Literature Review on Automated Machine Learning

Abstract

Machine Learning plays a pivotal role in our lives today. The extended influence on fields such as healthcare, security and finance are testament to the increasing awareness of its applications. Yet, the knowledge and labour-intensive nature of developing these models proved an obstacle to the inexperienced. Even for experts, these processes may require long periods of development which has prompted for automation. Automated machine learning (AutoML) aims to reduce the reliance of human intervention by automatically building machine learning applications without extensive knowledge of statistics and machine learning. This review will introduce the AutoML problem and survey on current AutoML methods.

1. Introduction

According to No Free Lunch Theorem (D. H. Wolpert, 1997), there are no algorithms that can achieve good performance on all learning problems with equal importance. Coupled with the sensitivity of some models to hyperparameters tuning such as SVM, define the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem.

Automation aims to balance the trade-off between model's performance and generalization across all tasks. Specifically, AutoML aims to fulfils certain functions:

1) Robust to all datasets and achieves decent performance
2) Less assistance from humans requiring less domain knowledge
3) High computational efficiency by returning outputs within limited budget
4) Deal with different types of problems such as supervised and unsupervised learning

AutoML can be applied to various aspect of the machine learning pipeline, namely Feature Engineering, Model Selection, Architecture Search and Hyperparameter Optimization. This is demonstrated by the general machine learning pipeline as shown in Figure 1. These approaches are not unprecedented as many applications such as Auto-sklearn and Auto-Weka has attempted to automate some of these aspects. The rest of this review introduces different strategies implemented for these areas.

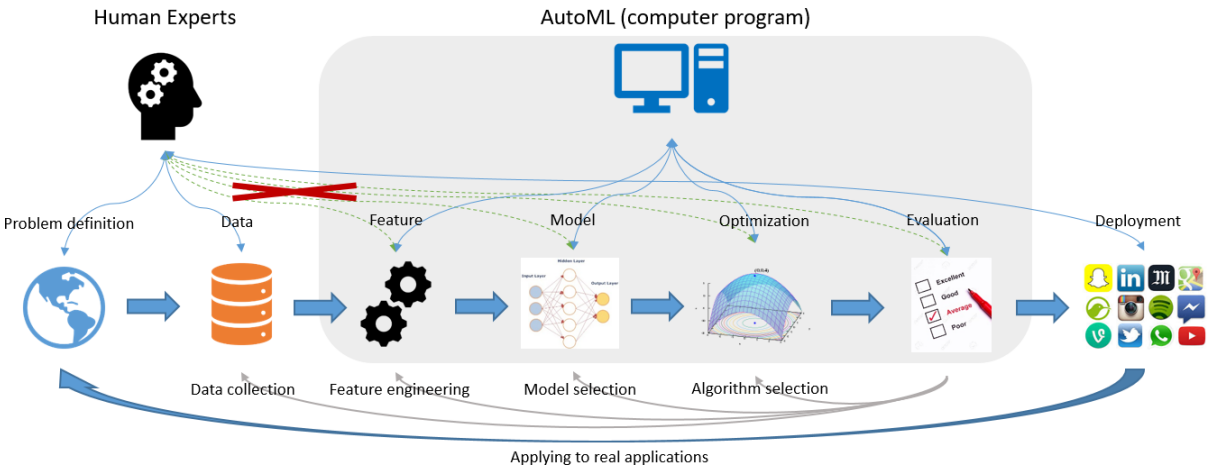

Figure 1: Demonstrate the areas in which AutoML can be implemented in a regular pipeline. (Yao, Wang, Chen, Dai, Li, Tu, Qiang, Yang, 2018)

2. Feature Engineering

The quality of features is a prerequisite to many machine learning tasks. However, current approaches require domain knowledge and AutoML aims to remove the need for such human intervention. The problem can be broken down into 3 sub-problems, feature generation, feature extraction and feature selection.

2.1 Feature Generation

Generation of effective features work on the intuition that highly informative features result from manipulations of elementary ones by transforming each feature individually or combine several of them together. This enhances the robustness of the model and increase the representative ability of original features.

In particular, ExploreKit (Gilad Katz, 2016) employs transformation operations to the original features to generate new features. ExploreKit divided them into 3 sub-operations, unary, binary and higher order.

Unary operators are operations applied on single features. Some examples include discretizers which converts continuous and datetime features to discrete values. For instance, numeric features can be discretised by partitioning them into equal ranges. These conversions are necessary in some classification algorithms such as Decision Trees and has also been shown to contribute to performance. Normalizers also helps by fitting continuous features into specific distributions. These unary operations provide us with transformations of continuous features which is the basis for higher–order operators.

Binary operators are operations applied on a pair of features. Basic arithmetic operations which include “+”, “-“, “x” are used to combine the original features to construct new ones.

For higher order operations, multiple features are utilised to generate new features. This includes grouping of multiple features and further applying statistical means to further enhance their discriminative ability.

On the other hand, Deep Feature Synthesis (James Max Kanter, 2017) has explored the possibility on generating features from relational datasets. In short, entity features are first generated by considering the features and their values in the table corresponding to the entity. Next, relational features are derived by jointly analysing two related entities via both forward and backward relationships. The features are then recursively generated across entities until the terminating condition is met.

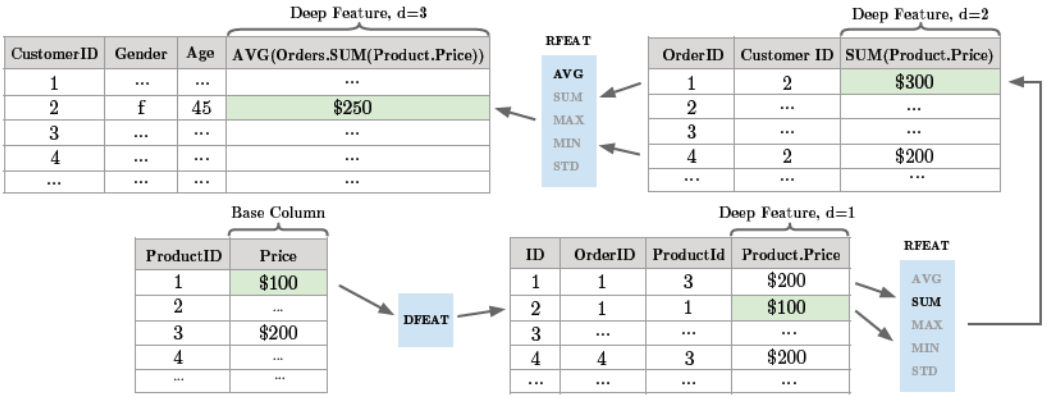

Figure 2: Example for feature generation in Deep Feature Synthesis

2.2 Feature Extraction

Feature extraction is a dimensionality reduction process through some mapping functions, which extracts informative and non-redundant features according to some specific metrics. This ensures that the features are discriminable in the feature space. While there are many known approaches such as Principle Component Analysis (PCA), Linear Discriminant Analysis (LDA), autoencoders and feed-forward neural networks which uses the hidden units of a pretrained model as extracted features, there seemed to be lesser research on the automation of such extraction.

2.3 Feature Selection

Feature Selection aims to simplify the model to avoid overfitting to ensure robust performance across datasets. This is done through reducing irrelevant features which often exhibits high correlation. Other than statistical tests, popular approaches including hierarchical rankings of features has been explored by various researches.

Deep Feature Synthesis (James Max Kanter, 2017) employs 2 techniques sequentially by first using truncated SVD transformation on generated features and select n components of the SVD. Next, they attempt to rank each SVD-feature by calculating its f-value w.r.t to the target value and select the x% highest ranking features.

ExploreKit employ similar hierarchical approach by training a ranking classifier to assign a score to each candidate feature. This classifier is trained using meta-features, both of the datasets and the features themselves. The former can be seen as characteristics of the dataset which may affect the effectiveness of features. In ExploreKit, some of these meta-features used are the general statistics on the dataset (number of features, size of datasets), initial evaluation (metric performance of the ranking classifier), entropy-based (Information Gain of the features in each subgroup) and feature diversity (tests' statistic from similarity tests). The latter represents the interactions of analysed features with other candidate features. The meta-features include entropy and statistical tests, general information on parent features and operators. A wrapper feature evaluation is then used to determine the error reduction obtained by applying the classifier on the joint set of meta-features to assign labels to each candidate feature and thereafter their meta-features. The labelled set is used to train the ranking classifier via supervised learning.

Mentioned above, Deep Feature Synthesis and ExploreKit has successfully automate some aspects of feature engineering into their systems. The evaluation of these approaches and their applications can be seen in Figure 3.

Researches	Advantages	Applications
ExploreKit	Human-understandable operations Do not require entity relations Leverage experience from previous datasets	Supervised Classification

Deep Feature Synthesis	Success-to-effort ratio is low Leveraging on relationships in relational data End-to-end system with other focus on hyperparameters tuning Requires for human intervention by specifying the relationships in advance	Supervised Classification

Figure 3: Evaluation of current automated feature engineering methods

2.4 Auto-Augmentation

Data augmentation can significantly improve the generalization ability of deep learning models. While previous attempts to ensure robustness hardcodes invariances into the model architecture directly, data augmentation proved to be easier. However, despite augmentation improvements on a particular dataset, they often do not transfer to other datasets as effectively. Auto-Augmentation aims to automatically search for improved data augmentation policies to ensure transferability and generalization.

AutoAugment (Ekin D. Cubuk, 2018) aims to formulate the problem of finding the best augmentation policy as a discrete search problem where policy consists of many sub policies. Each sub-policy consists of two operations: an image processing function such as translation, rotation and the probabilities and magnitudes with which the functions are applied. The search algorithm has two components: a controller, which is a recurrent neural network (RNN), and a training algorithm. The RNN samples a data augmentation policy which is used to train a neural network with a fixed architecture (Figure 4). Validation accuracy on the child network is trained to convergence and sent back to update the controller. At the end of the search, the best 5 policies are concatenated into a single policy.

In recent years, variants of AutoAugment has surfaced, namely the Fast AutoAugment (Sungbin Lim, 2019) and RandAugment (Ekin D. Cubuk B. Z., 2019). The former recommends a more efficient search strategy based on density matching as it does not require backpropagation for network training for each policy evaluation. It also aims to improve the generalization performance by learning the augmentation policies which treat augmented data

as missing data points of training data and recover them using Bayesian Optimization in the policy search phase. RandAugment focus on a unified optimization of the model weights and data augmentation policy. To support the parameter-free procedure, a fixed magnitude schedule is used instead of searching over optimal magnitudes for each transformation and by always selecting a transformation with uniform probability. Henceforth, the reduced search space allows training on target task with no need for a separate proxy task which assumes that proxy task provides a predictive indication of the larger task.

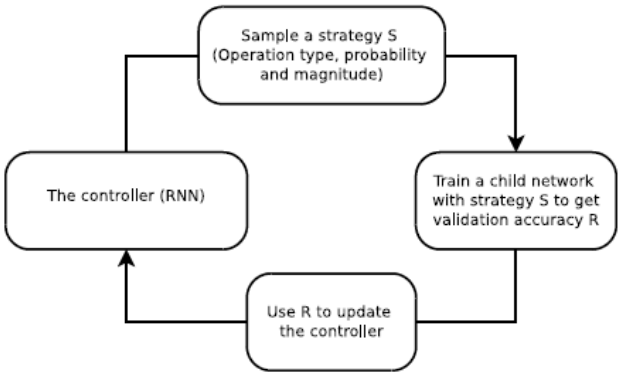

Figure 4: Framework of using AutoAugment's search method (Reinforcement Learning)

Specifically, for the field of object detection (Barret Zoph E. D.-Y., 2019), researches have applied auto-augmentation using similar algorithms to AutoAugment with the addition a set of simple transformations that may be applied to the bounding boxes. The evaluation of the algorithms is reflected in Figure 5.

Researches	Advantages	Applications
AutoAugment	Transferable learned policies Accuracy improves with more sub-policies	Image Classification
Fast AutoAugment	Improved efficiency and generalization performance Does not train model parameter from scratch again	Image Classification

RandAugment	Interpretable hyperparameters with only 2 additional parameters Reduced search space allows training on target task with no need for a separate proxy task Regularization strength may be tailored to different model and dataset sizes due to the implementation of parameterization Possible to use optimisation like naive grid search to finetune the hyperparameters	Image Classification
Augmentation for OD	Transferable learned policies onto object detection tasks	Object Detection

Figure 5: Evaluation of current automated augmentation methods

3. Model Selection

Selection of model, specifically the neural architecture has traditionally been designed by humans and has an influence on the model's performance. There exists a trade-off between performance and computational resources such that over-parameterized models often produce good performance though consuming huge resources. To overcome this, many methods attempted to use shared parameters by accumulating gradients for all networks and update the parameters via stochastic gradient descent. Researches has also focus on multi-objective NAS, where non-dominating sorting are used to consider more objectives such as minimizing training time and improving accuracy. This also protects the larger models which tends to be dominated by smaller models if multi-objective is not considered. Common approaches to the problem include gradient-based learning, evolutionary algorithms and reinforcement learning. Figure 6 provides the evaluation on these approaches.

3.1 Gradient Based

Gradient based algorithms involve network parameter optimization and architecture optimization. The former optimizes the parameters in the standard layers (i.e., convolution, batch normalization, fully connected layer) while latter learns the pattern of accurate network architectures. It is a commonly used approach due to its differentiable nature and can be manifested in Neural Architecture Optimization (NAO) (Renqian Luo, 2018).

Existing methods conduct architecture search in discrete space is highly inefficient due to the exponentially growing search space when number of choices increases. NAO suggest embedding neural network architecture into a continuous vector space via an encoder. This encoder is a single layer LSTM with the hidden states being used as the continuous representation of the architecture. A predictor then takes the output and predicts its accuracy by building a regression model to approximate the final performance of an architecture. Performance predictor and encoder hence allow us to perform gradient based optimization in the continuous space to find the embedding of a new architecture with better accuracy. A decoder then maps the continuous representation of the network back to its architecture where the decoder is also an LSTM model with attention mechanism to make exact recovery easier. The three components are jointly trained in a multitask setting which is beneficial to continuous representation. Specifically, starting from an architecture, better continuous representation is obtained by moving the embedding along the gradient direction induced by the predictor.

3.2 Evolutionary Based

The basis of evolutionary algorithms is to initialize a set of models and evolve for better architectures. The generation of offspring via crossover and mutation maintains a set of well-performed architectures that cover a vast space. However, they are usually time consuming and search cost is much more expensive due to the evaluation procedure of each individual.

This algorithm is manifested in the paper on continuous evolution (Zhaohui Yang, 2019) where individuals in the population represent architectures derived in the SuperNet which is initialized with cells and blocks. The individuals are generated through several benchmark operations such as crossover and mutation. In correspondence to the multi-objective approach presented, non-dominated sort strategy is adopted to select architectures with different model sizes and accuracies during the architecture optimization step. Their corresponding cells in the SuperNet are then updated for subsequent optimization. Evolution procedure in the next generation continuously execute based on the updated SuperNet. At the last generation, the

architectures will be tuned having previously trained based on shared parameters which reduces the computational complexity of separately training these different architectures.

3.3 Reinforcement Learning

Reinforcement learning stems from the observation that the structure and connectivity of a neural network can be specified by a variable-length string using a recurrent network (Barret Zoph, 2016). It makes use of a recurrent network as a controller to predict sequence of operations and generate different architectures hence designing a search space that decouples the complexity of an architecture. Every prediction is carried out by a SoftMax classifier and then fed into the next time step as input. The list of tokens that the controller predicts can be viewed as a list of actions to design an architecture for a child network. Once the controller RNN finishes generating an architecture, a neural network with this architecture is built and trained. At convergence, the accuracy of the network on a held-out validation set is recorded and the parameters of the controller RNN are then optimized to maximize the expected validation accuracy of the proposed architectures. This accuracy is used as the reward signal and backpropagate to train the controller. Skip connections or branching layers can be used to widen the search space by decide the layers as inputs to current layer. However, given an architecture, reinforcement learning requires to train it for a large number of epochs and evaluate its performance to optimize the controller, making the searching stage less efficient.

3.4 Semi-Supervised

Neural architecture search relies on a good controller to generate better architectures or predict the accuracy of given architectures. However, training them requires abundant pairs of architectures and their accuracy which is costly to evaluate. Hence, there exists a trade-off between number of architecture-accuracy pairs and training time of each candidate architecture. SemiNAS (Renqian Luo X. T.-Y., 2020) aims to leverage on numerous unlabelled architectures to improve the controller without evaluation and thus nearly no cost. This can be applied to many NAS algorithms and the research took NAO as basis for implementation. First, an initial controller is trained with a small set of architecture-accuracy data pairs which is then used to predict the accuracy of more randomly generated architectures without the need for evaluation. Unlabelled architectures can be obtained through random generation and mutation as in NAO. Lastly, the generated data pairs are added to the original data to further improve the controller via gradient optimization.

Algorithms	Specifications	Applications
Gradient Based	Differentiable hence more efficient (+) Searched architectures suffer from lack of variety (-)	Text to speech Image Classification
Evolutionary	Maintains a set of well-performed architectures that cover a vast space (+) Maximally utilizes the learned knowledge in the latest evolution generation, such as architectures (+) Search cost is much more expensive due to the evaluation procedure of each individual (-)	Image Classification
Reinforcement Learning	Cell determine by controller can easily generalize to different task (+) Enables transferability as the complexity of the architecture is independent of the depth of the network and the size of input images (+) Need to train for large number of epochs and evaluate its performance to optimize controller (-)	Image Classification Character Language modelling task Emulate Simulations

Figure 6: Evaluation of current neural architecture search algorithms

4. Hyperparameter Optimisation

Machine learning models have become more complexed testament by the growing number of tuning parameters associated with them. These hyperparameters helps to generalizes to new, unseen data and must be properly tuned to maximise models' performance. However, the increased complexity has made computational costs an obstacle. Automation to find the

optimal hyperparameters in the vast search space has thus been explored with various success. The evaluation of some of the popular approaches can be found in Figure 7.

4.1 Random Search

Together with grid search, random search is a naive search approach making no assumptions about the search space. Random search (James Bergstra, 2012) is capable of optimizing high-dimensional, non-convex function on the basis of working on low effective dimensionality. Grid search meanwhile suffers from curse of dimensionality as the number of joint values grows exponentially with the number of hyperparameters. Despite, the rate of convergence depends on the smoothness and is exponential in the number of dimensions in the search space.

4.2 Bayesian Optimization

Bayesian optimization (Jasper Snoek, 2012) works by assuming the unknown continuous function was sampled from a prior distribution and maintains a posterior distribution for this function as observations are made. A probabilistic model is then constructed by using information available from previous evaluations to make decisions about the next data point to evaluate. Hence, it aims to optimise functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try. Bayesian Optimisation consist of 2 subparts: a prior distribution and the kernel as well as an acquisition function.

4.2.1 Prior Distribution and Kernel

Prior distribution expresses assumptions about the function being optimized. Bayesian Optimisation is often associated with Gaussian Processes to model the learning algorithm's generalization performance due to its tractable posterior distribution and flexibility. The distribution is represented by a mean and covariance function which is govern by the kernel selection. However, the unclear choice of kernels proved to be a challenge for most attempts.

4.2.2 Acquisition Function

Acquisition function constructs a utility function from the model posterior, allowing us to determine the next point to evaluate. Commonly used criterions are Expected Improvements (EI), Probability of Improvement (PI) and Upper Confidence Bound (UCB).

4.3 Hyperband

Hyperband (Lisha Li, 2016) works on the basis of allocating more resources to promising hyperparameter configurations while quickly eliminating poor ones. It achieves by speeding

up random search via adaptive resource allocation and early stopping. Common approaches for elimination utilize successive halving, allocating a budget to a set of hyperparameter configurations, evaluate the performance of all configurations, throw out the worst half, and repeat until one configuration remains. However, there is a trade-off to make between the number of configurations to try and how quickly 2 configurations differentiate themselves. Hyperband first perform grid search for different s where the larger s is, the more exploratory it is, implying less resources allocated to each model, requiring it to terminate earlier. The hyperparameters are sampled via a random search in the hyperparameter space. Successive halving is then implemented to remove the inferior models, allocating the excess resources to the superior models. When the last model is left after iterations of eliminations, it is then compared with other brackets of s for the optimal model which is where the balance in the exploration and exploitation trade-off problem can be found.

4.4 Population Based Training (PBT)

PBT (Max Jaderberg, 2017) aims to optimise both the parameters of the model theta and the hyperparameters h jointly while leveraging on information sharing across a population of workers concurrently running optimisation. It employs gradient-based learning to train the workers to do partial evaluation and using the collection of partial solutions in the population to additionally perform meta-optimisation. When ready for evaluation the workers are then chosen to be exploited or continue exploring by T-test selection or truncation selection. Exploiting will mean abandoning the current solution and focus on a more promising one while exploring will either randomly perturb the hyperparameters with noise or resampling hyperparameters from the originally defined prior distribution. At the end of the iterations, the better models will be left in the population.

4.4.1 Automated Stopping

It is important to terminate evaluation early for some problems where computation cost is high, to free resources for more promising hyperparameters. Some metrics to determine if certain parameter setting is unpromising well before evaluation is finished will be discussed. Performance Curve Stopping Rule (Daniel Golovin, 2017) requires performing regression on the performance curves and make a prediction of the final objective value of an evaluation via training on past evaluations or measurements taken during current evaluation. If the probability of exceeding the optimal value found thus far is sufficiently low, early stopping is requested. Median Stopping Rule stops a pending evaluation at step s if the best objective value is strictly

worse than the median value of the running averages of all completed evaluation at the same step. Other methods included using Bayesian parametric regression with a carefully designed kernel that measures similarity between performance curves can be used.

4.5 Transfer Learning

In cases when number of trials per study is relatively small, but there are many of such studies, it is useful to leverage on data from prior studies to guide and accelerate current study. Building a larger gaussian process regressor that is trained on both the prior and the current study however is way too expensive. In a regression problem, transfer learning (Daniel Golovin, 2017) can do so by stacking gaussian process regressors, where each regressor is built using the data from the associated study and regresses on the residual of the objective with respect to the predictions of the regressor below it. A non-linear regressor is then fitted to approximate the residuals with respect to the stacked regressor's prediction given test data. The weighted sum of both stacked and residual regressor will give the prediction.

Researches	Specifications	Tested on
Random Search	Can easily be parallelised (+) Low effective dimensionality (+) Can optimize high-dimensional, non-convex function with unknown smoothness (+) Runs can be either abandoned or restarted without jeopardizing the experiment (+) Although requires time for one training run, when run in parallel requires the use of more computational resources (-)	Neural Networks Works for most
Bayesian Optimisation	More useful for lower dimensional problem (+) Result of previous run is used as knowledge to determine the next point (+)	Latent Dirichlet Allocation Structured SVMs

	Function evaluation may involve time-consuming optimization procedure since model is retrained from scratch (-) Appropriate choice for the covariance function/ kernel is needed (-) Hard to parallelised, requires information from previous runs (-)	Convolutional Neural Network
Hyperband	Makes minimal assumptions (+) Requires a budget that is only log factors larger than that of Successive Halving (+) Leverages down sampling to boost the number of configurations that are evaluated (+) Suited for high dimensional problems where more evaluations are needed (+) Hyperparameters which need to be tuned (-)	Deep Learning Kernel Based Learning Multi-class classification models Convolutional Neural Network
Population-Based Training	Does not require sequential runs (+) Can be easily parallelised (+) Decentralised and asynchronous, requiring minimal overhead and infrastructure (+) Can perform online adaptation of hyperparameters in non-stationary settings (+)	Generative Adversarial Networks Deep Reinforcement Learning Supervised Learning

Transfer Learning	Scale well with many prior studies (+) Achieve better results in lesser trials if priors are similar to problem of interest (+) Limiting the number of trials reduces the cost to train, hence more resources for tuning (+) Will be effective only if prior studies are well chosen (-)	Image Classification

Figure 7: Evaluation of current automated hyperparameter optimization methods

5. Optimizer Search

It is crucial for our optimizer to be a generalist, to learn on a large number of different tasks. Hence, parameters of optimizers should be initialized to ensure maximal performance on the new task after the parameters have been updated through one or more gradient steps. The evaluation will focus on 2 aspects, initialization of parameters and generation of an update rule and will be presented in Figure 7.

5.1 Parameters Initialization

Meta-learning proved to be effective on the caveat that it requires prior experience, treating tasks as training examples. Specifically, Model-Agnostic Meta Learning (Chelsea Finn, 2017) might be favoured in the case of AutoML since it is compatible with any model trained with gradient descent and thus applicable to a variety of different learning problems. The intuition of using meta learning is to have a meta-learner to learn on a large number of different tasks and is adaptable as more data becomes available. These tasks are known as the meta-training data while individual task is split further into support and query data.

Firstly, a task which can be viewed as a dataset is randomly sampled from the distribution of tasks. The dataset is then split randomly into support and query data for meta-training. At the start, the parameters theta are randomly initialised and tuned with the support data for a couple of gradient steps. The updated parameters, label as theta* are then used to test against the query set and the loss based on the objective function is then backpropagated to update the parameters theta. This process goes on for a specified number of iterations. Specifically, the meta-learner is used to generate a set of task specific parameters via optimising towards the meta-objective.

The meta-learner here can be combined with recurrent (Yan Duan, 2016) or convolutional neural networks (Nikhil Mishra, 2017). The former uses its memory to store information about previous queries and function evaluations as hidden states to make decisions. The latter combines temporal convolutions and soft attention to allow for aggregation of information from past experience using attention mechanism to pinpoint specific pieces of information from a potentially infinitely large context.

It is important to also note that other than for optimizer search, different meta-objective can be defined to customize meta-learning for different purposes.

Meta-Learner	Advantages	Applications
Gradient (MAML)	Does not introduce any learned parameters Does not require a particular learner architecture since it is updated using gradient not learned update Is consistent since it is using gradient approach, always converging to optimum Can apply to variety of loss functions and any differentiable objective Make no assumption on the form of the model, other than parametrized by parameter vector theta	Regression Image Classification Deep Learning Reinforcement Learning
Recurrent Neural Network	Does not involve either matrix inversion Can run in parallel and not need the optimizer to rely on specific ordering during parallel Temporally linear dependency bottlenecks their capacity to perform sophisticated computation on a stream of inputs (-)	Image Classification Reinforcement Learning Neural Art

SNAIL (Convolution)	Easier to train than traditional RNNs Do not rely on any application-specific architectural components Can be efficiently implemented such that entire sequence can be processed in a single forward pass	Supervised Learning Reinforcement Learning Few-shot Image Classification

Figure 7: Evaluation of current meta-optimizer choice for meta-learning

5.2 Update Rule

Choosing the right optimization method is crucial when training deep learning models. While there are many advanced optimization methods, designing optimization methods, however, is can be challenging due to the non-convex nature of the problem.

Reinforcement learning (Irwan Bello, 2017) has shown to be able to conduct neural optimizer search. In the research, the controller generates strings corresponding to update rules, which are then applied to a neural network to estimate the update rule's performance. The update rules are generated by first selecting 2 operands, then 2 unary functions to apply to the operands before finally a binary function that combines the outputs of the unary function. This performance is then used to update the controller so that the controller can generate improved update rules over time. The update rules being in the form of equations, can be transferred to other optimization tasks. This approach however focuses on deep neural network architectures and have achieved competitive performance against common optimizers in image classification and machine translation.

6. Conclusion

The review has look at some of the areas of machine learning that has been automated. While there are many attempts and researches to automate the pipeline, few have managed a complete framework that works end to end. Even so, they do not apply to the different types of datasets. This is unsurprising since the search space is vast and requires huge computational resources to do so. Meta-learning however, seemed promising to deal with all areas of the pipelines since they work on the general intuition of "learning to learn" and have been proven to work on selected areas with great effect. Thus, meta-learning may be further investigated to build a general framework for all tasks and areas in the pipeline. Further, there seemed to be lesser

focus on data cleaning and collection phase in the research community which may be helpful when dealing with less structured data.

References

D. H. Wolpert, W. G. Macready, “No free lunch theorems for optimization,” IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, Apr. 1997

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, Yang Yu, Taking the Human out of Learning Applications: A Survey on Automated Machine Learning, 2019

Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, Yang Yu

Gilad Katz, Eui Chul Richard Shin, Dawn Song, ExploreKit: Automatic Feature Generation and Selection, 2016

James Max Kanter, Kalyan Veeramachaneni, Deep Feature Synthesis: Towards Automating Data Science Endeavors, 2017

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le, AutoAugment: Learning Augmentation Policies from Data, 2018

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, Sungwoong Kim, Fast AutoAugment, 2019

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, Quoc V. Le, RandAugment: Practical automated data augmentation with a reduced search space, 2019

Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, Quoc V. Le, Learning Data Augmentation Strategies for Object Detection, 2019

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, Tie-Yan Liu, Neural Architecture Optimization, 2018

Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, Chang Xu, CARS: Continuous Evolution for Efficient Neural Architecture Search, 2019

Barret Zoph, Quoc V. Le, Neural Architecture Search with Reinforcement Learning, 2016

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, Tie-Yan Liu, Semi-Supervised Neural Architecture Search, 2020

James Bergstra, Yoshua Bengio, Random Search for Hyper-Parameter Optimization, 2012

Jasper Snoek, Hugo Larochelle and Ryan P. Adams, Practical Bayesian Optimization of Machine Learning Algorithms, 2012

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, Ameet Talwalkar, Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization, 2016

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, Koray Kavukcuoglu, Population Based Training of Neural Networks, 2017

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, D. Sculley, Google Vizier: A Service for Black-Box Optimization, 2017

Chelsea Finn, Pieter Abbeel, Sergey Levine, Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, 2017

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, Pieter Abbeel, RL2: Fast Reinforcement Learning via Slow Reinforcement Learning, 2016

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, Pieter Abbeel, A Simple Neural Attentive Meta-Learner, 2017

Irwan Bello, Barret Zoph, Vijay Vasudevan, Quoc V. Le, Neural Optimizer Search with Reinforcement Learning, 2017