An Evolutionary Approach to Variational Autoencoders

Jeff Hajewski
Department of Computer Science
University of Iowa
Iowa City, IA, USA
jeffrey-hajewski@uiowa.edu

Suely Oliveira
Department of Computer Science
University of Iowa
Iowa City, IA, USA
suely-oliveira@uiowa.edu

***Abstract*—Variational autoencoders are an important tool in the domain of generative data models and yet they remain difficult to design due to a lack of intuition as to how to best design the corresponding encoder and decoder neural networks in addition to determining the size of the latent dimension. Furthermore, there is no definitive guidance on how one should structure these networks. Designing an effective variational autoencoder typically requires fine-tuning and experimenting with different neural network architectures and latent dimensions, which, for large datasets, can be costly both in time and money. In this work we present an approach for designing variational autoencoders based on evolutionary neural architecture search. Our technique is efficient, avoiding redundant computation, and scalable. We explore how the number of epochs used during the neural architecture search affects the properties of the resulting variational autoencoders as well as study the characteristics of the learned latent manifolds. We find that evolutionary search is able to find highly performant network even when the networks are evaluated after only two epochs of training. Using this insight we are able to dramatically reduce the overall computational requirements of our neural architecture search system applied to variational autoencoders.**

***Index Terms*—variational autoencoder, neural architecture search, evolutionary algorithm, distributed system**

I. Introduction

Despite the recent success of many deep learning techniques, many of them rely on on labeled data. Unfortunately this labeled data can be expensive to create because it requires experts to manually label each individual piece of data. Unsupervised learning's ability to find patterns and learn underlying distributions of the data without requiring labeled data is part of what makes unsupervised learning so appealing.

Variational autoencoders [1], a generative unsupervised learning technique, have experienced success in representation learning. Variational autoencoders are similar to traditional autoencoders [2]–[4] only in their design, consisting of two neural networks referred to as the encoder and decoder networks. Where a typical autoencoder learns a reduced representation of its input data, the variational autoencoder learns a latent probability distribution of the input data, which can be used to generate new data samples. Variational autoencoders have also been used in areas such as image captioning, educational data mining [5], [6], and even outlier removal [7].

Despite their wide success, there is little work or guidance on how one should design the neural networks that make up the variational autoencoder as well as how that design should change depending on the task — this includes determining the dimension of the latent space. This is made more difficult by the fact that training and evaluating a given variational autoencoder is a time and resource intensive task, which makes each iteration costly.

Neural architecture search tries to solve this problem by treating it as an optimization problem: find the neural network architecture that minimizes the validation loss for a given problem. It has achieved great success in designing neural networks that match or beat current state-of-the-art results [8]–[11]. The two main approaches to neural architecture search are reinforcement learning and evolutionary algorithms; we focus on the evolutionary algorithm approach in this paper.

In this work we present an efficient evolutionary approach to building variational autoencoders. The efficiency in our algorithm comes from caching caching previously seen architectures and imposing an immutability constraint on the genotypes (the encoded representation of the neural network architectures), both of which are largely inspired by functional programming. We run this algorithm on a distributed system that trains and evaluates candidate neural network architectures in parallel.

The rest of this paper is organized as follows. We introduce variational autoencoders in Section II, followed by a brief overview of evolutionary neural architecture search and a description of our evolutionary algorithm in Section III. In Section IV we discuss related work and in Section V we describe our experiments and present their results.

II. Variational Autoencoders

Variational autoencoders are an unsupervised learning technique that build a generative model of the data. Given a dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$ of m datapoints in $\mathbb{R}^n$, we assume the data is generated from a distribution $p(\mathbf{x}|\mathbf{z})$, where $\mathbf{z}$ is sampled from the parameterized latent distribution $p(\mathbf{z})$. The space from which $\mathbf{z}$ is sampled is referred to as the *latent space*, and is typically a lower dimensional space than that of $\mathbf{x}$. We can

978-1-7281-3783-4/20/$31.00

derive the distribution of $\mathbf{x}$ using the conditional and prior distributions $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$.

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \tag{1}$$

The latent distribution is commonly assumed to be the multivariate Gaussian, $N(\mu, \sigma^2 \cdot \mathbb{I})$, where $\mathbb{I} \in \mathbb{R}^{d\times d}$ is the identity matrix of size $d\times d$ for latent dimension d. We can approximate the posterior distribution $p(\mathbf{x}|\mathbf{z})$ via a neural network [12] and sampling $\mathbf{z} \sim N(\mu, \sigma^2 \cdot \mathbb{I})$. Of course, this latent distribution works both ways — if we can generate a sample $\mathbf{x}$ given a sample z from the latent distribution via $p(\mathbf{x}|\mathbf{z})$, then we should be able to generate a latent sample $\mathbf{z}$ given a sample $\mathbf{x}$ from the data distribution via a distribution $p(\mathbf{z}|\mathbf{x})$. As with the distribution $p(\mathbf{x}|\mathbf{z})$, we can approximate $p(\mathbf{z}|\mathbf{x})$ with a neural network. We refer to this approximate distribution as $q(\mathbf{z}|\mathbf{x})$. In practice, we assume $\mathbf{z} \sim N(\mu, \sigma^2 \cdot \mathbb{I})$ and use the neural network to generate μ and σ, rather than $\mathbf{z}$ itself. The learning process tries to minimize the distance between the posterior probability distribution, $p(\mathbf{x}|\mathbf{z})$ and the true distribution $p(\mathbf{x})$. The total loss is defined as the sum of the Kullback-Leibler (KL) divergence [13], which measures the similarity between two probability distributions, and the expected reconstruction error, and is given by equation (2).

$$L = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[-\log p(\mathbf{x}|\mathbf{z})] - KL(q(m\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{2}$$

III. Evolutionary Neural Architecture Search

Neural architecture search (NAS) is a family of algorithms and techniques for designing neural networks. The two most popular approaches are reinforcement learning and evolutionary techniques. We use an evolutionary search algorithm in this work because evolutionary algorithms are simpler to implement and understand in addition to typically requiring less computational resources when compared to their reinforcement learning based counterparts. The main challenge the evolutionary algorithm needs to overcome is that of exploration versus exploitation. We need to make sure we give every architecture a fair evaluation in the sense that it has trained long enough such that the evaluation phase, where it is tested on validation data, accurately assesses its effectiveness on the task at hand. On the other hand, given a finite amount of computational resources, we want to maximize the number of explored architectures. These two aspects of evolutionary neural architecture search are at odds with each other — longer training time consumes computational resources, leaving less resources for the remaining search.

Formally, the problem we solve with neural architecture search is given by equation (3), where L is a loss function, ψ is a network architecture, and $\mathcal{A}$ is the space of all neural network architectures.

$$\Psi^* = \min_{(\psi_e,\psi_d)\in\mathcal{A}} L(\mathbf{X}, \psi_d(N(\psi_e(\mathbf{X})))) \tag{3}$$

Where $\Psi = (\psi_e, \psi_d)$ and $N(\psi_e(\mathbf{X}))$ represents the sample $\mathbf{z} \sim N(\mu, \sigma^2 \cdot \mathbb{I})$ for $\mu, \sigma^2 = \psi_e(\mathbf{X})$. Note that the search is actually for two network architectures, the encoder and decoder architectures. Using the reparameterization trick suggested by Kingma et al. [1], the loss function we use given by equation (4).

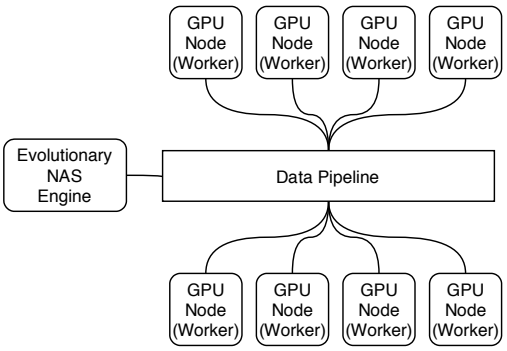


Figure 1: High-level overview of the system architecture.

$$L(\theta_x, \theta_z; \mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L} \log p(\mathbf{x}|\mathbf{z}) + \frac{1}{2}\sum_{j=1}^{J}(1 + 2\log(\sigma_j) - \mu_j^2 - \sigma_j^2) \tag{4}$$

This trick simply reparameterizes the prior distribution of $\mathbf{z}$ from $N(\mu, \sigma^2 \cdot \mathbb{I})$ to

$$\mathbf{z} = \mu + \sigma^2 \cdot \epsilon$$

where $\epsilon \sim N(0, 1)$. The training process is described by Algorithm 1. Lines 7 and 8 show the sampling and reparameterization of $\mathbf{z}$. The function `AdamUpdate` encapsulates the computation of the gradient and the parameter updates of the encoder and decoder neural networks via stochastic gradient descent.

Algorithm 1 Variational autoencoder training algorithm.

1: **procedure** TrainVAE($\mathbf{X}, \Psi, n, m$)
2: $(\psi_e, \psi_d) \leftarrow \Psi$ ▷ Destructure VAE
3: **for** $i = 1$ **to** n **do** ▷ n - # of epochs
4: **for** $j = 1$ **to** $\lceil |\mathbf{X}|/m \rceil$ **do**
5: $\mathbf{xs} \leftarrow$ `sampleMB`$(\mathbf{X}, m)$ ▷ Sample mini-batch
6: $(\mu, \sigma^2) \leftarrow \psi_e(xs)$
7: $\epsilon \leftarrow N(0, \mathbb{I})$
8: $\mathbf{z} \leftarrow \mu + \sigma^2 \cdot \epsilon$
9: $\hat{\mathbf{xs}} \leftarrow \psi_d(\mathbf{z})$
10: `AdamUpdate`$(\Psi, L(\mathbf{xs}, \hat{\mathbf{xs}}))$
11: **end for**
12: **end for**
13: **end procedure**

A. System Architecture

Figure 1 shows the architecture of the distributed system we used to perform this search. The system separates model

generation, which uses very little computational resources, and model evaluation, which uses extensive computational resources, across different node types. Model generation does not require any specialized hardware while model evaluation requires a GPU. The evolutionary NAS engine handles the evolution of the network architecture. Generated network architectures are encoded in Protocol Buffers [14] and sent to the GPU nodes (referred to as workers) via gRPC [15], [16]. One of the advantages of this approach is that the data pipeline is agnostic to the type of data it transmits, which makes the overall system more flexible in its application.

The training and evaluation of the candidate neural network architectures is done asynchronously by the workers. A central work queue–part of the data pipeline–feeds tasks to the workers who complete the training and evaluation and return the results. We are able to achieve near-linear scaling in the number of workers because the system workload is computationally bounded (rather than bounded by the speed of communication) and because the workers are stateless, allowing the system to recover after a worker failure. A worker sends its result back through the data pipeline back to the NAS engine. The engine uses these evaluations to generate the next population of candidate architectures. The system infrastructure is written in Go while the workers are written in Python. The code is freely available on GitHub at `github.com/j-haj/brokered-deep-learning`.

B. Evolutionary Algorithm

We use $(\mu + \lambda)$ selection where μ parents generate λ offspring and we select the top μ individuals to survive the next generation. Fitness is defined as the reciprocal of the loss, giving it the nice properties that the loss is easily computed from the fitness and that high fitness corresponds to small loss. The possible mutation operations are appending a new layer to the network or modifying an existing layer. We limit the number of layers to five for each of the neural networks and limit the layer type to linear layers. This results in fully connected neural networks with no more than five hidden layers. If a genotype (that is, the encoding that represents a given neural network architecture) tries to append a layer when it has the maximum allowed layers it falls back to modifying a randomly selected existing layer. Layer sizes range from 50 to 1,000 (inclusive) in steps of 50. This process occurs for both the encoder and decoder networks. Additionally, the genotype can mutate the latent dimension, which allows the search to explore different latent spaces. Latent dimension ranges from 10 to 100 (inclusive) in steps of 10. Since the encoder and decoder networks evolve independently, there are a total of $10 \cdot 20^{10} \approx 10^{14}$ or 100 trillion different architectures.

Part of the efficiency of our system comes from the evolutionary algorithm that drives the NAS engine. The set `seen` tracks architectures that have been seen previously. If the search happens to find a previously seen architecture (lines 8 and 9) it sets the fitness to -1, which results in the genotype being dropped during the selection process. Because selection always keeps the top μ individuals, a previously seen individual will either still be in the top set of individuals or will have been replaced with a better performing individual. Either way, we can safely discard the re-discovered genotype.

Algorithm 2 Generational evolutionary neural architecture search.

```
1:  procedure EvoNASSearch(n, m)
2:      maxGeneratsion ← m
3:      seen ← ∅                          ▷ Tracks seen architectures
4:      p ← InitializePopulationOfSize(n)
5:      for i = 1 to maxGenerations do
6:          p′ ← p ∪ p.produceOffspring()
7:          for o ∈ p′ do
8:              if o.isEvaluated() or o ∈ seen then
9:                  if o ∈ seen then
10:                     o.fitness ← −1    ▷ Drop genotype
11:                 end if
12:                 continue              ▷ Skip evaluation
13:             end if
14:             Send o to task queue
15:             seen.add(o)
16:         end for
17:         r ← ReceiveResults()
18:         ProcessResults(r)
19:         p ← SelectTopLambda(r, λ)     ▷ (μ + λ)
20:     end for
21: end procedure
```

We achieve an additional improvement in algorithm efficiency by making the genotypes immutable. Once a genotype has been evaluated–by building, training, and evaluating the corresponding variational autoencoder–its fitness will not change. This is a simple aspect of the algorithm but avoids redundant evaluations, which is particularly important for larger neural network sizes. A natural question is how genotypes are mutated in the evolutionary algorithm if they are immutable — we create a clone of the genotype and modify it during the creation process for both gene mutations and crossover.

IV. Related Work

Although there has been extensive work exploring different aspects of variational autoencoders [7], [17]–[19] , to the best of our knowledge there is no prior work exploring evolutionary neural architecture search techniques to designing variational autoencoders. Previous work most similar to this topic is that of [20], [21] explores the use of evolutionary neural architecture search techniques to autoencoders. Aside from our focus on variational autoencoders rather than traditional autoencoders, our work differs in that it uses an evolutionary algorithm that always generates valid network architectures and does not require the encoder and decoder networks to be the same architecture.

Much work has been done on the domain of neural architecture search, mostly applied to classification. Work such as [8], [11], [22] focuses on reinforcement learning based approached to neural architecture search. More related to our work is

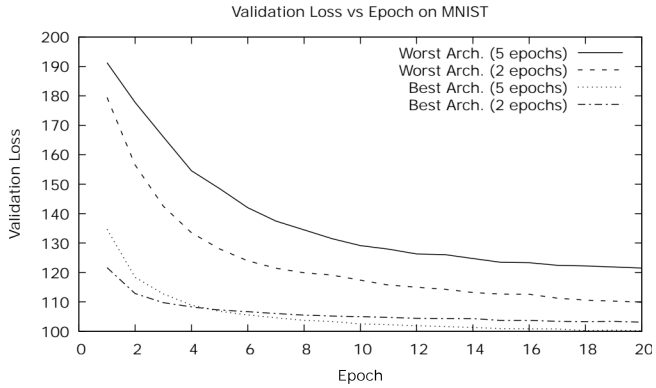


Figure 2: Comparison of validation loss over number of epochsfor the best and worst architectures on the MNIST dataset. Thenetwork was evaluated after either two or five epochs duringthe architecture search.

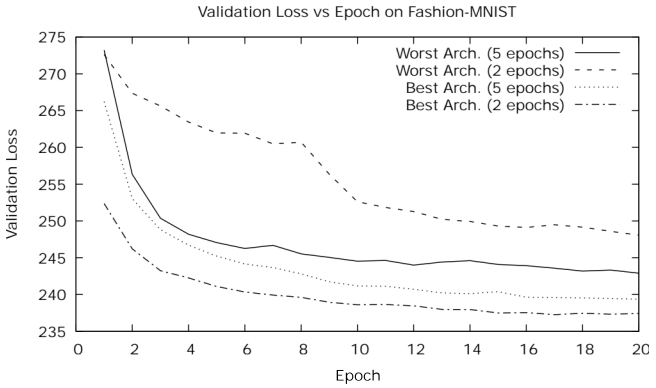


Figure 3: Comparison of validation accuracy over number ofepochs for the best and worst architectures on the Fashion-MNIST dataset. The network was evaluated after either twoor five epochs during the architecture search.

that of [9], [23], [24], which focuses on evolutionary based approaches. A majority of the prior work in reinforcement learning and evolutionary algorithm-based approaches to neural architecture focuses on image classification, rather than learning generative models. Despite the difference in application with our work, many of these techniques are applicable to other areas of application, such as neural architecture search applied to variational autoencoders.

V.E XPERIMENTS

Our experiments explore the ability of evolutionary neural architecture search to find effective variational autoencoders. We evaluate the found architectures by exploring their latent spaces as well as illustrating the impact of the number of epochs used during the architecture search on the training speed of the found architecture.

All experiments were performed using Nvidia K80 GPUs on Amazon Web Services servers. We use PyTorch [25] to implement the neural networks. The variational autoencoder architectures are evaluated using the MNIST [26] and Fashion-MNIST [27]. We use a mini-batch size of 128 for training, Adam [28] for updating the neural network parameters, and unless otherwise noted, results are based on 20 epochs of training. Due to resource constraints we were unable to explore larger datasets such as CIFAR-10 [29].

Table I shows the best and worst architectures found in the two and five epoch searches for both the MNIST and Fashion-MNIST dataset. The architecture description consists of three parts separated by a vertical line. The first part represents the encoder network architecture where the linear layer sizes are separated by commas. The second part (in between the vertical bars) represents the latent dimension, and the final comma-separated list represents the decoder architecture.

A. *Effect of epochs on search*

Although the number of epochs used in the evolutionary search algorithm may seem like a minor detail, getting this hyper-parameter correct has a real impact on the progress and results of the search process. More complex neural networks– that is, neural networks with more parameters–typically need more passes through the data to get to a point where their loss begins to level off. This is clear in both Figures 2 and 3. The challenge is that the epoch at which this plateau begins is not known *a priori* and changes with the neural network architecture.

Figure 2 shows comparison of validation loss with respect to epoch for the best and worst architectures found during the evolutionary neural architecture search. During the search, the neural network architectures are evaluated after either two epochs or five epochs. As expected, the architectures found from searching with two epoch selection start at a lower loss and their loss decreases more rapidly. This is a result of the selective pressure resulting from the early evaluation after only two epochs. The found architectures have adopted the feature they attain better performance earlier in their training. One would expect that architectures that are given more epochs to train prior to selection may have a better overall performance because the search is able to better evaluate the architecture, rather than simply selecting the networks that train faster. This is seen in the top performing architectures but not exhibited in the worst performing architectures.

Figure 3 shows the same comparison as Figure 2 but for the Fashion-MNIST dataset. The Fashion-MNIST experiment shows similar, if not more extreme, results to the MNIST experiment. The interesting aspect of these results is that the the best architecture found in the two epoch search always outperforms the best architecture found the five epoch search. This result is surprising on two levels. As mentioned above, we expect the longer training time allows the search process to better evaluate candidate network architectures. Additionally, even if the additional training received in the five epoch search does not help the search, it shouldn't hurt the search either. Most likely this is a result of the two epoch search happening to find an ideal architecture that the five epoch search, by

Table I: Found architectures

Dataset	Search Epochs	Rank	Architecture
MNIST	2	Best	$850\|20\|1000$
		Worst	$200, 500\|20\|1000, 650, 50$
	5	Best	$800, 900\|30\|150, 1000$
		Worst	$850, 700, 400\|30\|850, 50, 1000, 900, 50$
Fashion-MNIST	2	Best	$1000\|20\|350, 1000$
		Worst	$950, 850, 250, 200\|10\|150, 500, 750$
	5	Best	$1000\|60\|700, 750$
		Worst	$900, 50, 700\|60\|50, 750, 600, 950$

chance, did not discover.

B. Latent space manifold

We can visualize the learned manifold by reducing the latent dimension to two and sampling from the inverse of the Gaussian cumulative distribution function on the unit square $[0, 1] \times [0, 1]$. Figure 4 shows the learned manifolds from both the best VAE architecture found in the two epoch search, shown in Figure 4a, and the best VAE architecture found in the five epoch search, shown in Figure 4b. It is important to note that this analysis is purely qualitative and does not say anything about which model is preferable. Although there are similarities between the two manifolds, they are surprisingly quite different. The manifold depicted in Figure 4a consists of about 50% footwear, while the manifold in Figure 4b is only about 15% footwear. We point this out because although we would expect the manifolds to differ from each other–they are completely different network architectures–it is intriguing the learned manifolds are so different.

C. Sampling the latent space

Figure 5 compares samples from the best architectures found in the two and five epoch searches. Recall from Figure 2 that the five-epoch search achieved a better loss after 20 epochs when compared to the best architecture from the two epoch search. The digits in Figure 5a are blurrier than the digits in Figure 5b. For example, many of the 8s in Figure 5a are harder to discern compared to the 8s of Figure 5b, where the lines are more solid and the centers more pronounced. In general, most digits in Figure 5b have thicker lines, making them more defined and legible. It is clear comparing these two figures that the variational autoencoder found in the five epoch search has learned a more accurate distribution of the data than the architecture found in the two epoch search. This is likely due to the greater learning capacity, as the neural networks in the five epoch architecture are larger and have more parameters. Larger networks are more flexible models due to the increased number of learnable parameters with the trade-off that they are also more likely to overfit the data.

D. Observations

One aspect of the search we noticed was that architectures containing information bottlenecks, a triple of layers consisting of two high node count layers surrounding a layer of size 50, tended to perform poorly compared with other architectures. For example, the worst performing network for Fashion-MNIST from the five epoch search, shown in Table I, has a layer of 900 nodes, followed by a layer of 50 nodes, followed by a layer of 700 nodes. By the time data gets to the 700 node layer much of the information transferred from the 900 node layer to the 50 node layer is lost simply because the 50 node layer does not have the capacity to transfer it. Once this information is lost it cannot be recovered and the overall performance of the network suffers as a result. This is largely unsurprising — a common heuristic when designing neural networks is to gently decrease the dimension of the layers or gently increase the dimension of the layers.

VI. Conclusion

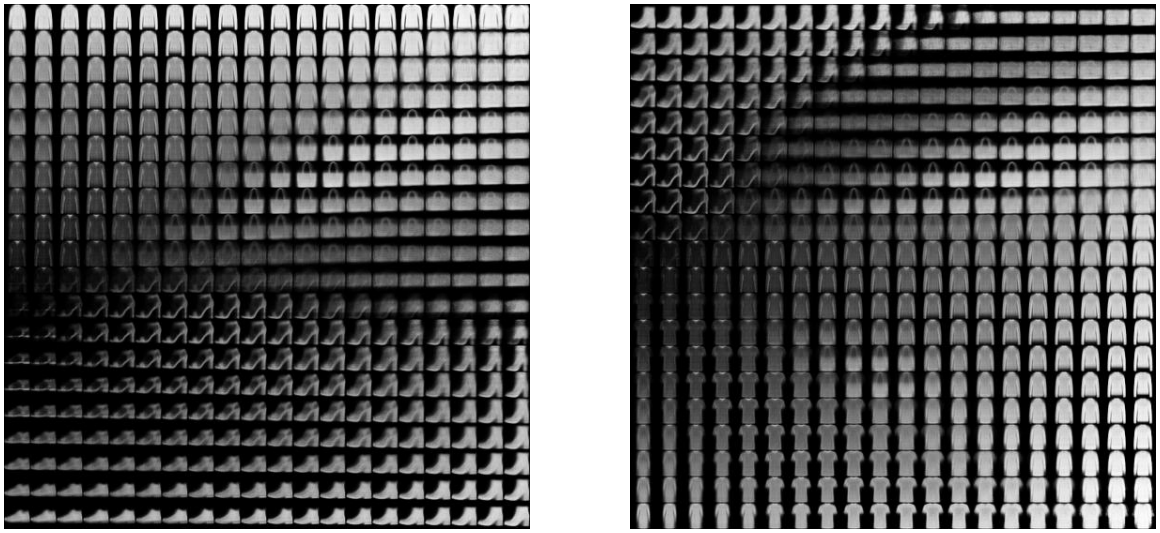
We have presented an evolutionary algorithm for evolving variational autoencoders and demonstrated its effectiveness on the MNIST and Fashion-MNIST datasets. Our algorithm is able to efficiently find high-performing architectures for variational autoencoders. One area for future work is comparing the generational algorithm we used with a tournament-style, or steady-state [30], [31], this would improve the throughput of the system by reducing idleness at the workers; however, it would also impact the selection process, possibly resulting in some genotypes surviving that would have otherwise been removed from the population. Another area of future work is studying this approach on more complex datasets; due to limits on computational resources we were unable to explore larger datasets.

Neural architecture search is a promising area of deep learning. Continued efforts in improving its efficiency and widening its areas of application will have a positive impact on the field of deep learning. We have shown that indeed neural architecture search can be an effective approach in building generative models. This is important, as generative models will likely play an important role in future deep learning and artificial intelligence systems.

References

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[2] T. Hrycej, *Modular Learning in Neural Networks: A Modularized Approach to Neural Network Classification*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1992.



(a) (b)

Figure 4: (a) Learned manifold from the best architecture found in the two epoch search, trained for 20 epochs. (b) Learned manifold from the best architecture found in the five epoch search, trained for 20 epochs.

(a) (b)

Figure 5: (a) Samples of $p(\mathbf{x}|\mathbf{z})$ from the best architecture found in the two epoch search after 100 generations, trained for 20 epochs. (b) Samples from $p(\mathbf{x}|\mathbf{z})$ from the best architecture found in the five epoch search after 100 generations, trained for 20 epochs.

[3] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, 2010.

[4] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), pp. 1096–1103, ACM, 2008.

[5] M. Curi, G. Converse, J. Hajewski, and S. Oliveira, "Interpretable variational autoencoders for cognitive models," in *International Joint Conference on Neural Networks*, 2019.

[6] G. Converse, M. Curi, and S. Oliveira, "Autoencoders for educational assessment," in *International Conference on Artificial Intelligence in Education*, 2019.

[7] B. Dai, Y. Wang, J. Aston, G. Hua, and D. P. Wipf, "Hiddent talents of the variational autoencoder," *CoRR*, vol. abs/1706.05148, 2017.

[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.

[9] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017.

[10] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, (New York, NY, USA), pp. 1061–1068, ACM, 2013.

[11] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the 35th*

International Conference on Machine Learning (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 4095–4104, PMLR, 7 2018.
[12] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
[13] S. Kullback, "Letter to the editor: The kullback-leibler distance," *The American Statistician*, vol. 41, pp. 340–341, 11 1987.
[14] K. Varda, "Protocol buffers: Google's data interchange format," tech. rep., Google, 6 2008.
[15] Google, "grpc. Accessed on March 2019"
[16] X. Wang, H. Zhao, and J. Zhu, "Grpc: A communication cooperation mechanism in distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 27, pp. 75–86, July 1993.
[17] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," in *Advances in neural information processing systems*, pp. 3738–3746, 2016.
[18] D. Kingma, T. Salimans, R. Josefowicz, X. Chen, I. Sutskever, M. Welling, *et al.*, "Improving variational autoencoders with inverse autoregressive flow," 2017.
[19] X. Hou, L. Shen, K. Sun, and G. Qiu, "Deep feature consistent variational autoencoder," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1133–1141, IEEE, 2017.
[20] S. Lander and Y. Shang, "Evoae – a new evolutionary method for training autoencoders for deep learning networks," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2, pp. 790–795, 7 2015.
[21] M. Suganuma, M. Ozay, and T. Okatani, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 4771–4780, PMLR, 7 2018.
[22] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.
[23] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *CoRR*, vol. abs/1711.00436, 2017.
[24] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 2902–2911, PMLR, 2017.
[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.
[26] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.
[27] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.
[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.
[29] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"
[30] R. Enache, B. Sendhoff, M. Olhofer, and M. Hasenjäger, "Comparison of steady-state and generational evolution strategies for parallel architectures," in *Parallel Problem Solving from Nature - PPSN VIII* (X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, eds.), (Berlin, Heidelberg), pp. 253–262, Springer Berlin Heidelberg, 2004.
[31] A.-C. Zăvoianu, E. Lughofer, W. Koppelstätter, G. Weidenholzer, W. Amrhein, and E. P. Klement, "Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems," *Know.-Based Syst.*, vol. 87, pp. 47–60, Oct. 2015.