Automatic model selection for neural networks

David Laredo

***Abstract*—Neural networks and deep learning are changing the way that artificial intelligence is being done. Efficiently choosing a suitable model (including hyperparameters) for a specific problem is a time-consuming task. Choosing among the many different combinations of neural networks available gives rise to a staggering number of possible alternatives overall. Here we address this problem by proposing a fully automated framework for efficiently selecting a neural network model given a specific problem (whether it is classification or regression). Our proposal focuses on a distributed decision-making algorithm for keeping the most promising models among a pool of possible models. We hope that this approach will help non-expert users to more effectively identify neural network based models and hyperparameter settings appropriate to their applications, and hence to achieve improved performance.**

***Index Terms*—artificial neural networks, model selection, hyperparameter tuning, distributed computing, evolutionary algorithms.**

I. Introduction

MACHINE learning studies automatic algorithms that improve themselves through experience. Given the large amounts of data currently available in many fields such as engineering, biomedical, finance, etc, and the increasingly computing power available machine learning is now practiced by people with very diverse backgrounds. Increasingly, users of machine learning tools are non-experts who require off-the-shelf solutions. The machine learning community has aided these users by making available a variety of easy to use learning algorithms and feature selection methods as WEKA [1] and PyBrain [2]. Nevertheless, the user still needs to make some choices which not may be obvious or intuitive (selecting a learning algorithm, hyperparameters, features, etc).

Recently, neural networks have gained a lot of attention due to the newer models (CNN, RNN, Deep Learning, etc.) and their flexibility and generality for solving a large number of problems: regression, classification, natural language processing, recommendation systems, just to mention a few. Furthermore, there are a lot software libraries which makes their implementations easy to use (tensorflow, keras, kaffe, etc.). Nevertheless, the task of picking the right neural network model (hyperparameters included) can be even more complicated than that of other algorithms. Given the popularity of neural networks, specially among non computer scientist we will focus our efforts in this study to them and leave other algorithms for future work.

Usually, the process of selecting a suitable machine learning model for a particular problem is done in an iterative manner. First, an input dataset must be transformed from a domain specific format to features which are predictive of the field of interest. Once features have engineered users must pick a learning setting appropriate to their problem, e.g. regression, classification or recommendation. Next users must pick an appropriate model, such as support vector machines (SVM), logistic regression, any flavor of neural networks (NN). Each model family has a number of hyperparameters, such as regularization degree, learning rate, number of neurons, and each pf these must be tuned to achieve optimal results. Finally, users must pick a software package that can train their model, configure one or more machines to execute the training and evaluate the model's quality. It can be challenging to make the right choice when faced with so many degrees of freedom, leaving many users to select a model based on intuition or randomness and/or leave hyperparameters set to default. Certainly this approach will usually yield suboptimal results.

This suggests a natural challenge for machine learning: given a dataset, to automatically and simultaneously chose a learning algorithm and set its hyperparameters to optimize performance. As mentioned in [1] the combined space of learning algorithm and hyperparemeters is very challenging to search: the response function is noisy and the space is high dimensional, involves both, categorical and continuous choices and containes hierarchical dependencies (e.g. hyperparameters of the algorithm are only meaningful if that algorithm is chosen). Thus, identifying a high quality model is typically costly (in the sense that entails a lot of computational effort) and time consuming.

Distributed and cloud computing provide a compelling way to accelerate this process, but also present additional challenges. Though parallel storage and processing techniques enable users to train models on massive datasets and accelerate the search process by training multiple models at once, the distributed setting forces several more decisions upon users: what parallel execution strategy to use, how big a cluster to provision, how to efficiently distribute computation across it, and what machine learning framework to use. These decisions are onerous, particularly for users who are experts in their own field but inexperienced in machine learning and distributed systems.

To address this challenges we propose NeuroTuner a flexible and scalable system to automate the process of selecting artificial neural network models.

II. The CASH problem

In this section we introduce and formally describe the model selection problem, for this section we borrow the definitions given in [3]. This work focuses on supervised learning: learning a function $f : \mathcal{X} \mapsto \mathcal{Y}$ with finite $\mathcal{Y}$. A learning algorithm A maps a set $\{d_1, \ldots, d_n\}$ of training data points $d_i = (\mathbf{x_i}, \mathbf{y_i}) \in \mathcal{X} \times \mathcal{Y}$ to such a function. Most learning

D. Laredo is with the Department of Mechanical Engineering, University of California, Merced, CA, 95340 USA e-mail: dlaredorazo@ucmerced.edu.

algorithms A further expose hyperparameters $\lambda \in \mathbf{\Lambda}$, which change the way the learning algorithm A_λ works. One example of hyperparameters is the number of neurons in a hidden layer of an ANN, another common example is the learning rate α of a neural network. These hyperparameters are typically optmized in an "outer loop" that evaluates the performance of each hyperparameter configuration using cross-validation.

A. Model selection

Given a set of learning algorithms $\mathcal{A}$ and a limited amount of training data $\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}, \ldots \mathbf{x_n}, \mathbf{y_n})\}$, the goal of model selection is to determine the algorithm $A^* \in \mathcal{A}$ with optimal generalization performance. Generalization performance is estimated by splitting $\mathcal{D}$ into disjoint training and validation sets $\mathcal{D}_t$ and $\mathcal{D}_v$ respectively, learning function f by applying A^* to $\mathcal{D}_t$, and evaluating the predictive performance of this function on $\mathcal{D}_v$. Using k-fold validation, which splits the data into k equal sized partitions $\mathcal{D}_v^1, \ldots, \mathcal{D}_v^k$ and sets $\mathcal{D}_t^i = D \backslash D_v^i$ for $i = 1, \ldots, k$ the model selection problem is written as:

$$A^* \in \underset{A \in \mathcal{A}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A, \mathcal{D}_t^i, \mathcal{D}_v^i\right), \tag{1}$$

where $\mathcal{L}\left(A, \mathcal{D}_t^i, \mathcal{D}_v^i\right)$ is the loss achieved by A when trained on $\mathcal{D}_t^i$ and evaluated on $\mathcal{D}_v^i$.

B. Hyperparameter optimization

The problem of optimizing the hyperparameters $\lambda \in \mathbf{\Lambda}$ of a given learning algorithm A is conceptually similar to that of model selection. Some key differences are that hyperparameters are often continuous, that hyperparameter spaces are often high dimensional, and that we can exploit correlation structure between different hyperparameter settings $\lambda_1, \lambda_2 \in \mathbf{\Lambda}$. Given n hyperparameters $\lambda_1, \ldots, \lambda_n$ with domains $\Lambda_1, \ldots, \Lambda_n$, the hyperparameter space $\mathbf{\Lambda}$ is a subset of the crossproduct of these domains: $\mathbf{\Lambda} \subset \Lambda_1 \ldots \Lambda_n$. This subset is often strict, such as when certain settings of one hyperparameter render other hyperparameters inactive. For example, the parameters determining the specifics of the third layer of a deep belief network are not relevant if the network depth is set to one or two. More formally, following [4], we say that a hyperparameter λ_i is conditional on another hyperparameter λ_j, if λ_i is only active if hyperparameter λ_j takes values from a given set $V_i(j) \subsetneq \Lambda_j$; in this case we call λ_j a parent of λ_i. Conditional hyperparameters can in turn be parents of other conditional hyperparameters, giving rise to a tree-structured space [5] or, in some cases, a directed acyclic graph (DAG) [4]. Given such a structured space $\mathbf{\Lambda}$, the (hierarchical) hyperparameter optimization problem can be written as:

$$\lambda^* \in \underset{\lambda \in \mathbf{\Lambda}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A_\lambda, \mathcal{D}_t^i, \mathcal{D}_v^i\right), \tag{2}$$

In this study we consider the more general combined algorithm selection and hyperparameter optimization (CASH). That is we intend to optimize both problems at the same time.

III. Literature Review

Automatic model selection has been of research interest since the uprising of deep learning. This is no surprise since selecting an effective combination of algorithm and hyperparameter values is currently a challenging task requiring both deep machine learning knowledge and repeated trials. This is not only beyond the capability of layman users with limited computing expertise, but also often a non-trivial task even for machine learning experts [6].

To make machine learning accessible to non-expert users, researchers have proposed various automatic selection methods for machine learning algorithms and/or hyperparameter values for a given supervised machine learning problem. These methods' goal is to find, withind a pre-specified resource limit (usually specified in terms of time, number of algorithms and/or combintations of hyperparameter values), an effective algorithm and/or combination of hyperparameter values that maximize the accuracy measure on the given machine learning problem and data set. Using an automatic selection method method, the machine learning practitioner can skip the manual and iterative process of selecting and efficient combination of of hyperparameter values and neural network model, which is high labor intensive and requires a high skill set in machine learning.

In the recent years a number of tools have been made available for users to automate the model selection and/or hyperparameter tuning, in the following we present a brief survey of the most popular methods.

A. AutoWEKA

Auto-WEKA [7] is a system designed to help machine learning users by automatically searching through the joint space of WEKA's learning algorithms and their respective hyperparameter settings to maximize performance using a state-of-the-art Bayesian optimization method. AutoWEKA addresses the CASH problem by treating all of WEKA as a single, highly parametric machine learning framework, and using Bayesian optimization to find a strong instantiation for a given dataset. AutoWEKA also natively supports parallel runs (on a single machine) to find good configurations faster and save the N best configurations of each run instead of just the single best. AutoWEKA is tightly integrated with WEKA and does provide support for Multilayer Perceptrons (MLP).

B. Auto-sklearn

Auto-sklearn [8] is Auto-WEKA's sister package, it uses the same Bayesian optimizer but comprises a smaller space of models and hyperparameters, however it includes additional meta-learning techniques.

C. TuPAQ

TuPAQ [6] is a system designed to efficiently and scalably automate the process of training predictive models. One of its main features is a planning algorithm which decides on an efficient parallel execution strategy during model training while identifying new hyperparameter configurations and proactively

eliminating models which are unlikely to provide good results. TuPAQ is aimed at large scale machine learning, it builds on top of the well know Apache Spark. TuPaq only focuses on classification problems and considers only three model families (Support Vector Machines, Logistic Regression and nonlinear SVMs), each with several hyperparameters. TuPAQ performs batching to train multiple models simultaneously and deploys bandit resource allocation to allocate more resources to the most promising models. TuPAQ does not provide support for neural networks.

IV. An evolutionary framework for the CASH problem

While there is a number of methods for automatic model selection and hyperparameter tuning, the most popular ones still have room for improvement. In the case of AutoWEKA and Auto-sklearn they do not provide good support for large machine learning problems, nor provide support for distributed computing. TuPAQ on the other hand, provides wide support for distributed computing, maximizing the use of computational resources through the use of sophisticated optimizations, nevertheless its restricted to only classification problems and does not provide support for neural networks.

We propose to implement a system for automatically selecting the most fitting neural network architecture (only fully connected networks in the first stage) for a given problem, whether it is classification or regression. Furthermore, we plan that the system should be scalable and should be able to be used in distributed computing environments, allowing it to be usable for large datasets and complex models. To achieve the latter we propose to build our system using Ray [9] which is a distributed system designed with large scale distributed machine learning in mind.

For this work we will consider three major architectures of neural networks, namely multilayer percpetrons (MLPs) [10], convolutional neural networks (CNNs) [11] and recurrent neural networks (RNNs) [12]. Each one of these architectures can be built by stacking together a *valid* combination of any of the four following layers: fully connected layers, recurrent layers, convolutional layers and pooling layers.

We say that a neural network architecture is *valid* if it complies with the following set of rules, which we derived empirically from our practice in the field:

- A fully connected layer can only be followed by another fully connected layer
- A convolutional layer can be followed by a pooling layer, a recurrent layer, a fully connected layer or another convolutional layer
- A recurrent layer can be followed by another recurrent layer or a fully connected layer.
- The first layer is user defined according to the type of architecture chosen (MLP, CNN or RNN)
- The last layer is always a fully connected layer with either a softmax activation function for classification or a linear activation function for regression problems

A. The fitness function

In order to steer the search in the most promising search directions, a carefully designed fitness function is required. The framework's goals are to generate a neural network architecture with good predictive power for the class of problem at hand while keeping the complexity of the network as low as possible. Measuring the predictive power of the network is straightforward; having a valid neural network we can assess its predictive power by training it on the set $\mathcal{D}_t$ and then evaluating the predictions using the set $\mathcal{D}_v$. A more robust approach would be performing a k-fold cross-validation which splits the data into k equal sized partitions $\mathcal{D}_v^1, \ldots, \mathcal{D}_v^k$ and sets $\mathcal{D}_t^i = D \backslash D_v^i$ for $i = 1, \ldots, k$.

Let $A \in \mathcal{A}$ be a certain neural network architecture trained on set $\mathcal{D}_t$, let also $\mathcal{P}_A(\mathcal{D}_v)$ represent the performance of the neural network A when tested using validation set $\mathcal{D}_v$ and the user-defined performance indicator p, where p is usually any of the metrics listed in Table II . Using k-fold cross-validation the average performance of the algorithm can be written as

$$p = \frac{1}{k} \sum_{i=1}^{k} \mathcal{P}_A(\mathcal{D}_v^i) \tag{3}$$

For measuring the complexity of the architecture we consider the number of trainable weights w of the neural network which is a good indicator of how complex the architecture is.

Using p and w we propose the following fitness function

$$f = p + \lambda w, \tag{4}$$

where $\lambda \in [0, 1]$ is a scaling factor that indicates how much does the number of trainable weights w affects the overall fitness of the neural network. By setting $\lambda = 1$ the preference is given to very compact architectures, while $\lambda = 0$ will only care about architectures that find the best possible value for p regardless of their complexity.

B. Evolutionary algorithms

The main part of the framework consists of an evolutionary algorithm, evolutionary algorithms (EAs) are a family of methods for optimization problems. The methods do not make any assumptions about the problem, treating it as a black box that merely provides a measure of quality given a candidate solution. Furthermore, EAs do not require the gradient when searching for optimal solutions, making them very suitable for applications such as neural networks.

In the following we describe the very basics of evolutionary algorithms as an introduction for the reader.

Every evolutionary algorithm consists of a population of individuals (sometimes EAs are also referred as population based algorithms). Each individual in the population is indeed a potential solution to the optimization problem. Individuals are generally encoded, this encoded solution is often called a genotype while the actual representation of the genotype in the domain of the problem is referred as a fenotype, for our application the fenotype represents the neural network architecture while the genotype will be defined later on. Each solution is evaluated using a so-called fitness function, where

the function represents how does the individual performs with respect to a certain metric.

At every iteration a new generation of solutions is generated by using crossover and mutation operators. Crossover operator is an evolutionary operator used to combine the information of two parents to generate new offspring while the mutation operator is used to maintain genetic diversity from one generation of the population to the next.

The basic template for an evolutionary algorithm is the following

Algorithm 1 Basic Evolutionary Algorithm

Let $t = 0$ be the generation counter
Create and initialize an n_x-dimensional population, $\mathcal{C}(0)$, to consist of n_s individuals
while stopping condition not true **do**
  Evaluate the fitness, $f(\mathbf{x}_i(t))$, of each individual, $\mathbf{x}_i(t)$
  Perform reproduction to create offspring
  Select the new population, $\mathcal{C}(t+1)$
  Advance to the new generation, i.e. $t = t + 1$
end while

Among the many different choices for evolutionary algorithms three major trends currently lead the way, we refer to the genetic algorithms (GAs), evolutionary strategies (ES) and genetic programming (GP) [10].

C. Encoding neural networks as genotypes

In order to perform the optimization of neural network architectures a suitable encoding for the neural networks is needed. A good encoding has to be flexible enough to represent neural network architectures of variable length while also making it easy to verify the *validity* of a proposed neural network architecture.

While array based encodings are quite popular for numerical problems, they often use a fixed-length genotype. While it is possible to use an array based representation for encoding a neural network, this would require the use of very large arrays, furthermore verifying the validity of the encoded neural network is hard to achieve. Three-based representation as that used in genetic programming enables more flexibility when it comes to the length of the genotype, nevertheless imposing constraints for building a valid neural network requires traversing the entire tree or making use of complex data structures every time a new layer is to be stacked in the model.

For this work, the chosen encoding is list-based, that is, the genotype is represented as a list of arrays, where the length of the list can be arbitrary. Each array within the list represents the details of a given layer as described in Table III. A visual depiction of the array is presented in Table I.

Layer type	Neuron number	Activation function	CNN filter size	CNN kernel size	CNN stride

TABLE I: Visual representation of a neural network layer as an array.

Encoding the neural network as a list of arrays presents two big advantages over other representations. First, the number of layers that can be stacked is in principle arbitrary. Second, the validity of an architecture can be verified, in constant time, every time a new layer is to be stacked to the model, this is due to the fact that in order to stack a layer in between the model one just needs to verify the previous layer and the layer ahead to check for compatibility.

D. The evolutionary operators

Since the encoding chosen for this task is rather peculiar, no specific operators exist for it. Instead of trying to create new operators for this task we take common evolutionary operators and adapt them to our encoding, in the following we describe the operators and their adaptations.

1) Mutation operator:

2) Crossover operator:

REFERENCES

[1] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[2] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rucksties, and J. Schmidhuber. Pybrain. *JMLR*, 11:743–746, 2010.

[3] Thornton C., Hutter F., Hoos H., and Leyton-Brown K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *KDD*, 2013.

[4] Hutter F., Hoos H., Leyton-Brown K, and Stutzle T. Paramils: and automatic algorithm configuration framework. *JAIR*, 36(1):267–306, 2009.

[5] Bergstra J., Bardenet R., Bengio Y., and Kegl B. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

[6] Sparks ER., Talwalkar A., Smith V., Kottalam J., Pan X, and Gonzales JE. Automated model search for large scale machine learning. In *SoCC*, pages 368–380, 2015.

[7] Thornton C., Hutter F., Hoos H., Leyton-Brown K, and Kotthoff L. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *JMLR*, 2016.

[8] Feurer M., Klein A., Eggensperger K., Springenberg J., Blum M., and Hutter F. Efficient and robust automated machine learning. In *NIPS*, volume 17, pages 1–5, 2015.

[9] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889, 2017.

[10] Douglas P. Engelbrecht. *Computational Intelligence. An Introduction*. Wiley, 2007.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[12] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

Metric name	Definition
Root Mean Squared Error	Regression
Accuracy	Classification
Precision	Classification
Recall	Classification
F1	Classification

TABLE II: Common performance metrics for neural networks

Cell number	Cell name	Data Type	Represents	Applicable to	Values
0	Layer type	Integer	The type of layer. See table IV	MLP/RNN/CNN	$x \in \{1, \ldots, 5\}$
1	Neuron number	Integer	Number of neurons/units in the layer	MLP/RNN	$8 * x$ where $x \in \{1, \ldots, 128\}$
2	Activation function	Integer	Type of activation function. See table IV	MLP/RNN/CNN	$x \in \{1, \ldots, 4\}$
3	Filter size	Integer	Number of filters generated by the layer	CNN	$8 * x$ where $x \in \{1, \ldots 64\}$
4	Kernel size	Integer	Size of the kernel used for convolutions	CNN	3^x where $x \in \{1, \ldots, 6\}$
5	Stride	Integer	Stride used for convolutions	CNN	$x \in \{1, \ldots, 6\}$

TABLE III: Details of the representation of a neural network layer as an array.

Index	Type of layer	Activation function
1	Fully connected	Sigmoid
2	Convolutional	Hyperbolic tangent
3	Recursive	ReLU
4	Pooling	Not applicable

TABLE IV: Mapping from indexes to layer/function.