv1*, ..., *block5_conv1*.

(4) The total loss is weighted sum of the two previously defined losses:

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

With respect to the pixel values of the result image as variables, $\mathcal{L}_{total}$ is differentiable, therefore optimal intensities can be calculated via back-propagation.

* We must also mention, that improvements were also done to speed up the work of NST, see paper [9].

Surprisingly, NST was not a subject of researches related to direct data augmentation until the end of 2019. The first experiment was to measure, if NST can improve a classifier's performance by creating stylised images, therefore the training dataset will have more variance, see paper [20] for further details. In 2020, NST was also used in medical fields for dermatological data augmentation, with the same motivations there as mentioned in the introduction, see paper [13].

3. Proposed Method

Our hypothesis is that a sufficiently used NST can be used to synthesize retina images with style to gain more variance, and therefore a classifier trained on the augmented data would be more robust to outliers, creating less false positives. To test this, we executed the following plan:

(1) In the beginning, we use the original 20 DRIVE images [3] to measure the performance of the classifying Convolutional Neural Network (CNN). The results we are mainly looking to improve is specifity and training time, see section 5 for explaining evaluation metrics.
(2) We create stylized images with the NST algorithm. For getting desirable results, one must address many properties of the algorithm: losses and respective weights, the way of representing style, etc. In the end, we will have 20 times the number of styles images.

(3) The CNN is re-trained, but now on the augmented dataset, and we compare the two classifier's performances.

4. Experiment

4.1. **Used frameworks.** The CNN we used in our experiment, called *Retina U-net*, was implemented by Orobix, resources can be found at [14]. The idea of the U-network was first presented in paper [16], with the motivation to create encode-decoder framework for medical image processing. The network in subject could be marked as a tiny U-net, as its size does not even approach the one presented in the original, having only ≈ 470.000 parameters.

The network processes retinal images in patches with size 48×48 pixel, encodes them in convolution-dropout-convolution-maxpooling manner, and decodes with upsampling in the end of the same structure, convolution-dropout-convolution-upsampling. All convolutions are 3×3, dropout layers were used with 0.2 probability. In the beginning we start with 32 kernels and double the numbers after each maxpooling, up to 128, and halving after upsampling, back to 32. We used the *Adam* optimizer to train the network, with no final activation and Binary Cross-Entropy loss functions.

The predicted images were thresholded to gain binary images, this cut-off was set in the interval $[0.15; 0.25]$ after multiple trials.

Retinal images were obtained from the DRIVE database. The database consists of 40 fundus images, 20 for training and 20 for testing purposes, each given with a manually segmented blood vessel map, as well as an image mask that separates the background. Each image has the size of 584×565 pixel.

In the beginning, we measured the stand-alone performance of the Retina U-net by training it on 10.000 patches (500 extracted randomly from each image, see pictures 2) and evaluating it later on the test patches. Test patches were obtained by first expanding the test images to the size of 624×576, and then making 13×12 regular patch cuts.

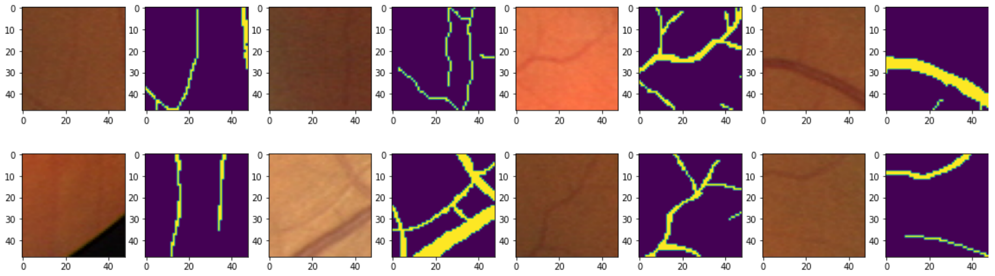


Figure 2. Patches extracted from retinal images

After initial result, we now turn to use the NST algorithm. We chose 3 style images: *Composition VII* from Wassily Kandinsky, *Starry Night* from Vincent van Gogh and *The Great Wave Off Kanagawa* from Hokusai, see pictures 3. The content and style loss weights were chosen 10^{-6} and 10^{-3} respectively and optimization process were executed also with *Adam* on the VGG19 network.

FIGURE 3. Used styles

After stylization, the same patch extraction-training-evaluation procedure was executed, see the results in the next section and examples of stylized retina in the appendix.

Upon applying the transformation to the training set, the difference shift in colors was measured in the fundamental way: calculating the mean and standard deviation for each color channel. This is due to get insights, expectations before going through the classification process again:

Looking at the values, what first thing that meets the eye is that for each transformed image, the mean values are much more regular, are closer to each

TABLE 1. Average value and standard deviation of colors among the training images, compared with the augmented ones (scaling from 0 to 255)

	R	**G**	**B**
Original	181.89	97.78	57.37
	45.19	28.57	17.26
Composition VII	153.99	119.57	98.93
	23.00	19.12	15.97
Great Wave	145.10	118.12	104.28
	21.15	17.25	15.92
Starry Night	154.81	115.62	97.14
	23.41	19.00	17.48

other in contrast to the original training set: the red values decreased by about $30/255 = 11.76\%$, while the green and blue channel values increased by roughly $20/255 = 7.84\%$ and $40/255 = 15.68\%$. While this means shift occurs, each channel's standard deviation interval shrank to $[-25, 25]$. This will be presented with images in the appendix, but this means, that the same content, the blood vessels, must be extracted from the new, augmented images, which are more homogeneous in color, therefore the new CNN has to be more sensitive to small changes in order to perform well.

5. Results and discussion

5.1. **Statistical measures used.** The final results we calculated follow the traditional evaluation metrics and statistics used in image segmentation. Upon getting the model predictions, a usual thresholding scheme is used to obtain binarized images. These images were matched with their corresponding ground truth segmentation, and all four class scores are calculated: *True positives* (TP), *True Negatives* (TN), *False Positives* (FP) and *False Negatives* (FN). After this, the following measures were used:

(1) *Accuracy* (ACC): the overall performance of classifying a seen pixel correctly. While being a widely used measure, ACC has its limitations and can be misleading in cases, for example when the label set imbalanced.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

(2) *Sensitivity* (true positive rate, SENS) and *Specificity* (true negative rate, SPEC): both measures the percentage of misclassification in vessel and non-vessel cases respectively. High SENS can be interpreted that the classifier pays attention to small details, that is, it can find tiny vessels, while high SPEC means that it is robust enough not to be distracted with noise.

$$SENS = \frac{TP}{TP + FN} \quad SPEC = \frac{TN}{TN + FP}$$

(3) *Balanced Accuracy* (BACC): makes up for the imbalanced cases, where ACC is misleading, commonly said as average true predictive power.

$$BACC = \frac{SENS + SPEC}{2}$$

(4) *Precision* (PREC): another measure for positive prediction ratio besides SENS. In this case we compare true vessels pixels to those that

are marked as vessel, and get a view about how well the algorithm separates noise from blood vessels.

$$PREC = \frac{TP}{TP + FP}$$

(5) *Matthew's Correlation Coefficient* (MCC): while no single number can capture a classifier's performance, MCC is regarded to be one of the best so far. It ranges between $[-1; 1]$ with 1 perfect predictive power, 0 meaning randomness and -1 meaning failure.

$$\begin{aligned} MCC = & \\ & = \sqrt{PREC \cdot SENS \cdot SPEC \cdot NPREC} \\ & - \sqrt{(1 - PREC) \cdot (1 - SENS) \cdot (1 - SPEC) \cdot (1 - NPREC)} \end{aligned}$$

where NPREC is the negative class precision.

5.2. **Evaluation and Discussion.** The aformentioned final scores can be seen in table 2, where we took the average in both cases.

We can see that the augmentation technique indeed helped to improve our CNN's classification power, with returning positive difference in almost each category, except in SPEC. Although these results look promising with respect to employing NST in retinal image augmentation, we want to mention that this method seems far from done. We list some number of parameters that need to be set correctly:

- The network used is an unexplored part of NST, but representions do depend on that the underlying network previously learnt. Could it be, that a network trained on retinal images could perform NST better than the VGG-networks?
- Related to the previous, but a different network might need other layers to be chosen to represent content and style weights $\mathcal{L}_{content}, \mathcal{L}_{style}$ should be chosen accordingly as well.

Table 2. Scores on the original and the augmented dataset, and respective differences (positive means improvement on the augmented data)

	ACC	**PREC**	**SENS**	**SPEC**	**BACC**	**MCC**
Original Data	95.95	84.31	66.53	98.78	82.66	72.64
Augmented Data	96.49	84.44	74.17	98.65	86.41	77.09
Difference	0.53	0.13	7.63	-0.12	3.75	4.44

- The concept of style loss are examined thoroughly in previous works (e.g. *histogram losses*), and that is one thing could enhance a retinal image augmenting NST.
- If we follow the patch-based strategy, then there might be a number of patches, where creating synthesized images no longer affects the classifier's performance. Therefore, this image augmentation technique should be performed only when limited data is available.
- Additional metrics for measuring differences can be applied before retraining the CNN, to get more insights. One can mention the normalized cross-correlation, or measuring the euclidean distance of feature maps on a normally trained U-net.

This concludes our work. We have seen that there are ways to deploy NST in medical image processing and we are eager to see and continue with further improvements.

6. Acknowledgments

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] Al-Rawi, M., Qutaishat, M., and Arrar, M. An improved matched filter for blood vessel detection of digital retinal images. *Computers in biology and medicine 37*, 2 (2007), 262–267.

[2] Chaudhuri, S., Chatterjee, S., Katz, N., Nelson, M., and Goldbaum, M. Detection of blood vessels in retinal images using two-dimensional matched filters. *IEEE Transactions on medical imaging 8*, 3 (1989), 263–269.

[3] Drive: Digital retinal images for vessel extraction. https://drive.grand-challenge.org/.

[4] Fraz, M. M., Remagnino, P., Hoppe, A., Uyyanonvara, B., Rudnicka, A. R., Owen, C. G., and Barman, S. A. Blood vessel segmentation methodologies in retinal images–a survey. *Computer methods and programs in biomedicine 108*, 1 (2012), 407–433.

[5] Gatys, L. A., Ecker, A. S., and Bethge, M. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015).

[6] Gatys, L. A., Ecker, A. S., and Bethge, M. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2414–2423.

[7] Gatys, L. A., Ecker, A. S., Bethge, M., Hertzmann, A., and Shechtman, E. Controlling perceptual factors in neural style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 3985–3993.

[8] Jing, Y., Yang, Y., Feng, Z., Ye, J., Yu, Y., and Song, M. Neural style transfer: A review. *IEEE transactions on visualization and computer graphics* (2019).

[9] Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision* (2016), Springer, pp. 694–711.

[10] Li, Y., Wang, N., Liu, J., and Hou, X. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036* (2017).

[11] Marín, D., Aquino, A., Gegúndez-Arias, M. E., and Bravo, J. M. A new supervised method for blood vessel segmentation in retinal images by using gray-level and moment invariants-based features. *IEEE Transactions on medical imaging 30*, 1 (2010), 146–158.

[12] Nguyen, U. T., Bhuiyan, A., Park, L. A., and Ramamohanarao, K. An effective retinal blood vessel segmentation method using multi-scale line detection. *Pattern recognition 46*, 3 (2013), 703–715.

[13] Nyíri, T., and Kiss, A. Style transfer for dermatological data augmentation. In *Proceedings of SAI Intelligent Systems Conference* (2019), Springer, pp. 915–923.

[14] Orobix: Retina u-net. https://github.com/orobix/retina-unet.

[15] Risser, E., Wilmot, P., and Barnes, C. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893* (2017).

[16] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (2015), Springer, pp. 234–241.

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

[18] Wang, C., Zhao, Z., Ren, Q., Xu, Y., and Yu, Y. Dense u-net based on patch-based learning for retinal vessel segmentation. *Entropy 21*, 2 (2019), 168.

[19] Zhang, B., Zhang, L., Zhang, L., and Karray, F. Retinal vessel extraction by matched filter with first-order derivative of gaussian. *Computers in biology and medicine 40*, 4 (2010), 438–445.

[20] Zheng, X., Chalasani, T., Ghosal, K., Lutz, S., and Smolic, A. Stada: Style transfer as data augmentation. *arXiv preprint arXiv:1909.01056* (2019).

Appendix A

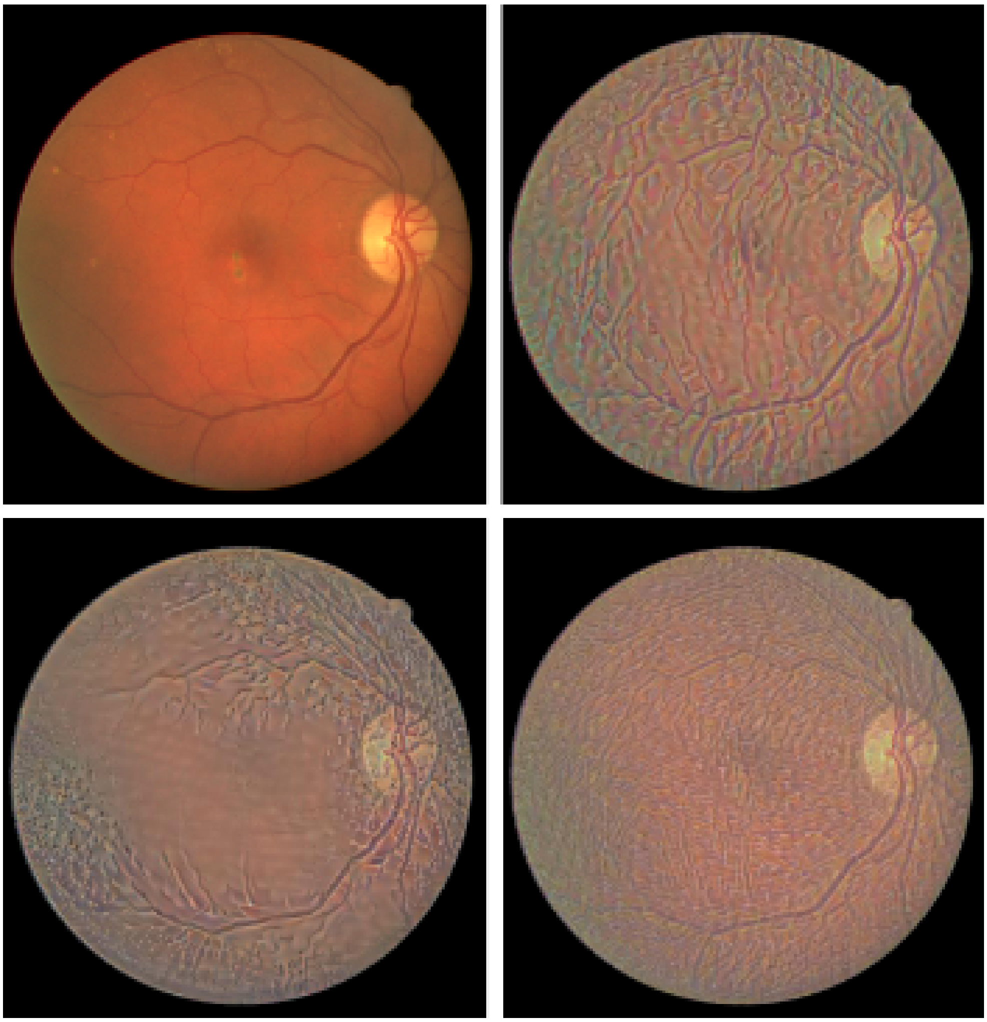


Figure 4. Retina images with corresponding stylization, from left to right row-wise: original, Composition VII, Great Wave, Starry Night

ELTE Eötvös Loránd University, Budapest, Hungary
Email address: `p3kxga@inf.elte.hu`

J. Selye University, Komárno, Slovakia
Email address: `kissae@ujs.sk`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.06

TEMPORAL DISCOUNTING FOR MULTIDIMENSIONAL ECONOMIC AGENTS

FLORENTIN BOTA

Abstract. Individuals frequently place a higher value on money and goods today than they would in the future. This is known as temporal or time discounting, and most economic models include discount functions to represent such utility over time.

In this paper we evaluated traditional models with experimental data from the scientific literature and constructed our own samples for comparison. In addition, we evaluated the prediction accuracy of the models and proposed new hybrid solutions. Our investigation aims to contribute to a better understanding of human nature in complex processes.

1. Introduction

We propose a new unified computational model that represents the human decision-making process in complex systems, such as economy. The model is created using a bottom-up, data-driven approach[5] and will provide an effective tool for developing realistic Multidimensional Economic Agents (MEA). This paper focuses on the *rational* part of the model, where we analyze the standard paradigm, specifically the time-preference or temporal discounting phenomenon observed in economy.

Intertemporal choice is an elegant and simple economic theory introduced by John Ray in 1834 and formulated by Irving Fisher (1930) [9, 25], who created a model that includes an individual's impatience, contrary to Keynes (1936) [13], who emphasized on current income in relation with consumption. In this model the consumers make time-based decisions to maximize their lifetime satisfaction.

Figure 1 is a representation of intertemporal choice of the consumer subject to the utility preferences and the budget constraint. The *indifference*

Received by the editors: 3 April 2021.

2010 *Mathematics Subject Classification.* 68Q19, 91B44.

1998 *CR Categories and Descriptors.* J.4 [**Computer Applications**]: Social and Behavioral Sciences – *Economics*; I.6.4 [**Computing Methodologies**]: Simulation and Modeling – *Model Validation and Analysis* .

Key words and phrases. economics, temporal discounting, intertemporal decisions.

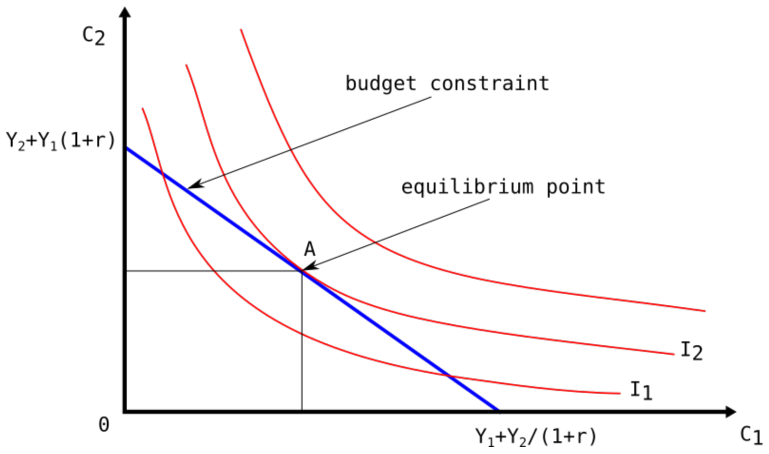


FIGURE 1. Intertemporal choice exerted by the consumer

curves[19] $(I_1, I_2 ... I_n)$ can be part of an *indifference map* and illustrate different bundles of goods between which a consumer is indifferent. There is an infinity of indifference curves in *microeconomic theory* and in this case they represent different utility levels. The slope of the indifference curve is the *marginal rate of substitution* [1] of goods consumption in different time periods (how many extra goods would you need to consume in t_2 to give the same level of utility if you consumed one in t_1).

An interesting concept of this model is that we can consume now money that we make in the future by borrowing that amount. The graph is a model of saving and consumption with the interest rate of r over time periods (C_1, C_2 represent consumption over periods of time and Y_t represents income for time t). The consumer will have to maximize utility $U(C_1, C_2)$ but under the constraint represented below:

$$C_1 + C_2/(1+r) = Y_1 + Y_2/(1+r) \tag{1}$$

Human behavior is inherently difficult to model, due to the dynamic interactions that can be observed between agents. However, experimental data shows that there are (unwritten) social rules that can be used to model human reactions in some environments[1].

In this paper I will discuss the existing discount functions and our new proposed hybrid models. We conducted our own experiment for sample data

[1]The marginal rate of substitution (MRS) represents the rate at which economic agents will substitute one good for another while maintaining the same utility .

and tested the proposed functions. The findings showed better results and a good contribution to our multidimensional economic agent.

The remainder of the article is structured as follows. Section 2 is focused on the concept of temporal discounting and other related papers. Section 3 will cover current experiments from the scientific literature and the zero-shot capability of GPT-3 regarding time preference. Section 4 describes our proposed functions and the methodology we employed, then in Section 5 we will examine the results. Section 6 presents our conclusions and directions for future research.

2. Background

2.1. **Temporal discounting.** Temporal discounting (*delay discounting, time preference*) refers to the observed phenomenon where agents value the same good differently based on the time of consumption. Certainly, a good can be any product that is desired and provides *utility*[2] to a consumer.

There are many models that represent temporal discounting or time preference. The models are simplified mathematical versions of discounting in a complex system and are used to explain, analyze and predict behavior and interactions of economic agents. Several differences between the models exist, specifically between the discount functions and we will present some of the most used variations.

The model we presented earlier assumes that the consumers make choices by discounting the present value of their consumption and income exponentially into the future, using the same interest rate. In other words, in ideal markets, both firms and individuals borrow or lend until their marginal rate of substitution between consumption today and consumption tomorrow equals the interest rate [25, 12].

Before continuing to mathematical formulae of the function, we should explain several important economic terms for those without an economic background. The *discount factor*, let's call it δ, is the amount a future value must be multiplied with in order to get the present value and can be defined with the formula

$$\delta = \frac{1}{1+\rho} \tag{2}$$

We can extract the *discount rate* as

[2]In economics, *utility* is a measure of the total satisfaction received from consuming a good or a service. It was introduced by Daniel Bernoulli in 1738

$$\rho = \frac{1-\delta}{\delta} \tag{3}$$

The discount rate ρ refers to the interest rate used in discounting. More intuitive examples can be found in the fourth chapter of *Behavioral Economics* by Edward Cartwright (2011) [7] or in the scientific literature in general. A more general equation for future cash flows is:

$$PV = FV * \frac{1}{(1+r)^n} \tag{4}$$

where PV = Present Value, FV = Future Value, Discount Factor = $\frac{1}{(1+r)^n}$, n = time and r = discount rate

2.2. **Exponential discounting.** Exponential discounting is a very simple way to model choice over time and is by far the most common way used in economics, because of it's simplicity. The standard economic model of exponential discounting was proposed by Samuelson in 1937[23] and the general formula is:

$$f_E(D) = e^{-kD} \tag{5}$$

where $f(D)$ is the discount factor, D is the delay and k is a parameter which determines the rate at which value decreases with the time delay. A larger k can be associated with a steeper discounting of the value of a future reward[11].

We can define total utility in this case by:

$$u_T = \sum_{t=1}^{T} \delta^{t-1} u_t \tag{6}$$

where u_t is the utility in time t, and δ is the exponential discount factor. We can rewrite that in continuous time as:

$$u^T = \int_0^T e^{-\rho t} u_t \tag{7}$$

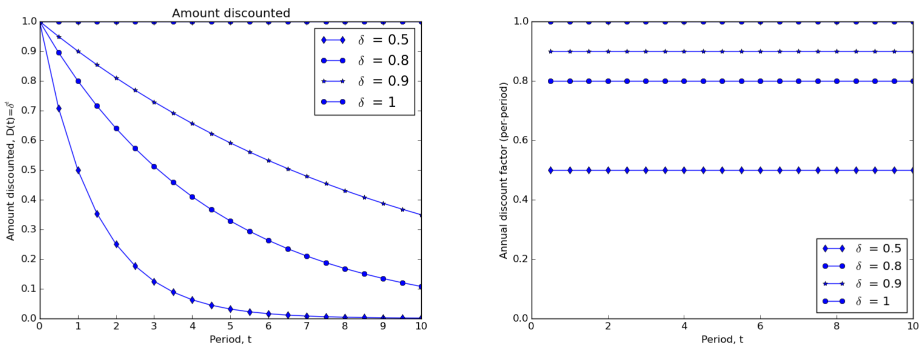
In Figure 2a we plotted the exponential discount function for different values of δ. The top line is the theoretical limit where no discounting occurs.

There are several anomalies in this model[17], recognized even by Samuelson[23] when he proposed the DU model. He stressed that "it is completely arbitrary to assume that the individual behaves so as to maximize an integral of the form envisaged in (7)"

One of the most prominent anomalies is the *constant rate of discount assumption.* Empirical evidence suggests that discount rates fall over time. We can prove the constant rate by calculating the change in discounting over two consecutive periods of time:

$$\frac{D(t)}{D(t-1)} = \frac{\delta^t}{\delta^{t-1}} = \delta \tag{8}$$

We can observe the constant factors in Figure 2b.



(A) Exponential discount function δ^t versus t

(B) Annual discount factor (per-period), $e^{\ln(f_E(t))/t}$

Figure 2. Temporal discounting

2.3. **Hyperbolic discounting.** With the exponential discount function we assumed a constant discount factor δ and "discount rate"[21] ρ so that:

$$\delta_t = \delta^t = \left(\frac{1}{1+\rho}\right)^t \tag{9}$$

Much of the empirical data from both humans and animals contradicts the predictions of exponential discounting [10]. An alternative notion of *hyperbolic discounting* was developed by psychologists (Ainslie, 1975; Chung & Herrnstein, 1967; Herrnstein, 1981; Rachlin, 1970), and **Mazur** (1987) formalized the current standard hyperbolic model [24]:

$$V = \frac{A}{1+kt} \tag{10}$$

where k is a discounting parameter that scales the degree of preference for immediate rewards. Hyperbolic discounting corresponds to simple interest[22].

Hyperbolic discounting also permits time-inconsistency. For example you may agree to wait an year and a month for a larger cash prize instead of taking a smaller prize after an year, but you might change your mind after an year and take the money. The Mazur hyperbolic discounting model was initially developed to describe pigeon data and tends to "overpredict subjective value at shorter delays, while underpredicting it at longer delays" [16].

Several researchers have modified his model by adding more parameters to better fit the data. **Rachlin** (2006) added an exponent σ to the time delay, which allows a more flexible relationship between value and delay:

$$V = \frac{A}{1 + kt^{\sigma}} \tag{11}$$

2.4. **Quasi-Hyperbolic discounting.** The "quasi-hyperbolic" discount function, proposed by Laibson (1997) [14] as the "Golden Eggs" model assumes that demand follows profits, and it illustrates why consumers have asset-specific marginal consumption propensities. According to the model, financial creativity may be to blame for the continuing downturn in US savings rates.

$$f_{QH}(D) = \beta \times \delta^{D} \tag{12}$$

3. Experiments and related work

3.1. **Thaler experiment.** Thaler [25] conducted a study where he asked respondents to state an amount that would be equivalent to receiving \$15 now. The time periods were one month, one year and ten years. The average response was \$20 for one month, \$50 for one year and \$100 for 10 years (We can calculate the corresponding annual discount rates (R): 345%, 120% and 19%)[10]. The same pattern was found by Uri Benzion, Amnon Rapoport, and Joseph Yagil (1989), Gretchen B. Chapman (1996), Chapman and Arthur S. Elstein (1995), John L. Pender (1996), Daniel N. Heller (1993), Stevens, Jeffrey R. (2016).

We used:

$$R = \frac{\ln(\frac{FV}{PV})}{t}[*100] \tag{13}$$

where R is the annual discount rate in percentage, FV is the future value, PV is the present value and t is the time period.

The rates were calculated using compound interest with continuous compounding: $FV = PV * e^{Rt}$, where R is the decimal equivalent of the rate of interest expressed as a percentage and t represents time.

The e constant was discovered by Jacob Bernoulli in 1683 by studying compound interest. The problem was as follows: "An account starts with \$1 and pays 100 percent interest per year. If the interest is credited once, at the end of the year, the value of the account at year-end will be \$2. What happens if the interest is computed and credited more frequently during the year?"

Bernoulli noticed that with continuous compounding the account value will reach \$2.7182818....

3.2. **Benzion et al. experiment.** Another anomaly is the *magnitude of payoff effect.* The model implicitly assumes that an individual's rate of discount is independent of the size or magnitude of a payoff. Once again, the empirical studies suggests that individuals discount less when faced with larger payoffs. Benzion, Rapoport and Yagil (1989) examined this aspect by varying the amount of money (\$40, \$200, \$1000, \$5000) and the time periods (0.5,1 2 and 4 years). They found in all scenarios that the discount rates decrease as the amount of money increases (0.228, 0.18, 0.16, 0.123 for a two year period). In Figure 3, the inferred discount rate is :

$$R = (F/P)^{1/t} - 1 \tag{14}$$

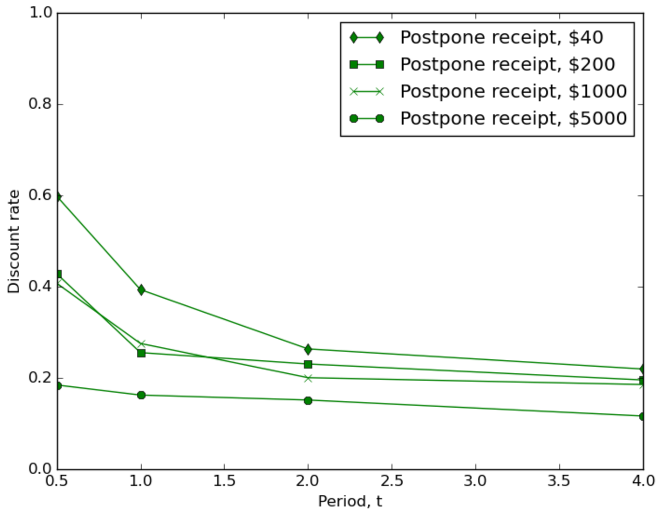
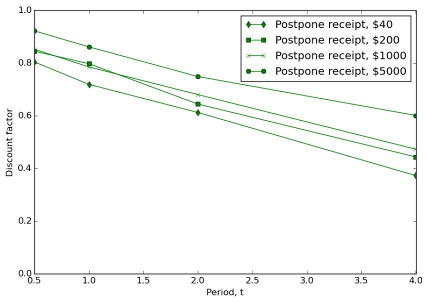


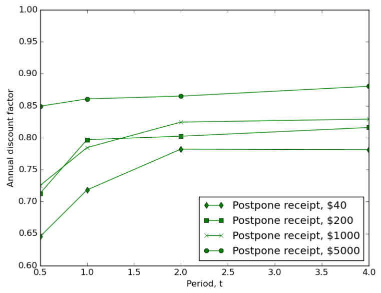
FIGURE 3. Discount rate - Benzion et al.

In Figure 3 we plot the results of scenario A from Benzion et al. [2] (postpone a receipt), where a person has just earned \$y for his or her work financially solid public institute. Instead of receiving the money, the person is told that there is a temporary shortage of funds and is assured payment of another amount of \$y over t times periods from now.

If we take the raw average responses and calculate the discount factor ($DF = PV/FV$), we can plot the results from Figure 4a. With that we compute the annual discount factor from Figure 4b [7]. Figure 4b shows that there is a short-term impatience (The discount factor is higher the longer they had to wait) and an *absolute magnitude effect* (the larger the sum of money, the larger the estimated discount factor).



(A) Amount discounted



(B) Annual discount factor

FIGURE 4. Benzion et al. experiment

Next find the best parameter δ from $D(t) = \delta^t$ that fits the experimental data from Figure 4a. We plot the resulting exponential function in Figure 5.

3.3. **Benzion and Yagil experiment.** This experiment[3] conducted by Benzion and Yagil reexamined the behavior of subjective discount rates across several dimensions: financial scenario, time delay and the monetary sum of the cash flow. They used subsamples of 105 subjects from undergraduates, graduates and higher academic degree. The emerging pattern is similar with other experimental data from literature: the discount rates are decreasing with the time delay and the sum of the cash flow, and are higher for a postpone-a-receipt scenario than for a postpone-a-payment scenario. We can observe their results in Figure 6a, where the mean discount rates are plotted over time t. In their survey they used a scenario A (postponing a receipt), scenario B(postponing a payment), 3 time periods (0.5, 2 , 5 years) and 3 sum variations (\$200, \$600, \$5000)

Based on (14) we can determine the future discounted value:

$$F = P(1 + R)^T \tag{15}$$

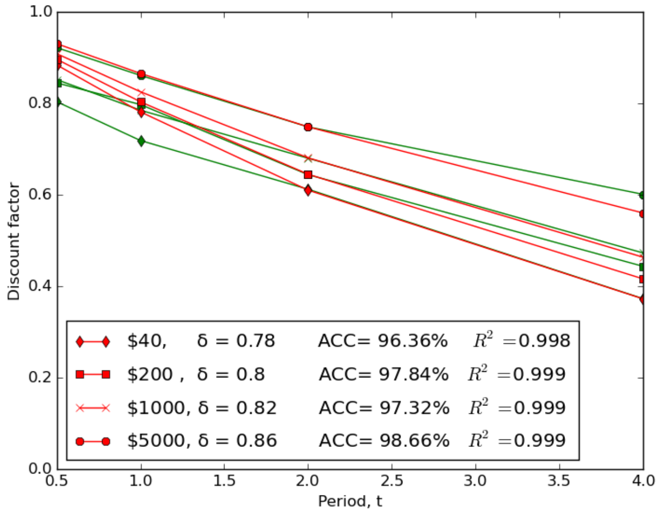
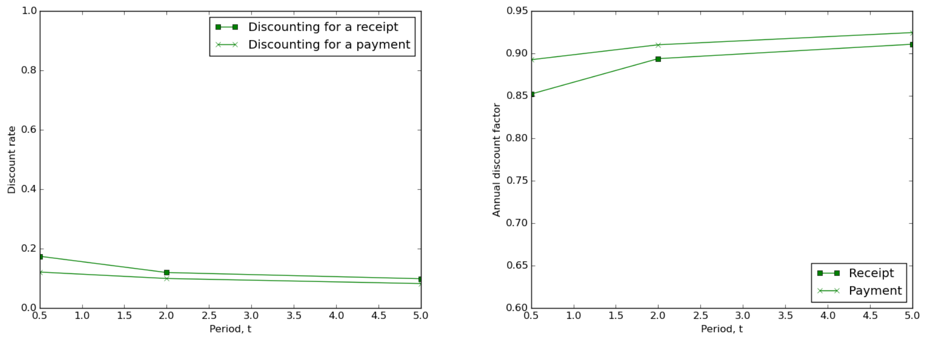


FIGURE 5. Fitted exponential function - Benzion et al.



(A) Amount discounted (B) Annual discount factor

FIGURE 6. Uri Benzion and Joseph Yagil Experiment

where F is the future value of a cashflow, P is the present value, R represents the discount rate and T the time period.

3.4. **A general model of temporal discounting.** Wouter and Samuel [4] argued that certain behaviours like impulsivity are inexplicable with classic models. As a result, psychological models of temporal discounting have now effectively replaced classical economic theory.

This is consistent with our own results in Section 5 and represents a positive context for our own proposed solution in Section 4.2.

In [4] they presented a brain-based discounting model that overcomes some constraints, while retaining much of the practical structure of the hyperbolic discount equation.

They used neuroscience-based theory to create a new model that accounts for several well-known contextual effects, with a simplified discount function seen in (16).

$$D_\tau = \omega\delta_1^\tau + (1-\omega)\delta_2^\tau \tag{16}$$

where ω indicates the relative involvement of each system in a given decision (McClure, Ericson et al.)[15]

3.5. **GPT-3 Experiment.** Generative Pre-trainer Transformer 3 (GPT-3) is a language model developed by OpenAI[6], an artificial intelligence research and deployment organization. The model uses historical values to forecast future data (autoregressive) and is based on feed-forward neural networks. GPT-3 is trained with 175 billion parameters, making it a state-of-the-art language model and the largest one at its launch, in 2020. The previous largest model was Microsoft's Turing NLG, with 17 billion parameters, 10 times smaller than GPT-3[20].

Their scaled up approach significantly enhanced task-agnostic, few-shot efficiency[6], competing with other state-of-the-art fine-tuning models[18].

Although this model is usually employed in NLP use cases, the generative attribute with the options of zero-shot and few-shot learning make it a good candidate for our experiment. I could not find any examples of such studies in the literature being conducted so far.

We tested the standard GPT-3 model in making temporal choices by using the OpenAI Playground and the Q&A preset with *temperature* set initially to 0. Temperature is a parameter for stochastic values and controls the randomness of the response. A value of 0 causes the engine to be deterministic, which means it will always produce the same output for a given input text, and a value of 1 causes the engine to take the most chances and use the most *imagination.*

There are variations in this strategy and we noticed them right away. We were able to communicate with the model using written English and receive written answers as responses, equivalent to our human survey.

We used adapted questions from our survey, to make things easier for the model. For example:

”Would you prefer \$ 50 now or \$ 500 in a year?”

The answers were formulated like this:

”I would prefer \$ 50 now.”

Table 1. GPT-3 Q&A answers for temporal choices

Value now (\$)	In a year (\$)	Answer	Temperature
50	500	”I would prefer \$ 50 now.”	0.0
50	5000	”I would prefer \$ 50 now.”	0.0
50	50000	”I would prefer \$ 50 now.”	0.0

We experimented with the model’s temperature parameter and quickly noticed that the zero-shot version (no added training) always prefers the offered non-zero value in the present, regardless of the future amount.

By training the model with several examples we are able to simulate more human-like responses and this will be the main research for a future study in our MEA project.

4. Methodology

In this section, we will discuss the approach we used to validate existing models as well as the experiments we conducted to demonstrate the accuracy of our proposed hybrid model.

We find an optimum δ for the exponential function and plot the results in Figure 7. In the following sections we will also use this data to measure predictive accuracy between the models and the experimental data.

4.1. The dataset.

After our study of scientific literature, we were concerned that most of the articles and experiments offered interpreted data with average discount rates and average responses. We wanted sample raw data with original answers to get a better understanding of the topic.

Therefore we conducted a survey with students from Babes-Bolyai University of Cluj-Napoca and other participants from an on-line community (reddit). There were a total of 118 responses, with 52 respondents being computer science undergraduate students and 66 online respondents from Europe and USA. We used Google Forms as a survey tool, to gather the responses.
By using the template:

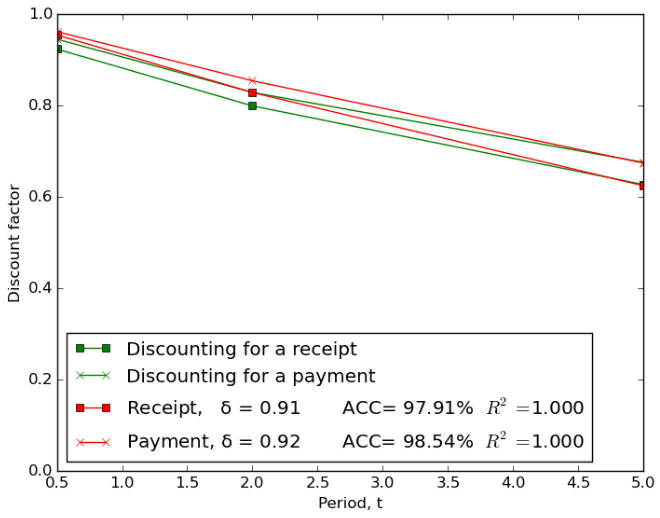


FIGURE 7. Fitted exponential function - Uri Benzion and Joseph Yagil, 2002

TABLE 2. Sample survey answers

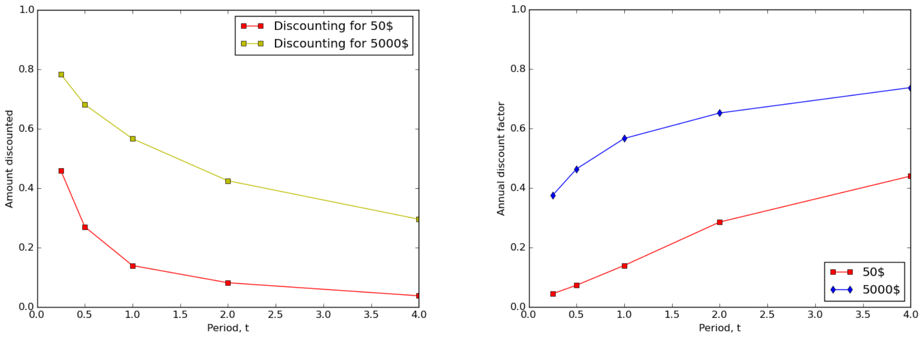
				50$	**5000$**				
3 mo	6 mo	1 yr	2yr	4yr	3 mo	6 mo	1 yr	2yr	4yr
200	400	600	1000	2000	8000	10000	15000	20000	35000
75	100	300	500	1000	5999	7420	8499	9999	14000
60	75	100	200	400	6000	7500	8000	15000	25000
65	80	100	150	300	6000	6666	8000	10000	12500
100	300	1000	2000	5000	10000	10000	12000	15000	20000
75	100	300	500	1000	5200	5500	6000	7000	10000
52	54	56	60	70	5200	5500	6000	7000	8000
300	500	700	1000	2000	6000	8000	10000	15000	20000
55	60	80	130	300	5100	5500	6000	7000	8000

You are indifferent to Y$ now vs X $ in t years. Write the X amount below we questioned the subjects to state indifference for receiving money over 6 months, 1 year, 2 years and 4 years periods. The money amount varied (\$50 and \$5000). We can observe a sample set with the answers in Table 2 below.

We specifically asked the following question:

> "You have a bank savings plan which just achieved maturity and the bank manger offers you the choice to invest again in another similar savings plan but with a different maturity time. What amount of money would make you COMPLETELY indifferent about receiving the relevant sum today or receiving a larger sum in the future?"

Based on the results in Table 2 we calculated the discount factors, which can be seen in Figure 8.



(A) Amount discounted (B) Annual discount factor

FIGURE 8. Discounted values in our experiment

As compared to older experimental evidence discussed in this paper, the findings show some variations, but they are are consistent with more recent research [8].

4.2. **Proposed functions.** In this context, we suggest two new functions that can be thought of as hybrids of existing, simplified models. We believe that by using multiple parameters, they can conform better to modern human behaviour, particularly given the anomalies we discovered in the literature.

Equation (17) describes **Hybrid Exponential-Hyperbolic** discount function:

$$hyb_{EH}(x) = \frac{\delta^x}{1 + (\alpha \times x)} \tag{17}$$

and equation (18) describes the **Hybrid Quasi-Exponential-Hyperbolic** discount function.

$$hyb_{QEH}(x) = \beta \times \frac{\delta^x}{1 + (\alpha \times x)} \tag{18}$$

In these functions, δ is the *discount factor* and α represents the discounting parameter that scales the degree of desire for instant gratification, with β being used to capture present-time bias. In our experiments, the parameters are estimated using the ordinary least squares (OLS) method.

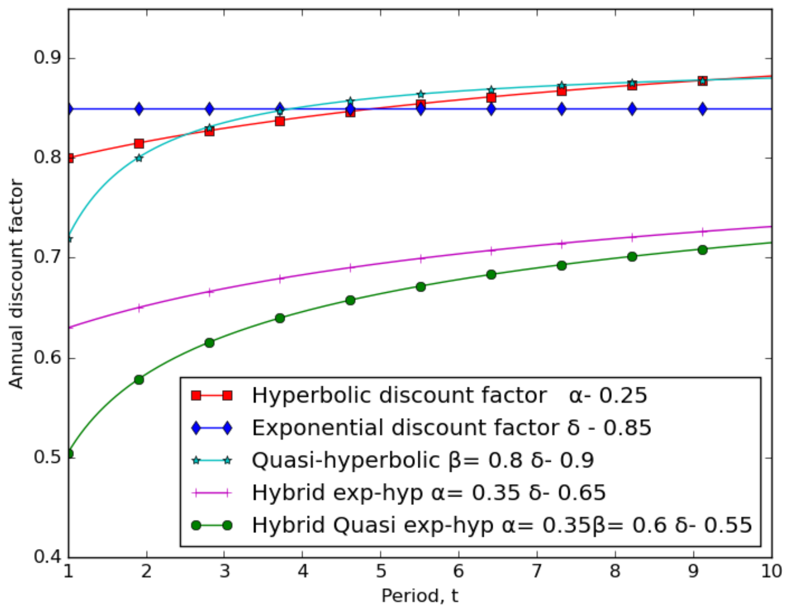


FIGURE 9. Annual discount factor, function comparison

Figure 11 shows a comparison of current approaches and our suggested hybrid functions for the annual discount factor.

We can observe similarities between the quasi-exponential-hyperbolic function and quasi-hyperbolic, and between exponential-hyperbolic with the hyperbolic discount function. As shown in the following section, the variations are important in our analysis.

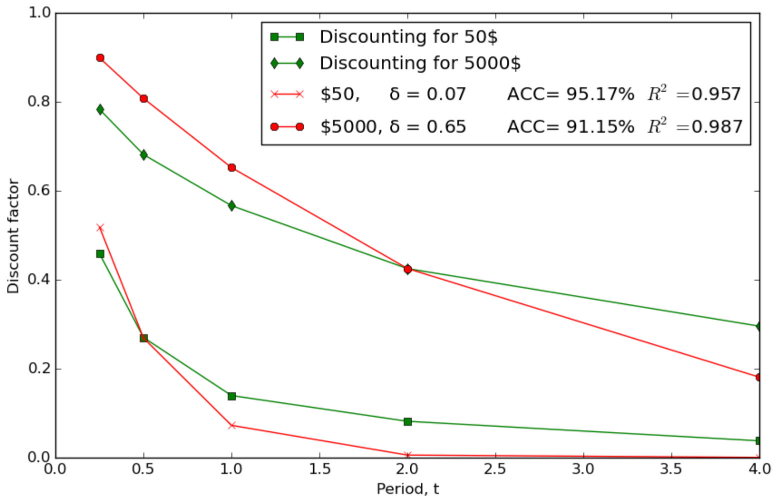


Figure 10. Fitted exponential function, using survey data

5. Results

We can observe the discount factors in Figure 10, where we fitted the exponential discount function to our data. The outcome shows a **95.17**% accuracy for the 50$ discount and **91.15**% for the 5000$ questions.

This suggests that the exponential discount function doesn't fit well on our experimental values, relative to the experiments of Benzion et al. in Figure 5. We believe this is due to the participants' lack of economic awareness or to inconsistencies in the model compared to our results. It also provides us with an ideal setting for testing the models in a real-world situation where data is sparse or uncertain.

A sample of the answers from our survey can be observed in Table 2. We tested all multiple functions on our survey data and the results can be observed in Table 3.

The visual representation can be seen in Figure 11, where our hybrid functions performed very well, with **99.39 %** accuracy for Quasi-Exponential-Hyperbolic and **98.65 %** accuracy for Exponential-Hyperbolic.

The standard model of exponential discounting did the worst, with **89.02 %**, which is consistent with other examples in the recent literature [4] [8].

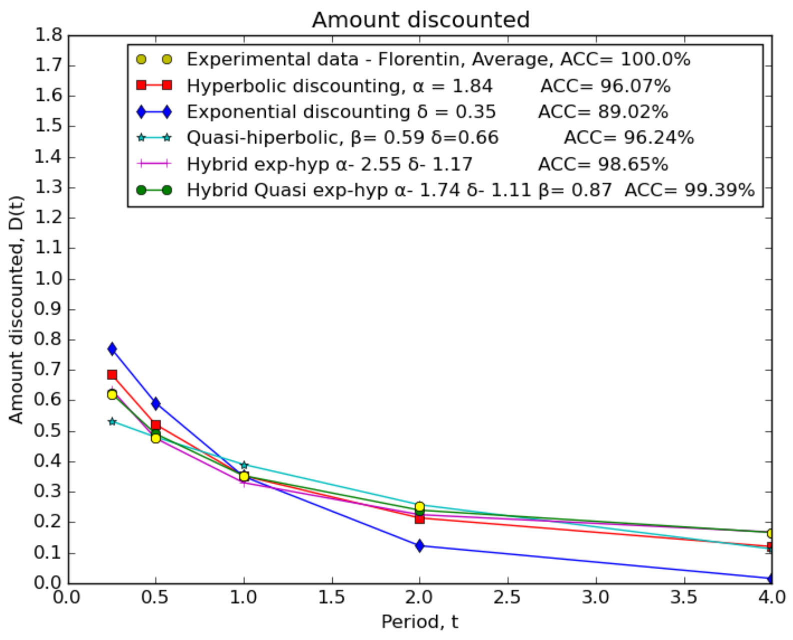


FIGURE 11. Accuracy on real data (average)

TABLE 3. Accuracy results for our experimental data

Discount function	ACC
Hybrid Quasi-Exponential-Hyperbolic	99.39 %
Hybrid Exponential-Hyperbolic	98.65 %
Quasi-Hyperbolic	96.24 %
Hyperbolic	96.07 %
Exponential	89.02 %

6. Conclusions and further work

In this article, we provided an analysis of the temporal discounting phenomena found in economic processes. Using studies from the literature, we evaluated existing discount functions and proposed new hybrid solutions based on our own experimental data.

Our proposed functions performed well and we determined very good results for the Quasi-Exponential-Hyperbolic function, with up to **99.39%** accuracy.

According to our findings, the standard functions performed well on older data sets and struggled to model behavior on newer data, while our proposed solutions show a very good potential to model consumers constraining their own future choices.

We also obtained intriguing results with zero-shot and GPT-3 learning in the form of inter-temporal preference (always accepting the present value), yet further investigation is required.

This paper represents an important phase in our MEA[5] research, in which we developed a theoretical model that can be used to simulate and forecast human actions in complex scenarios.

Further work will include a more in-depth analysis on few-shot learning models such as GPT-3 and their ability to simulate inter-temporal choices.

References

[1] Alahi, A., Ramanathan, V., Goel, K., Robicquet, A., Sadeghian, A.A., Fei-Fei, L., Savarese, S.: Chapter 9 - learning to predict human behavior in crowded scenes. In: Murino, V., Cristani, M., Shah, S., Savarese, S. (eds.) Group and Crowd Behavior for Computer Vision, pp. 183–207. Academic Press (2017)

[2] Benzion, U., Rapoport, A., Yagil, J.: Discount rates inferred from decisions: An experimental study. Management science **35**(3), 270–284 (1989)

[3] Benzion, U., Yagil, J.: Decisions in financial economics: An experimental study of discount rates. Advances in Financial Economics **7**, 19–40 (2002)

[4] Van den Bos, W., McClure, S.M.: Towards a general model of temporal discounting. Journal of the experimental analysis of behavior **99**(1), 58–73 (2013)

[5] Bota, F., Simian, D.: Embedding human behavior using multidimensional economic agents. In: Simian, D., Stoica, L.F. (eds.) Modelling and Development of Intelligent Systems. pp. 3–19. Springer International Publishing, Cham (2021)

[6] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)

[7] Cartwright, E.: Behavioral economics, vol. 22. Routledge (2014)

[8] Esopo, K., Mellow, D., Thomas, C., Uckat, H., Abraham, J., Jain, P., Jang, C., Otis, N., Riis-Vestergaard, M., Starcev, A., et al.: Measuring self-efficacy, executive function, and temporal discounting in kenya. Behaviour Research and Therapy **101**, 30–45 (2018)

[9] Fisher, I.: The theory of interest. New York **43** (1930)

[10] Frederick, S., Loewenstein, G., O'donoghue, T.: Time discounting and time preference: A critical review. Journal of economic literature **40**(2), 351–401 (2002)

[11] Green, L., Myerson, J.: Exponential versus hyperbolic discounting of delayed outcomes: Risk and waiting time. American Zoologist **36**(4), 496–505 (1996)

[12] Keller, L.R., Strazzera, E.: Examining predictive accuracy among discounting models. Journal of Risk and Uncertainty **24**(2), 143–160 (2002)

[13] Keynes, J.M.: General theory of employment, interest and money. Atlantic Publishers & Dist (2007)

[14] Laibson, D.: Golden eggs and hyperbolic discounting. The Quarterly Journal of Economics **112**(2), 443–478 (1997)

[15] McClure, S.M., Ericson, K.M., Laibson, D.I., Loewenstein, G., Cohen, J.D.: Time discounting for primary rewards. Journal of neuroscience **27**(21), 5796–5804 (2007)

[16] McKerchar, T.L., Green, L., Myerson, J., Pickford, T.S., Hill, J.C., Stout, S.C.: A comparison of four models of delay discounting in humans. Behavioural processes **81**(2), 256–259 (2009)

[17] Musau, A.: Modeling alternatives to exponential discounting (2009)

[18] Nagabandi, A., Kahn, G., Fearing, R.S., Levine, S.: Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 7559–7566. IEEE (2018)

[19] Pareto, V.: Manuale di economia politica con una introduzione alla scienza sociale (manual of political economy). Milano: Societa Editrice Libraria (1919)

[20] Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: Zero: Memory optimizations toward training trillion parameter models. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–16. IEEE (2020)

[21] Rasmusen, E., et al.: Some common confusions about hyperbolic discounting. In: Working Paper (2008)

[22] Read, D.: Intertemporal choice. Blackwell handbook of judgment and decision making pp. 424–443 (2004)

[23] Samuelson, P.A.: A note on measurement of utility. The Review of Economic Studies **4**(2), 155–161 (1937)

[24] Stevens, J.R.: Intertemporal similarity: Discounting as a last resort. Journal of Behavioral Decision Making **29**(1), 12–24 (2016)

[25] Thaler, R.H.: Some empirical evidence on dynamic inconsistency. Quasi rational economics **1**, 127–136 (1981)

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

Email address: `florentin.bota@ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.07

LEADER ELECTION IN A CLUSTER USING ZOOKEEPER

MANUELA PETRESCU

Abstract. This paper presents an algorithm for flexible and fast leader election in distributed systems using Apache Zookeeper for configuration management.

The algorithm proposed in this paper is designed for applications that do not use symmetric nodes so they need a specialized election process or for applications that require a more flexible approach in the leader election process. The algorithm proposes a different approach as it allows assigning prioritizations for servers in the cluster that are candidates to become a leader. The algorithm is flexible as it takes into consideration during the leader election process of the different server settings and roles, network properties, communication latency or specific application requirements.

1. INTRODUCTION

In general, distributed systems are designed to use symmetric nodes - all nodes have similar roles or responsibilities. However there are situations where a specific type of processing must be done on a single node, critical processes or there are situations when it is more efficient to do the processing on a single node at a time. In this case, in order to ensure a high degree of availability in case there is a failure of the leader node, any other viable node from the cluster can and should assume the leader role. So far, most election algorithms focused on efficiency in terms of size of communication between nodes and maximum latency until a leader is elected. Also, many algorithms assign a uniform role to each node during the election procession as each node can vote either for itself or for any other node, in general leading to a broadcast type of communication until a leader is elected.

In this paper we propose a different approach, whereby using a third party coordination tool we can complete the leader election in fewer steps but relying on a temporary election node.

Received by the editors: 25 September 2020.

2010 *Mathematics Subject Classification.* 68U99.

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

Key words and phrases. leader election, algorithm, cluster, ZooKeeper.

Why another algorithm?

The algorithms running over ZooKeeper subject raise interest as the scientific community is still trying to find a solution to improve their consistency and performance, as recently presented papers prove. In 'Strong and Efficient Consistency with Consistency-Aware Durability' (2020), ORCA algorithm is proposed[1]; ZabAA and ZabAC algorithms were proposed in [6]; ZabCT algorithm in [7]. ORCA is presented as a modified version of ZooKeeper that implements CAD (Consistency-aware Durability) and also cross-client monotonic reads. There are experimentally results that suggest that ORCA provides strong consistency while closely matching the performance of weakly consistent ZooKeeper[1]. The fact that a communication network latency influences the leader election process is treated in other research papers that implemented a prototype algorithm based on ZooKeeper in order to emulate wide area systems in which the transmission delays can have a huge impact over the efficiency[4]. Other research proposes a model based on watchers in ZooKeeper and define a watch as a trigger that causes an event to be dispatched to the client whenever the watched resource changes its state. Due to the fact that the processes are asynchronous and as a consequence, the network latency gives rise to multiple possible orderings of network messages; so the model was improved in order to enable consistency [2,3]. Another proposal for a leader election algorithm for replicated services that are based on a leader, updates propagation and client request was POLE (Performance-Oriented Leader Election)[5], the algorithm selects the leader depending on an application specificity. The specificity can be defined as a metric, for example the recovery time or request latency can be used. The Pole algorithm was evaluated using ZooKeeper and the results showed that just optimizing the latency of consensus does not translate into lower latency for clients. An important conclusion from our results is that obtaining a general strategy that satisfies a wide range of requirements is difficult, which implies that configurability is indispensable for practical leader election [5].

However, none of the above algorithms relate to applications that have apart from generic constraints (server capacity, network latency), other constraints, for example the new leader should belong to a cluster that is geographically located in a different cluster from the previous leader. This behaviour differentiates it from the other algorithms, thus, the proposed approach is generic and flexible.

2. Apache Zookeeper

Zookeeper is an open source Apache project, which was designed as a service that propagates changes in the distributed systems using an improved, reliable and easy to understand method. It offers a centralized service that provides configuration management capabilities, naming information, distributed synchronization, group services, configuration information and leader election receipts over clusters in distributed systems [10,11,14].

2.1. **Leader Election Process in Zookeeper.**

As soon as a new leader is elected, it begins to serve the client's requests. Each client request contains a command with data to be applied to the state machine. The leader appends the command to its log and begins the notification process for the other servers. After the log entry from the leader was replicated on the majority of servers, the leader applies the command in its state machine. In fact, the entry is committed and the leader sends an acknowledge message to the client and informs the other read replicas servers (followers). When a follower receives the acknowledge message regarding a committed entry, it updates its own state machine based on that message. The inconsistencies that might appear between the leader's log and the follower's logs are solved by pushing the server's log version to the follower's log versions [8,9]. The protocol used in case of network errors is that the leader should try indefinitely to send messages to the followers. Data consistency is guaranteed by timers usage, so the followers logs will contain only valid data [12].

In the following we present some definitions related to Zookeeper [15]:

- **znode** - The basic data structure used by Zookeeper. It can contain some data, additional z-nodes children and several attributes (creation time, version number, so on.)
- **zk-session** - A standard TCP session established between the client and the Zookeeper server. The Zookeeper server permanently monitors the session for interruptions or timeouts by sending periodical probes. If the client fails to respond within the configured timeframe, a session may be expired and all the ephemeral z-nodes are automatically removed. A connection is established with any Zookeeper node from the cluster. If the chosen node fails, the connection migrates to another available node. This is transparent for the client.
- **watches** - A zookeeper client can configure various watches on selected z-nodes so it is informed of any change happening on these z-nodes.

2.2. **Consistency guarantees.** According to the specification [15], Zookeeper provides the following consistency guarantees:

- **”Sequential Consistency** : Updates from a client will be applied in the order that they were sent.
- **Atomicity** : Updates either succeed or fail – there are no partial results.
- **Single System Image** : A client will see the same view of the service regardless of the server that it connects to. i.e., a client will never see an older view of the system even if the client fails over to a different server with the same session.
- **Reliability** : Once an update has been applied, it will persist from that time forward until a client overwrites the update. This guarantee has two corollaries:
  - If a client gets a successful return code, the update will have been applied. On some failures (communication errors, timeouts, etc) the client will not know if the update has been applied or not. We take steps to minimize the failures, but the guarantee is only present with successful return codes.
  - Any updates that are seen by the client, through a read request or successful update, will never be rolled back when recovering from server failures.
- **Timeliness**: The clients view of the system is guaranteed to be up-to-date within a certain time bound (on the order of tens of seconds). Either system changes will be seen by a client within this bound, or the client will detect a service outage.”

By providing the above mentioned consistency guarantees Zookeeper can be used to build higher-order primitives such as queues, locks, two-phase commit protocols and leader elections for other solutions.

2.3. **Leader election in ZooKeeper using SEQUENCE—EPHEMERAL flags algorithm.** In ZooKeeper documentation, the proposed leader election algorithm is based on the usage of two flags called *SEQUENCE |EPHEMERAL*. The flags are used when creating znodes that represent ”proposals” of clients. The ephemeral znodes exist as long as the session that created the znodes is active; when the session ends the ephemeral znodes are deleted. For the sequence znodes: based on a request issued to Zookeeper when creating a z-node, Zookeeper can append a monotonically increasing counter to the end of path. The appended counter is unique to the parent znode [16].

The basic idea is to have a znode, named "/election", and that each znode creates a child znode "/election/guid-n_" with both flags Sequence and Ephemeral. The sequence flag is used to automatically append a sequence number greater than any one number previously appended to a child of "/election". The implications are that the process that created the znode having the smallest appended sequence number is the leader node [13].

Additionally, the leader failure case must be treated in order to insure the selection of a new node to become a leader. The simplest solution is to have all application processes checking constantly the current smallest znode, and in case the smallest znode is not replying checking if they should be the new leader. However this approach causes an undesired effect as all the processes receive a notification after the leader has failed, and they initiate the process to obtain the current list of children nodes from "/election". The number of the servers/znodes is directly proportional with the number of operations that ZooKeeper servers have to process. The optimization used in order to avoid this scenario is to check the next znode down on the znodes sequence. The algorithm written in pseudocode is the following [13]:

```
Create znode z with path "ELECTION/guid-n_" with both SEQUENCE and EPHEMERAL flags;
Let C be the children of "ELECTION", and i be the sequence number of z;
Watch for changes on "ELECTION/guid-n_j", where j is the largest sequence number such that j ¡ i and n_j is a znode in C;
Upon receiving a notification of znode deletion:
Let C be the new set of children of ELECTION;
If z is the smallest node in C, then execute leader procedure;
Otherwise, watch for changes on "ELECTION/guid-n_j", where j is the largest sequence number such that j <i and n_j is a znode in C;
```

Although we understand that this algorithm is just a basic example and it was not designed to be used directly in production as is, we believe that it is useful to analyse some of the shortcomings of this proposed algorithm and to provide an improved alternative.

Based on our experience the algorithm proposed by Zookeeper team has the following issues:

- No flexibility regarding leader election - the oldest node alive, with the lowest sequence is always elected as leader.
- The fact that a leader is elected does not always translate into that node actually becoming a leader. Depending on application the transition to leader status can be an elaborate process which may last

longer or it may fail. Only this transition has completed the leader is actually active and this moment should be used to notify the other nodes that the election process has been completed.

3. Algorithm description

As the previous paragraph detailed, most of the election algorithms in distributed systems were focused on efficiency in terms of size of communication between nodes and maximum latency until a leader is elected. However the efficiency in the election process does not guarantee the efficiency of the system during the processing phase. Moreover, many algorithms assign a uniform role to each node during the election process, the nodes are equals and each node can vote either for itself or for any other node. This approach is leading in general to a broadcast type of communication until a leader is elected. In this paper we propose a different approach, in which the nodes are assigned different priorities, their vote can have a different impact and weight. The algorithm uses Zookeeper as a third party coordination tool; using this tool, the leader election process can be completed using node's predefined priorities and can provide additional guarantees regarding the election process.

3.1. **Node roles.** Although, in general, all the nodes in the cluster can be identical, they perform various roles during the operational lifetime.

- Election node - This node runs the election process.
- Leader node - This node performs some critical activity which must be done on a single instance at a time.

These roles are dynamic and transitory, meaning that, in general, there is no static configuration regarding which node is a leader or an elector node. Any valid node can assume these roles.

3.2. **Zookeeper data structure.** In order to manage the cluster configuration and the election process the algorithm uses three parent znodes:

- **nodes** - contains one ephemeral *znode* for each active node. Each znode contains more detailed information about each cluster member.
- **election** - contains one *znode* with emphflags ephemeral |sequential for each active node. Used to select the election node, by default the node with the lowest sequence.
- **leader** - contains only two *znodes*:
  - **elected** - created by the election algorithm, identifies the next leader candidate
  - **current** - created by the leader candidate

3.3. **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.

3.4. **Process startup.** First of all, during startup each process must register in the cluster by connecting to a common Zookeeper cluster. This involves the following steps:

- Connecting to Zookeeper which starts a *zk-session.*
- Creating a *znode* under a certain path with node information (nodeId). This z-node is ephemeral, meaning that is automatically removed when the *zk-session* times out. This node is not used in the election process, but it only contains some useful instance information such as IP address, location (site) and possible other info.
- Creating a *znode* in order to register as a potential election node. This node is created with a sequential flag, meaning that Zookeeper will allocate a unique, sequential id to each node. The data value for this node is also the *nodeId.* This is used to select the election node - the node which will run the election process.
- Registering Zookeeper watches on cluster *znode, election znode* and leader *znode.*

3.5. **Election process.** The election process is triggered by any change in the list of *znodes* under the *election path.* Every time a new node is added to the cluster or there is a failure and one node stops, the associated *zk-session* is timed-out and the ephemeral *znodes* created by this process are removed from the Zookeeper repository. These changes are notified immediately to all the remaining nodes. By reading the remaining *election z-nodes* and comparing its own *nodeId* only the oldest process alive (with the lowest sequence assigned by Zookeeper) will execute the election process. This node will assume the temporary role of ***elector node***.

The election process can be designed to be highly flexible by assigning different priorities to different nodes. Some of these election strategies are discussed in the next section. But, in all cases, at the end of the election process the algorithm chooses one candidate as the next leader. There are cases when this candidate is the actual leader because a non-leader node exited the cluster, so nothing else happens and the process stops here.

At the end of the election process, if the elected node is different from the current leader, we create a new ***elected znode*** with the nodeId of the new leader candidate.

From here the next processing happens in parallel as all the nodes also monitor the znodes under the *leader* path:

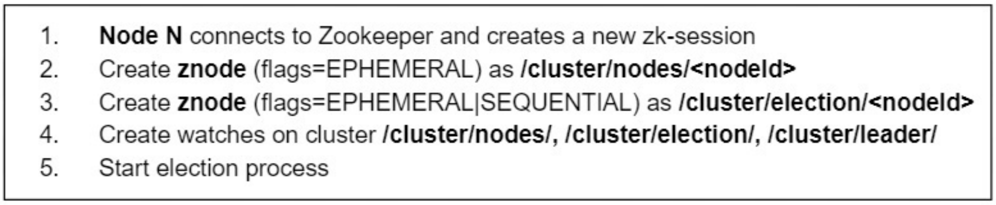
The existing leader node: For it, the presence of a new elected leader may mean that it must voluntarily give up the leader role. At the end of this process it deletes the ***leader znode***. A leader or candidate node monitors the *"elected znode"* and if it was replaced by a different z-node it must immediately stop the leader role or the leader transition process.

The elected leader node: When a node detects the presence of a *new elected zone* which matches its own id, it automatically starts a process to become a leader. But, before that, it announces its intention to become a leader by creating the ***leader znode*** with a specific status - PROGRESS. If the *leader znode* already exists - maybe because the existing leader has not removed it yet, then this creation is retried after a short delay. After successfully creating the *leader znode* it executes the required procedures and after that it updates the *leader znode* with status READY, meaning that the cluster has a new leader which is ready for processing.

3.6. **Leader transition watchdog.** Additionally, for improved robustness of the solution we can include a leader transition watchdog. If an elected leader does not manage to become leader in X seconds, the election node will run the algorithm again by excluding the previously selected leader.

3.7. **Additional considerations.** The election algorithm runs in the callback thread used by Zookeeper client library for notifications, which means that the process is not re-entrant. If the cluster configuration changes while the election process is running, the process simply runs again when the next notification is delivered. This implies that an elected node must always be ready to abort the leader transition at any time, even if just started.

3.8. **Principal methods.** A. Node startup:

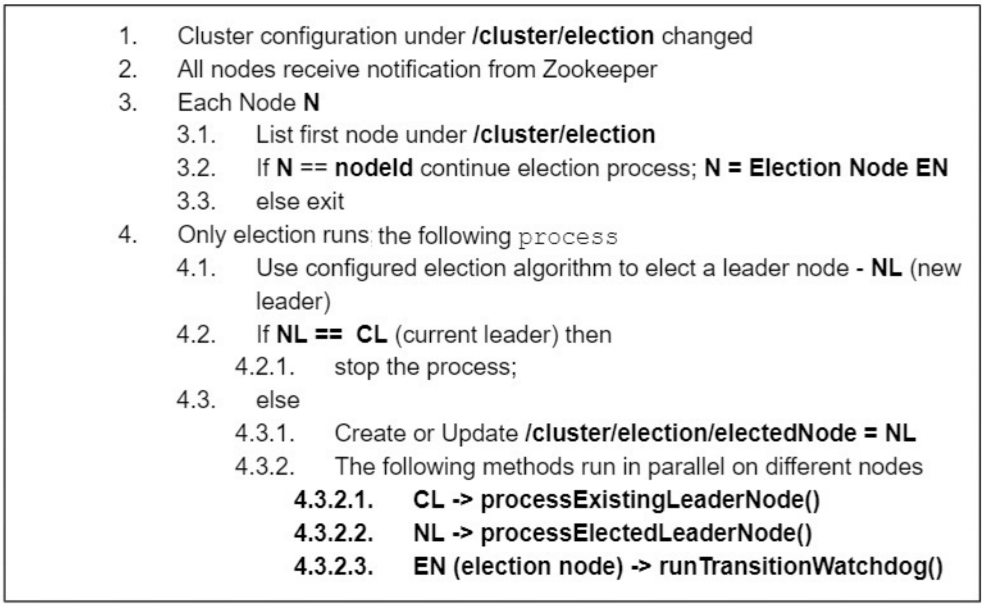


1. **Node N** connects to Zookeeper and creates a new zk-session
2. Create **znode** (flags=EPHEMERAL) as **/cluster/nodes/<nodeId>**
3. Create **znode** (flags=EPHEMERAL|SEQUENTIAL) as **/cluster/election/<nodeId>**
4. Create watches on cluster **/cluster/nodes/, /cluster/election/, /cluster/leader/**
5. Start election process

Election process is also triggered when any node under **/cluster/election/**

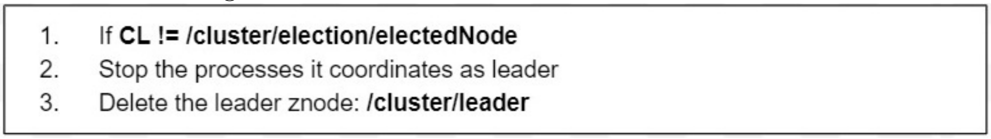
changes, which means when one node disconnects or when a new one re-joins the cluster.

B. Election:



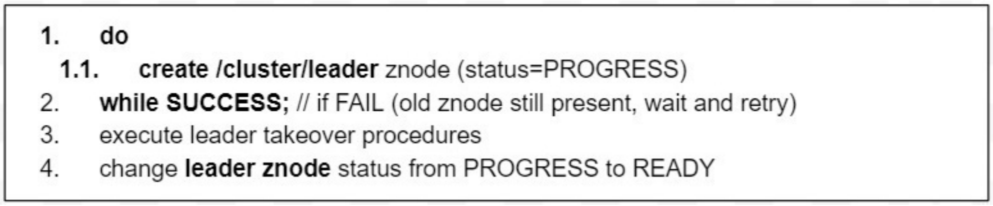
```
1.  Cluster configuration under /cluster/election changed
2.  All nodes receive notification from Zookeeper
3.  Each Node N
    3.1.  List first node under /cluster/election
    3.2.  If N == nodeId continue election process; N = Election Node EN
    3.3.  else exit
4.  Only election runs the following process
    4.1.  Use configured election algorithm to elect a leader node - NL (new
          leader)
    4.2.  If NL ==  CL (current leader) then
        4.2.1.  stop the process;
    4.3.  else
        4.3.1.  Create or Update /cluster/election/electedNode = NL
        4.3.2.  The following methods run in parallel on different nodes
            4.3.2.1.  CL -> processExistingLeaderNode()
            4.3.2.2.  NL -> processElectedLeaderNode()
            4.3.2.3.  EN (election node) -> runTransitionWatchdog()
```

C. ProcessExistingLeaderNode



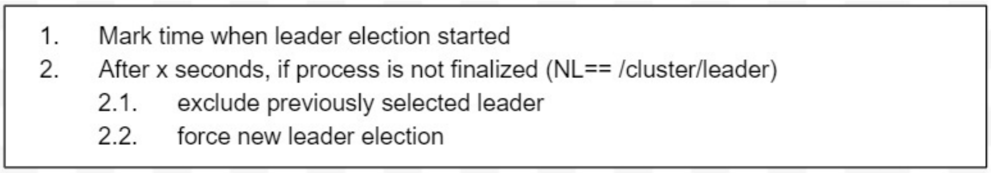
```
1.  If CL != /cluster/election/electedNode
2.  Stop the processes it coordinates as leader
3.  Delete the leader znode: /cluster/leader
```

D. ProcessElectedLeaderNode(candidate znode)



```
1.  do
  1.1.  create /cluster/leader znode (status=PROGRESS)
2.  while SUCCESS; // if FAIL (old znode still present, wait and retry)
3.  execute leader takeover procedures
4.  change leader znode status from PROGRESS to READY
```

E. RunTransitionWatchdog



```
1.  Mark time when leader election started
2.  After x seconds, if process is not finalized (NL== /cluster/leader)
    2.1.  exclude previously selected leader
    2.2.  force new leader election
```

3.9. **Election strategies or policies.** The most simple election strategies would be to simply pick the first available node, based on their nodeId order or, alternatively, to use a round-robin method.

Another possibility is to assign all the nodes to different sites or data centers, based on their geographical location. For various reasons, nodes belonging to a particular site are preferred over others. This site preference or priority is not static, but it change dynamically during normal operation:

- If some critical error disables an entire site, for instance due to failure of some shared network or storage equipment. In these cases, even if there are nodes available in the primary site, it may be better/safer to move the processing to the backup site.
- If there is a planned maintenance operation which impacts all the nodes in site, the administrator can simply move the leader to another site by temporarily assigning a higher priority to the backup site.

The election algorithm can be configured to support multiple strategies and pick the most appropriate one based on the exact circumstances when the election is run.

4. Conclusion and Future Work

There are applications that have specific rules, applications that are processing sensitive data and for which a cloud installation is out of discussion. For contingency reasons, the servers are split into clusters located in different places and have additional constraints such as: the new leader should belong to a different cluster from the previous leader. None of the mentioned algorithms is enough flexible to allow this approach. All the algorithms have a predefined standard set of constraints and they adjust the election process and the algorithm behavior based on the same parameter or set of parameters. Our algorithm permits to set different priorities for the znodes, influencing the chances to be elected and offering a wider set of methods to customize the election process.

The proposed algorithm was designed to offer a lot of flexibility regarding the criterias used in a leader election process, so it maps on a range of applications that require a customized approach. Even if it adds a new layer of processing, it allows to prioritize the servers in the election process, thus enabling a high degree of customization for each application type, taking into account not only different metrics such as latency, but also requirements such as the locations of the leader server. There are critical applications that require a DR site (disaster recovery site), where the nodes should replicate the information posted and processed in the live site. For these types of applications, the leader election process has other constraints: in case a leader fails, the processing should be automatically moved to the other site and the new leader should be selected as one of the nodes from that site. The proposed algorithm addresses these requirements and can also accommodate other application's specific requirements.

Another benefit added by our proposed algorithm is that the leader transition happens in two stages: first the new leader is notified and second, only after the successful completion of the transition process the new leader announces that the new leader is ready to receive requests. This ensures that if the leader transition does not proceed as planned, the process can be retried by another candidate.

The algorithm can be easily extended into a multi-tenant operation, where there are multiple leaders at the same time, one for each critical resource. This can be achieved simply by using the Zookeeper znode hierarchy which is modeled like a tree. In such a multi-tenant operation we would have one dedicated data structure as described in the Process Startup phase for each tenant, so that each tenant runs completely isolated from others.

Another possibility is to slightly change the algorithms to support more than one leader at the same time for the same resource. Such an approach is similar to a configuration with multiple Active and Spare nodes (or primary/backup architectures) - where spare nodes are not actually stopped, but idle and waiting to resume processing or take over the leader/active node as required.

The future work will consist in developing and running a set of tests in order to check how the system will behave under heavy loading and also to try to find out if there are vulnerabilities in the proposed algorithm.

References

[1] Artho, C., Banzai, K., Gros, Q., Rousset, G., Ma, L., Kitamura, T., Yamamoto, M., Model based testing of Apache ZooKeeper: Fundamental API usage and watchers. Software Testing, Verification and Reliability, 2019, DOI:10.1002/stvr.1720

[2] Artho C, Gros Q, Rousset G, Banzai K, Ma L, Kitamura T, Hagiya M, Tanabe Y, Yamamoto M. Model-based API testing of Apache ZooKeeper. Proc. 2017 IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2017): Tokyo, Japan, 2017; pp. 288-298.

[3] Becker D., Junqueira F., Serafini M., Leader Election for Replicated Services Using Application Scores.,2011, DOI 7049. 289-308. 10.1007/978-3-642-25821-3 15.

[4] EL-Sanosi I.,Ezhilchelvan P.,Improving the Latency and Throughput of ZooKeeper Atomic Broadcast, Imperial College Computing Student Workshop, 2018, pp. 3:1–3:10, ISBN 978-3-95977-059-0, DOI 10.4230/OASIcs.ICCSW.2017.3

[5] EL-Sanosi I. , Ezhilchelvan, P., Improving ZooKeeper Atomic Broadcast Performance by Coin Tossing, Lecture Notes in Computer Science, 2017, pp.249-265. DOI:10.1007/978-3-319-66583-2_16

[6] Ganesan A., Alagappan R., Arpaci-Dusseau A., Arpaci-Dusseau R., Strong and Efficient Consistency with Consistency-Aware Durability, 18th USENIX Conference on File and Storage Technologies, Santa Clara, CA, 2020, ISBN 978-1-939133-12-0
[7] Hunt P, Konar M, Junqueira F, Reed B. ZooKeeper: Wait-free Coordination for Internet-scale Systems. Proc. USENIX Annual Technical Conf., USENIXATC, USENIX Association: Boston, USA, 2010; 11'11. DOI:doi=10.1.1.178.5750
[8] Junqueira F., Reed B. ZooKeeper: Distributed Process Coordination. O'Reilly, 2013, ISBN-13: 978-1449361303
[9] Medeiros A., ZooKeeper's atomic broadcast protocol: Theory and practice, Helsinki University of Technology, 2012, DOI: 10.1.1.473.1373
[10] Medeiros A., ZooKeeper's atomic broadcast protocol: Theory and practice,2012, retrieved from http://www.tcs.hut.fi/Studies/T-79.5001/reports/2012-deSouzaMedeiros.pdf
[11] Petrescu M., Replication in Raft vs Apache Zookeeper, Advances in Intelligent Systems and Computing, 2020, ISSN 2194-5357
[12] Petrescu M., Petrescu R., Log replication in Raft vs Kafka, Studia Universitas Babes-Bolyai, 2020, DOI 10.24193/subbi.2020.2.05
[13] Santos, N. H., Andre M.S., Latency-aware Leader Election.,2009, DOI 10.1145/1529282.1529513.
[14] ZooKeeper 3.6 Documentation / ZooKeeper Recipes and Solutions, 2020, retrieved from https://zookeeper.apache.org/doc/r3.6.2/recipes.html
[15] ZooKeeper 3.6 Documentation / ZooKeeper Programmer's Guide, 2020, retrieved from https://zookeeper.apache.org/doc/r3.6.2/zookeeperProgrammers.html
[16] ZooKeeper 3.6 Documentation / The ZooKeeper Data Model, 2020, retrieved from https://zookeeper.apache.org/doc/current/zookeeperProgrammers.html#Ephemeral+Nodes

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

Email address: `mpetrescu@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.08

BIBTEX FOR THE ROMANIAN LANGUAGE

MARIAN MUREŞAN

Abstract. BibTeX was created by Oren Patashnik and Leslie Lamport in 1985 according to https://en.wikipedia.org/wiki/BibTeX and turned out to be a very useful software product. Nevertheless most of the present scientific paper in mathematics and computer science, and not only, are written in English. Our aim was to offer to the Romanian scientific paper writers a variant of the BibTEX whose output agrees with the Romanian language grammar.

1. Presentation

Many years ago I faced to the following issue: how to use the LATEX and BibTEX tools to write a book in Romanian so that the output fulfils the requirements of the Romanian grammar. The problem was the bibliography part since certain rules are not longer true. A trivial example is the following: in Romanian there no word *and* but instead of it there is the word *şi* .

There in 1996 we started to study how BibTEX is constructed and how can it be modified. As a result of this attempt a first version of the Romanian BibTEX was released at July 1st 1996. Later on two improved were performed and released at November 30th 1997 and February 9th 2009, respectively. Our package AMSP-MR1.BST accepts input according to the rules of BibTEX and supplies output with certain changes oriented to the Romanian language.

The latest version of our Romanian BibTEX is uploaded on the author's web site http://marianmuresan.wordpress.com. Obviously our product is free.

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania
Email address: `mmarian@math.ubbcluj.ro`

Received by the editors: 2 May 2021.
2010 *Mathematics Subject Classification.* 97P40, 97R30.
1998 *CR Categories and Descriptors.* D.3.3 [Language Constructs and Features]: Modules, packages.
Key words and phrases. BibTEX.