EvoCasPer: An Evolutionary Cascade Approach for Noisy Data Classification

Robert Neil McArthur[1]

Research School of Computer Science, Australian National University u6956078@anu.edu.au

Abstract. This paper proposes a new neural network architecture for noisy data classification called EvoCasPer. EvoCasPer is primarily inspired by two other prominent architectures - NEAT and CasPer [9, 10]. EvoCasPer combines the structural rigidity that CasPer contains whilst still allowing for the freedom NEAT provides in evolving the network's topology. The goal of this constrained freedom is to allow for the network to evaluate noisy input with reduced chances of overfitting in a still adaptable manner. This paper critically evaluates EvoCasPer's performance on the task of noisy data classification when compared to CasPer and several standard feedforward neural networks [3]. To do so, noise from a normal distribution with varying standard deviation was added to patient data in the sars-cov1 dataset [4]. EvoCasPer, CasPer [10] and four feedforward neural networks with varying single hidden layer size were trained to classify whether the patients were normal, had high blood pressure, had pneumonia or had SARS based on the noisy medical data input. The performance of the models was assessed on the ability to classify the noisy data while still having the ability to classify data with noise removed. It was found that EvoCasPer consistently achieved good values for each of these metrics while mitigating the issues of frequent underfitting and overfitting problems found in the other models.

Keywords: EvoCasPer · CasPer · Evolutionary Algorithms · Noisy Data · Classification.

1 Introduction

In a perfect world, one would not have to worry about dealing with noisy data. Unfortunately, our world is not perfect and many things, such as robotic sensors, have seemingly random amounts of noise added to what would be otherwise expected. Yet, decisions must still be made based on our noisy view of the world. Hence, it is important to explore different methods for the purposes of classifying noisy data.

This paper proposes a new architecture called EvoCasPer, which is primarily inspired by the CasPer and NEAT models [10, 9]. The goal of the paper is to evaluate the proposed method as an extension of an existing work [3] which is classifying data with noisy input. The base of the EvoCasPer model is primarily built off of CasPer. CasPer is a form of cascade neural network which begins with a single hidden neuron. From here, the network is trained a certain number of steps before using Rprop a new neuron is added and the process is repeated (in this paper, we use the Adam optimiser instead). As this happens, the weights are distributed into three regions named L1, L2 and L3 and each given a separate learning rate (such that L1 is much larger than L2, which is greater than L3). The L1 region represents weights that feed into the newly added neuron. The L2 region represents weights from the new neuron to the output neurons, while the L3 region is everything else. This relation allows primarily the new neuron to be trained to minimise error while everything else can still be adjusted (to a lesser extent). The architecture is depicted in figure 1 for the sake of simplicity. EvoCasPer extends upon this idea by allowing the network's topology to be evolved by applying masks to the weights and biases of the model. This allows the network to pick and choose, so to say, which features (or derived features for the hidden neurons) are most important in order to reduce the network's error. Additionally, when neurons are added to the network is also controlled by evolution. All of this allows for greater expressivity than CasPer while still maintaining the rigidity, which was found to work well for noisy data classification [3]. This was only a brief overview of the new model and further details are provided in the methodology section.

This paper uses the sars-cov1 dataset [4] to analyse the results of the new EvoCasPer architecture when compared with a standard feedforward neural network and CasPer in handling noisy data classification. The sars-cov1 dataset contains medical data for patients who are healthy, have pneumonia, have high blood pressure, or are infected with SARS. There are 1000 patients in each category, giving 4000 total data points. The medical data contains fuzzy values for four different temperature recordings (at different times), systolic and diastolic blood pressure, and nausea (abdominal pain is ignored). For each of these seven metrics, there are three fuzzy values representing whether the metric should be classified as slight, moderate or high. These, after pre-processing, form the 21 input features for the classification problem. 80% of the data was put into a training set, with the rest falling into a test set. Many

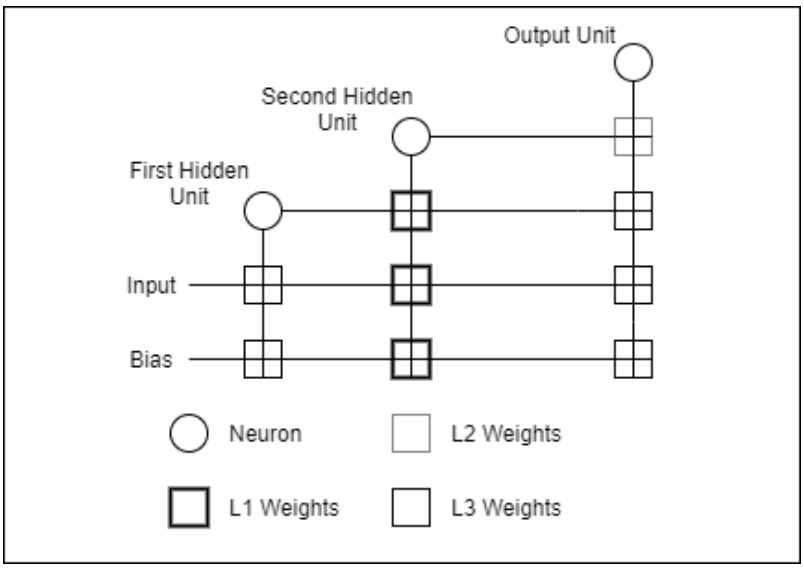

Fig. 1. The CasPer architecture with weights after a second hidden unit is added. The units along a vertical line are summed before an activation function is applied.

numeric values for the columns are non-existent, as shown in figure 2. This could lead to problems with running predictions on previously unseen data.

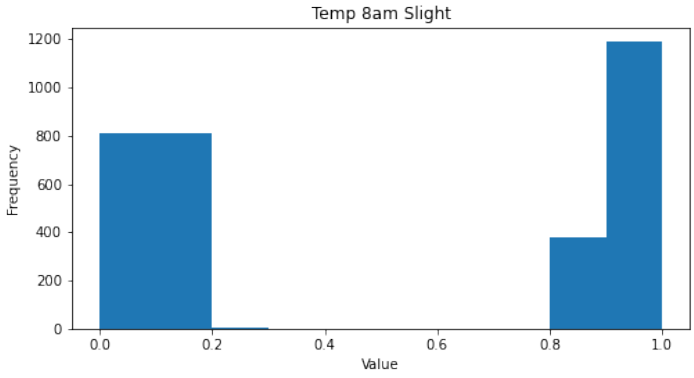

Fig. 2. Most of the columns in the input data look like this before pre-processing.

Three copies of the training data were created, each one having a separate normal distribution in $\{\mathcal{N}(0,\sigma) \mid \sigma \in \{0.6, 0.9, 1.2\}\}$ added to it to simulate noise. The mean and standard deviation of these training sets was calculated so that it could be normalised to have a mean of zero and standard deviation of 1 (by subtracting the mean and dividing by the standard deviation for each column). This was done to standardise the input neurons so that each would have an equal effect since the beginning of training. The same normalisation transformation was applied to three copies of the test data (subtracting the training data's mean before dividing by the training data's standard deviation) for the consistency of values between each set pair. After pre-processing is complete, the resulting data distribution looks as in figure 3 in each column.

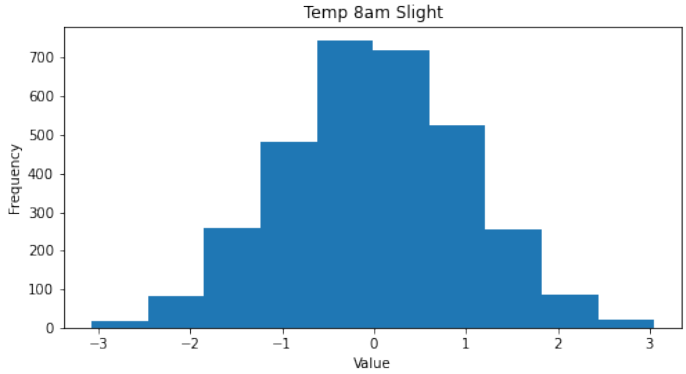

Fig. 3. Most of the columns look like this after pre-processing is completed.

2 Related Works

The classification of noisy input data is a well-studied problem, especially in the realms of image classification. A common technique applied when dealing with noisy data is the usage of a denoising autoencoder [11], assuming there is access to data without noise. A denoising autoencoder works by corrupting the input data randomly, before feeding it through an autoencoder in an attempt to recover the initial data. One paper [7] layers different variants of the denoising autoencoder with a convolutional neural network to classify noisy images. The results outperformed existing methods on images corrupted with 50% noise.

Another paper [8] details different methods for training neural networks when noisy data is present with a greater focus on regression. The paper details methods surrounding reducing the impact of noisy data on learning. Some examples of this involve changing how the error is calculated, instead relying on metrics that are better suited against outliers. This includes the least mean log squares, simulated annealing for least median of squares, least trimmed absolute values and iterative least median of squares.

3 Methodology

3.1 Criteria

To determine which architecture of EvoCasPer, CasPer, and a standard feedforward neural network is better with noisy data classification, there are two conditions to be assessed. Firstly, when the noise is removed from the data, the accuracy of the model should remain high while the loss of the model remains low. This is to ensure the model is still learning about the underlying structure of the data rather than memorising the noise. This can be done by simply assessing the models on the test set (which doesn't have noise added). If each model's loss remains low and accuracy remains high here after training, it has successfully learnt how to handle normal data from noisy data. Secondly, on the noisy data, the accuracy of the model should be high and the loss of the model should remain low. If this is the case, then the model has effectively learned to treat the noisy data as if there were no noise.

3.2 EvoCasPer

The EvoCasPer model is structured in the exact same fashion as the CasPer model depicted in figure 1. There is always a fully connected layer between the input and output and hidden neurons are stacked in the same way as the CasPer model. The innovative factor of the EvoCasPer model, however, is that evolution is performed over the network's topology. Each weight in EvoCasPer's model has an associated value which is either 0 or 1 (with the exception of the fully connected layer from the input to output). If this number is 1, the connection is as normal. Otherwise, if the number is 0, the corresponding weight/bias is zeroed out such that no connection is present. This allows for the connections which are present in the model to evolve, allowing it to pick and choose which neurons to take information from and which neurons may have an effect on the output. Additionally, the timing where neurons are added to the network is controlled by random mutations.

More specifically, the population for evolution is n different EvoCasPer models (implemented with $n = 12$). The genotype for evolution are the connections that are present and not present in each model. Each model is trained

for twenty epochs before the loss on the training set for each model is calculated as a fitness score. The $\frac{n}{2}$ (six) best performing models (lowest fitness) are used for reproduction with elitism to reform the population size. The reciprocal of each 0.1 plus fitness is calculated as a use of the chance an individual is selected for reproduction (0.1 is added to avoid the possible case of the fitness being 0, causing a division by zero). Using these probability scalings, two random individuals are selected. The weights and biases of the fully connected layer from input to output (which is always present) is averaged between the two individuals to form the fully connected layer for the new child. Averaging the weights and biases in such a manner was found to have the smallest impact on initial loss for the creation of the new child (especially considering how this should cause the fully connected layers to evolve together). Then, the topology is evolved, for each possible connection that would be in the normal CasPer model, whether the connection is present is sampled randomly from the two parent networks. For example, for weight #1, the evolution process samples this weight from *parentA*. If the connection is present in *parentA*, the connection (including weight/bias values) is copied across. Otherwise, the connection remains disconnected. An issue arises when one parent has a larger size (as in the number of hidden layers) than another parent. In order to resolve this issue, imagine that the network with the smaller size has the same size as the larger one. In order for this to be possible, all new connections there would need to be zeroed out. Therefore, when evolving two networks of different sizes, we simply assume that the smaller network is the same size as the larger one with zero for all new weights and biases needed for reach the larger equivalent size, before reproducing. Mutations are performed by, for each weight/bias which can be evolved, connections are randomly disconnected or connected with probability 0.03. Additionally, a new layer is added to a new model by mutation with probability 0.8. When initialising new layers, on average 3% of all connections are zeroed out. In order to stop the zeroed out weights/biases from changing during training, the respective gradients are also zeroed out during backpropagation. Additionally, in order to solve the zero-weight problem when a disconnected connection becomes connected (and begins with weight zero), the LeakyReLU activation function is used to ensure the gradient is non-zero at zero. Performing this process over successive generations allows for neurons to be slowly added to the network and connections to be chosen such that an optimal model structure for the problem is found. For the underlying CasPer model, the Adam optimiser is used with L1 weights 0.6, L2 weights 0.3 and L3 weights 0.1. Cross entropy loss is also applied. Each model in the population is trained for twenty epochs, before the population is reproduced. This process is repeated eighty times before the final results are achieved.

3.3 Feedforward Neural Network Implementation

A simple feedforward neural network was created. The neural network contains one hidden layer of specifiable size (this included 10, 20, 50 and 100 for experimentation). The input layer of size 21 is fully connected to the hidden layer and the LeakyReLU activation function is applied. The hidden layer is then fully connected to the four output neurons, which represent the four classes. Cross entropy loss is applied and the Adam optimiser is used with a learning rate of 0.08. When training, for every 20 epochs beginning with 35 (i.e. 35, 55,...) if the error had not decreased by at least 1%, training was terminated.

3.4 CasPer Implementation

The CasPer implementation [10] consists of several parts. The input layer is always fully connected to the output layer. The input layer is also fully connected to every hidden neuron. Further, to each hidden neuron, all of the previous hidden neurons are also connected. The LeakyReLU activation is applied to each hidden neuron. Finally, in addition to the input layer being fully connected to the output layer, each hidden neuron is also connected to the output layer. The learning rate for the L1 region (weights that connect to a newly created hidden neuron) has a learning rate of 0.15. The L2 region (weights from a newly created neuron to the output neurons) has a learning rate of 0.03. Further, the L3 region (everything else) has a learning rate of 0.01. The Adam optimiser is used with these learning rates and cross-entropy loss is applied. Weight decay was not applied because it was found in initial experimentation that it generalised well to the task. New neurons were added to the network if the epoch is equal to $15 + 20 \cdot \text{CasPer.}\textit{hidden\_neurons}$ and the error had decreased by at least 1%. If the error had not decreased by at least 1%, training was terminated.

3.5 Experimentation

An experiment over the training data was repeated two times. For each of the three training and test data pairs, the EvoCasPer model and CasPer model was trained, as well as the four feedforward neural networks. The training rules for CasPer and the feedforward neural networks were applied up to a maximum of 2000 epochs. Loss and accuracy graphs were created for each model training and the final loss and test loss were recorded.

4 Results and Discussion

4.1 Results

The average final train loss and test loss are recorded in the two tables below for each dataset with noise added. FF n refers to the feedforward neural network with n hidden neurons.

Table 1. Average training loss for each model and noisy dataset after training.

Noise Standard Deviation (σ)	EvoCasPer	CasPer	FF 10	FF 20	FF 50	FF 100
0.6	$1.7 \cdot 10^{-10}$	$5.4 \cdot 10^{-2}$	$5.8 \cdot 10^{-2}$	$2.1 \cdot 10^{-2}$	$1.5 \cdot 10^{-5}$	$7.5 \cdot 10^{-6}$
0.9	0.13	0.32	0.34	0.26	$6.7 \cdot 10^{-2}$	$2.6 \cdot 10^{-5}$
1.2	0.38	0.64	0.61	0.54	0.30	0.22

Table 2. Average test loss for each model and noisy dataset after training.

Noise Standard Deviation (σ)	EvoCasPer	CasPer	FF 10	FF 20	FF 50	FF 100
0.6	0.0	$1.1 \cdot 10^{-3}$	$4.7 \cdot 10^{-4}$	$3.9 \cdot 10^{-5}$	0.0	$1.0 \cdot 10^{-9}$
0.9	$1.7 \cdot 10^{-2}$	$5.6 \cdot 10^{-2}$	$5.3 \cdot 10^{-2}$	$4.5 \cdot 10^{-2}$	$5.1 \cdot 10^{-3}$	$1.7 \cdot 10^{-4}$
1.2	0.13	0.22	0.21	0.19	0.18	0.12

4.2 Discussion

The results show that in all cases, EvoCasPer significantly outperforms CasPer in all cases, with more mixed results when compared with the later feedforward neural networks. It is noted that the number of experiments was indeed small, which did affect the CasPer results from underfitting and feedforward neural network results with a lack of overfitting as observed in an earlier work [3]. Still, EvoCasPer seems to significantly outperform the results shown in that work in terms of training and test loss. To work out exactly why this is the case, we must analyse the loss and accuracy graphs.

A significant issue for the feedforward neural network approaches was the chance of overfitting. Figure 4 shows two loss graphs for $\sigma = 1.2$ dataset and the feedforward neural network with 100 hidden neurons. The figure shows that sometimes the neural network fits very well to the data but at other times, the test loss remains high. This indicates that the model sometimes overfits to the noise in the dataset. Even smaller neural networks don't fare much better when it comes to overfitting, despite the good performance of the feedforward neural network with 50 hidden neurons, overfitting shown in the worst case (figure 5).

If we compare this to the CasPer implementation, however, we get in the overfitting worst case two graphs as shown in figure 6. The spikes in the loss correspond with when a neuron is added to the CasPer neural network. As can be seen, the training loss follows a slight sigmoidal trend with the loss accelerating lower initially before slightly tapering back and the termination condition is reached. There are some spikes in the test loss, particularly in the second image. These primarily coincide with the spikes in training loss. Towards the end of training, the spikes in test loss look as though they begin to outpace that of training loss. When the CasPer model begins overfitting, it appears to do so to a lesser extent than when the feedforward neural network does. However, it appears it is certainly possible in a scenario for the test loss spikes to outpace those of the training loss spikes and hence cause overfitting.

Another significant issue for the CasPer implementation is the chance of underfitting. If for the initial few neurons, the loss doesn't decrease by all too much, training simply terminates. This can readily be observed in figure 7.

Let us now compare this to how EvoCasPer works. Firstly, the EvoCasPer model decides for itself when to add new neurons through evolution and has no early termination criteria (the behaviour of which is shown in figure 9). Neurons are randomly added and connections are evolved and mutated until the loss is low enough for a species with a new neuron to become dominant.

Additionally, the evolutionary process in EvoCasPer helps to protect itself against overfitting. This is because large jumps in test loss also usually coincide with large jumps in training loss. This training loss takes longer to converge so is removed. This mechanism is best demonstrated by figure 9.

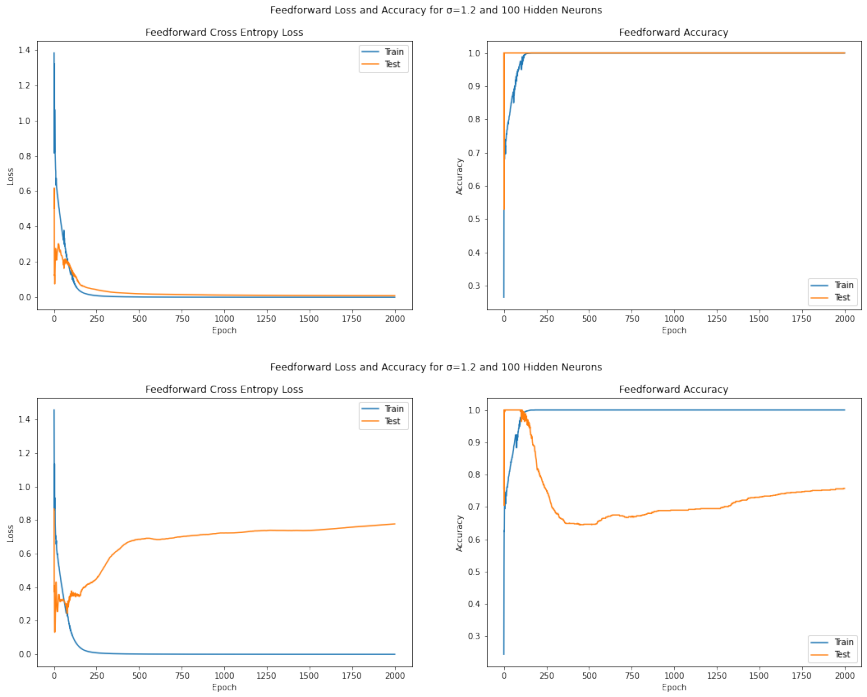

Fig. 4. Two loss/accuracy graphs for the feedforward neural network with 100 hidden neurons and $\sigma = 1.2$ [3]

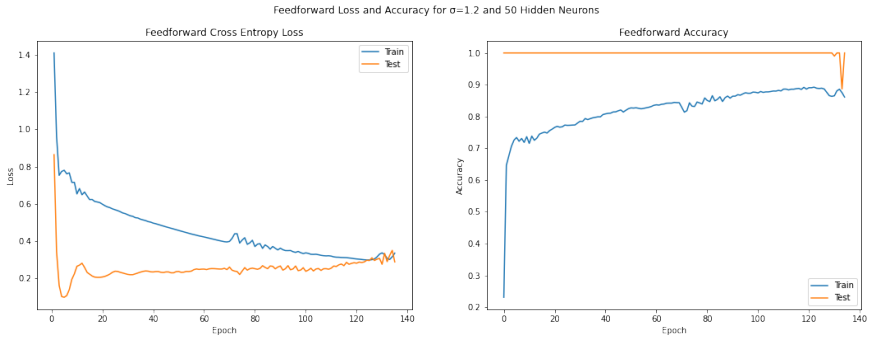

Fig. 5. The worst loss/accuracy graphs for the feedforward neural network with 50 hidden neurons and $\sigma = 1.2$ [3]

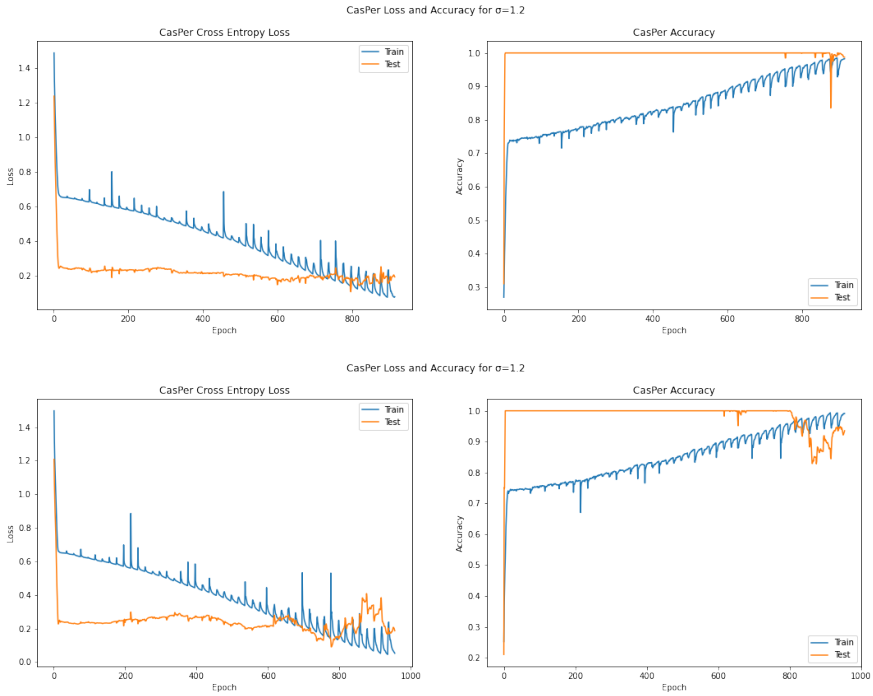

Fig. 6. The worst two loss/accuracy graphs for CasPer with $\sigma = 1.2$ [3]

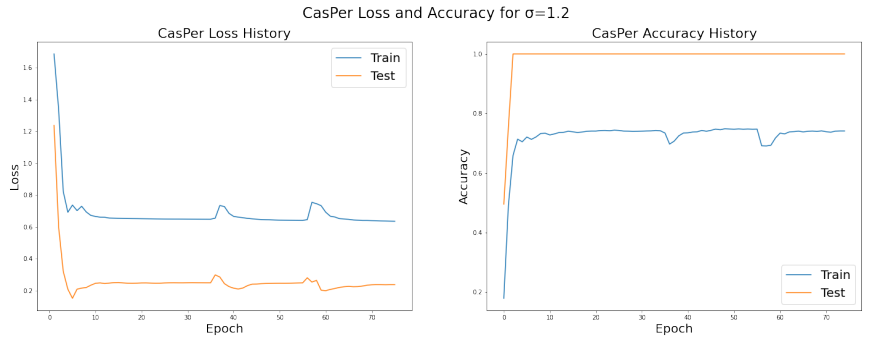

Fig. 7. CasPer has a reasonably high probability to underfit if the loss doesn't reduce after adding the first couple of neurons

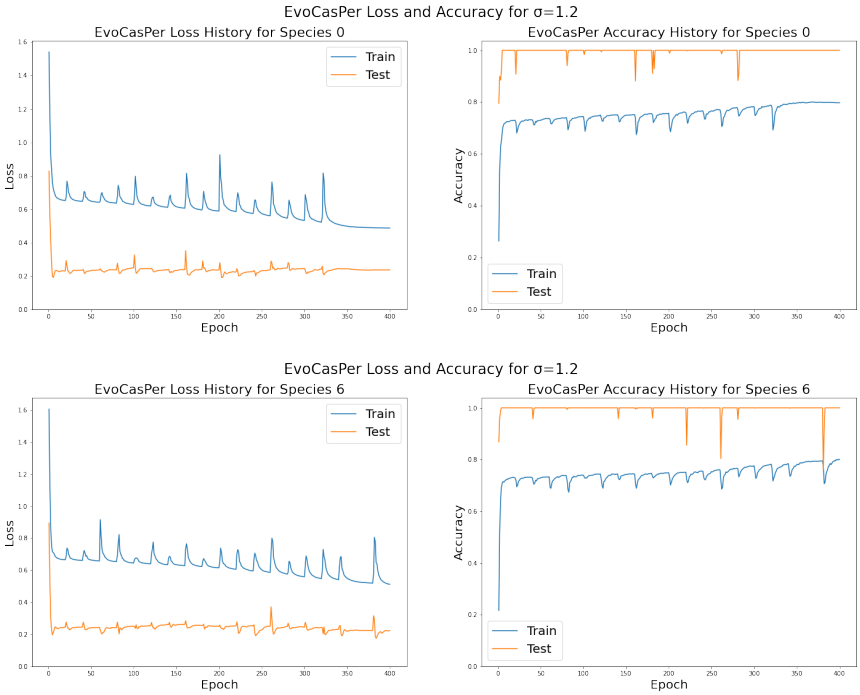

Fig. 8. These two species in EvoCasPer have neurons added at different points (as seen by spikes).

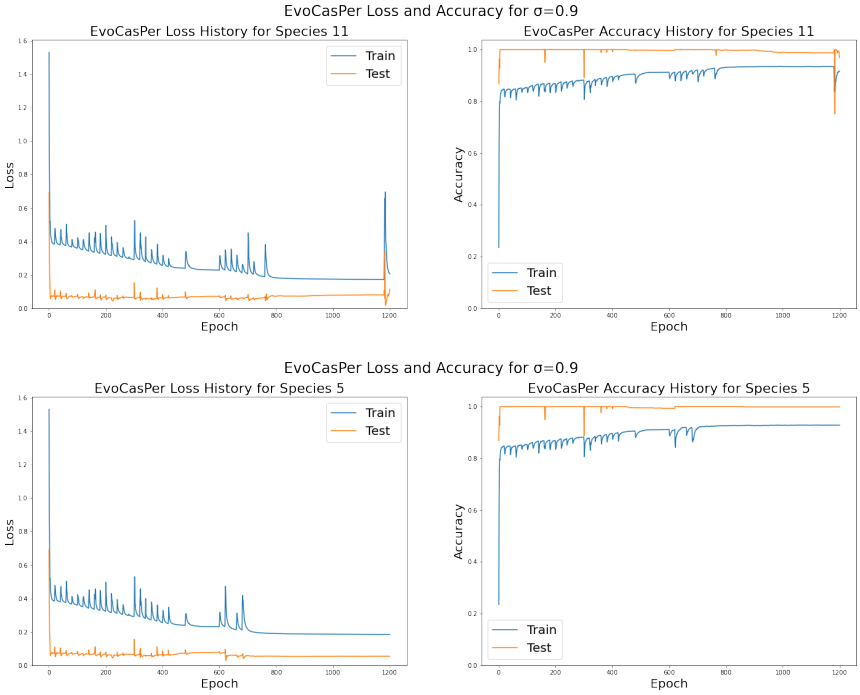

Fig. 9. Species 11 was removed while species 5 remains. Species with fewer bumps in the test graph are more likely to survive

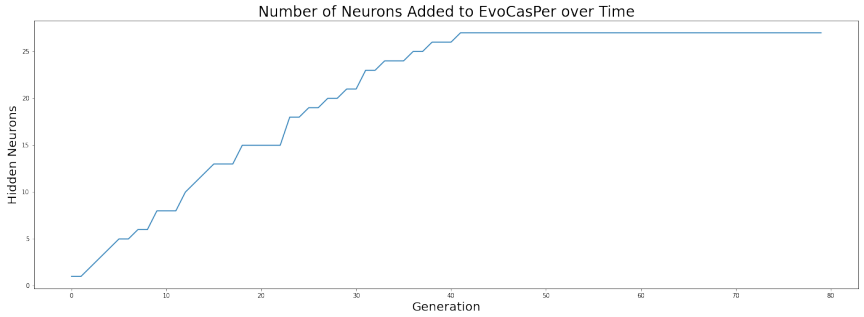

Fig. 10. The number of neurons in the best performing EvoCasPer model for each generation

This overfitting mitigation potential is further illustrated by figure 10. Observe that as the generations progress, the rate at which the number of hidden neurons is added in the best performing species of the population decreases Until it plateaus at 27 in this instance. This reflects that the model cannot gain any more performance by adding another neuron as doing so causes such a jump in training loss that the model with more neurons cannot converge in time (which would increase the risk of overfitting if it remained in the population).

There are several conclusions that we can make from all of this. Firstly, is that feedforward neural network models are very binary when dealing with noisy classification problems. Provided they work well on noisy data, they either generalise very well or very poorly to data with noised removed. Meanwhile, provided the CasPer model does not underfit on the noisy data, the training loss converges excellently while the test loss remains low (with a race as it sometimes starts increasing towards the end). However, EvoCasPer performs similarly to CasPer though underfitting avoidance is built into how the model trains and overfitting avoidance is done in the architecture implicitly as the rate at which new neurons can be added slows. Provided, the feedforward neural network works well on the train data, when the noise is removed and unseen test data observed, the feedforward neural network either performs exceptionally well or exceptionally poor. Meanwhile, normal CasPer usually performs well on the test data with slight issues of overfitting towards the end of training, and early termination at the beginning of training. However, the way the EvoCasPer model behaves helps to mitigate these overfitting and underfitting issues, performing well on the train dataset with noise, and test dataset without noise. Hence, by the criterion established at the beginning of the section, the EvoCasPer model performs the best when tasked with classifying noisy data. It is worth an important mention that training the EvoCasPer model takes much longer than the others as it is essentially training twelve CasPer models at a time.

4.3 Limitations and Future Work

There are several key limitations of this research, however. The main limitation is that the experiment was only repeated twice. It may very well be the case that, in some cases, the EvoCasPer model fails to converge or causes large test loss. That being said, during personal experimentation when setting up the main experiment and tuning hyperparameters, this was not observed. Further, the lack of early termination and constant trials of adding neurons should help avoid underfitting (especially when considering 12 species are trialled at a time). Further, the behaviour of the model should ensure overfitting is avoided when the number of training epochs between generations is sufficiently low. That being said, repeating the experiment additional times would lead to much greater confidence in the results. Additionally, it would allow us to analyse exactly when and how often the CasPer and feedforward neural network models fail through overfitting or underfitting.

Another key limitation is the layout of the feedforward neural networks. It only contains one hidden layer. Perhaps adding more layers would increase the performance of the model overall. That being said, the universal approximation theorem suggests that one hidden layer is enough to learn any function with sufficiently many neurons [2]. Moreover, the greater complexity of an additional layer could lead to underfitting on the noisy training datasets, still yielding poor performance for the test datasets for noise removed.

Another area this research is slightly lacking in is the quantification of the effect that the weight disconnection EvoCasPer has on the performance of the model. Currently, it is mainly hypothesised that it is the mutation around the addition of neurons that is primarily responsible for the reduction of overfitting in EvoCasPer when compared to the CasPer model. In a follow-up paper, it would be worth measuring the effect that topological evolution of

weight/bias disconnections in the model has on the performance of the model, or whether it is only worth evolving the number of hidden layers.

In addition to all of this, models other than CasPer and feedforward neural networks could be evaluated for the purposes of noisy data classification when compared to EvoCasPer. One such model that could be used, which is also part of the inspiration for EvoCasPer, is NEAT [9]. NEAT allows the entire topology of an artificial neural network to change over time. Adding more freedom to CasPer by implementing EvoCasPer increased the performance of the model. Perhaps increasing the freedom of the model further by employing NEAT could lead to further gains. Another model which could be used is the resource-allocation network [5]. A resource allocation network continuously adds new neurons to respond to pockets of input data the network would otherwise perform badly on. This may allow the network to recognise certain areas and patterns within the noise and is worth experimenting with.

A final avenue of further research is the employment of additional techniques from the growing area of neural network surgery [1, 6]. Currently, it is unclear as to whether the current method of picking and choosing weights from two different models is the most effective way of combining two neural networks while minimising the new loss. Perhaps employing techniques from neural network surgery could lead to superior overall performance.

5 Conclusion

This paper proposed a new model called EvoCasPer, combining CasPer and topological evolution, and evaluated its performance for classifying noisy data. The sars-cov-1 dataset [4] was split into a train set and test set. Three copies were created of the train and test sets and a normal distribution from $\{\mathcal{N}(0, \sigma) \mid \sigma \in \{0.6, 0.9, 1.2\}\}$ was added to each train dataset. Additional normalisation was applied between each training dataset and corresponding test dataset. The new EvoCasPer model, CasPer model and four feedforward neural networks (with a single hidden layer of size in $\{10, 20, 50, 100\}$) were trained over each training dataset and evaluated on the corresponding test dataset with no noise. The results showed that when a feedforward neural network worked well on predicting from a noisy training dataset, it either performs very well or very badly with the test dataset with no noise. Contrastingly, when CasPer worked well on predicting from a noisy training dataset, performance was still mostly good on the test dataset with no noise though with early termination issues as well as slight overfitting troubles towards the end. EvoCasPer, however, takes the advantages of CasPer and manages to effectively mitigate the issues of underfitting and overfitting. Despite longer training time, due to EvoCasPer's reliability in having simultaneously low noisy training error and low test error with no noise, by the criterion established, EvoCasPer is the best suited model of those attempted to the problem of noisy input classification.

References

1. Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 (2019)
2. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural networks **4**(2), 251–257 (1991)
3. McArthur, R.: Casper: A constructive approach for noisy data classification. ANU Bioinspired Computing Conference **4** (2021)
4. Mendis, B.S.U., Gedeon, T.D., Kóczy, L.T., et al.: Investigation of aggregation in fuzzy signatures. In: 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems. vol. 406 (2005)
5. Platt, J.: A resource-allocating network for function interpolation. Neural computation **3**(2), 213–225 (1991)
6. Raiman, J., Zhang, S., Dennison, C.: Neural network surgery with sets. arXiv preprint arXiv:1912.06719 (2019)
7. Roy, S., Ahmed, M., Akhand, M.A.H.: Noisy image classification using hybrid deep learning methods. Journal of Information and Communication Technology **17**(2), 233–269 (2018). https://doi.org/10.32890/jict2018.17.2.8253, http://www.e-journal.uum.edu.my/index.php/jict/article/view/jict2018.17.2.8253
8. Rusiecki, A., Kordos, M., Kamiński, T., Greń, K.: Training neural networks on noisy data. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) Artificial Intelligence and Soft Computing. pp. 131–142. Springer International Publishing, Cham (2014)
9. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
10. Treadgold, N.K., Gedeon, T.D.: A cascade network algorithm employing progressive rprop. In: International Work-Conference on Artificial Neural Networks. pp. 733–742. Springer (1997)
11. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on Machine learning. pp. 1096–1103 (2008)