Review of Literature on Growing Neural Networks

Max Ploner
HU Berlin
Science of Intelligence

1 Introduction

This article aims to summarize the existing literature concerned with growing artificial neural networks. For each paper it will list the most significant contribution. The following four questions will guide the summary of each paper:

1. **Why** are models grown? What is the goal or metric the approach is evaluated on?
2. **When** are the models grown?
3. **Where** are the models grown?
4. **How** are the the new parts initialized?

Each paper tries to make progress in answering at least one of the questions. Hence, they can be used to categorize these papers.

1.1 Topics of Interest

This articles reviews publications on Artificial Neural Networks (ANN or simply NN) which are grown in some way or another. It is focuses on networks trained using backpropagation with gradient descent and excludes research areas such as self-organizing maps (Boinee, De Angelis, and Milotti 2003), growing neural gas (GNG, Fritzke 1994), self-organizing networks (Piastra 2009), and growing neural forests (Palomo and López-Rubio 2016).

Additionally, we do not consider an ANN to be grown if simply another classification head is introduced when a new task is encountered in an continuous learning (CL) setting. Instead, in continuous learning settings, we require the shared parts of the network to be grown.

2 Categorization

Table 1: Papers according to the three questions (see Section 1).

Short Title	Year	Why?	When?	Where?	How?	Paper
Net2Net	2016	Knowledge transfer for NAS(?), already mentions lifelong learning (no experiments)	Single growth event	Not dynamic: Width is uniformly grown, new layers are added towards the end	Function-preserving transforms (identity matrix)	Tianqi Chen, Goodfellow, and Shlens (2016)
Network Morphism	2016	Knowledge transfer	Single growth event	Not dynamic: Width is uniformly grown, new layers are added towards the end	Function-preserving transform, less sparse init.	Wei, Wang, Rui, et al. (2016)
Progressive Nets	2016	Continual Learning	On new tasks	New columns	Random init	Rusu et al. (2016)
NAS using Net Transforms	2017	NAS	Fixed schedule	Decided by RL agent	Function-preserving transforms	Cai, Tianyao Chen, et al. (2017)
NASH	2017	NAS	Iterativly grow and train a set of networks, then pick the best	Randomly selected (multiple alternatives)	Function-preserving transforms	Elsken, Metzen, and Hutter (2017)
NeST	2018	NAS		Neurons which will exhibit large gradients		Dai, Yin, and Jha (2018)
Path-Level Transformations	2018	NAS	Fixed schedule	Decided by RL agent	Function-preserving transforms	Cai, Yang, et al. (2018)
Compacting & Picking	2019					Hung et al. (2019)
Progressive Stacking	2019	Accelerate pre-training	Fixed schedule	Duplicated layers added on top	Duplication of existing layers	Gong et al. (2019)
Firefly	2020	NAS and CL	Fixed Schedule	Decided based on gradient information	Function-preserving transforms	Wu et al. (2020)
GradMax	2022	NAS	Fixed Schedule	Fixed (GradMax could be adapted for this decision)	By maximizing the gradient of new parts using SVD	Evci et al. (2022)

3 Summaries of the Reviewed Publications

The following sections give short summaries of each of the publications which we deemed relevant.

3.1 Net2Net: Accelerating Learning via Knowledge Transfer (Tianqi Chen, Goodfellow, and Shlens 2016)

Tianqi Chen, Goodfellow, and Shlens (2016) introduce the idea of training a larger student network from an existing smaller teacher network by using *function-preserving transformations*. These transformations (*Net2Net* operations) allow the rapid transfer of learned knowledge and omits the need to retrain the larger network from scratch.

The authors propose two operations two increase the student network's size:

1. Growing in width: adding more units in each hidden layer and
2. growing in depth: adding more hidden layers.

Growth along the **width** dimension is achieved by randomly splitting the original neurons (*Net2WiderNet* operation, see Figure 1). Input weights of new neurons are copied from existing and the output weight of existing neurons is equally distributed among all copies (the old neuron and all new copies).

If no dropout is used, Tianqi Chen, Goodfellow, and Shlens (2016) propose to add a small noise to the input weights to break the symmetry.

Growth along the depth dimension is achieved by adding new layers which are initialized with the identity function. This requires idempotent activation functions: the activation function ϕ needs to chosen such that $\phi(\mathbf{I}\phi(\mathbf{v}))$ for any vector $\mathbf{v}$. For rectified linear units (ReLU) this is the case, for some the identity matrix may be replace with a different matrix, in some cases it may not be as easy to construct an identity layer.

The experiments are conducted on an Inception network architecture (Szegedy et al. 2014), a convolutional neural network (CNN). They show that rapid transfer of knowledge through the two types of network transformations is possible, allowing the faster exploration of model families contained in this architecture space.

3.2 Network Morphism (Wei, Wang, Rui, et al. 2016)

Wei, Wang, Rui, et al. (2016) follow a very similar path to Tianqi Chen, Goodfellow, and Shlens (2016): *function-preserving transformations* are used to grow a parent (or "teacher") network to a child (or "student") network while maintaining the same function.

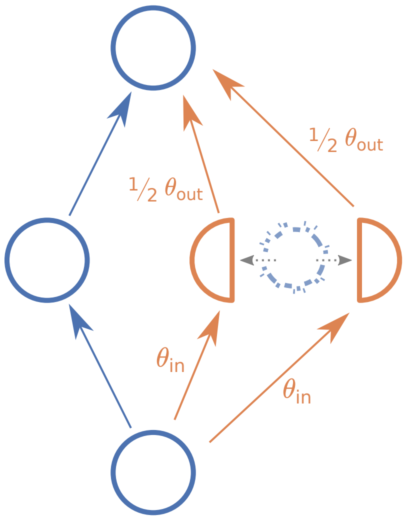

Figure 1: Illustration of *Net2WiderNet*. Here, a single neuron is split in two parts. However, multiple neurons can be split in one operation and each neuron may be split in multiple parts

Wei, Wang, Rui, et al. (2016) point out that using an identity layer for growing in depth (which they refer to as "IdMorp") may be sub-optimal as it is extremely sparse. Additionally, they reiterate the requirement of idempotent activation functions, which they deem insufficient.

Through an iterative procedure, a convolutional layer is decomposed into two layers, retaining a large number of non-zero entries.

Wei, Wang, and C. W. Chen (2019) furhter improve the decomposition method in order to minimize the performance drop after transforming (growing) the network.

Instead of relying on idempotent activation functions, Wei, Wang, Rui, et al. (2016) introduce parametric activation functions for new layers: A parameter a interpolates between the identity function and the non-linear activation function. a is initialized with one such that there is essentially no activation function. Over the course of future training, the parameter can be learned to make the activation function non-linear [for an example see Figure 2 or the parametric rectified activation units (PReLU), He et al. (2015)].

3.3 Progressive Neural Networks (Rusu et al. 2016)

Rusu et al. (2016) develop *Progressive Networks* for tackling catastrophic forgetting. The idea is to grow networks when learning new tasks. The older

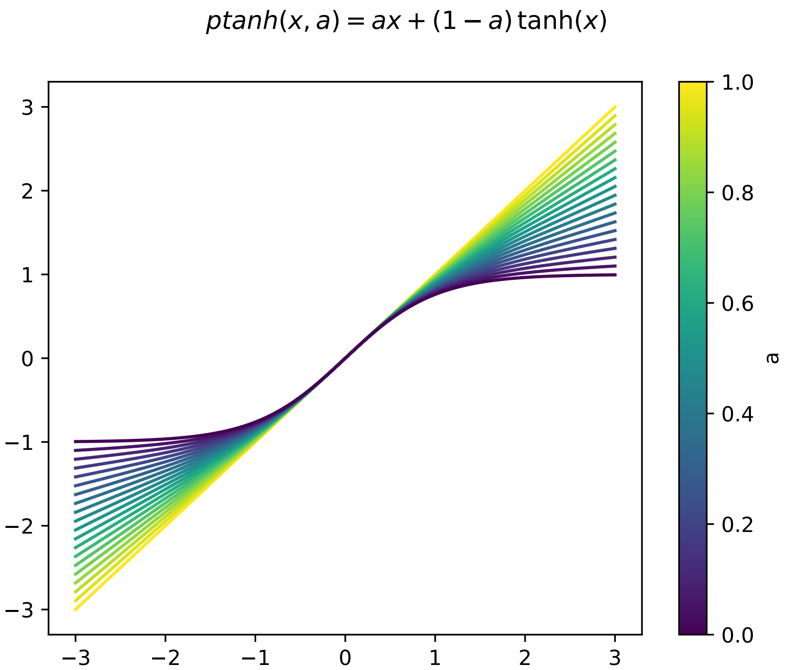

Figure 2: Illustration of an parametric tanh function: with $a = 1$ the function is equal to the identity function, with $a = 0$ it is equal to tanh.

parts of the networks are frozen and their function incoporated using adapters to allow for knowledge transfer from earlier tasks. Each time a new tasks is learned, the network is further extended (a new column is added).

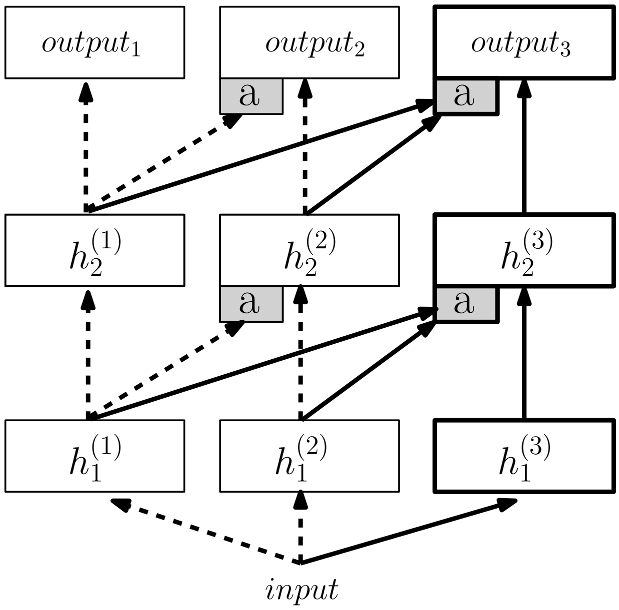

Figure 3: Figure from Rusu et al. (2016) illustrating the use of columns and adapters.

During inference (as well as during training), a task identifier is needed to select the column which matches the current task. By freezing the older parts of the networks during training, the performance on tasks learned in early training is guaranteed to remain stable, as the respective weights (and therefore the models function) cannot change.

3.4 Efficient Architecture Search by Network Transformation (Cai, Tianyao Chen, et al. 2017)

Cai, Tianyao Chen, et al. (2017) propose using a reinforcement learning (RL) agent as a meta-controller in order to decide when and where the network is grown (using function-preserving transformations).

By using variable-length strings (see Zoph and Le 2017) to represent the network architecture, an RL agent can be used to generate a function-preserving transformation (Tianqi Chen, Goodfellow, and Shlens 2016).

The network architecture is encoded using an bidirectional LSTM and the encoding is then fed to a number of actor networks which decides whether and where transformations should be applied. For each possible network transformation there is one actor network. For an illustration, see Figure 4.

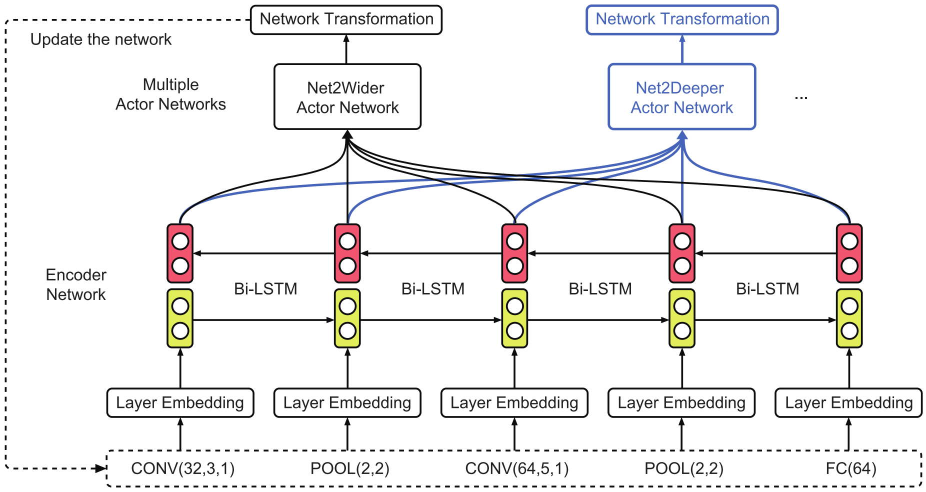

Figure 4: Illustration of the architecture embedding. Figure from Cai, Tianyao Chen, et al. (2017)

In each growth phase, 10 networks are sampled from the meta-controller and trained for 20 epochs (on image datasets CIFAR-10 and SVHN). Based on the accuracy on held out validation data (acc_v), a reward for the meta-controller is calculated. Instead of directly using the accuracy as reward signal, Cai, Tianyao Chen, et al. (2017) propose using a non-linear transformation in order to increase the reward if the accuracy is already high (an increase of 1% starting at 90% is more difficult than starting at 60%):

$$\tan(acc_v \times \frac{\pi}{2})$$

3.5 Simple And Efficient Architecture Search for Convolutional Neural Networks (Elsken, Metzen, and Hutter 2017)

Elsken, Metzen, and Hutter (2017) propose an iterative NAS algorithm (Neural Architecture Search by Hillclimbing; short: NASH) which – in each growth step – produces a set of grown child networks (using function-preserving transformations). Each child is trained for a small number of epochs before the most promising candidate is chosen. This best performing child is then used for repeating the procedure (see fig. 5).

Additionally, they use a different set of network morphisms (or *function-preserving transformations*) such as an interpolating layer (similar to the parametric activation functions in Wei, Wang, Rui, et al. 2016): Here an

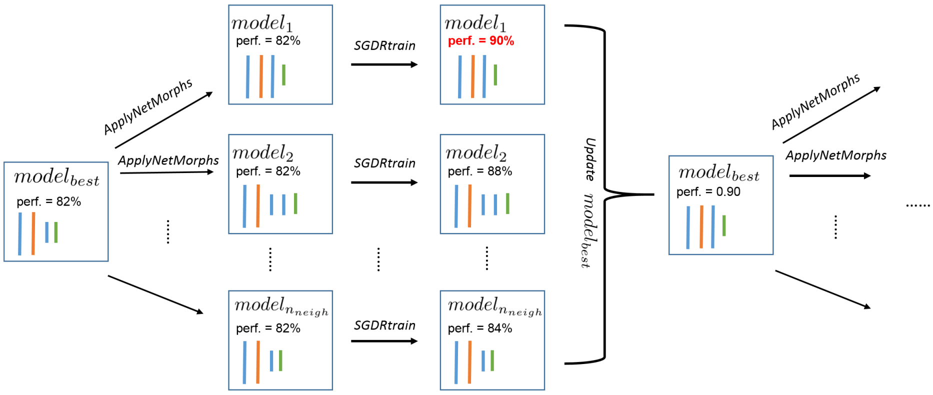

Figure 5: Illustration of the NASH algorithm. A set of children is grown and trained. Then the best candidate is chosen. Figure from Elsken, Metzen, and Hutter (2017).

existing layer is replaced by an affine combination of the existing layer and some new ones (starting with all weights of the new layers being 0, and the weight of the existing layer to be 1).

3.6 NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm (Dai, Yin, and Jha 2018)

Dai, Yin, and Jha (2018) utilize growth with network architecture search (NAS) in mind. They note that trial-and-error approaches are inefficient as a process and can (as a product) lead to inefficient architectures which might far more parameters than required. To combat these issues, they propose NeST, which trains weights as well as the architecture.

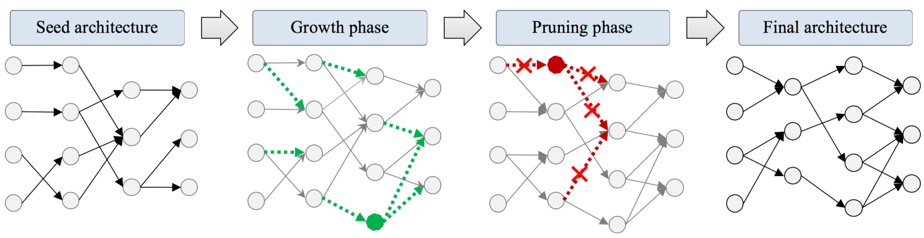

Figure 6: Illustration of the steps for synthesizing an architecture using NeST (figure from Dai, Yin, and Jha 2018).

NeST starts with an initial small network (a *seed architecture*). In a first phase, the network is grown by adding new connections based on their gradient (assuming they already existed with an weight of 0), and growing new neurons in a layer l in order to connect existing neurons n and m in layers $l-1$ and $l+1$ which if they were connected directly, exhibited a large gradient magnitude:

$$G_{m,n} = \frac{\partial L}{\partial u_m^{l+1}} x_n^{l-1} \geq threshold$$

Here, u_m^{l+1} is the sum of incoming activiations of neuron m in layer $l+1$ and x_n^{l-1} is the activation of neuron n in layer $l+1$. The threshold is calculated using a growth proportion.

In a second phase, weights are iteratively prruned. Between each pruning step, the network is retrained to recover its performance.

3.7 Path-Level Network Transformation for Efficient Architecture Search (Cai, Yang, et al. 2018)

This publication offers an incremental extension to enable branched architectures using function-preserving transformations (Tianqi Chen, Goodfellow, and Shlens 2016) and growing the model using a RL agent based meta-controller as in Cai, Tianyao Chen, et al. (2017).

Cai, Yang, et al. (2018) propose *path-level* transformations which allows the branching of neural networks (whereas Tianqi Chen, Goodfellow, and Shlens (2016) initially proposed just growing deeper and wider). Instead of restricting the architecture space to sequences of layers, Cai, Yang, et al. (2018) represent their network architecture as trees.

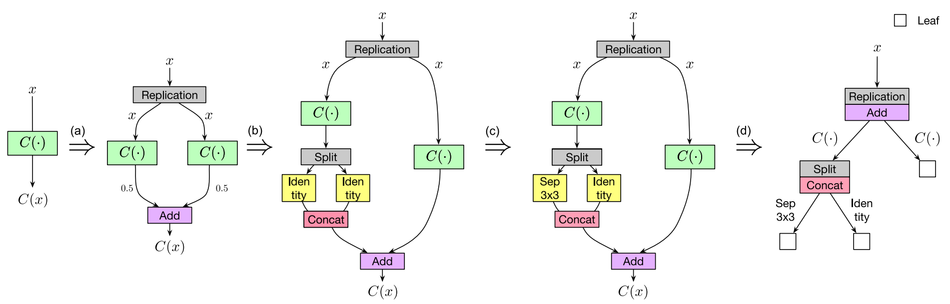

Figure 7: Illustration of a series of network transformations. The last part shows the tree-structure of the transformation. Figure from Cai, Yang, et al. (2018).

Each *path-level* transformation follows either an *add* or a *concatenation* merge scheme. In the *add* scheme, a layer is replaced by two copies and each of their outputs is multiplied by 0.5. This is similar to splitting a neuron, except on a layer level. Transformation (a) in Figure 7 shows such a transformation.

In the *concatenation* scheme (step (b) in Figure 7), the outputs dimensions (in a fully connected layer: neuron outputs, in a convolutional layer: output channels, etc.) are split among the different branches and the output of each branch is later concatenated. This introduces branches while preserving the function and each branch is unique.

These two schemes do not introduce a significant change to the network. However, in combination with the existing operations (in Tianqi Chen, Goodfellow, and Shlens 2016), this can lead to a variety of branched architectures.

3.8 Compacting, Picking and Growing for Unforgetting Continual Learning (Hung et al. 2019)

comming soon

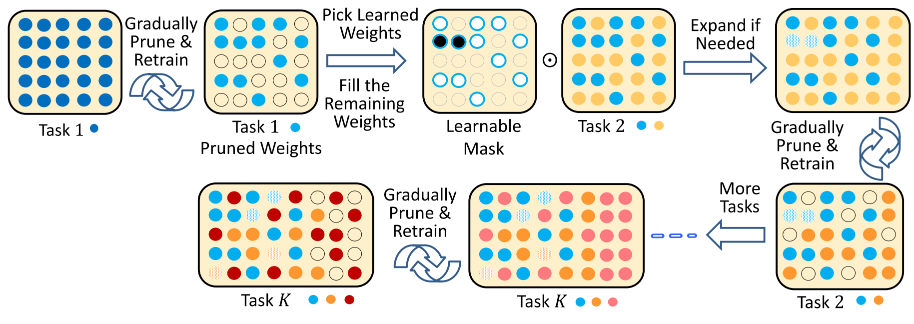

Figure 8

3.9 Efficient Training of BERT by Progressively Stacking (Gong et al. 2019)

Gong et al. (2019) observe, that self-attention distributions across different layers of well-trained BERT model typically exhibit a large degree of similarity. Hence, they propose starting the pre-training of BERT models with a smaller number (3) of hidden layers. Over the course of the training, these pre-trained layers are duplicated twice (and added on top, see Figure 9) and trained between each stacking operations in order to differentiate the layers.

By training with few layers for a large portion of the pre-training, Gong et al. (2019) can reduce the pre-training time by $\sim 35\%$ with only a small loss of performance.

3.10 Firefly Neural Architecture Descent: A General Approach for Growing Neural Networks (Wu et al. 2020)

Wu et al. (2020) propose alternating between training and growth steps. In each growth step, the network is grown to be wider and deeper. During each growth step, multiple candidate elements (neurons or layers) are temporarily added to the network. The contribution of each candidate part is multiplied

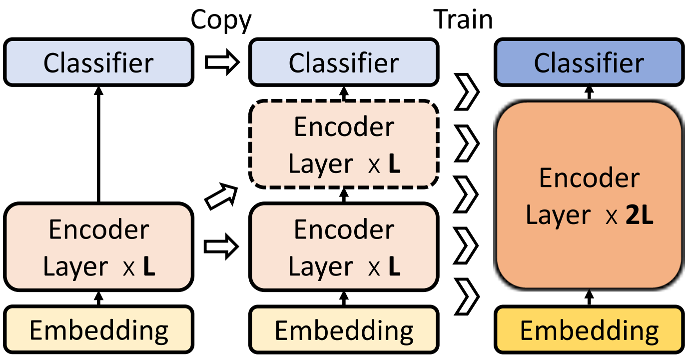

Figure 9: In each stacking step, the number of layers is doubled (Gong et al. 2019).

with some step size ϵ to maintain the original function. By using the training data (or some portion of it), one can then calculate how beneficial these new parts might be during future training. Wu et al. (2020) show that by using Taylor approximation, this reduces to looking a the gradients of these new parts.

Additionally, Wu et al. (2020) test their approach in a CL task-incremental setting. For each task, a neuron mask is created (which can be retrieved using the available task identifier). This allows the model to share structure while maintaining its function on old tasks and hence to maintain a good average accuracy even after multiple tasks have been learned.

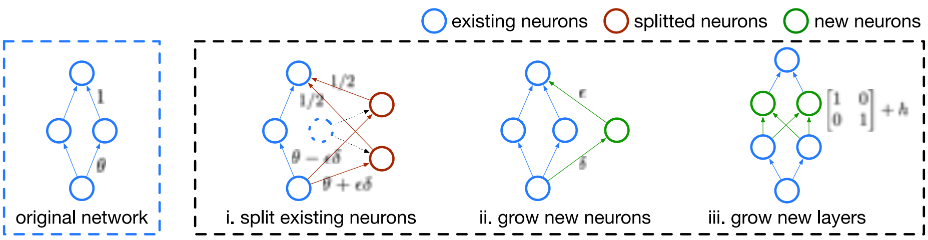

Figure 10: Figure from Wu et al. (2020) illustrating how new neurons can be added.

3.11 GradMax: Growing Neural Networks Using Gradient Information (Evci et al. 2022)

Evci et al. (2022) focus on the question **how** new neurons are initialized. They propose initializing new neurons such that the gradient norm of new weights are maximized while maintaining the models function. By enforcing large gradient norms of the new weights, the objective function is guaranteed to decrease in the next step of gradient descent.

When using a step size of $\frac{1}{\beta}$ on a function with a β-Lipschitz gradient, the loss is upperbounded by:

$$L(W_{new}) \leq L(W) - \frac{\beta}{2}\|\nabla L(W)\|^2$$

While a constant Lipschitz constant generally does not necessarily exist in neural networks the authors use this as a motivation to assume that large gradient norms will lead to large decreases in the loss function after the next

In GradMax, the maximum gradient norms (with some constraint) are approximated using singular value decomposition (SVD). The authors additionally provide experiments using optimization to produce large gradient norms instead of using the closed-form solution of SVD. While they find that SVD usually produces better results, it can only be used, if the activation function returns 0 given an input of 0.

The authors note that this idea could also be utilized to select **where** new neurons should be grown. The decision where to add new neurons could be made by looking at the singular values (e,g, selecting the largest or adding a neuron, once the singular value reaches a threshold). This idea is very similar to the strategy of Wu et al. (2020) which use a very similar technique to choose **where** to grow neurons (but use a different initialization strategy).

References

Boinee, P., A. De Angelis, and E. Milotti (July 2003). "Automatic Classification Using Self-Organising Neural Networks in Astrophysical Experiments". In: *arXiv:cs/0307031*. arXiv: cs/0307031. URL: http://arxiv.org/abs/cs/0307031 (visited on 05/06/2021).

Cai, Han, Tianyao Chen, et al. (Nov. 2017). "Efficient Architecture Search by Network Transformation". In: *arXiv:1707.04873 [cs]*. arXiv: 1707.04873 [cs]. URL: http://arxiv.org/abs/1707.04873 (visited on 05/06/2021).

Cai, Han, Jiacheng Yang, et al. (July 2018). "Path-Level Network Transformation for Efficient Architecture Search". In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, pp. 678–687. URL: https://proceedings.mlr.press/v80/cai18a.html (visited on 10/18/2022).

Chen, Tianqi, Ian Goodfellow, and Jonathon Shlens (Apr. 2016). "Net2Net: Accelerating Learning via Knowledge Transfer". In: *arXiv:1511.05641 [cs]*. arXiv: 1511.05641 [cs]. URL: http://arxiv.org/abs/1511.05641 (visited on 11/15/2021).

Dai, Xiaoliang, Hongxu Yin, and Niraj K. Jha (June 2018). "NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm". In: arXiv:1711.02017. DOI: 10.48550/arXiv.1711.02017. arXiv: 1711.02017 [cs]. URL: http://arxiv.org/abs/1711.02017 (visited on 10/17/2022).

Elsken, Thomas, Jan-Hendrik Metzen, and Frank Hutter (Nov. 2017). *Simple And Efficient Architecture Search for Convolutional Neural Networks*. arXiv: 1711.04528 [cs, stat]. URL: http://arxiv.org/abs/1711.04528 (visited on 10/25/2022).

Evci, Utku et al. (Feb. 2022). *GradMax: Growing Neural Networks Using Gradient Information*. DOI: 10.48550/arXiv.2201.05125. arXiv: 2201.05125 [cs]. URL: http://arxiv.org/abs/2201.05125 (visited on 05/28/2022).

Fritzke, Bernd (1994). "A Growing Neural Gas Network Learns Topologies". In: *NIPS*.

Gong, Linyuan et al. (2019). "Efficient Training of BERT by Progressively Stacking". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 2337–2346. URL: https://proceedings.mlr.press/v97/gong19a.html.

He, Kaiming et al. (Dec. 2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, pp. 1026–1034. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.123. URL: http://ieeexplore.ieee.org/document/7410480/ (visited on 10/19/2022).

Hung, Steven C. Y. et al. (Oct. 2019). "Compacting, Picking and Growing for Unforgetting Continual Learning". In: *arXiv:1910.06562 [cs, stat]*. arXiv: 1910.06562 [cs, stat]. URL: http://arxiv.org/abs/1910.06562 (visited on 11/15/2021).

Palomo, E. J. and Ezequiel López-Rubio (2016). "Learning Topologies with the Growing Neural Forest". In: *Int. J. Neural Syst.* DOI: 10.1142/S0129065716500192.

Piastra, Marco (June 2009). "A Growing Self-Organizing Network for Reconstructing Curves and Surfaces". In: *2009 International Joint Conference on Neural Networks*, pp. 2533–2540. DOI: 10.1109/IJCNN.2009.5178709. arXiv: 0812.2969. URL: http://arxiv.org/abs/0812.2969 (visited on 05/06/2021).

Rusu, Andrei A. et al. (Sept. 2016). "Progressive Neural Networks". In: *arXiv:1606.04671 [cs]*. arXiv: 1606.04671 [cs]. URL: http://arxiv.org/abs/1606.04671 (visited on 10/14/2021).

Szegedy, Christian et al. (Sept. 2014). *Going Deeper with Convolutions*. DOI: 10.48550/arXiv.1409.4842. arXiv: 1409.4842 [cs]. URL: http://arxiv.org/abs/1409.4842 (visited on 10/19/2022).

Wei, Tao, Changhu Wang, and Chang Wen Chen (July 2019). "Stable Network Morphism". In: *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851955.

Wei, Tao, Changhu Wang, Yong Rui, et al. (Mar. 2016). "Network Morphism". In: *arXiv:1603.01670 [cs]*. arXiv: 1603.01670 [cs]. URL: http://arxiv.org/abs/1603.01670 (visited on 05/06/2021).

Wu, Lemeng et al. (2020). "Firefly Neural Architecture Descent: A General Approach for Growing Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 22373–22383. URL: https://proceedings.neurips.cc/paper/2020/file/fdbe012e2e11314b96402b32c0df26b7-Paper.pdf.

Zoph, Barret and Quoc V. Le (Feb. 2017). *Neural Architecture Search with Reinforcement Learning*. DOI: 10.48550/arXiv.1611.01578. arXiv: 1611.01578 [cs]. URL: http://arxiv.org/abs/1611.01578 (visited on 10/25/2022).