Probabilistic Program Neurogenesis

Charles E. Martin[1] and Praveen K. Pilly[1]

[1]HRL Laboratories, Malibu, CA 90265
cemartin@hrl.com

Abstract

We present a new method for addressing the challenge of continual learning wherein an agent must adapt to new tasks while maintaining high performance on previously learned tasks. To accomplish this, an agent must identify previously acquired information that generalizes to the new task while also adapting its internal model to learn information that is specific to the new task. Our approach is based on neurogenesis, which involves adding new neurons to a previously trained neural network in an intelligent way. To our knowledge, we are the first to leverage probabilistic programming within the framework of evolutionary computation to optimize the growth of neural networks for continual learning. Through a series of experiments, we show that our approach is able to consistently find better performing solutions than genetic algorithms and it is able to do so faster.

Introduction

In this work, we consider the challenge of continual learning. Agents that act and learn in the real world must develop the skills to handle a wide variety of different tasks. These tasks are inevitably associated with different distributions of observations and responses. The diversity of tasks encountered in complex environments may make the task of continual learning seem improbable at best, and impossible at worst. However, tasks are often related to one another. For example, some of the visual attributes that an autonomous vehicle learns for detecting cars on the road may also be used when learning to identify motorcycles. Thus, an agent may be able to avoid learning a new task from scratch. The key is for the agent to efficiently determine what information from previous tasks is reusable for a new task and to update its internal decision making models with new, task-relevant information while avoiding catastrophic forgetting. Succinctly, an agent must be able to learn new tasks efficiently while not forgetting how to perform previously learned tasks.

There are a variety of methodologies for tackling the continual learning problem [Parisi *et al.*, 2019]. Existing approaches can be roughly divided into three categories: those that focus on consolidating synapses in a neural network that are critical to retain previous knowledge [Kirkpatrick *et al.*, 2017; Zenke *et al.*, 2017; Aljundi *et al.*, 2018; Kolouri *et al.*, 2019]; those that employ either an explicit memory buffer or a generative model to be able to interleave the learning of new knowledge with that based on stored or generated data for the old tasks [McClelland *et al.*, 1995; Ans *et al.*, 2004; Atkinson *et al.*, 2018; Ketz *et al.*, 2019]; and those that dynamically change the structure of the neural network to accommodate new knowledge [Rusu *et al.*, 2016; Draelos *et al.*, 2017; Yoon *et al.*, 2017]. Here, we address the challenge of continual learning by intelligently adding new neurons to an existing neural network (neurogenesis) so that it learns to solve a new task without forgetting how to solve previously learned tasks. The challenge is in determining how many neurons to add and where to add them so that a desired accuracy on the new task is achieved with as few new neurons as possible. In our approach the network is able to learn to leverage knowledge that was acquired when learning previous tasks, but which is relevant to the new task. This information sharing can substantially reduce the number of new neurons that need to be added to achieve a desired level of performance.

Our approach takes a unique angle towards neurogenesis by treating it as an optimization problem and using a novel combination of probabilistic programming and evolutionary computation. Specifically, we leverage the Estimation of Distribution Algorithm (EDA) framework. However, instead of representing the distribution over solutions using a Bayesian network, which is most common, we use a probabilistic program designed specifically for neurogenesis.

This paper is organized as follows. We first discuss relevant background and then move on to our methodology, which covers neurogenesis and our estimation of distribution algorithm. We then describe our probabilistic program and learning process. Next, we describe our experimental setup and results, and lastly provide a discussion and concluding remarks.

Background

The approaches described here involve modifying the architecture of a neural network to enable continual learning

without catastrophic forgetting. In Rusu *et al.* [2016], when a new task arrives, a new sub-network is added to the parent network to facilitate learning. The sub-network receives connections from the parent network so that it can integrate previously learned useful information, but the weights in the parent network remain fixed. This approach prevents catastrophic forgetting, but does not scale well with increasing numbers of tasks. In contrast, our approach learns where and how many neurons to add.

Draelos *et al.* [2017] proposed the neurogenesis deep learning (NDL) model to incrementally train an autoencoder on new MNIST digits by adding new neurons at each layer and employing a pseudo-rehearsal process called "intrinsic replay" for preserve the performance on the old MNIST digits. Yoon *et al.* [2017] proposed a dynamically expanding network (DEN) that selectively adapts network weights and also expand network structure at each layer as needed using group sparse regularization in an online manner to facilitate the sequential learning of tasks. These approaches are computationally efficient because they only leverage *local* statistical properties of neurons and/or layers. However, this is also a disadvantage because they will often fail to identify the best solutions. In contrast, our approach encodes an efficient *global* search process and thus has a higher probability of finding optimal or near-optimal solutions.

Another popular approach is to use principles of self-organization to incrementally "grow" a neural network. Recent examples include Part and Lemon [2017] and Mici *et al.* [2018]. These approaches share a similar advantage in terms of computational efficiency, but also the disadvantage of not performing a global search like our approach.

Methodology

In this section, we describe the components of our approach. We start with a description of our neurogenesis-based approach to the continual learning problem. Next, we describe the estimation of distribution algorithm [Larraaga and Lozano, 2001], which is our overarching optimization method. Then, we discuss the probabilistic program learner (PPL) that generates new individuals and is the heart of our approach. Lastly, we describe the learning procedure used to adapt the parameters of the PPL so that it focuses on promising regions of the search space.

Neurogenesis for Continual Learning

In this work, we focused on adding new neurons to fully connected, feed-forward neural networks, however, our approach is equally applicable to recurrent and convolutional networks. Our method is initialized with a neural network that has been previously trained on one or more tasks. The neurogenesis process begins when a new task is presented. Here, we do not address the need to identify the arrival of a new task, but in practice, assuming that the input distribution for the new task is different, one can identify changes

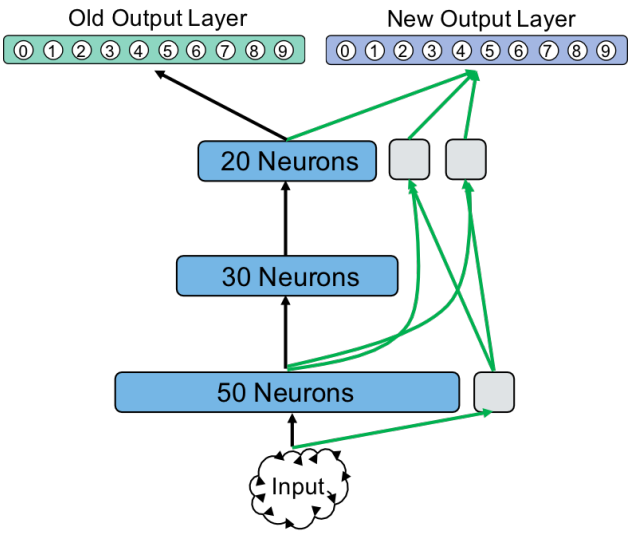

Figure 1: Example of neurogenesis used in our approach, illustrated here for continual learning of optical character recognition tasks. The layers of the parent network are shown in blue, while the newly added neurons are shown in gray. The black connections have fixed weights, while the green connections have trainable weights. A new head (shown in purple) is added to act as the output layer for the new task. This prevents the green connections, whose weights are trained on the new task, from interfering with the parent network's performance on the old task.

in the statistics of the neuron activations in the penultimate layer of the network to infer that a new task has arrived. An example of neurogenesis is shown in Figure 1. The gray blocks are new neurons. The weights on the black connections were learned from a previous task and are fixed, while the green connections are new and have trainable weights. First, a new output (classification/prediction) layer is added to the network. Next, new neurons are added and connections are established. When new neurons are added to a layer it effectively creates a new layer that emits new connections (green) to the new neurons in the nearest layer above, which may be the output layer. Additionally, if new neurons are added to the first hidden layer, then new connections are established from the input layer to these neurons. Restrictions on the connectivity patterns of new connections ensure that the performance of the network on old tasks is not altered. This is because the activations of new neurons do not affect the activations of old neurons.

Estimation of Distribution Algorithm

Estimation of Distribution Algorithms (EDAs) are a class of evolutionary computation algorithms [Larraaga and Lozano, 2001]. Unlike most EC algorithms, EDAs explicitly encode a probability distribution over the search space from which

new individuals are generated. Rather than using mutation and crossover operators, an EDA adapts the parameters of a probabilistic model during the optimization process so that new individuals are increasingly likely to be selected from good regions of the search space. In our approach the probabilistic model is a probabilistic program, which is described in the next subsection.

Figure 2 shows a diagram of our EDA. The process begins by generating a new population of individuals using the program. Each individual is a neural network and is trained on a pre-defined task, such as a classification problem. Next, the fitness of each individual is computed. We use a fitness function that captures the trade-off between classification accuracy and model complexity. The fitness function is given by

$$Fitness = k \cdot accuracy - complexity, \quad (1)$$

where $accuracy \in [0, 1]$ and is the accuracy on the new task, $complexity$ is the model complexity given by the number of new neurons divided by the number of neurons in the parent network, and $k \in R^+$. The selection process then proceeds in two separate steps. First, one individual is selected using tournament selection. This provides some degree of exploration since the individual with the highest fitness may not be selected. Second, we use a hall-of-fame that always and only contains the best performing individual found over the course of the entire optimization process. If the fitness of the best performing individual in the current population exceeds the best fitness found thus far, then it becomes the individual in the hall-of-fame. The last step is to update the parameters of the probabilistic program. This is done using the newly selected individual and the individual in the hall-of-fame, which may be the same. Updating the parameters tends to adjust the probability distribution represented by the probabilistic program in such a way that it is more likely to sample from good regions of the search space. At this stage the EDA either generates a new population using the updated program or terminates. In this work the termination criterion is that the elapsed wall-clock time has exceeded 21 min, though other criteria are possible, such as those based on a maximum number of iterations or fitness level.

Probabilistic Program Learner

Our approach uses a probabilistic program as a generative model to create new individuals. It has trainable parameters that are adjusted through a learning process, so that over the course of the optimization, individuals are sampled with increasing frequency from the best regions of the search space. We refer to this component as the *Probabilistic Program Learner* (PPL). Figure 3 shows a diagrammatic representation of the PPL. The purple boxes are random variables that specify the characteristics of the individuals, such as the number of new neurons to add. The light blue boxes are latent (unobserved) random variables that control the

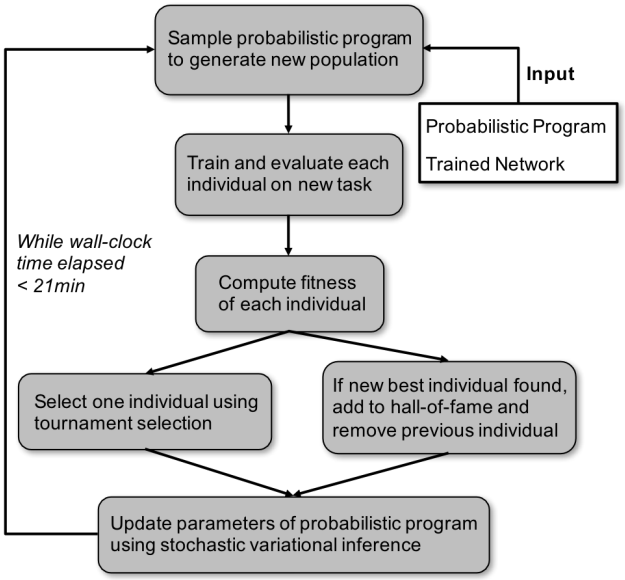

Figure 2: Our EDA used to execute neurogenesis. The uniqueness of our approach lies in the probabilistic program that learns to generate good performing individuals.

shapes of the probability distributions governing the aforementioned characteristics. These boxes each contain a descriptive name, the data type of the random variable, and the form of the probability distribution from which it is drawn. The dark blue boxes represent learnable parameters, hence the word "learnable" in the acronym PPL. They show the names of the parameters, their data types, and initial values. The program input (gray box) specifies a list of the maximum numbers of new neurons that may be added to any layer and the number of layers in the parent network, both of which are fixed throughout the optimization process.

We now describe the stochastic process of sampling from the PPL to generate a new individual. The sub-process shown at the top of Figure 3 begins by sampling a parameter $p_s \sim$ *Dirichlet*(c), where c is a learnable parameter. Next, an index is drawn from $Ind_{max} \sim$ *Multinomial*$(1, p_s)$. This index is used later in the program to access the array MaxSizes. The next sub-process begins by sampling a list of parameters $p_n \sim$ *Beta*$(alphas, betas)$, where $alphas$ and $betas$ are learnable parameters. The parameters p_n are the individual probabilities of selecting each of the layers in the parent network to receive new neurons. Next, the layers that will receive new neurons are selected according to $L \sim$ *Bernoulli*(p_n), where L is a Boolean array. The last sub-process selects how many new neurons to add to each layer $m \in N$. The control logic allows new nodes to be added only to those layers selected in the second sub-process (i.e., $L[m] == True$). The probability of adding a single new neuron to layer m is drawn from $p_a \sim$ *Beta*$(alpha_m, beta_m)$, where $alpha_m$ and $beta_m$ are

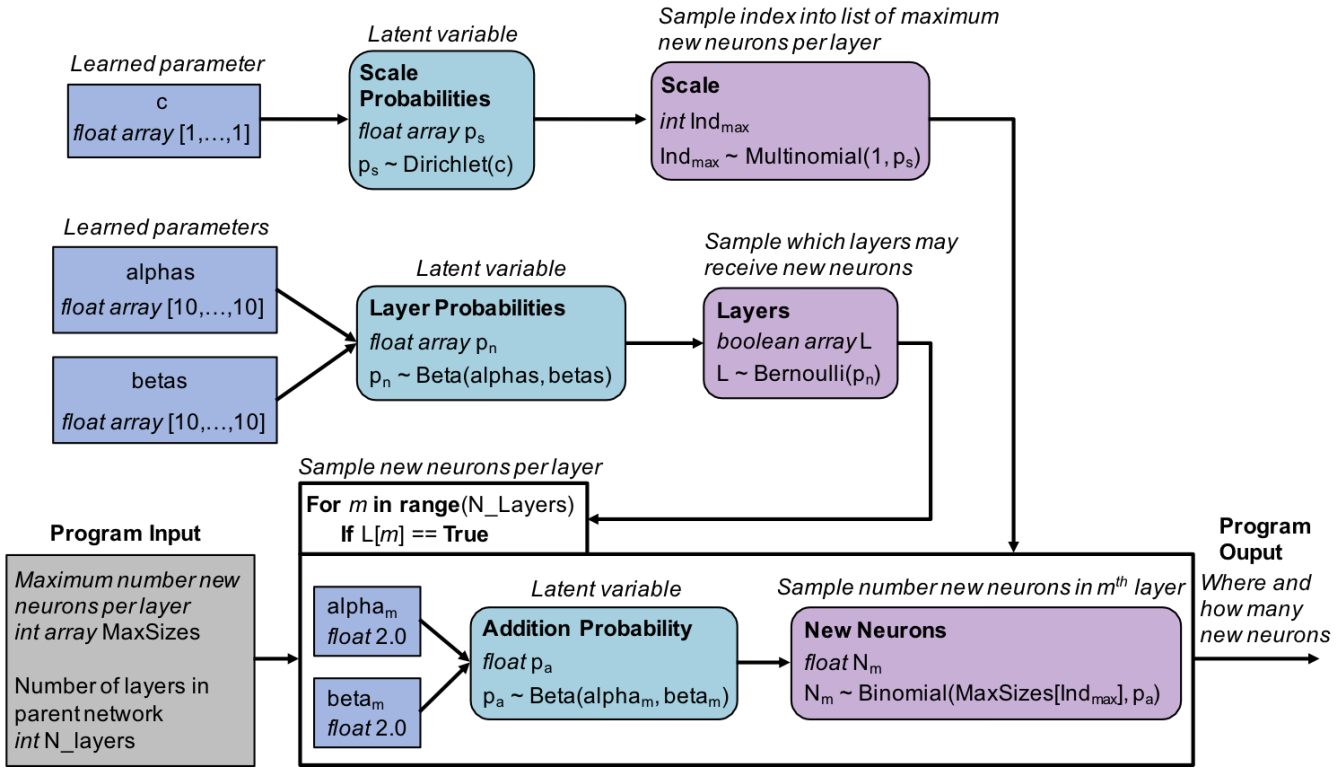

Figure 3: Illustration of the probabilistic program learner used in our approach.

learnable parameters. The total number of new neurons to add to the m^{th} layer is sampled from a binomial distribution according to $N_m \sim Binomial(MaxSizes[Ind_{max}], p_a)$, where $MaxSizes[Ind_{max}]$ is the maximum number of new nodes that can be added to any layer. Once the loop is completed, the program output specifies where and how many new neurons to add, which is sufficient to build a new individual.

Learning

In each generation of the optimization process, after selection has occurred, the learnable parameters of the probabilistic program are updated based on the attributes of the selected individuals. We refer to these attributes as observable variables and denote them by x. For each individual, the observables are the maximum number of new neurons per layer (Ind_{max}), the layers that are permitted to receive new neurons (L), and the number of new neurons in the m^{th} layer (N_m). The random variables that are not observable are referred to as latent variables and are denoted by z. The latent variables in our model are p_s, p_n, and p_a, which are described in the previous section. For each individual, the learning algorithm updates the parameters (c, $alphas$, $betas$, $alpha_m$, and $beta_m$) so that the probabilistic program is more likely to generate similar individuals in subsequent generations. We used stochastic variational inference as the learning algorithm [Hoffman *et al.*, 2013].

Experimental Setup

We applied our approach to a variant of the MNIST optical character recognition problem [LeCun *et al.*, 1998]. Our setup is shown in Figure 4. First, a "parent" neural network was trained to classify gray-scale images of handwritten digits from 0 to 9 (old task). The training dataset consisted of 1000 randomly selected images with approximately 100 images per class. The parent networks used in our experiments achieved an average testing accuracy of 87.4%. The images were flattened into a 748-dimensional vector. The parent network consisted of 3 hidden layers with hyperbolic tangent (tanh) activation functions. There were 50, 30, and 20 neurons in the first, second, and third hidden layers, respectively.

A new task was created by randomly generating a permutation mask and applying it to each of the images in the dataset. The training dataset for the new task consisted of 1000 randomly selected, permuted images with approximately 100 images per class. The permutation mask was created by randomly selecting two non-intersecting sets of pixel indices, and then swapping the corresponding pixels in each image. In our experiments, 50% of the pixels in each image were modified. The resulting new task was similar enough to the old task that some information from the parent network

was still valid, but different enough that adding new neurons significantly improved performance on the new task. If no new neurons were added, the parent network would experience catastrophic forgetting (loss of performance on the old task) due to the change in the input distribution. We set the maximum number of new neurons that could be added to any layer at 50. This leads to a total of $50^3 = 125,000$ unique individuals.

We set the parameter k in Equation 1 to 10.0. We found that smaller values of k resulted in the complexity term dominating the fitness, which resulted in a fairly simple fitness landscape with the global optimum being achieved by adding only 1 to 3 neurons at *any* layer. In contrast, setting $k = 10.0$ provided better balance between accuracy and complexity, and consequently, a more challenging optimization problem with many good, but suboptimal, local minima. For this setting, the global optimum is achieved by adding 16 new neurons to the first hidden layer and no new neurons to the second and third hidden layers. However, good, but sub-optimal, local minima can be achieved by adding new neurons to only the second or third hidden layers.

We used a genetic algorithm (GA) as basis for comparison with our approach. Genetic algorithms [Whitley, 1994] are a good fit to this problem due to the discrete nature of the search space. For the GA, an individual was encoded as a vector of length 3, where the values of the components indicated the number of new neurons to add in each of the three hidden layers. The maximum number of new neurons that could be added to any layer was 50. We used a population size of 30 and tournament selection with a tournament size of 3. Among the selected population, an individual was chosen for crossover with another randomly chosen individual with probability 0.5 and was chosen for mutation with probability 0.2. Once selected for mutation, each entry in the individual was mutated uniformly at random with probability 0.3 to a value in the interval [0, 50]. The relatively high mutation rate was found to prevent pre-mature convergence to poor solutions. We used two-point crossover with the crossover points being selected uniformly at random.

Results

We compared the performance of our approach to those of the GA using the experimental setup described in the previous section. We ran a total of 150 trials for the two approaches combined. Each trial started with a newly initialized population, and parameters in the case of the PPL, and then the optimization process was run for 21 minutes of wall-clock time. Wall-clock time was used because both the GA and our approach are randomized search algorithms and on average one generation of our approach takes longer than the GA due to the probabilistic inference. The average accuracy attained on the new task by the solutions from our approach was 87.7% and for the GA it was 87.6%. The relationship between accuracy and fitness is shown in Eq. 1. The fact that the accuracy is close to that of the parent network on the old task (87.4%) illustrates the effectiveness of neurogenesis for continual learning. The typical solution found by PPN was 12-17 new neurons added only to the first layer. The typical solution found by the GA had greater variability and was either 11-20 new neurons added only to the first layer or 13-20 new neurons added only to the second layer. Based on a large number of experimental trials, we observe that the global optimal solution is approximately 16 new neurons added only to the first layer.

The results of the first analysis are shown in Figures 5 and 6. Each figure shows the average best fitness achieved by the PPN (blue) and GA (red) as a function of elapsed run time in minutes. Figure 5 is from 2 to 5 minutes and the Figure 6 is from 5 to 21 minutes. Based on the curves in Figure 5, it can be seen that on average PPN reaches near optimal solutions (a fitness of about 60.0) within the first 2 minutes of simulation time, whereas it takes the GA about 5 minutes to reach a comparable level of fitness. Figure 6 shows that in the long run our approach continues to improve and outperform the GA.

The next analysis examined the consistency with which the PPN and GA were able to find solutions that achieved particular fitness levels. Figure 7 shows the fraction of trial runs on which the best fitness found exceeded various lower bounds. The results for the PPN are in blue and those of the GA are in red. We can see that for each fitness lower bound on the x-axis, our approach exceeds the success frequency of the GA, and for the higher (more difficult to achieve) fitness levels (> 61) the success rate of our approach is at least double that of the GA. These results demonstrate that our approach consistently finds better solutions than the GA.

Discussion

We have demonstrated the effectiveness of our approach by showing that it is able to consistently find better performing solutions than the GA and it is able to do so faster. We believe that one of the primary reasons for this is that the probabilistic program represents a distribution over distributions, which promotes exploration and thus the discovery of better solutions more quickly. As can be seen in Figure 3, the parameters (p_s, p_n, p_a) of the distributions governing the observable random variables (Ind_{max}, L, N_m), which control the structural attributes of an individual, are themselves (latent) random variables. When running the program in the forward direction to generate a new individual, we must sample the latent random variables first and then use them to determine the distribution over the observables.

Another reason for the effectiveness of our approach is that it is easy to incorporate prior knowledge about a domain directly into the search process. Our probabilistic program, rather than using crossover and mutation operators to implicitly define the search distribution, enables the designer/user

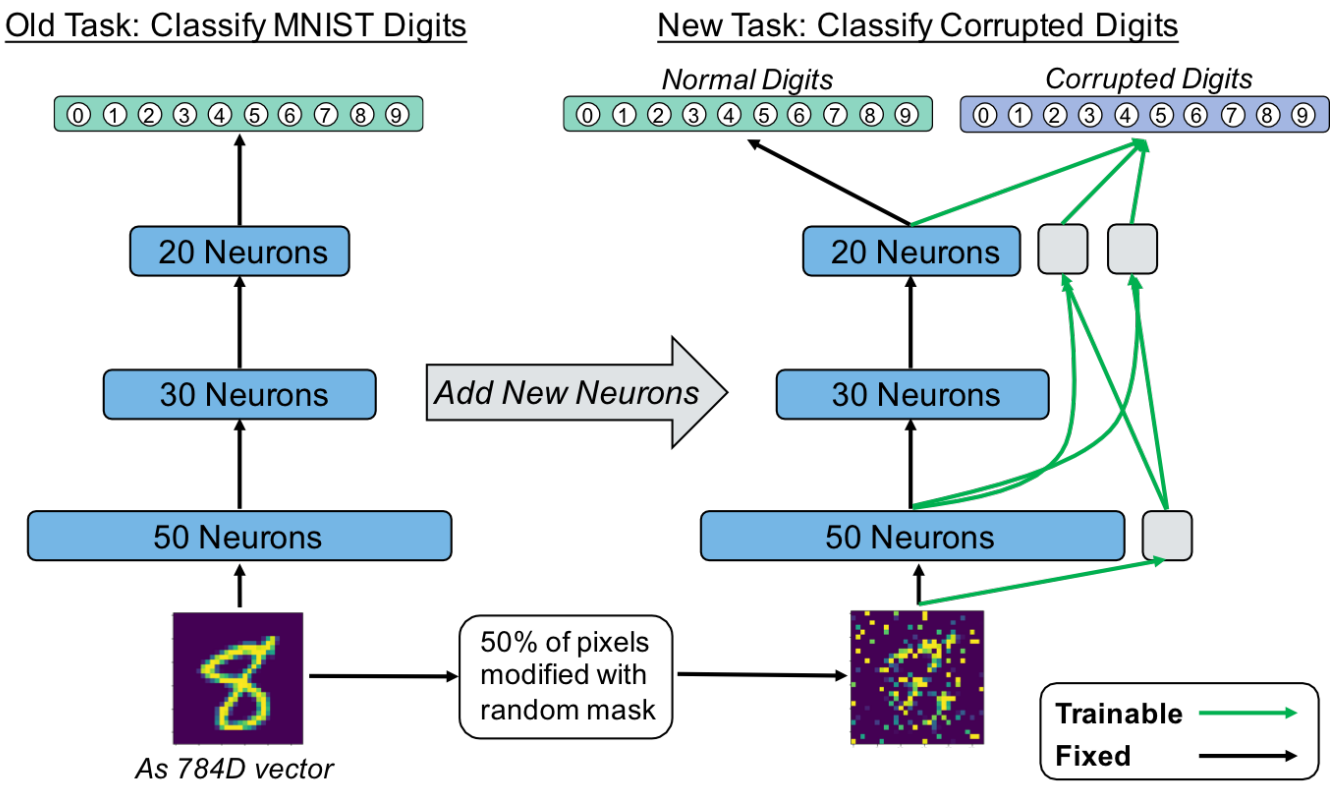

Figure 4: Illustration of our experimental setup. On the left, a *parent* network is trained to classify images of handwritten digits (0-9). A new task is created by randomly generating a permutation mask that randomly and uniformly selects 50% of the pixels to be swapped. The mask is then applied to each image. This way the new task is different, but still related to the old task. Neurogenesis is shown on the right. Our approach identifies where new neurons should be placed to minimize network complexity while achieving good classification performance on the new task (corrupted digits). The new (green) connections have trainable weights, while the weights on the old (black) connections remain fixed to prevent catastrophic forgetting.

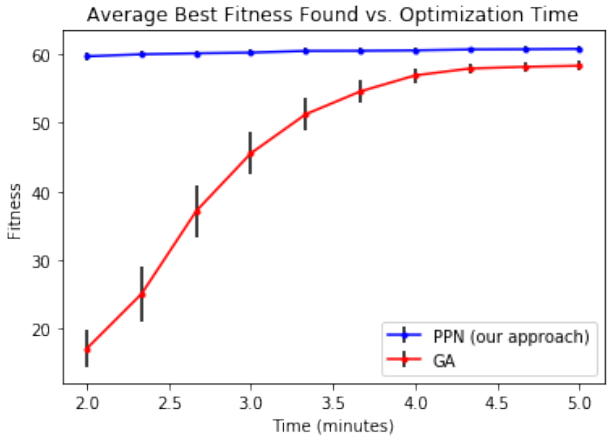

Figure 5: Comparison of the average best fitness found over the first 5 minutes of optimization time. The fitness values used for plotting have been shifted by -8.0 and scaled by 100.0. This was done solely for the purpose of improving visual interpretability of the results. Our approach (blue) quickly finds near optimal solutions, while the GA (red) is slower to find good solutions. Our approach starts much higher due to the informative prior in the probabilistic program. As a point of reference, when a parent network makes predictions on the new task prior to neurogenesis ($complexity = 0$ in Eq. 1) the expected fitness is -400 ($accuracy = 0.4$). The error bars are 90% confidence intervals.

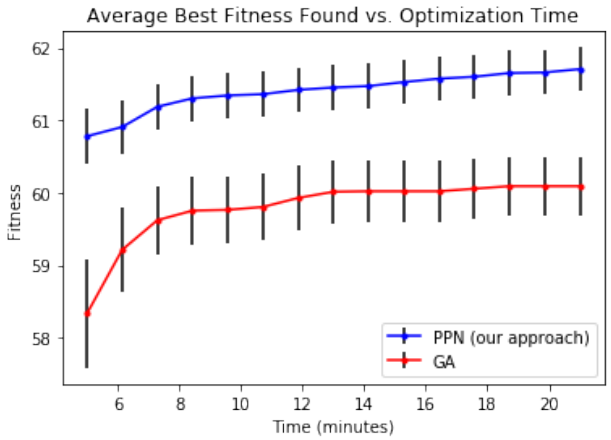

Figure 6: Comparison of the average best fitness found over the next 5 to 21 minutes of optimization time. The fitness values used for plotting have been shifted by -8.0 and scaled by 100.0 to improve readability. Our approach (blue) continues to improve, while the GA (red) stagnates. The error bars are 90% confidence intervals.

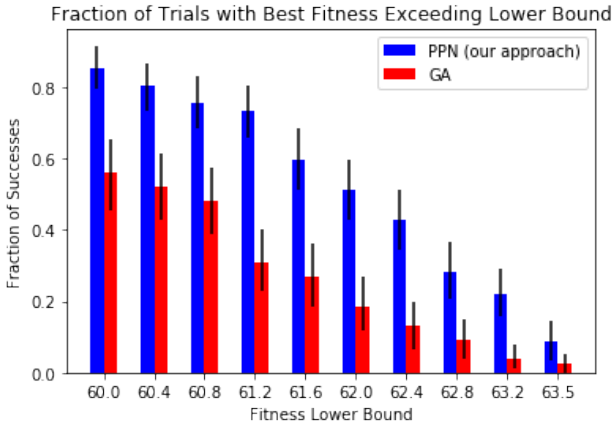

Figure 7: Comparison of how often our approach (blue) finds solutions with fitness values exceeding various lower bounds. The fitness values used for plotting have been shifted by -8.0 and scaled by 100.0 to improve readability. For each fitness lower bound, our approach exceeds the GA. The error bars are 90% confidence intervals.

to build in semantics that are believed to be important. We were able to take advantage of this with the portion of the program that selects the layers that are permitted to receive new neurons. Our original formulation of the PPL did not have this logic, but we quickly realized based on empirical observations that it is important to efficiently explore regions of the search space where one or more layers do not receive any new neurons. Subsequently, a simple modification to our program enabled this enhancement.

Conclusions

We have demonstrated a new approach to neurogenesis based on probabilistic program learning. Our method helps to address the continual learning problem wherein learning occurs sequentially on different tasks. In this setting, when learning a new task, it is of utmost importance to avoid catastrophic forgetting of previously learned tasks. However, previously learned information that generalizes to the new task should be leveraged to limit growth in the complexity of the learning model.

Past work on neurogenesis has largely focused on employing ad hoc rules or leveraging local statistical properties of neurons and/or layers [Draelos *et al.*, 2017; Yoon *et al.*, 2017] to decide the number and location of the new neurons. In contrast, our approach employs an efficient global search process using a probabilistic program, which has a higher probability of finding optimal or near-optimal solutions. Moreover, the probabilistic program process allows for easy integration of new features and rules to better regulate the addition of new neurons. In the future we will also validate our framework for continual learning us-

ing reinforcement learning tasks (e.g., those from the OpenAI Gym) and more challenging datasets such as ImageNet [Krizhevsky *et al.*, 2012], and make comparisons with additional methods such as evolution strategies. Another avenue we plan to pursue is extending our approach to convolutional neural networks (CNNs). In this application, instead of adding new neurons, PPL would add new feature maps to the convolutional layers.

Acknowledgments

We thank Risto Miikkulainen, Sebastian Risi, Andrea Soltoggio, and Jean-Baptiste Mouret for conceptual discussions surrounding the work. This material is based upon work supported by the United States Air Force and DARPA under contract no. FA8750-18-C-0103. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA.

References

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

Bernard Ans, Stephane Rousset, Robert M French, and Serban Musca. Self-refreshing memory in artificial neural networks: Learning temporal sequences without catastrophic forgetting. *Connection Science*, 16(2):71–99, 2004.

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Achieving deep reinforcement learning without catastrophic forgetting. *arXiv*, 1812.02464, 2018.

T.J. Draelos, N.E. Miner, C.C. Lamb, J.A. Cox, C.M. Vineyard, K.D. Carlson, W.M. Severa, C.D. James, and J.B. Aimone. Neurogenesis deep learning: Extending deep networks to accommodate new classes. *International Joint Conference on Neural Networks (IJCNN)*, pages 526–533, 2017.

M.D. Hoffman, D.M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Nicholas Ketz, Soheil Kolouri, and Praveen K. Pilly. Using world models for pseudo-rehearsal in continual learning. *arXiv*, 1903.02647, 2019.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Soheil Kolouri, Nicholas Ketz, Xinyun Zou, Jeffrey Krichmar, and Praveen K. Pilly. Attention-based selective plasticity. *arXiv*, 1903.06070, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

P. Larraaga and J.A. Lozano. Estimation of distribution algorithms: A new tool for evolutionary computation. vol. 2. *Springer Science Business Media*, 2001.

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

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419, 1995.

Luiza Mici, German Parisi, and Stefan Wermter. Self-organizing neural network architecture for learning human-object interactions. *Neurocomputing*, 307:14–24, 2018.

G.I. Parisi, R. Kemker, J.L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

Jose Part and Oliver Lemon. Incremental online learning of objects for robots operating in real environments. In *Proceedings of the Joint IEEE Int. Conf. on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 304–310, 2017.

A.A. Rusu, N.C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv*, 1606.04671, 2016.

D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

J. Yoon, E. Yang, J. Lee, and S.J. Hwang. Lifelong learning with dynamically expandable networks. *arXiv*, 1708.01547, 2017.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3987–3995. JMLR.org, 2017.