2021 12th International Conference on Information and Communication Systems (ICICS) | 978-1-6654-3351-8/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICICS52457.2021.9464618

Automatic Hyperparameter Optimization for Arbitrary Neural Networks in Serverless AWS Cloud

Alex Kaplunovich
Department of Computer Science
University of Maryland
Baltimore, USA
akaplun1@umbc.edu

Yelena Yesha
Department of Computer Science
University of Maryland
Baltimore, USA
yeyesha@umbc.edu

***Abstract*— Deep Neural Networks are the most efficient method to solve many challenging problems. The importance of the subject can be demonstrated by the fact that the 2019 Turing Award was given to the godfathers of AI (and Neural Networks) Yoshua Bengio, Geoffrey Hinton, and Yann LeCun. In spite of the numerous advancements in the field, most of the models are being tuned manually. Accurate models became especially important during the novel coronavirus pandemic.**

Many day-to-day decisions depend on the model predictions affecting billions of people. We implemented a flexible automatic real-time hyperparameter tuning approach for arbitrary DNN models written in Python and Keras without manual steps. All of the existing tuning libraries require manual steps (like hyperopt, Scikit-Optimize or SageMaker). We provide an innovative methodology to automate hyper-parameter tuning for an arbitrary Neural Network model source code, utilizing Serverless Cloud and implementing revolutionary microservices, security, interoperability and orchestration. Our methodology can be used in numerous applications, including Information and Communication Systems.

Keywords*— *Neural Networks, Cloud, Hyperparameter, Automation, Optimization, Server-less, Machine Learning, Data Science, AWS

I. INTRODUCTION

Various scientists (including James Bergstra [1], [3], and [7]) have been working on the problem of hyperparameter optimization. Their solutions require to identify available hyperparameters manually before the tuning starts for each model (the hyperopt library[13]). We decided to eliminate the manual steps completely, and have implemented a fully automated methodology for optimization, saving a lot of time and money. The Twenty First century with its advanced technologies is the time to eliminate manual Machine Learning.

What are the Hyperparameters? They are the values – numeric or textual – needed to configure a multi-layer (deep) neural network model. These parameters affect the network behavior and predictions.

They include:

- Number of layers
- Number of neurons
- Dropout rate
- Learning rate
- Number of epochs
- Activation function
- Optimization function

It is important to mention that every layer has a separate set of hyperparameters. One of the intuitive solutions would be to run our model for all possible parameter combinations (grid search). That approach is not practical because of size of our search space. Curse of dimensionality [9] points that the number of grid search trials is exponential in the number of search dimensions. For example, if we have five pa-rameters, each taking just ten values, we will have to run our model 10^5 times to exhaustively search our dimension space. We have an exponential dependency. It is physically impossible to run so many experiments, even with virtually unlimited cloud resources.

Our goal is to find a practical solution to optimize the hypermaraments in reason-able amount of time. Neural networks usually run on expensive GPU instances, for which the cloud prices range from 90 cents an hour to 48 dollars an hour. It has been demonstrated by James Bergstra et al. in [1] and [7] that randomly chosen hyperparameters are much more efficient than exhaustive grid search or manual search. The number of parameters can be very large and it will be physically impossible to use all possible value permutations to find the optimal subset. We dedicate our research to James Bergstra.

There are several hyperparameter optimization Python libraries available – Scikit-optimize [18], [19], and [21], RandomizedSearchCV [20] or hyperopt [13]. These libraries require a developer to create a dictionary of parameters and their values manually. We were unable to find an automatic tool that will be optimizing hyperparameters for an arbitrary model.

The research community needs a generic automatic approach. Our methodology handles all the possible neural

networks (CNN, RNN, ANN, LSTM, etc.). We run the tuning automatically given the two important input sources – model code and data files to train and validate the neural network. Our methodology starts automatically and notifies the stakeholders upon completion via cloud function triggering. We apply powerful parser and dynamic code generation to implement model tuning on the fly in the Cloud using the hyperopt library and providing novel cloud architecture, scalability, spot instance procurement and results saving in our Serverless functions.

II. PROPOSED APPROACH

The major technical contribution of our methodology is automation, cloud interoperability and orchestration, wise software architecture and elimination of manual steps.

There are numerous moving parts required for model tuning:

- Cloud function triggering when a new model is uploaded
- Parser invocation
- Updated model saving
- Cloud functions invocating and triggering
- GPU spot instance template configuration
- Spot instance request placement
- Spot instance spawning
- Cloud function interaction and data exchange
- GPU instance termination
- User notification
- Results saving into DynamoDB
- Methodology monitoring
- Security policy implementation and maintenance
- Terminating resources upon completion
- Cloud and serverless integration
- Error handling and fault tolerance
- Concurrency and scalability enablement

Our approach implements the complete pipeline, making it look easy from the data scientists perspective. We have carefully designed our methodology for all the working parts to interact and function together. It is a challenging scientific software architecture work.

All stages of our pipeline are designed to work together via triggered cloud functions. Once the pipeline starts, no human interaction is required. Our approach works for any machine learning model, processing source code, learning about its insights and transforming the code into an optimized model. It is a generic approach producing better performing models and serving the data science community to allow re-searchers to concentrate on their algorithms without implementation of the routine manual tuning steps. Our approach provides substantial cost saving, because our pipeline spawns cloud spot instances to train the models. These instances save up to 90% in hardware cost. Moreover, we have designed the approach with zero idle time – the instances will be terminated as soon as the experiments are completed.

Manual hyperparameter specification is time consuming and tedious step that can be automated. Our method has proven that it is possible to tune models automatically without data scientists spending their precious time specifying parameters and their potential values. Our approach is an enabler for researches to spend their time more efficiently and productively.

Innovation and automation are the stepping stones of our method, providing the unified methodology to find optimal hyperparameters for an arbitrary Machine Learning model code. Even today, very few ML techniques are trying to deal with source code to find the insights of the model. Our cloud expertise helped us to design the state-of-the-art integration with the ecosystem. We feel that it is necessary to design a process that will calculate the optimal hyperparameter values for supervised neural networks code, while allowing experienced data scientists to write their own algorithms for data filtering, processing, model creation, and neural network business logic. We are bridging the gap and addressing the need for ML automation.

The methodology pipeline will parse and analyze the code and find all the possible hyperparameters for optimization. Our approach implements the following major steps to produce the tuned deep neural networks model:

- Keras layers hyperparameters metadata
- Parsing a Python machine learning model
- Identifying the hyperparameter space
- Updating the code if the original model
- Running the updated model with the hyperopt library in the Cloud
- Saving the results and sending notifications
- Integrating with Serverless and Cloud ecosystem

A. Keras layers hyperparameters metadata

Keras is a leading machine learning library, simplifying the neural network creating and implementing the basic convolutional (CNN), recurrent (RNN) and long-short-term-memory (LSTM) and other pipelines. We have designed and developed an input spreadsheet containing the model hyperparameter metadata. Users do not use metadata directly, it is being loaded by our methodology during startup. Each metadata line specifies a single hyperparameter details. The metadata is stored safely and securely in the protected Cloud location. The following columns are describing the parameters:

Layer, ParamName, Hptype, ParamOrder, Base, Type, Min, Max, Int, and Set.

Below is the detailed description of the columns. Our metadata guides the pipeline to identify the parameters and their ranges:

- Layer – a name of the Keras class representing a Neural Network layer (Dropout, Activation, Dense, etc.)

- ParamName – a name of the hyperparameter (rate, activation, etc.)
- Hptype – the type of the parameter values distribution from the hyperopt library (choice, uniform, quniform, etc.)
- ParamOrder – order of the parameter in the class initialization list of arguments
- Base – base class name. Used if the same parameter is shared between a number of subclasses (for example, for multiple activations or optimizers)
- Type – parameter type (n for number, s for string)
- Min – minimum value for numeric parameters
- Max – maximum value for numeric parameter
- Int – a Boolean flag identifying that the parameter value should be integer
- Set – set of parameter values for string parameters (for example, "relu, softmax, sigmoid" values for the activation parameter)

For security reasons, only system has access to the file to make sure it can not be tempered, deleted or modified by malicious forces, trying to prevent us from providing the best models to the data science community.

B. Parsing a Machine Learning Model

Unlike the majority of the model tuning libraries, we start our pipeline from the model code. Such an approach distinguishes us from the rest, makes our methodology more powerful and efficient. Source code parsing is an extremely powerful tool. We can find anything about the model, and update any node, block, code line or function. Since parsing provides the access to the original code, we can learn everything about the model structure and architecture. After parameters detection, we provide a complex transformation of the model code to be tuned in the Cloud.

To demonstrate the benefits of the method, we can say, that it is technically possible to transform the model to another framework, language or library, since event-based parser provides us the access to every single model line and detail. Automatic model transformation is the future of computer science and artificial intelligence. As we have demonstrated in [32], it is possible to convert from one programming language to another automatically.

Assorted parsing libraries are available to the research community. We found the most reliable and flexible one, supporting indentation, modern frameworks and Py-thon3. We found that the AST event parser (using node visitor pattern) is the most appropriate for our tasks. It provides an access to all the model constructs, including assignments, expressions, method calls, number parsing, or print statements. It is possible to transform a node implementing any desired behavior via

AST parsing and unparsing. We provide the following interfaces during model processing:

- collect all the variables and their assigned values
- find all the DNN model layers and its parameters
- replace hyperparameter values with the value placeholders compatible with the hyperopt library

High level programming languages, like Python, allow to implement the same thing in many different ways. For example, a parameter can be passed by value, by string, or by a variable holding an object. Our approach can handle:

- specifying the full or practical object classpath
- object creation as a parameter
- object reference as a parameter
- implicit object creation and assignment to a variable
- parameters passed by name or location
- functional model programming

```
1  from keras.models import Sequential
2  from keras.layers import Conv2D
3  from keras.layers import MaxPooling2D
4  from keras.layers import Flatten
5  from keras.layers import Dense
6  from keras.layers import ReLU
7  # Initialising the CNN
8  classifier = Sequential()
9  # Step 1 - Convolution
10 classifier.add(Conv2D(32, 3, 3, input_shape = (64, 64, 3), activa
11 # Step 2 - Pooling
12 classifier.add(MaxPooling2D(pool_size = (2, 2)))
13 activator = ReLU(negative_slope=0.1)
14 # Adding a second convolutional layer
15 classifier.add(Conv2D(32, 3, 3, activation = activator))
16 classifier.add(MaxPooling2D(pool_size = (2, 2)))
17 # Step 3 - Flattening
18 classifier.add(Flatten())
19 # Step 4 - Full connection
20 classifier.add(Dense(64, activation = keras.layers.ReLU(negative_
21 classifier.add(Dense(1, activation = 'sigmoid'))
22 # Compiling the CNN
23 classifier.compile(optimizer = 'adam', loss = 'bina-ry_crossentro
```

Fig. 1. Detecting model constructs for hyperparameters identification.

We can see in Figure 1 that ReLu function can be passed as a string (line 10), as a variable (line 15), and as inline initiation (line 20). Moreover, a class can be referenced with full path, partial path or just its name ReLu (line 13). Our methodology can recognize all these cases to detect model hyperparameters on the fly.

C. Identifying the hyperparameter space

Creating a space of hyperparameters is challenging, especially from a generic model code. We designed a robust approach to dynamically parse the model using the ASP parser and generate Python code containing all the model parameters and their values on the fly. Detecting parameters correctly is crucial, it affects the model performance and prediction results. Data scientists do not have to provide a single hint about the parameters, they just have to write the model, and our methodology will take care of the rest. The example of generated search space can be viewed at Figure 2.

The search space consists of a dictionary with keys – generated parameter names, indexes, and the value ranges. The values depend on the parameter type and nature. We targeted to handle generic cases with as many parameters as necessary and

our pipeline turned out to be working very well. For the record, we index all the hyperparameters, to distinguish between activation1 and activation11. The parameter candidate values will be passed to the hyperopt fmin method during the tuning phase of our pipeline.

Our parameters can be of the following types:

- hp.choice – enumeration of the values
- hp.uniform – specified every value in the range [min, max] equally likely
- hp.quniform – uniform for integer values

```
from hyperopt import hp
space = {
'filters0':hp.choice('filters0',
[32,64,128,256]),
'kernel_size2_0':hp.quniform('kernel_size2_0
', 2.0,4.0, 1),
'kernel_size2_1':hp.quniform('kernel_size2_1
', 2.0,4.0, 1),
   'activation2':hp.choice('activation2',
["softmax", "elu", "selu", "softplus",
"softsign", "relu", "tanh", "sigmoid",
"hard_sigmoid", "linear"]),
   'pool_size3':hp.quniform('pool_size3',
2.0,4.0, 1),
   'filters4':hp.choice('filters4',
[32,64,128,256]),
'kernel_size6_0':hp.quniform('kernel_size6_0
', 2.0,4.0, 1),
'kernel_size6_1':hp.quniform('kernel_size6_1
', 2.0,4.0, 1),
'pool_size8_0':hp.quniform('pool_size8_0',
2.0,4.0, 1),
'pool_size8_1':hp.quniform('pool_size8_1',
2.0,4.0, 1),
   'units8':hp.choice('units8', [64, 128,
256, 512]),
   'activation9':hp.choice('activation9',
["softmax", "elu", "selu", "softplus",
"softsign", "relu", "tanh", "sigmoid",
"hard_sigmoid", "linear"]),
   'units10':hp.choice('units10', [64, 128,
256, 512]),
   'activation11':hp.choice('activation11',
["softmax", "elu", "selu", "softplus",
"softsign", "relu", "tanh", "sigmoid",
"hard_sigmoid", "linear"]),
   'optimizer12':hp.choice('optimizer12',
["sgd", "rmsprop", "adadelta", "adagrad",
"adam", "adamax", "nadam"]),
   'epochs14':hp.choice('epochs14', [50,
100, 200])
}
```

Fig. 2. Generated space parameter for hyperopt.

The possible value range of hyperparameters is derived from the metadata file described in the previous section. Because our pipeline is code-based, the data scientists or researchers can always review the detected parameters and update them for any reason. They can delete unnecessary keys, change the default parameter ranges, or rename parameters.

D. *Updating the code of the original model*

The most important stage of our pipeline is generating model code. It will be executed during hyperparameter tuning. We are generating the transformed model code from the original one. We utilize the advantaged of the AST parser usage by updating statements or creating new functions when necessary. From the previous steps, we already have full access to the model parameters search space, the nature of model layers, and the collected variables.

We will be optimizing the model in the Cloud. During the AST parsing phase, we generate a method, which will be called by the framework multiple times to tune the hyperparameters. The method contains the model architecture code along with parameters and a dictionary of desired metrics. The framework can use these metrics in the future to select the next set of hyperparameters, and to analyze the results. The detected search space is integrated into the method, and all the hyperparameter values are passed with the argument "params".

Our AST parser stores function definitions and blocks of code as one tree element with children. Such a hierarchical structure helps us to group top-level statements together and place them into one function with domain space parameters. During this step, we generate a code calling the hyperopt fmin function, including all the necessary imports and creating the actual model runner function passed to fmin. Figure 3 demonstrates the code calling the hyperopt tuning function fmin with all the necessary parameters. The code is generated automatically and inserted into updated model.

```
from hyperopt import STATUS_OK, fmin, Trials
start = datetime.datetime.now()
trials = Trials()
MAX_EXECUTIONS=50
best = fmin(modelOptFunction, space,
algo=tpe.suggest, max_evals=MAX_EXECUTIONS,
trials=trials)
print ('best: ', best)
print ('trials: ', trials)
outParams(best, space)
end = datetime.datetime.now()
```

Fig. 3. Generated Call to fmin – tuning method of hyperopt.

E. *Running the updated model with the hyperopt library in the Cloud*

Running the model and getting the tuned hyperparameters is the ultimate goal of our research. All the previous steps of the pipeline were helping us to achieve our goal, calling the "fmin" method and optimize our model. To recall, hyperopt [13] is a state of the art Python library for hyperparameter optimization written by Bergstra et al. Research [30] confirms that it is still the best performing hyperparameter optimization library.

It is important to mention that executing a model could take a lot of time even in the Cloud, even on the most powerful multi-GPU instance. Pricing is another issue we try to optimize. Although AWS Cloud provides very powerful machine learning GPU instances, they cost a substantial amount of money (even the spot instances with seventy to ninety percent discount cost from $0.25 to $9.36 per hour).

The optimization process integrates the search space, state-of-the-art hyperparameter tuning algorithms and the Cloud ecosystem. The parameters include model tuning method, search space, tuning algorithm, and the number of evaluations. In fact, we implement dimensionality reduction, because we pass only five parameters to the optimization method "fmin". Out approach works for a potentially large search space containing hundreds of parameters.

F. Saving the results and sending notifications

After our model has been processed, tuned, and optimal hyperparameters have been detected, it is extremely important to save the results securely and reliably. The Cloud ecosystem provides multiple options to store results; we have chosen the most appropriate for our case – DynamoDB No-SQL database. We can achieve a millisecond latency for the database operations, providing scalability, reliability and concurrency. Moreover, since all the experiment results are located at one centralized place, we can always analyze them. It is also important that our data is stored securely and only authorized microservices can access it. We have implemented a state-of-the-art security mechanisms for cloud function to access required No-SQL tables and operations. We have been tailoring security policies for the Amazon's Identity and Access Management (IAM) service. It pro-vides a very granular list of JSON statements to specify the least privilege security paradigm.

G. Serverless Cloud integration

Serverless is a new revolutionary trend in Cloud Computing. It is becoming the mainstream for modern software applications. The granular microservice is deployed into the Cloud (as FAAS – Function as a Service), and the ecosystem takes care of container provision and termination, service discovery, scaling and load balancing. It is the best practice to utilize serverless architectures. Cloud function invocation can be automatically triggered by configured events. For our pipeline, the functions are triggered when model files are placed into a specified S3 bucket. As for pricing, we have to pay just twenty cents for one million function invocations.

Serverless technology is underrepresented in Machine Learning and our approach utilizes it. Serverless architectures help to automate the process, improve scalability and enforce zero idle server time, reducing overall cost. We have designed an integration of Machine Learning and Serverless Architectures to bridge the gap between the Cloud engineering and AI, providing flawless orchestration, administration, instance invocation and monitoring at scale. Cloud functions give us control and implement low level networking, security and cloud services integration [28]. Designing efficient Cloud applications require a lot of wisdom, creativity and software engineering design work.

As Peter Sbarski said in [27], "Serverless architectures are the latest advance for developers and organizations to think about, study, and adopt. This exciting new shift in architecture will grow quickly as software developers embrace compute services such as AWS Lambda. And, in many cases, serverless applications will be cheaper to run and faster to implement".

Our methodology solves multiple important Machine Learning, Security and Architecture problems:

• Results storage and aggregation – all results of our experiments will be stored at the same place, so we can incrementally analyze and graph them

• Scalability – running multiple models in parallel on different machines

• GPU Instance procurement – there is no need to have any local powerful computer. All instances will be created on the fly and terminated once our experiments are done

• The optimization task will be triggered automatically once the model code is placed in the S3 cloud bucket; the results will be saved into the Cloud and a notification (text message or email) will be sent to the author upon completion of the model tuning

Building the state-of-the art methodology is our goal. We help the data scientist community to improve their models. Our approach will help them to run multiple models in parallel, potentially with different datasets, utilizing multiple GPU instances. Figure 4 demonstrates the architecture of our approach. It is using the following AWS services – S3, AWS Lambda, DynamoDB, IAM, CloudFormation, EC2 spot instances, SES, and SNS.

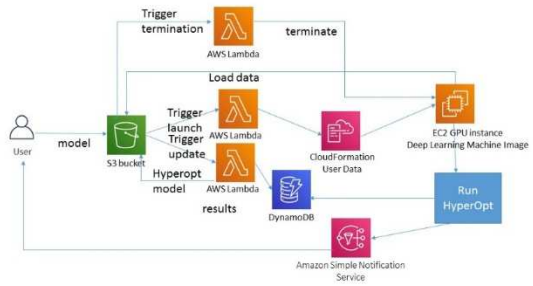

Fig. 4. Methodology Pipeline Architectural diagram.

Our methodology is triggered by placing an original Python Model code into the S3 bucket. The first Lambda serverless functions will parse the model, extract hyperparameters and dynamically create the updated Model code that will be suitable to run with hyperopt tuning. The updated model code is placed into the specified S3 bucket (Cloud Simple Storage). It will trigger the second Lambda function invocation.

The second Serverless Lambda function spawns a CloudFormation template to create an EC2 spot instance that will be used to run our experiments. For the machine image we will be using one of the preconfigured UNIX deep learning Machine Image, containing most of the Python data science libraries. We will pass User Data – a script used to initiate an instance – to install the missing libraries, copy the model code and data into the instance from S3, and run our python updated model code.

Fast GPU computers are not always ready or available for Data Scientists. That is why Cloud integration helps us to spawn an instance as needed, and terminate it upon completion of our experiments. To save instance cost, we can invoke spot in-stances from CloudFormation, saving up to 90% over on-demand price. CloudFormation templates help us to automate

the whole Cloud ecosystem infrastructure creation and orchestration with a single yaml script.

The code generated by our updater, will execute hyperopt hyperparameter tuning, output results into the DynamoDB database and notify the user via SNS notification service. After the results are saved, all the resources will be terminated providing zero idle time. The cloud DynamoDB database stores all of our results for further analysis. Serverless function invocations are asynchronous. Multiple models can run at the same time in the Cloud providing efficiency, scalability and reliability.

H. Representative Model

We would like to present a generalized model architecture example. Deep Learning models contain multiple layers. For Convolutional Neural Networks (CNN) we have several convolutional layers, followed by pooling layers. This architecture reduces dimensionality because previous layers "condense" the information, reducing the number of parameters and speeding up model training. Figure 5 demonstrates a typical architecture of a convolutional neural network. It contains five convolutional layers, each followed by a ReLU activation function and a max pooling layer. ReLU is an activation function that increases non-linearity in the input. It is a piecewise linear function, that outputs zero for negative inputs and the value for positive inputs. It can be defined as a positive value of its argument, or max(0, x).

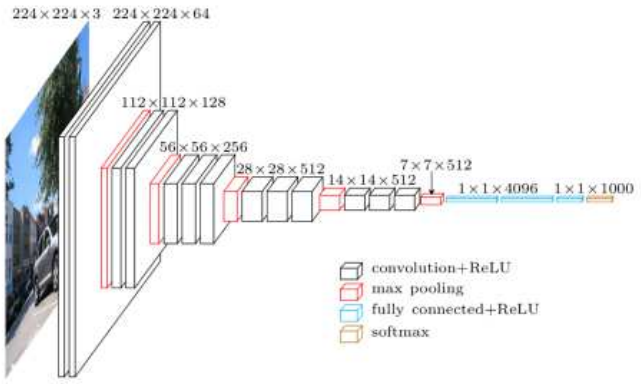

Fig. 5. VGG16 Convolutional Neural Network Architecture.

Max pooling does the actual dimensionality reduction by picking the maximum number in the rectangular filter. For example, pooling layer can reduce input of size 4x4 to the size of 2x2. Besides max pooling, we can use average, global max and global average layers. A researcher can use a pooling flavor depending on the nature of the problem, architecture of the network and use case. The global functions pick the maximum or average values for the whole filter, not just for the pool size rectangle.

After a combination of convolutions, we need to flatten the results and add one or more fully connected (Dense) layers to the model. The final layer needs to use the softmax activation function for multi-label classification, and the sigmoid function for binary classification. It is important to mention that the last fully connected layer should have the number of units equal to the total number of classification labels.

TABLE I. HYPERPARAMETER EXAMPLES FOR LAYERS OF THE REPRESENTATIVE MODEL

Layer	**Hyperparameters**
Conv2D (convolution)	Filters, kernel_size, strides, activation
MaxPooling2D (pooling)	Pool_size
Dense (fully connected)	Units, Activation, kernel_initializer, bias_initializer
Dropout	rate

Dropout layers are used to prevent overfitting. They randomly set input weights to zero with a specified frequency during training time. Dropout layers can be placed between any other layers. Each layer of the representative model has a set of hyperparameters affecting its performance. Table I contains examples of hyperparameters for the representative model architecture.

III. RESULTS

During our research, we were able to achieve impressive results. We have executed our approach on many models. As we want to handle a generic neural network, we have been testing our approach on diverse model architectures. We have covered convolutional, deep, recurrent, long short-term memory and many other models. We have utilized the most popular data science datasets, including MNIST, cifar, cats/dogs, and blockchain.

TABLE II. SELECTED RESULTS

Model	**Layers**	**Network Type/ # Params**	**Code Lines**	**Exec Time (hour)**	**Loss**	**Data Size**
Urban noises	5	DNN 8	200	1	.02	4G
Bitcoin Forecast	4	DNN 8	190	50	.01	.25G
Cat/dog	11	CNN 6	65	64	.003	.5G
MNIST	6	CNN 7	90	8	.02	45MB
Translate	5	RNN 6	300	10	.029	20 MB
Cifar100	7	CNN 38	158	47	1.2e-7	150MB

Table II demonstrates our results for the most commonly used model architectures. As you can see, our methodology was able to achieve very low loss for all the experiments. It is especially impressive because the results were generated automatically from the model source code. No human interaction or manual steps were involved to produce optimized

models with accuracy of up to 97%. Such models will be very good candidates to solve the most challenging security problems in classification, regression, clustering and dimensionality reduction.

We were able to integrate our methodology with the Cloud ecosystem and server-less architectures, boosting productivity and providing almost infinite scalability to model tuning. In the future, we would like to run our tool on a multitude of machine learning models to get more generic results for arbitrary models created by diverse researchers.

IV. Scientific Challenges Addressed

Our methodology pipeline addresses the following challenges:

• Maintenance, Procurement and Cost of the hardware used to tune models. There is no need to acquire expensive GPU machines and configure them. Our methodology starts all the necessary instances and terminates them when done. Moreover, we utilize cloud spot instances at the 70-90% discount prices.
• Eliminating manual steps during Machine Learning model hyperparameter optimization.
Automation is a very important feature needed by the Data Science community. Our approach provides better model tuning.
• Concurrent Model Tuning and Scalability.
Unlike in-house hardware, our methodology spawns all the necessary cloud functions or spot instances to conduct all the requested experiment. There is no limit to the concurrent models because each server and Lambda function is independent.
• Fault Tolerance and Reliability.
The Cloud integration provides a range of tools monitoring the services, retrying attempts if some call failed and notifying the stakeholders about events of interest.
• Cloud Adaption for Machine Learning.
Our approach helps Data Scientists to utilize the Cloud faster, easier and cheaper. We do not require any configuration or specific cloud knowledge to use our methodology.
• Security.
We integrate hyperparameter optimization with state-of-the-art cloud ecosystem, tailoring to the least privileged principle. Every cloud function has a role with security policies required for the microservice to run. We provide only necessary access, and our functions cannot implement non-permitted operations, such as removing metadata or models from S3 buckets, or deleting results from the No-SQL DynamoDB database.

V. Conclusion and Future Work

Our research summarizes the previous work, including the existing optimization libraries using Bayesian optimization and other statistical methods. We were able to confirm that our optimization methodology is systematic and works for a generic model. It opens new opportunities to optimize Deep Neural Networks. Our approach will help Data Scientists to discover better models faster, design new algorithms and improve machine learning practices and new architectures. We provide an example of an efficient cloud architecture and interoperability for Machine Learning tuning. There will be more services and models improving our approach even further.

The current state-of-the art Machine Learning technology does not support generalized models and requires human interaction for hyperparameter tuning. We have added automation, interoperability, minimized costs, and provided real-time system triggering for generalized models. Given such an imminent need, we provide a service to the entire Data Science community. Our approach is bridging the gap that currently exists between the tuning methodology and the researchers' needs.

During our research, we were able to prove and verify that our approach works well and that it is possible to automatically extract hyperparameters to optimize them in the serverless cloud. We were able to successfully parse any given model and generate the code for the hyperparameter optimization. It has been confirmed that our generated code runs successfully and tunes the model hyperparameters. As numerous research papers [30] suggest, hyperopt provides the best results for model tuning and our platform utilizes it.

Nowadays, Cloud Computing is an integral part of software systems. Using the Cloud efficiently helped our methodology to become scalable, reliable, automatic and resilient. The modern cloud ecosystem provides almost limitless resources for computing and storage. Most of the services are self-managing and auto-scaling. They also provide constant monitoring and event-based automatic invocation. Using Serverless Lambda triggering, Single Storage Service, No-SQL DynamoDB, and CloudWatch event scheduling helped us to make our platform reliable and automatic. One of the biggest values of the Cloud is that all the services are integrated with each other. They can easily communicate and exchange data securely. Our platform helps cloud adoption in Machine Learning by utilizing innovative services and saving money by using spot instances and serverless functions. Our platform saves all the results safely and securely in DynamoDB No-SQL database to keep track of the experiments. We can query the database to review and analyze the results and draw graphs or diagrams.

In the future we would like to test our service on many more models, create verification metrics and collect the results in the Cloud for further analysis. A No-SQL database will be the best candidate for results storage. We also might want to add more parameters to our tool to make it more flexible and feature rich.

We would also like to extend our methodology to other frameworks like pytorch or MXNet, and languages such as Java, R, or Scala, to create Machine Learning models. It is just the beginning because the AI industry needs automation badly. Even nowadays, many machine learning steps are manual.

Although our research was performed in the Amazon Cloud, it can be easily ex-tended to other cloud providers like Google, Microsoft and IBM, because all of them implement serverless cloud functions, storage and databases. It is possible to create a generic solution which works for any cloud provider. We can call it an interoperable Cloud-agnostic Deep Learning tuning and optimization methodology, general tool that is independent from a specific cloud provider.

Our method gives Data Scientists the power to create their own Python model using their experience, techniques and domain knowledge. Unlike several other machine learning automation tools like AWS's SageMaker, Machine Learning services, or Google's AutoML, we do not oversimplify model creation and allow data scientists and researchers to write their own code. Cloud integration adds scalability, reliability and serverless integration to our system. One of the biggest advantages of our approach is that we take any model code, allowing the experts to implement the algorithms of their choice. This relives them from the manual, lengthy and tedious processes of parameter optimization, hardware integration and configuration.

In the very near future there will be vast adaption of quantum computing. Then quantum Machine Learning and AI will become even more impactful solving very difficult problems.

Our work [33] impacts the Data Science community because researchers can use our methodology to get better performing models automatically, while concentrating on the creative work of algorithm and architecture implementation. Innovation is always motivating the Machine Learning scientists to solve modern problems faster with superb precision. It opens new vistas to the research community due to the fact that it can capitalize our research, propelling data science research to new horizons, and solving other synergistic problems. Many existing ML approaches can use our methodology to address automation demand for hyperparameter optimization with-out manual steps.

Automatic hyperparameter optimization is vital for the whole Machine Learning industry, and our methodology is definitely helping data scientists to optimize their models and improve application performance. During our Novel Coronavirus challenging times, we can tune models in all the areas, especially the ones related to Information and Communication Systems. Nowadays, during COVID-19 pandemic, well per-forming models are especially important, because their predictions might affect lives of millions of people.

REFERENCES

[1] James Bergstra et.al. "Random Search for Hyper-Parameter Optimization", Journal of Machine Learning Research 13, 2012

[2] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization"

[3] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, Balázs Kégl, "Algorithms for Hyper-Parameter Optimization"

[4] X.C.Guoab,J.H.Yanga,C.G.Wuac,C.Y.Wanga,Y.C.Lianga, "A novel LS-SVMs hyper-parameter selection based on particle swarm optimization"

[5] Nitish Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

[6] G. Cybenko "Approximation by superpositions of a sigmoidal function"

[7] Bergstra, J., Yamins, D., Cox, D. D. (2013) Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. To appear in Proc. of the 30th International Conference on Machine Learning (ICML 2013)

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun,"Deep Residual Learning for Image Recognition", The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-77

[9] Francis Bach "Breaking the Curse of Dimensionality with Convex Neural Networks", 4/2017, Journal of Machine Learning Research 18 (2017) 1-53.

[10] About Keras models, Keras online documentation https://keras.io/models/about-keras-models/

[11] Metz, Cade "TensorFlow, Google's Open Source AI, Points to a Fast-Changing Hardware World", November, 2015

[12] Martín Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning", November 2016, Proceedings of the 12th USENIX Symposium

[13] James Bergstra, Dan Yamins, David D. Cox "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms", THE 12th PYTHON IN SCIENCE CONF, (SCIPY 2013)

[14] C.E. Rasmussen and C. Williams. Gaussian Processes for Machine Learning

[15] J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. In L.C.W. Dixon and G.P. Szego, editors, Towards Global Optimization, volume 2, pages 117–129. North Holland, New York, 1978

[16] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In ICML 2007, pages 473–480, 2007

[17] C. Bishop. Neural networks for pattern recognition. 1995

[18] Fabian Pedregosa et al.,"Scikit-learn: Machine learning in Python", Journal of machine learning research, volume 12, October 2011

[19] "Tuning the hyper-parameters of an estimator", scikit-learn documentation, https://scikit-learn.org/stable/modules/grid_search.html

[20] RandomizedSearchCV documentation, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html#sklearn.model_selection.RandomizedSearchCV

[21] skopt module documentation, https://scikit-optimize.github.io/

[22] Jonas Mockus: On Bayesian Methods for Seeking the Extremum. Optimization Techniques 1974: 400-404

[23] J. Mockus, V. Tiesis, and A. Zilinskas. "The application of Bayesian methods for seeking the extremum.", In L.C.W. Dixon and G.P. Szego, editors, Towards Global Optimization, volume 2, pages 117–129. North Holland, New York, 1978.

[24] Github hyperopt source and documentation, http://hyperopt.github.io/hyperopt/ and https://github.com/hyperopt/hyperopt/wiki/FMin

[25] C. E. Rasmussen and C. K. I. Williams. Gaussian Processes for Machine Learning. MIT Press, 2006.

[26] Kyle Gorman, Steven Bedrick, We need to talk about standard splits, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), August, 2019

[27] Peter Sbarski, "Serverless Architectures on AWS", 2017, Manning, ISBN 9781617293825

[28] Jonas, Eric, et al. "Cloud programming simplified: A berkeley view on serverless computing." arXiv preprint arXiv:1902.03383 (2019).

[29] Baldini, Ioana, et al. "Serverless computing: Current trends and open problems." Research Advances in Cloud Computing. Springer, Singapore, 2017. 1-20.

[30] Putatunda, Sayan, and Kiran Rama. "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost." Proceedings of the 2018 International Conference on Signal Processing and Machine Learning. 2018.

[31] Dosovitskiy, Alexey, et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv preprint arXiv:2010.11929 (2020).

[32] Kaplunovich, Alex. "ToLambda--Automatic Path to Serverless Architectures." *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*. IEEE, 2019.

[33] Kaplunovich, Alexander. *Real-Time Automatic Hyperparameter Tuning for Deep Learning in Serverless Cloud*. Diss. University of Maryland, Baltimore County, 2020.