# Evospnet: Evolutionary Split Networks

Abhimanyu Bellam
North Carolina State University
Raleigh, USA
Email: abellam2@ncsu.edu

***Abstract*—Neuroevolution suffers from extremely high training times and reduced performance with the increase in the dimensionality of a Neural Network. The algorithms used to evolve the weights requires the computing of the loss function very often, which compels one to use more resources to reduce time. To tackle the dimensionality and resource issues, I propose to split a neural network into parts which are evolved and re-combined itertively via a step-wise evolutionary strategy, aiming to achieve a definitely better solution as compared to it's population generated by different initialization schemes. Empirically, it is shown that the Evopsnet performs better than regular Neuroevolution via Differential Evolution. The code is available at: https://github.com/AbhimanyuBellam/evospnet**

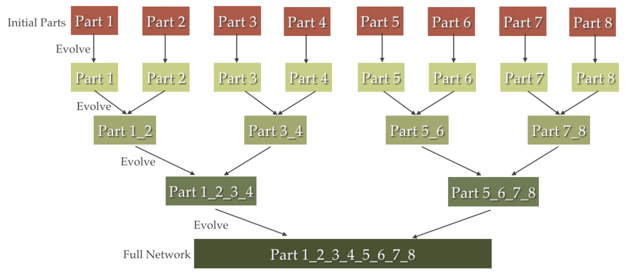


Fig. 1: Recombination sequence

## I. Introduction

Neuroevolution (NE) is the use of Evolutionary Algorithms (EA) to optimize the weights/parameters of a Neural Network (NN). It has been gaining ground due to the ability of EA to find the global optima, rather than getting stuck at a local optima since most functions in the context of neural networks are non-convex. NE was first introduced in 1989 in [1] where genetic algorithms were used to train a NN. In 2002, its popularity further increased when NeuroEvolution of Augmenting Topologies (NEAT) was introduced to improve not just the parameters but also the topology of the network [2]. Following this, there have been many works to enhance NEAT like CoevolutionaryNEAT [3], HyperNEAT [4], NEATfields [5] and odNEAT [6]. Inspite of its scope, the fact remains that it has not been able to replace gradient descent as the most popular and effective way to train a Neural Network.

The performance of optimization algorithms degrade with the increase in dimensionality of the solution space, making it very difficult to tune a NN with large number of parameters. This work contributes to NE based approaches by exploring a method to reduce the dimensionality of a network and training in parts for better search space exploration. Also, time and resources are taken into consideration, given their importance in NN training and inference. This method primarily focuses on a training methodology, however, it can lead to choosing a smaller network and therefore reducing inference time.

From an algorithmic view, Divide and Conquer approaches are used in many areas of computer science and in real life too. They are effective in solving problems because they try to solve sub-problems. This can be applied to NNs too. Instead of optimizing all the parameters at once, we can optimize parts of the network.

This work brings forward — using NN splitting to evolve in low dimensional spaces for solving sub-problems in the context of NE, along with a combination scheme 1 that is applicable to both NE and SGD based training too.

## II. Related Work

In 2018, a divide and conquer strategy was applied with NE by McDonnell et al. where a multi-class classification problem is divided into multiple one-vs-one and one-vs-all ensemble models and trained [7]. Kai et al. proposed structural objectives – guiding evolution to align neural networks with a user-recommended decomposition pattern and a high diversity in decomposition pattern [8]. In 2019, Gregory examined the usage of evolution in conjunction with gradient descent [9]. In 2021, Kamil et al, proposed an Ensemble Neuroevolution-based approach to optimize Neural architecture and hyperparameters [10]. In 2022, Google Cloud developed EvoJAX, a complex-problem solver toolkit that uses Neuroevolution with the belief that it can solve many “non well-behaved” problems showcasing it's ability and future prospect [11].

In the context of splitting neural networks, Kim et al. proposed a lightweight NN that learns to split the NN parameters into a hierarchy of many parts which use features that are disjoint [12]. Xin et al. designed deep neural networks (DNNs) and distributed its workload to camera sensors and aggregator under a resource constrained environment [13]. In [14] MalekHosseini et al. split the filters and feature maps from a larger Convolutional NN (CNNs) to smaller networks.

However, none of these methods use network splitting for NE using Differential Evolution (DE) and perform iterative combining of parts. The motivation for this work roots from [15] where a divide and conquer method was used to reduce the search space complexity to solve a Warehouse resource allocation problem using DE.

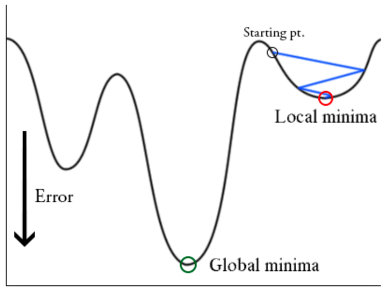


Fig. 2: Gradient Descent can get stuck at local minima. Source: [20]

## III. Problem Definition

### A. Background

The Universal Approximation Theorm (UAT) [16] says that a single layer NN having a non linear activation function can approximate any function to some degree. This shows the applicability of NN to solve a broad range of problems. The weights and architecture of a NN is its identity. The most common way to update the weights of a NN is via backpropagation of error. The end result is a lot of numbers that may not have any intrinsic meaning, but get the job done. Therefore, any set of numbers that has the best loss and accuracy values (in general) can be used. The function formed by the network parameters and data can be approximated using several methods. Evolutionary algorithms (EA) such as Genetic algorithm (GA) [18] and DE [17] are a class of algorithms that search the solution space using certain operations (nature-inspired) on its population. The population is a group of one-dimensional sequences, that interact with eachother and also with randomness to explore a space and converge to a solution. In general, these algorithms produce excellent results for low dimensional problems. These methods can be used to update the weights of a NN in place of backpropagation via SGD.

### B. What is the problem?

A NN is heavily impacted by its initalization [19]. Infact, an ensemble of NNs are sometimes used because of the lack of confidence in a NN's initialization. There is no guarantee that stochastic gradient descent (SGD) will take the solution to the global minima 2. Upon convergence, a NN reaches a local minima when trained via SGD. The fact that in an ensemble of trained NNs, the weights are not numerically equal and yet they show close but unequal loss values, indicate that they represent multiple local minimas in the solution space. Is it possible to improve the loss/accuracy of one model to something higher than an ensemble of similar models?

At the same time, what can be done to handle the high dimensionality of a NN that reduces the space of exploration for NE and also reduces resource requirements if a sufficient model is produced?

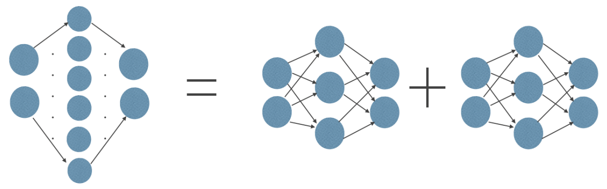
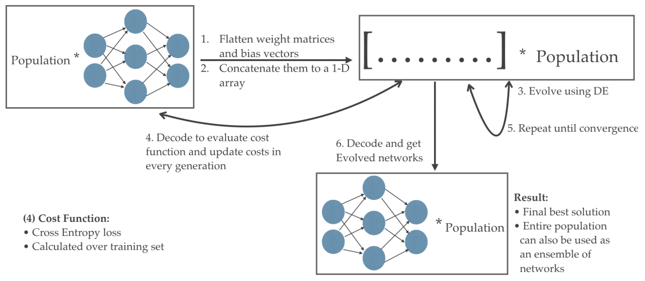


Fig. 3: Splitting method. Hidden nodes are equally distributed.



Population *
1. Flatten weight matrices and bias vectors
2. Concatenate them to a 1-D array
[.........] * Population
3. Evolve using DE
5. Repeat until convergence
4. Decode to evaluate cost function and update costs in every generation
6. Decode and get Evolved networks
(4) Cost Function:
• Cross Entropy loss
• Calculated over training set
* Population
Result:
• Final best solution
• Entire population can also be used as an ensemble of networks

Fig. 4: Training method

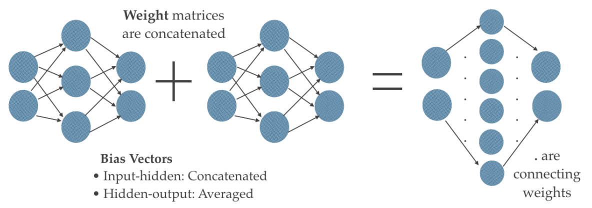
## IV. Evospnet

An NN with one hidden layer (ONN) is split into n parts. 3 shows a split into two parts. The sum of the number of hidden nodes in the parts would be equal to that in the large ONN. This reduces the dimensionality of the NN. Now, each part is evolved according to the method shown in 4, in which the search space is smaller. When the objective function is the same across all the parts, this formulation can be seen as a Divide and Conquer method to break the problem into sub-parts and solve the sub-parts first. Now, each part is combined with another part via algorithm 1, represented by 5 and evolved again through the same training method. This is done iteratively till the final network is reached in the sequence 1 by the algorithm 2. It is important to note that, if a part has achieved sufficient accuracy for a use case, the Multi-Level-Split-Evolution is stopped.

### A. Properties of ONN-Evospnet

*1) Evolution is not always needed- SGDspnet:* The process of iteratively combining parts need not involve NE, the weights can obviously be optimized by SGD.

*2) Split Invariant:* Suppose there are 40 hidden nodes in the parent NN. When they are split sequentially among 8 parts i.e., 40 to 5, 5, ..., 5, it is the same as splitting 40 into 20, 20;



Weight matrices are concatenated
Bias Vectors
• Input-hidden: Concatenated
• Hidden-output: Averaged
. are connecting weights

Fig. 5: Combining two networks into one.

**Algorithm 1** CombBin: Algorithm to combine two ONNs

```
Require: net1, net2, numLayers, NNDefinition
  $combinedNN \leftarrow NNDefinition()$
  for i in numLayers do
      $axis \leftarrow 1$
      $net1W \leftarrow net1[i].weights$
      $net1B \leftarrow net1[i].bias$
      $net2W \leftarrow net2[i].weights$
      $net2B \leftarrow net2[i].bias$
      if i is 0 then
          $axis \leftarrow 0$
      end if
      $combWmat = concatenate((net1W, net2W), axis)$
      if i is-last-layer then
          $combB \leftarrow \frac{net1B+net2B}{2}$
      else
          $concatenate((net1B, net2B), axis = 0)$
      end if
      $combinedNN[i].weights = combWmat$
      $combinedNN[i].bias = combB$
  end for
```

splitting 20 again to 10, 10; 10 again to 5, 5; because the bias is kept constant while splitting.

*3) Combination Variant:* Whenever two networks are combined, there is iterative averaging of bias which causes variation in combination.

## *B. Deep-Evospnet*

When there is more than 1 hidden layer, the weight matrices can be concatenated if the missing weights are set to random numbers or zeroes. Whereas the bias vectors between layers can be concatenated for all layers except for the last hidden-output connection which can be averaged. A NN formed by this combination would represent in a certain sense, an unstructured pruned network when all the missing weights are set to 0. However, there is still a chance for improvement because the accuracy of the combination could be less than any of it sub-parts.

## V. Experiments

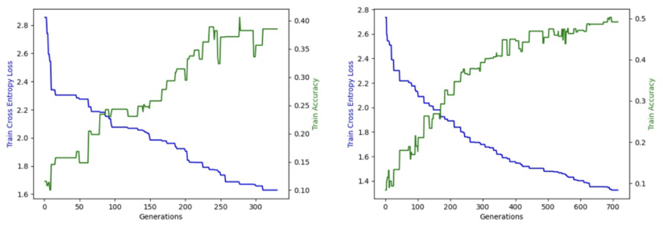
An ONN is used for all experiments in this work considering the constraint of resources for NE while also considering UAT. Note that only **train plots** are shown due to their similarity with test plots. The dataset for all experiments is the MNIST dataset [21]. A BasicNet is defined here as, a ONN having 784 input nodes (the flattend size of an image in MNIST), 5 hidden nodes and 10 output nodes (10 classes in MNIST).

## *A. SGD vs NE for BasicNet*

Figure 6 shows an increasing accuracy with increase in population and generations. Due to resource contraints, larger population and generations could not be run. Table I elucidates that SGD performs better. However, the pattern shown in the NE graphs imply that there is definitely a chance for further optimization.

**Algorithm 2** Multi-Level-Split-Neuroevolution

```
Require: NNLevels, NNDefinitions, RequiredAccuracy
Require: maxParts Initialize: NNLevelPopulations,
  for i in NNLevels do
      $L \leftarrow NNDefinitions[i]$
      $P \leftarrow NNLevelPopilations[i]$
      $BestSols \leftarrow ()$                    ▷ Store level's best sols
      if i!=0 then
          Send parts to CombBin and update CombParts
      end if
      for j in maxParts do
          $Population \leftarrow NNLevelPopulation[i]$
          $currBestSol \leftarrow Nil$
          if i!=0 then
              $currBestSol \leftarrow CombdParts[j]$
          end if
          $BestSol \leftarrow EvolvePop(CurrBestSol, P)$
          if $BestSol.accuracy \geq RequiredAccuracy$ then
              $Exit$
          end if
          $BestSols+ \leftarrow BestSol$
      end for
      $maxParts \leftarrow \frac{maxParts}{2}$
  end for
```



(a) Population=56 (b) Population=140

Fig. 6: NE with different population sizes

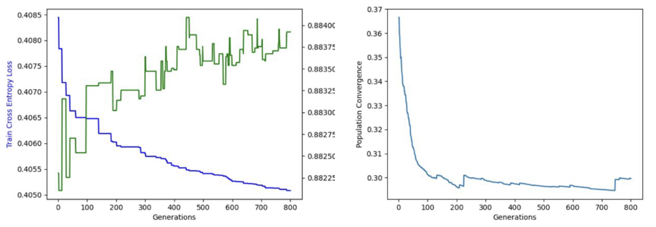
## *B. Ensemble of NNs trained via SGD as initial population for NE*

*1) As full population (SNEF):* The plots 7 shows the train loss reducing beyond the value attained via SGD, indicating that NE is able to push a NN further by a very small amount. The convergence graph 7b shows a rather unexpected visual where the convergence reduced for many generations indicating that the population is still quite varied. It therefore suggests that further optimization can be done.

*2) As half of population (SNEH):* In this case too NE is able to optimize further from SGD 8. Although, it does not do better than SNEF. Unlike SNEF, the convergence mostly has an increase, but after a certain point, there is almost a constancy. A speculative reason for the sudden jump in the convergence behaviour of SNEH could be that a random exploration moved the population in the same direction. However, further experiments would need to be conducted to understand it better.

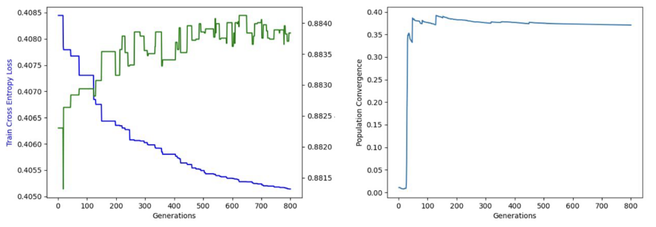
| | SGD | NE | SNEF | SNEH |
|---|---|---|---|---|
| Population size (NE) | Momentum:0.9 | 140 | 56 | 112 |
| Bounds (NE) | Batch size: 100 | [-1,1] | Weights[-1,1], Bias[-15,15] | Weights[-1,1], Bias[-15,15] |
| Generations (NE) | Epochs: 50 | 730 | 800 | 800 |
| Initialization | Kaiming Uniform | Uniform | SGD | Uniform, SGD |
| Train Accuracy | 0.8808 | 0.4907 | **0.8824** | 0.8823 |
| Train Loss | 0.4084 | 1.3255 | **0.4050** | 0.4051 |
| Test Accuracy | **0.8686** | 0.4877 | **0.8686** | 0.8680 |
| Test Loss | 0.4533 | 1.3158 | **0.4505** | 0.4510 |
| Training time | **9150 (Serial Ensemble 2.54 hr)** | 50127 (13.9 hrs) | 24091 (6.69 hr) | 45629 (12.67 hr) |
| CPU cores used | **1** | 7 | 7 | 7 |
| GPU Memory | **1.2** | 7.91 | 7.91 | 7.91 |

TABLE I: *Table comparing aspects of NN training via SGD and NE*



(a) Loss Graph (b) Convergence Graph

Fig. 7: NE's population fully initialized with ensemble of SGD trained NNs

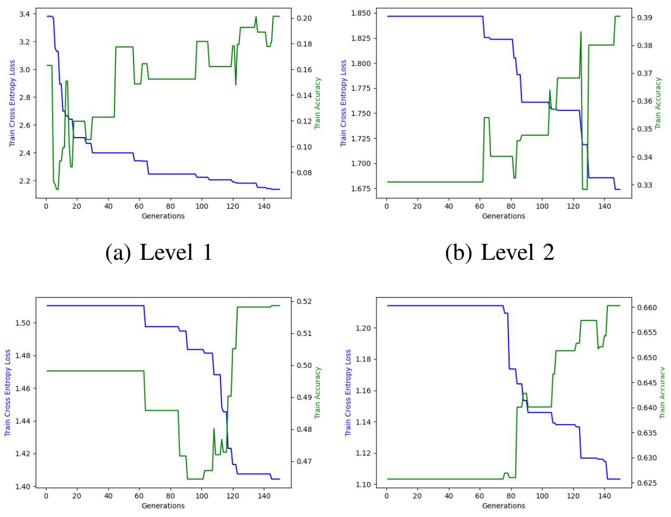


(a) Loss Graph (b) Convergence Graph

Fig. 8: Half of NE's population initialized with ensemble of SGD trained NNs

| | Basic Net 1 | Basic Net 2 | Combined Model |
|---|---|---|---|
| Train Accuracy | **0.8693** | 0.8677 | 0.7205 |
| Train Loss | 0.4568 | **0.4505** | 1.2276 |
| Test Accuracy | **0.8677** | 0.8649 | 0.7331 |
| Test Loss | 0.4669 | **0.4637** | 1.2405 |

TABLE II: *Comparison of metrics for NN before and after combining without training the combined model.*



(a) Level 1 (b) Level 2 (c) Level 3 (d) Level 4

Fig. 9: Multip step split Neuroevolution. The start loss of each part has improved from the previous part.

## C. *What happens when two NNs are combined?*

As expected, there is a decrease in all metrics II, but, it is only around 14 percent. This indicates that combining NN may not really be very bad, and can possibly serve as an excellent initialization for its training.

## D. *Multi-Step-Split-Neuroevolution (Evopsnet)*

In this experiment, 8 level 1 nets were combined to form 4 level 2 nets, which combined to give 2 level 3 nets, combining to 1 level 4 net. The graphs 9 show a continuous decrease in training loss for each level. III shows a clear increase in all metrics with increase in level.

## VI. Possible Future Work

1. Exploring how Evospnet with its initial population as an ensemble of SGD trained NNs would perform.

2. It can be extended to CNNs which are used frequently for image classification.

3. Evopsnet can be applied in a way that each part can work on parts of the dataset, which would be an interesting thing to see if on combination, would the combined network perform better on the combined data than the other two parts?

4. A function decomposition method may be used to split the cost function and Evopsnet can be applied in conjunction with combining cost function parts.

## VII. Discussion

In order to explore a search space efficiently a good amount of population is needed for EA to perform well. In the initial

| | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|
| Num of Hidden Layers | 5 | 10 | 20 | 40 |
| Train Accuracy | 0.2621 | 0.4135 | 0.5370 | **0.6592** |
| Train Loss | 2.01048 | 1.7192 | 1.3820 | **1.1033** |
| Test Accuracy | 0.2588 | 0.4135 | 0.5498 | **0.6586** |
| Test Loss | 2.0072 | 1.7050 | 1.3489 | **1.0818** |
| Training Time | 68773 | 103160 | 120353 | 128950 |

TABLE III: *Comparison of metrics for each level. Shows a clear increase in accuracy for each level up. Common information for all levels: initialization = uniform, population=120, generations=120, CPU-cores-used=8, GPU-memory = 8.9 GB.*

experiments of SGD vs NE, the population to explore such a large space was only 140, which is too small. The experiment Evopsnet has produced better results for lesser population and quite lesser generations. Experiments with generations of the order of 1000s would confirm the improving behaviour of Evospnet firmly. However, the compute time and resources consumed are incredibly higher than performing SGD, which puts Evospnet at a disadvantage. But, if one is able to prove that the accuracy attained by Evospnet is always higher than SGD based training, then one would have to compromise between resources, time and accuracy.

## References

[1] Montana, D. J., and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In Proceedings of the 11th International Joint Conference on Artificial Intelligence, Vol. 1, pp. 762–767.

[2] K. O. Stanley and R. Miikkulainen, ”Evolving Neural Networks through Augmenting Topologies,” in Evolutionary Computation, vol. 10, no. 2, pp. 99-127, June 2002, doi: https://doi.org/10.1162/106365602320169811.

[3] Stanley, K. O., and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research, 21:63–100.

[4] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. Artificial Life, 15(2):185–212.

[5] Inden, B., Jin, Y., Haschke, R., and Ritter, H. (2012). Evolving neural fields for problems with large input and output spaces. Neural Networks, 28:24–39.

[6] Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015). odNEAT: An algorithm for decentralised online evolution of robotic controllers. Evolutionary Computation, 23(3):421–449.

[7] McDonnell et al. (2018) “Divide and conquer,” Proceedings of the Genetic and Evolutionary Computation Conference [Preprint]. Available at: https://doi.org/10.1145/3205455.3205476.

[8] Kai et al. Guiding Neuroevolution with Structural Objectives. Evol Comput 2020; 28 (1): 115–140. doi:https://doi.org/10.1162/evco_a_00250

[9] Morse et al. ”Training Neural Networks Through the Integration of Evolution and Gradient Descent” (2019). Electronic Theses and Dissertations. 6714. https://stars.library.ucf.edu/etd/6714

[10] Faber et al. “Ensemble Neuroevolution-Based Approach for Multivariate Time Series Anomaly Detection,” Entropy, vol. 23, no. 11, p. 1466, Nov. 2021, doi: https://doi.org/10.3390/e23111466.

[11] Tang, Y. and Sato, K. (June 2022) EvoJAX: Bringing the power of neuroevolution to solve your problems — google cloud blog, Google. Available at: https://cloud.google.com/blog/topics/developers-practitioners/evojax-bringing-power-neuroevolution-solve-your-problems (Accessed: December 4, 2022).

[12] J Kim et al. SplitNet: Learning to Semantically Split Deep Networks for Parameter Reduction and Model Parallelization. Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1866-1874, 2017. https://proceedings.mlr.press/v70/kim17b.html.

[13] X. Dong et al., ”SplitNets: Designing Neural Architectures for Efficient Distributed Computing on Head-Mounted Systems,” 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 12549-12559, doi: 10.1109/CVPR52688.2022.01223.

[14] Malekhosseini et al. “Splitting Convolutional Neural Network Structures for Efficient Inference.” ArXiv abs/2002.03302 (2020): n. pag. doi: https://doi.org/10.48550/arXiv.2002.03302s.

[15] J. Malagavalli et al., ”Multi-Objective Differential Evolution with unbalanced Divide-and-Conquer Strategy for Warehouse Resource Allocation,” 2021 International Conference on Emerging Techniques in Computational Intelligence (ICETCI), 2021, pp. 26-33, doi: https://doi.org/10.1109/ICETCI51973.2021.9574070.

[16] Cybenko, G. (1989) “Approximation by superpositions of a sigmoidal function,” Mathematics of Control, Signals, and Systems, 2(4), pp. 303–314. Available at: https://doi.org/10.1007/bf02551274.

[17] Storn, R., Price, K. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997). https://doi.org/10.1023/A:1008202821328

[18] Michalewicz, Z. (1992) Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York. http://dx.doi.org/10.1007/978-3-662-02830-8

[19] Narkhede et al. M.S. A review on weight initialization strategies for neural networks. Artif Intell Rev 55, 291–322 (2022). https://doi.org/10.1007/s10462-021-10033-z

[20] A Continuous Space Neural Language Model for Bengali Language - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Gradient-Descent-Stuck-at-Local-Minima-18_fig4_338621083 [accessed 5 Dec, 2022]

[21] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.

[22] P. Virtanen et al., ‘SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', Nature Methods, vol. 17, pp. 261–272, 2020.

[23] Evgenia Papavasileiou, Jan Cornelis, Bart Jansen; A Systematic Literature Review of the Successors of “NeuroEvolution of Augmenting Topologies”. Evol Comput 2021; 29 (1): 1–73. doi: https://doi.org/10.1162/evco_a_00282.