Augmented Population Based Training*

Term Paper

Josias Moukpe†
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne Florida USA
jmoukpe2016@my.fit.edu

1 Introduction

Today, Artificial Neural Networks (ANN) have proven to be potent Machine Learning (ML) models, capable of learning and approximating any arbitrary function, given enough learnable parameters.

However, they aren't currently without flaws. Anyone who has developed an ANN to fit a particular application dataset would undoubtedly have to toil with the frustration of searching and tuning the numerous hyperparameters of the neural network model. This is because an ANN requires an adequate combination of hyperparameters values to perform its best. Those hyperparameters are not learned by the model, leaving this arduous task of hyperparameter optimization to the developer. To make things worse, the search space for those hyperparameters is often infinite. The general list of potential hyperparameters to tune includes but is not limited to:

1. Numerical hyperparameters
   - Number of hidden layers
   - Number of hidden units within each hidden layer
   - Learning rate
   - Number of training epochs
   - Momentum rate
   - Batch size (if input batches are being used)
   - Dropout rate
   - Weight Initialization tensor
   - Weight decay rate
   - Number of folds (if K-fold cross-validation is applied)
2. Non-numerical hyperparameters
   - Activation function (possibly at each unit)
   - Optimization algorithm
   - Loss function
   - Regularization techniques
   - Network topology (given cell type modules such as fully connected, convolutions, pooling, multi-head attention, etc.)

Furthermore, some hyperparameters can even be adaptively adjusted for performance gains (e.g., adaptive learning rate to reduce training time).

Unlike other machine learning approaches such as Decision Trees, ANNs have too many hyperparameters to tune. This results in a lot of development time spent fine-tuning them and retraining each time before optimally training for the best model for the chosen application. Deep Learning (DL), the branch of machine learning dealing with ANN, is currently more art than science. Unfortunately, that slows the progress made in DL, ML, and Computer Science as a whole. It's imperative to solve the problem of hyperparameter initialization and tuning by providing an automated or learned way to reliably set all the hyperparameters to the optimal value for any given task. This problem is also called Auto-ML or meta-machine learning [6].

2 Related Work

Our problem being hyperparameters search, we are aiming at providing a better, more reliable, and efficient method for hyperparameter initialization and tuning than the simple manual search. Other attempts at hyperparameter search or optimization are random search, grid search, automated hyperparameter tuning using Bayesian optimization or Genetic Algorithm, and Artificial Neural Network tuning using deep reinforcement learning [3]. Before exploring our approach, we first look at some state-of-the-art methods.

Manual search is the most basic and time-consuming approach, taking hours to months. The following technique that comes to mind is grid search, where every hyperparameter and its values are arranged on a grid and exhaustively searched to find the best combination of hyperparameter values. Unfortunately, Grid search is only applicable at low dimensions (2-4 hyperparameters to search) and is impractical at much higher dimensions where we usually need hyperparameter tuning for practical reasons. Random search improves on that, but at the cost of guaranteed optimality. Random search applies to larger search spaces and provides better results in less iteration than Grid search. Random search

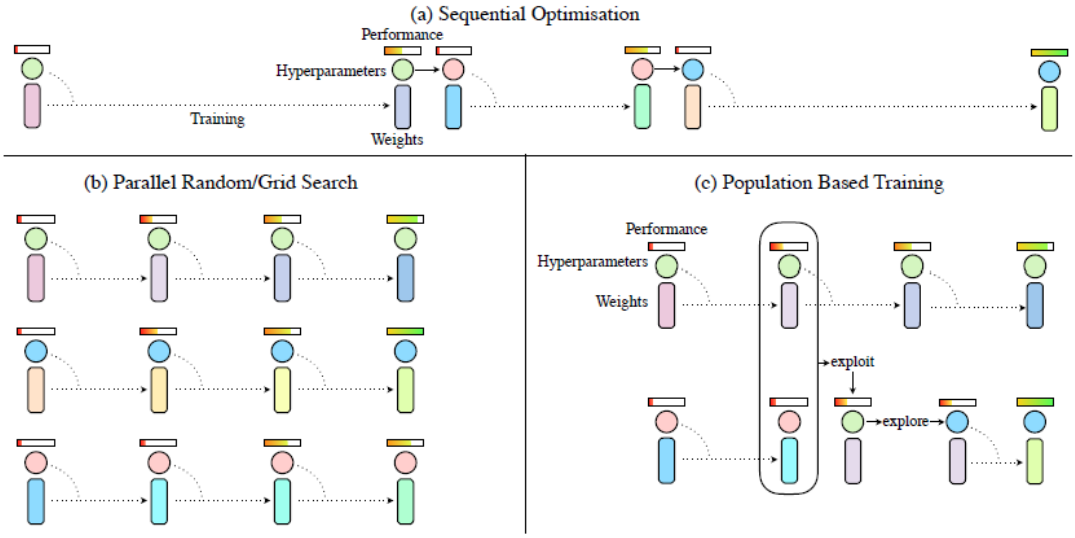

Figure 1. Processes of optimization

can also be run in parallel. Work done by Bergstra et al. demonstrates the superiority of Random search compared to Grid search and manual search [4]. However, like Grid Search, Random search doesn't leverage the information gained from the previous iterations; each new guess is independent of the previous one. Work done by Zoph et al. explores the application of using ANN or gradient-based methods to search optimal architectures (number of hidden layers, hidden units hyperparameters, and the type of layer units). This work leverages the information obtained at every guess and does better at providing the optimal model than just a random search [9]. However, since this work exploits a neural network to optimize the hyperparameters of another neural network, there's no significant reduction in hyperparameters to optimize. Finally, considerable strides have been made in applying evolution to hyperparameter optimization. Real et al. successfully applied genetic-based search to hyperparameter optimization with the introduction of aging evolution in the search [5]. Their work proved to systematically find more optimal models and find them quicker than Random search or ANN-based search.

Another more recent work by Li, Ang, et al. builds on it and provides a general framework for population-based training (PBT). Unlike other approaches, which first optimize the parameters then train the models, population-based training jointly optimizes hyperparameters and learnable parameters [7, 8]. In addition to the optimal models, this approach also provides hyperparameters schedules that more dynamically apply hyperparameters such as learning rate to optimally train models (not just a single hyperparameter value). Evolution-based methods have the added advantage of being able to be massively parallelized. However, PBT hasn't been developed to support deep architecture searches and doesn't cover design choices about the neural network topology. Our approach aims to augment PBT with architecture search to provide the optimal AutoML solution.

3 Approach

Our approach is Augmented Population Based Training and builds on top of the works by Real, Esteban, et al., Jaderberg, Max, et al., and Li, Ang et al. to devise a population-based method that not just optimize usual numeric hyperparameters such as learning rate, dropout rate, momentum, etc. But also searches for the optimal topology or architecture of the neural network given an initial set of architecture cells or modules (such as convolution, pooling, attention, fully connected, etc.) to compose from.

Our algorithm will work by providing an efficient way to encode and include the neural network topology in the population-based search, eventually coming up with the optimal model with the optimal topology and hyperparameter schedule in a single joint efficient optimization.

Our approach works by encoding the topology as a numeric list and adapting the genetic operators to optimize for topology as well, while the neural networks in the population are

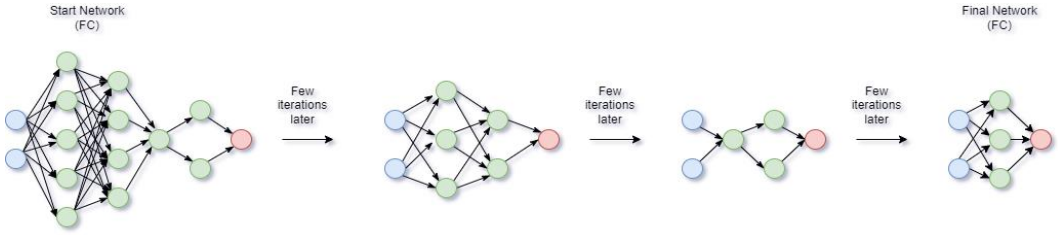

Figure 2. Evolution of ANN in population

Iris	Trial 0			Trial 1			Trial 2	
	Best	Most acc		Best	Most acc		Best	Most acc
Test Acc %	***94.24***	***96.6***		***91.24***	***94.88***		***66.42***	***68.8***
Num Params	***42***	***63***		***28***	***49***		***7***	***70***
Topology	***4,6,3***	***4,9,3***		***4,4,3***	***4,7,3***		***4,1,3***	***4,10,3***
Tennis	Trial 0			Trial 1			Trial 2	
	Best	Most acc		Best	Most acc		Best	Most acc
Test Acc %	***96.72***	***97.14***		***85.33***	***90.59***		***92.49***	***95.77***
Num Params	***72***	***120***		***12***	***96***		***12***	***24***
Topology	***10,6,2***	***10,10,2***		***10,1,2***	***10,8,2***		***10,1,2***	***10,2,2***
Identity	Trial 0			Trial 1			Trial 2	
	Best	Most acc		Best	Most acc		Best	Most acc
Test Acc %	***55.2***	***93.38***		***54.81***	***92.39***		***55.32***	***89.11***
Num Params	***16***	***144***		***16***	***160***		***16***	***144***
Topology	***8,1,8***	***8,9,8***		***10,1,2***	***8,10,8***		***8,1,8***	***8,9,8***

Figure 3. Summary of Results

simultaneous training their learnable parameters. Figure 1. and 2 illustrates the process.

Compared to manual sequential search, APBT is much faster, reducing the work of months or weeks to hours since it can automatically optimize for hyperparameters (possibly in parallel too). Unlike Grid search, it's able to search infinitely vast continuous spaces of hyperparameters because it leverages stochastic beam search capabilities of genetic algorithms. Unlike Random search, it's able to leverage the information gained from prior iterations to guide the search closer to its goal much more efficiently. This is because new members of the population are made based on the work done by older members (crossover operation). Finally, unlike PBT, it's able to optimize both topology and hyperparameters, not just hyperparameters given a fixed topology. Given the fact that a wrong topology can invalidate the most optimal hyperparameters, this feature of APBT gives it a strong advantage over other approaches.

3.1 Augmented Population Based Training

This part of the algorithm is identical to work done by [7]

Algorithm 1 Population Based Training (PBT)

procedure TRAIN($\mathcal{P}$) ▷ initial population $\mathcal{P}$
  for $(\theta, h, p, t) \in \mathcal{P}$ (asynchronously in parallel) **do**
    while not end of training **do**
      $\theta \leftarrow \texttt{step}(\theta|h)$ ▷ one step of optimisation using hyperparameters h
      $p \leftarrow \texttt{eval}(\theta)$ ▷ current model evaluation
      if $\texttt{ready}(p, t, \mathcal{P})$ **then**
        $h', \theta' \leftarrow \texttt{exploit}(h, \theta, p, \mathcal{P})$ ▷ use the rest of population to find better solution
        if $\theta \neq \theta'$ **then**
          $h, \theta \leftarrow \texttt{explore}(h', \theta', \mathcal{P})$ ▷ produce new hyperparameters h
          $p \leftarrow \texttt{eval}(\theta)$ ▷ new model evaluation
        end if
      end if
      update $\mathcal{P}$ with new $(\theta, h, p, t+1)$ ▷ update population
    end while
  end for
  return θ with the highest p in $\mathcal{P}$
end procedure

Figure 4. Original PBT procedure

3.2 Evaluate (fitness)

Evaluate is our fitness function. It takes an ANN, a validation/test set and returns the score based on a ration function f describe below:

$$f(accuracy, size) = {}^{1.09^{accuracy}}/_{1.02^{size}}$$

So, in steps, it computes the accuracy over the set, then get the ANN size, and finally apply the special ratio f on accuracy and size to get the performance score or fitness.

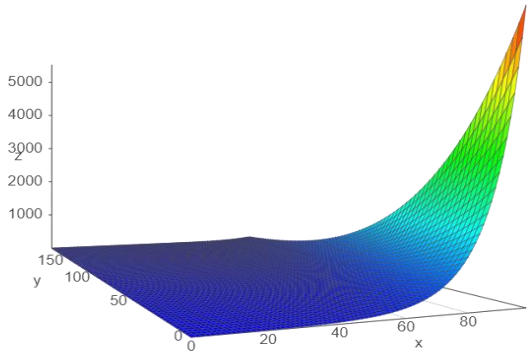

Figure 5 Visualization of ration function

3.3 Truncation selection

Our selection function is called truncation selection and is based on readiness and the population members. An ANN is simply "ready" after a set number of iteration has elapsed since last time it was ready.

Truncation selection works by first ranking the ANNs in the population by performance ratio. If current net is in the bottom 20% of the population, we sample another net uniformly from the top 20% of the population to exploit.

3.4 Exploit (crossover)

Our crossover function here is exploit. When a net is ready after a certain number of iterations, and is struggling in performance (bottom 20%), it can exploit the rest of the population by copying the topology, weights, and

hyperparameters of the top net selected from the population with truncation selection.

3.5 Explore (mutation)

Finally, we have explore which is our mutation function here. Following exploitation, the net experiences a perturbation in its hyperparameters, and topology before continuing training with the new hypermeters and topology. Hyperparameters are multiplied by either .8 or 1.2. Topology either gains a neuron, loses a neuron, or does nothing.

4 Empirical Evaluation

4.1 Evaluation Criteria

To solve AutoML, we propose an augmentation of a general framework for population-based training that searches for the optimal network architecture. To evaluate our approach, we will be attempting to find optimal hyperparameters for optimal ANN models on the following datasets:

- Iris Dataset
- Tennis Dataset
- Identity Dataset

The ANN models would be feedforward neural networks. We aim to demonstrate the superiority of our approach in providing the optimal models with the optimal set of hyperparameters in a quick, reliable, and computationally inexpensive fashion. The main performance metrics we will use to evaluate our method include:

- Effective number of hyperparameters needed (lower is better)
- Top Test Accuracy (the accuracy of the best model produced by the approach, higher is better).
- Model size (or the model's overall size expressed in terms of the number of parameter weights and biases in the model, lower is better).
- Accuracy per Size Ratio (higher is better)

With these metrics, we aim to demonstrate that our method performs better consistently across the board when compared to other similar approaches such as grid search, random search, and ANN search. We will also discuss secondary criteria such as how long it takes to run and how much space it takes when compared to backpropagation.

4.2 Experimental Data and Procedures

To evaluate our approach, we will be attempting to find optimal hyperparameters for optimal ANN models on the following datasets:

- Iris Dataset

For Iris dataset, inputs are already continuous and are not preprocessed. However, outputs are discrete categories and are therefore encoded using one-hot encoding. 20% of the dataset is used for validation. For experiment, the population size used is k=80, readiness is set to 220 epochs and the total number of epochs is 3000.

- Tennis Dataset

For Tennis dataset, both inputs and outputs are discrete categories and are therefore encoded using one-hot encoding. Again 20% of dataset used for validation. For experiment, the population size used is k=160, readiness is left at 220 epochs and the total number of epochs is 2600.

- Identity Dataset

For Identity dataset, no encoding for the inputs or outputs was required. The whole dataset was used as validation set. For experiment, the population size used is k=600, readiness is set to 400 epochs and the total number of epochs is 8000.

All experiments were run on the same device for 3 consecutive trials on each dataset and the results were all recorded.

4.3 Results and Analysis

Overall results are reflected on table 1. Detail graphs for the results are in exp-results.xlsx submitted with this paper**.**

Couple of observations we gather from the results are the following:

APBT works quite well in optimizing topology and hyperparameters jointly. It returns the highest absolute accuracy and effective accuracy networks, training them on optimal hyperparameters schedules. APBT only effectively requires 3 parameters (and mostly just 2). APBT appears to not be too sensitive to local minima or overfitting.

According the Figure 3. The algorithm doesn't just produce single value hyperparameters but produces schedules of those hyper parameters. Intuitively the algorithm learns to reduce the learning rate overtime to reduce chances of overshooting the objective, increase the momentum overtime to reduce chances of falling into local minima.

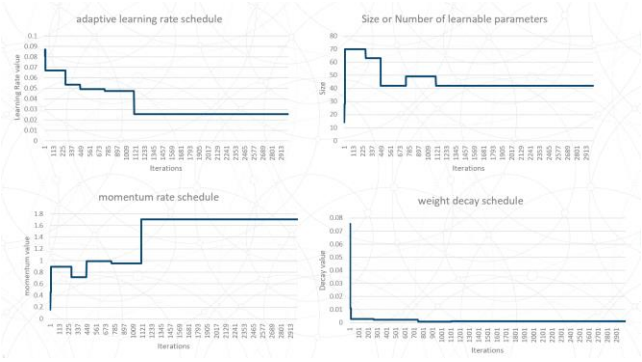

Figure 6 Hyperparameter schedules

Another observation we make is that higher k result in a more diverse population, which is great as long as computational resources are available for it.

Higher readiness in selection allows the ANNs to train for longer before sharing. However, at maximum readiness (total number of epochs), it's just k-random search as opposed to APBT. On the other hand, longer epochs ranges tend to allow the rest of the population to catch up in performance to the top performers.

The most effectively accurate ANN (highest accuracy for lowest size) is not always the most absolutely accurate.

In term of time of computation, APBT appears to use k (population size) times more time to compute than backpropagation. This can be alleviated with parallelism since APBT can be run in parallel. Moreover, in term of space, APBT uses k times more space than backpropagation.

APBT appears to be inconsistent with small k values and small dataset. On Identity, APBT was not able to find the minimal network.

5 Conclusion

To solve AutoML, we propose Augmented Population Based Training (APBT) to effectively optimize both topology and hyperparameters. APBT is found to work really well and require far less hyperparameters. It provides the most accurate, lowest size neural net, which is trained on an optimal schedule of hyperparameters. However, APBT is not perfect yet. APBT takes k (population size) times more space to run than Backpropagation, without parallelism runs k times longer than BP, and doesn't currently support early stopping. APBT can be inconsistent with smaller k values and very small datasets. Surprisingly, APBT struggles with finding optimal topology for Identity dataset. Nonetheless, to solve AutoML, APBT is a powerful step in the right direction. Further research could look into investigating issues with Identity or incorporating early stopping criteria in the training loop.

REFERENCES

[1] Koutsoukas, Alexios, et al. "Deep-Learning: Investigating Deep Neural Networks Hyper-Parameters and Comparison of Performance to Shallow Methods for Modeling Bioactivity Data." Journal of Cheminformatics, vol. 9, no. 1, 2017,
[2] Mitchell, Tom Michael. Machine Learning. McGraw-Hill, 1997.
[3] Ippolito, Pier Paolo. "Hyperparameters Optimization." Medium, Towards Data Science, 26 Sept. 2019,
[4] Bergstra, James, and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." Journal of Machine Learning Research 13, (2012) 281-305, 2 Dec. 2012.
[5] Real, Esteban, et al. "Regularized Evolution for Image Classifier Architecture Search." Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 4780–4789.,
[6] Gozzoli, Alessio. "Practical Guide to Hyperparameters Optimization for Deep Learning Models." FloydHub Blog, FloydHub Blog, 1 July 2020,
[7] Jaderberg, Max, et al. "Population-based training of neural networks." arXiv preprint arXiv:1711.09846 (2017).
[8] Li, Ang, et al. "A Generalized Framework for Population Based Training." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019,
[9] Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. In ICLR.