EPJ Web of Conferences **251**, 03036 (2021) https://doi.org/10.1051/epjconf/202125103036
CHEP 2021

Event Classification with Multi-step Machine Learning

Masahiko Saito[1,3,*], *Tomoe* Kishimoto[1,3], *Yuya* Kaneta[2], *Taichi* Itoh[2], *Yoshiaki* Umeda[2], *Junichi* Tanaka[1,3,**], *Yutaro* Iiyama[1], *Ryu* Sawada[1], and *Koji* Terashi[1]

[1]International Center for Elementary Particle Physics, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, Japan
[2]BrainPad Inc., 3-2-10 Shirokanedai, Minato, Tokyo, Japan
[3]Institute for AI and Beyond, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, Japan

Abstract. The usefulness and value of Multi-step Machine Learning (ML), where a task is organized into connected sub-tasks with known intermediate inference goals, as opposed to a single large model learned end-to-end without intermediate sub-tasks, is presented. Pre-optimized ML models are connected and better performance is obtained by re-optimizing the connected one. The selection of an ML model from several small ML model candidates for each sub-task has been performed by using the idea based on Neural Architecture Search (NAS). In this paper, Differentiable Architecture Search (DARTS) and Single Path One-Shot NAS (SPOS-NAS) are tested, where the construction of loss functions is improved to keep all ML models smoothly learning. Using DARTS and SPOS-NAS as an optimization and selection as well as the connections for multi-step machine learning systems, we find that (1) such a system can quickly and successfully select highly performant model combinations, and (2) the selected models are consistent with baseline algorithms, such as grid search, and their outputs are well controlled.

1 Introduction

Machine learning (ML), in particular, deep learning (DL), has evolved rapidly due to the availability of huge computing power and big data and has proven to be successful in many applications such as image classification, natural language translation, etc. In most ML approaches, a single task with a large model learned end-to-end is defined and trained to solve a given problem (see Fig. 1(a)). In most cases, this end-to-end approach provides state-of-the-art performance for a given problem in terms of precision and accuracy. However, we will adopt a different approach, which can still give acceptable precision and accuracy for a given problem: we connect some ML models, each of which can solve a part of a given problem as shown in Fig. 1(b). We call it Multi-step ML. Moreover, in some of Multi-step ML, we can assume that there are several different ML model candidates to solve the same sub-tasks as shown in Fig. 1(c). In this paper, ideas for the connecting of sub-tasks and their model selection are presented.

*e-mail: saito@icepp.s.u-tokyo.ac.jp
**e-mail: jtanaka@icepp.s.u-tokyo.ac.jp

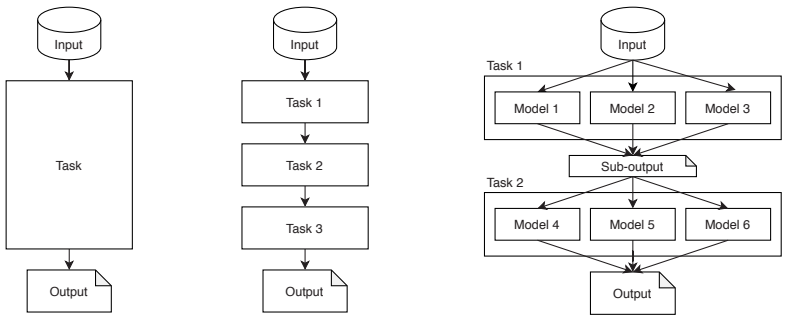
Multi-step ML

For a given task, we break it into several sub-tasks with known intermediate inference goals, find optimal ML models for each sub-task, resulting in the best model chain. From a different perspective, assuming that there are several tasks with well-defined or well-trained ML models, we solve a new task by combining them. The common interesting point is that there are multiple sub-tasks for a given task. ML models for each sub-task are relatively easy to build, or well-defined or well-trained ML models already exist.

Our approach may result in an interpretability versus accuracy trade-off when compared with end-to-end paradigms. The merits of our approach include:

- Domain knowledge is easily introduced into ML models for sub-tasks, and such ML models can be reused in other problems which involve common tasks,
- Intermediate data, which is the output of sub-tasks, provide the information to understand the behavior of the ML models, which can lead to an explainability of ML.

The simplest way to connect ML models is just to give the output of an ML model to the input of a next ML model. In this paper, we introduce more effective methods on the connection of ML models.



(a) Single-step (b) Multi-step (c) Multi-step with model selection

Figure 1. ML task flow

What type of problems can be useful for Multi-step ML?

Sub-tasks have to be defined so that a given problem is required to be expressed as a set of sub-problems, where such sub-problems should be recognized and their role, input and output are defined by a human. Said in this exaggerated manner, this might correspond to problems found in any community. A simple example is from some image classification problems: a sub-task to identify objects in an image and a sub-task to understand the context of the image using the identified objects can be separated [1, 2]. From the viewpoint of data flow, input data for Multi-step ML are produced via several steps; in other words, there is a hierarchical structure in the input data. For this type of data, we can solve the problem by defining sub-tasks for each step. Practically, a problem to use data produced by a simulation of any model or theory is optimal since the data could have a hierarchical structure or sub-tasks can be defined easily using supervised information. This matches studies in science fields, for example, experimental particle physics, as shown later.

NAS to select MLs

The situation shown in Fig. 1(c) might happen when there are different models requiring the optimization of ML model structures and hyperparameters. We also expect that the choice of ML in the intermediate steps is not unique, that is, it will depend on the goal of a given task. To select one of the models for each sub-task, in this paper, we use the idea of neural architecture search (NAS).

We demonstrate the usefulness and value of Multi-step ML on (1) the re-optimization of model weights after sequentially connecting multiple ML models, and (2) the selection of an ML model from multiple ML model candidates. For the latter, we adopt the idea of the differentiable architecture search (DARTS [3]) and the single path one-shot neural architecture search (SPOS-NAS [4]) to a task of particle physics, where two sub-tasks are defined to solve a given task. This is called *Model Selection with NAS (MSNAS)*.

Particle Physics

Experimental particle physics aims to understand the fundamental laws of nature and reveal unknown laws using a huge amounts of data. In collider physics experiments, each event[1] of data is produced from collisions using a high energy accelerator. An event has several particles, which are measured with detectors surrounding collision points. The classification of events is quite important in collider physics data analysis, where interesting signal events are separated from background events. ML has been used in collider physics research, for example, boosted decision trees (BDT) for event classification [5], and DL for event classification [6], jet imaging [7], etc.

2 Related Work

Multi-step ML can be categorized into so-called *Automated Machine Learning* (AutoML), for example, Ref. [8], where the hyperparameter optimization, meta-learning, NAS, etc. are described and discussed. The scope of AutoML is huge and continues to grow. One of differences from AutoML is that in this paper we focus on the connection of multiple ML models, where not only a task but also sub-tasks are defined by humans because each sub-task has its own purpose.

NAS was introduced to automatically design a network architecture for a given task with the best performance and less human intervention. The purpose of Multi-step ML is different from that of NAS, however, methods developed for NAS can be applicable to Multi-step ML. Several survey documents of NAS are found, for example, in Refs. [9, 10]: image classification [11–13], object detection [12, 14], etc. In some NAS algorithms, large computational resources are required, for example, due to the discrete search space of the architecture with the reinforcement learning. To overcome this issue, the idea of one-shot NAS [15, 16] is promising: ENAS [17], DARTS [3, 18, 19], ProxylessNAS [20], SPOS-NAS [4], SNAS [21] and so on. In DARTS, a differentiable calculation in the search space is introduced using the softmax function. In SPOS-NAS, a supernet training and architecture search is decoupled by using the single path one-shot approach. In NAS, a neural architecture is selected to achieve the best performance for a given task. On the other hand, in Multi-step ML, an ML model is selected by considering the performance of both a given task and its sub-tasks.

[1]The term of "event" corresponds to the term of "image" in the image classification using for example CIFAR-10.

3 Methods

In this paper, the idea of DARTS [3] and SPOS-NAS [4] is used to select one of the ML models. One of the motivations of these algorithms based on NAS is to improve computing complexity. By optimizing all model combinations simultaneously, instead of optimizing all model combinations separately, compute time reduces drastically because of avoiding repetitive model training. In a method based on DARTS, a network consists of parallelly connected ML models. All model weights in the network are optimized simultaneously for each mini-batch. In a method based on SPOS-NAS, a network consists of randomly sampled ML models. Model weights in the randomly selected model combination are optimized for each mini-batch.

Briefly summarizing these algorithms below, we explain what we have additionally done for connecting and selecting models.

3.1 Based on DARTS

DARTS represents a search architecture as a graph, where an edge corresponds to an operation. Each edge has an architecture weight (α), which is used to aggregate the outputs o_i on the same path with a softmax function $o = \sum_{i \in \text{path}} \text{softmax}(\alpha_i) \cdot o_i$. After the training of architecture weights, operations that have maximum α among the same path are selected as a final operation set.

We use this idea for model selection. The operations, which are represented as edges, are replaced with models for a sub-task. The outputs of each model are built using architecture weight α, called a model architecture weight hereafter, $y_t = \sum_{i \in \text{models}} \text{softmax}(\alpha_i) \cdot y_{t,i}$, where $y_{t,i}$ is an output of i-th model for the t-th sub-task. After the training of model architecture weights, the models that have maximum α among sub-tasks are selected as a final model set.

First, a pre-training is applied: every single model is individually trained using ground-truth data. Second, model architecture weights are optimized following the DARTS optimization, where model weights (w) and the model architecture weights (α) are optimized separately. Model architecture weights are updated to be optimal for validation data, while model weights are fitted to training data. Finally, with fixed models selected by model architecture weights, model weights of selected models are re-optimized for training data. The algorithm is outlined in Appendix. A.

A loss function used in DARTS for MSNAS is built using the loss functions of each sub-task:

$$\mathcal{L}(\boldsymbol{y}^{\text{true}}, \boldsymbol{y}^{\text{pred}}) = \sum_{t \in \text{task}} v_t \cdot \left(\mathcal{L}_t(y_t^{\text{true}}, y_t^{\text{pred}}) + \sum_{i \in \text{models}} \epsilon \cdot \mathcal{L}_t(y_t^{\text{true}}, y_{t,i}^{\text{pred}}) \right),$$

where $\mathcal{L}_t$, y_t^{true}, and y_t^{pred} are a loss function, a ground truth, and a model prediction for the t-th task, respectively. The first term is a loss function for aggregated outputs for each model. This term is necessary for differentiably updating model architecture weights. The second term is a loss function for each model. Without the second term, outputs for each sub-task do not converge to the targets y^{true}, because there are degrees of freedom that can significantly change the output values of individual models while not changing the loss function values due to interference between models connected in parallel. In this study, ϵ is fixed[2] to 1, and task weights (v_t) are hyperparameters with a normalization of $\sum_{t \in \text{task}} v_t = 1$, which was found to be reasonable values satisfying with both the validity of each model output and the performance of the whole task.

[2]Optimization of ϵ, e.g. depending on α instead of a fixed value, is future work.

We use an Adam optimizer with a learning rate of 10^{-3} for pre-training, model architecture weight determination, and post-training. The training is terminated after 100 epochs or if a valid loss does not decrease in 10 (20) epochs in pre/post-training (model architecture weight determination).

3.2 Based on SPOS-NAS

In Ref. [4], in the context of NAS, the supernet optimization (weight optimization) and architecture search are decoupled. Weights are optimized after selecting a single path of architectures with a uniform path sampling. Then, the architecture search is performed using the evolutionary algorithm.

We use this idea for model selection. Model weights in each sub-task are optimized after selecting a single path of models with a uniform sampling. Then, the model search, where the best model is selected for each task, is performed using the grid search algorithm instead of the evolutionary algorithm since the number of models is small in this study. The algorithm is outlined in Appendix. A.

A loss function used in SPOS-NAS for MSNAS is built using the loss functions of each model:

$$\mathcal{L}(\boldsymbol{y}^{\text{true}}, \boldsymbol{y}^{\text{pred}}) = \mathbb{E}_i \left[\sum_{t \in \text{task}} v_t \cdot \mathcal{L}_t(y_t^{\text{true}}, y_{t,i}^{\text{pred}}) \right],$$

where i-th models for the t-th sub-task are randomly sampled. In this study, task weights (v_t) are hyperparameters like DARTS in MSNAS, with a normalization of $\sum_{t \in \text{task}} v_t = 1$. The optimizer used and the strategy of training termination are the same as the DARTS method.

4 Experiments and Results

A toy problem from experimental particle physics is prepared to prove the concept of Multi-step ML. All datasets have been generated by Monte-Carlo simulation.

4.1 Problem and task

The problem used as an experiment in this paper is the classification of particle origin: one is a Higgs boson (H) and the other is a Z boson. The main differences between the two particles are their mass (125 GeV for H, 91 GeV for Z) and spin (0 for H, 1 for Z). Both particles promptly decay into a pair of τ-leptons ($H/Z \to \tau^{\pm}\tau^{\mp}$) with some probability, and the τ-leptons then decay into various particles, leaving an energy deposit in the detectors. With the signature left in the detector, τ-lepton candidates are reconstructed, then the particle origin is identified. In this paper, we separate this problem into two parts and define sub-tasks: the first task (Task 1) is the energy calibration (measurement) of a τ-lepton candidate, and the second one (Task 2) is the classification of H/Z using a pair of τ-lepton candidates where the input is the output from Task 1.

A loss function of Task 1 is defined as mean squared errors of τ-lepton momentum

$$\mathcal{L}_1 = \frac{1}{N_{\text{events}}} \sum_{i \in \text{event}} \left(\|\boldsymbol{p}_{1,i}^{\text{pred}} - \boldsymbol{p}_{1,i}^{\text{true}}\|^2 + \|\boldsymbol{p}_{2,i}^{\text{pred}} - \boldsymbol{p}_{2,i}^{\text{true}}\|^2 \right),$$

where $\boldsymbol{p_1}$ ($\boldsymbol{p_2}$) is a leading (sub-leading) τ-lepton momentum. For the Task 2, a binary cross-entropy loss

$$\mathcal{L}_2 = -y^{(\text{true})} \log y - (1 - y^{(\text{true})}) \log(1 - y)$$

is considered. For stable training, the output of Task 2 is defined as logits instead of probability (i.e. sigmoid(logits)) and used instead of the formula above. Output aggregation and loss function, therefore, are defined as

$$y = \sum_{i \in \text{models}} \text{softmax}(\alpha_i) \cdot y_i \quad \text{(DARTS method)},$$
$$\mathcal{L}_2 = max(y, 0) - y \cdot y^{(\text{true})} + \log\left(1 + e^{-|y|}\right).$$

In our problem setting, the two loss functions cannot be treated as having equivalent statistics: the loss of Task 1 is a χ^2 assuming a momentum resolution of 1 GeV, while the loss of Task 2 can be regarded as a negative log-likelihood based on Bernoulli distribution. To match the scale of two loss functions, the loss of Task 1 is scaled by 10^{-4}, i.e. the momentum is normalized by 100 GeV, in our experiments.

4.2 Dataset

The data was produced with particle physics simulation programs[3]. We use only hadronic τ-leptons that decay into hadrons, not electrons nor muons. Task 1 uses reconstructed-level jet 4-vectors and calorimeter/tracker information as input variables, then predicts truth-level τ-lepton momentum. Calorimeter/tracker information is given as 16x16 pixel images, which is expected to be used to estimate the momentum of neutrino from τ-leptons to calibrate τ-lepton momentum. The input/output formats of Task 1 and Task 2 and the pixel image examples are found in Appendix B. The transverse momentum p_T of Task 1 and Task 2 used in ML models is normalized by $p_\text{T} \leftarrow \log(0.1 + p_\text{T}(\text{GeV}))$, to fit the values into a reasonable range for machine learning algorithms. We have 50,000 events for both H and Z, where 60% for training, 20% for validation and 20% for test.

4.3 Models

Three kinds of models are prepared for each task. For Task 1, Multi-Layer Perceptron (MLP), CNN and a linear transformation method, called a scale factor method (Sf) hereafter, are defined. For Task 2, MLP, Long Short-Term Memory (LSTM), and a simple mass method (Mass) are defined. MLP, CNN and LSTM models are typical deep learning models, while Sf and Mass models are based on conventional methods used in collider particle physics. They are robust compared to deep learning models and are expected to be not the best models because of their simplicity.

MLP and CNN models for Task 1 consist of two blocks: image feature extraction and correction factor evaluation (see Appendix C). The second block is designed to output a momentum residual like ResNet [24]. CNN has a good domain bias for image recognition, while MLP does not. The MLP model for Task 1 is expected to be overfitted due to its large number of trainable parameters in this problem. A Sf model for Task 1 applies a linear transformation ($f(x) = ax + b$) for each variable (p_T, η, ϕ)[4].

An MLP model for Task 2 is a simple deep neural network with three hidden layers with 32 nodes. An LSTM model for Task 2 is built by three stacked LSTM modules with 32 hidden nodes. Two τ-leptons, ordering by jet p_T, are sequentially fed to the LSTM module. A Mass

[3]We use Monte Carlo (MC) simulation to generate data, which is a set of events. We generated MC events with Pythia8 [22] assuming 13 TeV proton-proton collisions like the Large Hadron Collider at CERN. The detector response was simulated using Delphes [23] with the CMS parameterization (delphes_card_CMS_PileUp.tcl.)

[4]A Sf model has six trainable parameters.

model for Task 2 calculates a system mass[5] of two τ-lepton candidates, then applies it to an MLP with 2 hidden layers with 64 nodes.

In all models above, ReLU is used as an activation function. The model hyperparameters, e.g. the number of layers, are determined by scanning them for each the single model.

In addition, two models: Zeros (the output is always 0) and Noise (Gaussian noise $\sim N(\mu = 0, \sigma^2 = 1)$) are prepared as dummy models and are used in the model selection studies. We expect that if DARTS works well, these models should not be selected. On the other hand, SPOS-NAS cannot have dummy models since weights of models (MLP, etc.) are largely affected if such dummy models are included in a single-path. Experiments on SPOS-NAS are performed for models not including these dummy models.

Before performing any studies, each model is pre-trained using ground-truth data from the simulation as explained in Section 3.

4.4 Results

4.4.1 Re-optimization of ML models used in multiple steps

We present the usefulness of the re-optimization of model weights with the sequentially connected with multiple trained ML models. We execute experiments with two strategies:

Without re-optimization : Train a Task 1 model, then train a Task 2 model using the outputs of the Task 1 model.

With re-optimization : Train Task 1 and Task 2 models separately using ground-truth data from the simulation, then build a connected model and train the model.

A performance (AUC) for Task 2 is measured for all model combination (excluding dummy models) with and without the re-optimization as shown in Fig. 2, where experiments are executed 20 times with different random seeds for the same dataset. For the re-optimization model, v_1 is set to zero[6]. Re-optimization after pre-training improves the performance of the final task for any model pairs. Pairs of (CNN, MLP) or (CNN, LSTM) have the highest AUC values in this experiment, and should be selected in MSNAS.

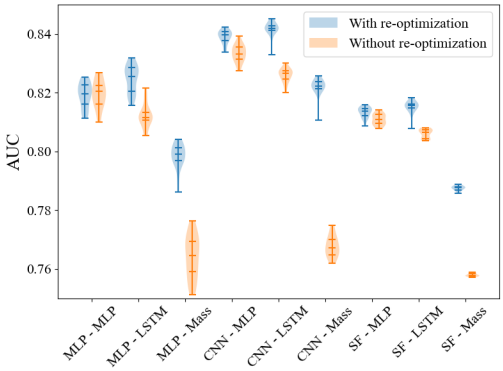


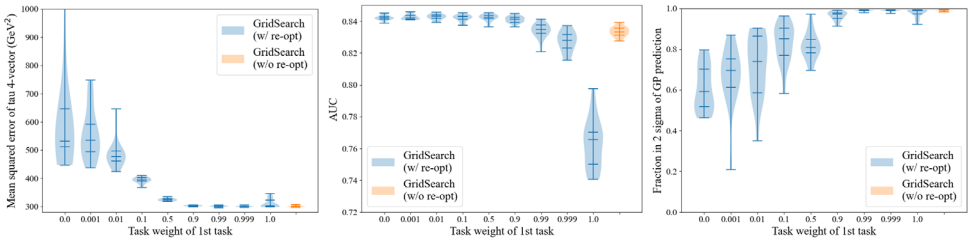
Figure 2. AUC of Task 2 over 20 runs for all model pairs with (blue) and without (orange) re-optimization. Horizontal lines in the violin plot show the quantile values for 0, 0.25, 0.50, 0.75, and 1.

Figure 3 (a) and (b) show loss-values of Task 1 (mean squared error) and AUC of Task 2 for the best model choices in each run as a function of Task 1 weight v_1. The larger the Task 1

[5]The system mass M is $\sqrt{(\sum_i E_i)^2 - (\sum_i p_{i,x})^2 - (\sum_i p_{i,y})^2 - (\sum_i p_{i,z})^2}$.

[6]Two task weights (v_1 and v_2) are normalized, i.e. $v_2 = 1 - v_1$ in this experiment.

weight, the more that the Task 1 loss value decreases, while the AUC values of Task 2 do not change significantly up to v_1 of 0.9. The choice of task weights, $v_1 = 0.9$ in this case, improves explainability (see the next section) while maintaining the performance of the final task.



(a) MSE of Task 1 (b) AUC of Task 2 (c) Fraction of events matching with Gaussian process predictions

Figure 3. Mean squared error of Task 1 (a) and AUC of Task 2 (b) as a function of Task 1 weight v_1 for the re-optimization model (blue) and without re-optimization (orange). There are 20 runs for each point. The fraction of events where the model outputs are within the two sigma range of a Gaussian process prediction (c) as a function of Task 1 weight v_1 for the re-optimization model (blue) and without re-optimization (orange).

Explainability

By splitting a large problem into sub-tasks, Multi-step ML is able to access intermediate states with readable forms. To evaluate the interpretability of intermediate states, we measure the fraction of outliers of the Task 1 output. As a reference model that predicts robust and controlled outputs, we use a Gaussian process (GP), which is a Bayesian machine learning technique predicting expected values with their uncertainties. A Gaussian process is implemented using GPyTorch package, and it is trained using the same training data. A validity of Task 1 output is defined as the fraction of events where a model prediction of Task 1 output is within the two sigma range predicted by Gaussian process[7].

A validity of Task 1 is shown in Fig. 3 (c) as a function of Task 1 weight (v_1). Strong constraints on Task 1's loss result in similar predictions with a Gaussian process model. Less constraints, on the other hand, give different predictions from the Gaussian process model over the uncertainties. Such a prediction cannot be regarded as the one expected, e.g. particle momentum in this case. A proper setting of task weights is required for the model to be explainable.

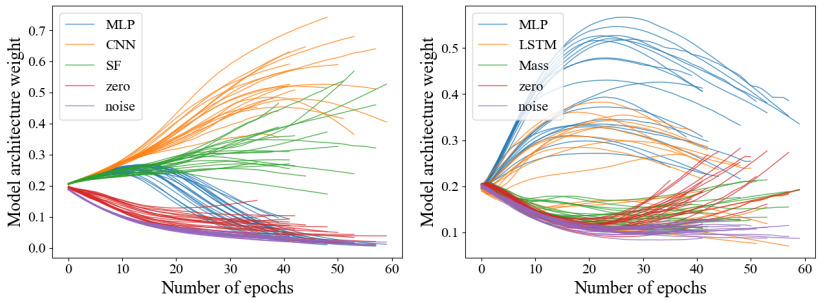
4.4.2 Selection of an ML model from multiple ML models in each step

We show results of the selection of an ML model using MSNAS. To compare results with grid search and see the stability of the model choice, we have 20 runs with the same dataset with different random seeds. In no run were Zeros or Noise models selected by either MSNAS (DARTS) or grid search.

The training process of the DARTS method is shown in Figs. 4, where model architecture weights (α) in Task 1 and Task 2 are shown as a function of the number of training epochs.

[7] A consistency within 2σ is independently evaluated for each scalar variables ($p_{\mathrm{T},1}, \eta_1, \phi_1, p_{\mathrm{T},2}, \eta_2, \phi_2$) for the simplicity.

The behavior of α looks similar in all the 20 runs. DARTS optimization, therefore, is robust for different initialization of each model weight. Model architecture weights of Zeros and Noise models decrease as the training proceeds, while model architecture weights of other ML models (MLP, CNN and LSTM) increases at the beginning of the training, as expected.



(a) α for Task 1 (b) α for Task 2

Figure 4. Model architecture weights (a) for Task 1 and (b) for Task 2 in DARTS training step as a function of epoch with different colors for different models. All training is terminated by early stopping, where the training is forced to be terminated when the validation loss does not decrease in the last 20 epochs.

Figure 5 shows which models are selected in this experiment as a function of Task 1 weight v_1. Grid search and SPOS-NAS select (CNN, LSTM) and (CNN, MLP) pairs in this order, while DARTS selects these two model pairs but with different fractions. Considering the difference of AUC between these pairs is small as shown in Fig. 2, DARTS can have a practical model selection power, but might have worse model selection ability than grid search and SPOS-NAS methods.

Performance of Task 1 (MSE) and Task 2 (AUC) is shown in Figs. 6 (a) and (b) as a function of Task 1 weight v_1. As expected, as Task 1 weight v_1 is larger, the performance of Task 1 improves while that of Task 2 (AUC) becomes worse. There is, however, a moderate point where the performance for the both tasks does not change largely from the best point. MSNAS gives a nearly optimal prediction for the last task with the intermediate data under control. A validity check by Gaussian process prediction is performed as shown in Fig. 6 (c). Grid search and SPOS-NAS have similar performance, while DARTS has a quite different prediction compared to a Gaussian process at small v_1, which will be investigated further.

4.4.3 Scalability

The compute complexity of a grid search is expressed by $O(\prod_{t\in\text{tasks}} N_{\text{model,t}})$, i.e. $O(N^2_{\text{models}})$ in our experiment, from the number of combinations for model pair. On the other hand, the compute complexity of DARTS and SPOS-NAS is $O(N_{\text{tasks}}N_{\text{models}})$ because these models are trained simultaneously.

We check the compute complexity as a function of the number of models. In this experiment, the same models as explained in Section 4.3 with dummy models excluded are used, i.e. the number of models for each task is three. To measure the scalability with a large number of trainable models, the models above are replicated with different initialization, where model weights of replicated models are not shared. Wall time for grid search, DARTS and

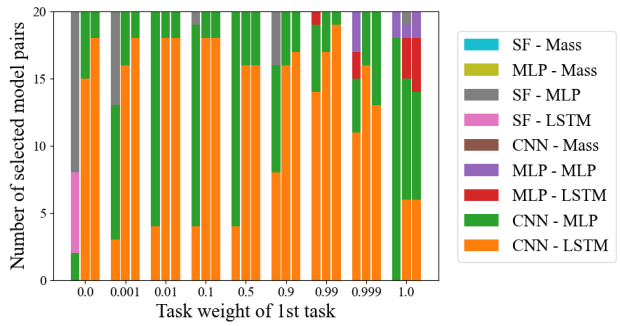
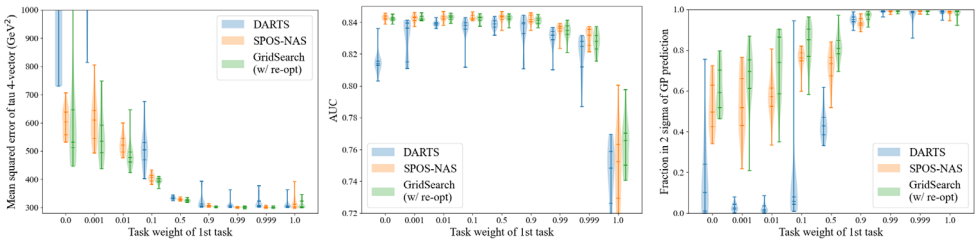


Figure 5. Selected model pairs in 20 runs as a function of Task 1 weight v_1 for DARTS, SPOS-NAS, and grid search from the left. Model pairs are stacked in the order of performance for Task 2 measured in Fig. 2.



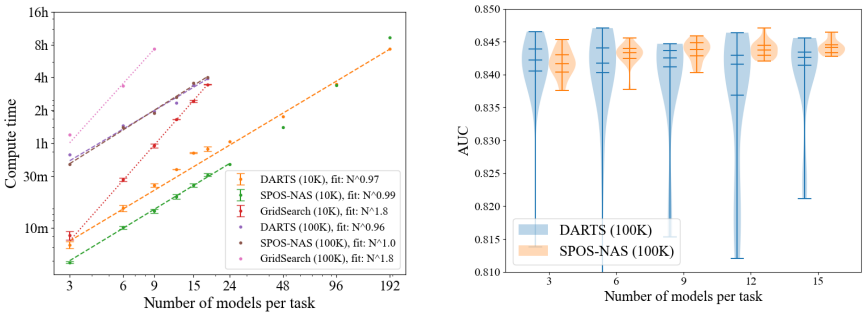
(a) Mean squared errors of Task 1 (b) AUC of Task 2 (c) Fraction of events matching with Gaussian process predictions

Figure 6. (a) Mean squared errors of Task 1, (b) AUC of Task 2 and (c) the fraction of events matching with Gaussian process predictions for DARTS (blue), SPOS-NAS (orange) and grid search with re-optimization model (green) as a function of a Task 1 weight v_1.

SPOS-NAS is shown in Fig. 7 (a) as a function of the number of models per task, where two different sized datasets are used to measure the wall time within the reasonable execution time. The dependency of the number of models follows our expectation: $O(N_{\text{models}})$ for DARTS and SPOS-NAS and $O(N^2_{\text{models}})$ for grid search. SPOS-NAS uses grid search instead of a evolutionary algorithm after the one-shot NAS, resulting in changing the power law from $O(N_{\text{models}})$ to $O(N^2_{\text{models}})$. This will be relaxed if a evolutionary algorithm is used. Performance of model prediction of Task 2 is stable against the number of models as shown in Fig. 7 (b). The DARTS and SPOS-NAS methods have good scalability for multiple-step machine learning problems.

5 Discussion

We have performed MSNAS based on NAS techniques (DARTS and SPOS-NAS) to connect and select ML models. From the perspective of performance, an ensemble of several ML models or different weight initialization, where all trained MLs are used for inference, may improve the performance of the final prediction. The purpose of Multi-step ML, however, is different as we introduced in Section 1: sub-tasks are defined for a given a large problem, which might lead to making ML models simpler. The number of model parameters can be reduced by selecting one of models with good performance, resulting in faster inference with lower computing resource requirements, e.g. memory. Splitting into and defining sub-tasks



(a) Wall time (b) AUC

Figure 7. (a) Wall time of MSNAS and grid search with two different-sized dataset 10K or 100K as a function of the number of models. NVIDIA Tesla A100 is used for all execution. DARTS and grid search are implemented using Tensorflow, and SPOS-NAS is implemented using PyTorch. Three (one) independent runs are executed for small (large) compute-time points drawn with (without) error bars. Dashed (MSNAS) and dotted lines (grid search) show fitted lines with a function of $f(x) = C \cdot x^a$. For SPOS-NAS, the fitting is applied up to the first six bins, because the dependency of the number of models is not followed by a single power law. (b) AUC of Task 2 for a full dataset (100K) as a function of the number of models, where 20 runs are executed for each point.

instead of building a large single task increases the explainability, where all the outputs of sub-tasks can be accessed. Moreover, it might be possible to know a proper domain for the problem from which a model is selected, e.g. if a LSTM model is selected, the problem has a strong correlation for a sequential structure.

Considering the demand from the machine learning community, hyperparameter optimization is also a hot topic. MSNAS is able to select optimal hyperparameters as well as optimal models, by setting multiple models with different hyperparameters as model candidates in a task. This method is scalable if each model has the small number of hyperparameter combination, e.g. $O(N_{\text{task}} N_{\text{hp\_comb.}})$ where each model has $N_{\text{hp\_comb.}}$ hyperparameter combination. However, it is not scalable if models have many types of hyperparameters, e.g. $O(N_{\text{task}} N_{\text{hp}}^{N_{\text{hp\_grid}}})$ when each model has N_{hp} types of hyperparameters which have $N_{\text{hp\_grid}}$ states. Practically, the application for the hyperparameter scan requires our method to work well for discrete parameters. This is a future challenge.

The task weights (v_i) are treated as a hyperparameter in this study. This parameter determines how much we respect the intermediate outputs. On the other hand, we can treat them as floating parameters like the studies used in multi-task learning [25–30]. This is also a future challenge.

For the model selection, SPOS-NAS has nearly the same performance as grid search, but DARTS does not. This is under investigation. Actually, our experiment is based on one specific problem, which is not familiar in the computing science field, so that we need more problems to test if our results are general.

For collider particle physics, the toy model used in this paper is a simplified model to demonstrate MSNAS methods. In a more realistic case, there are more tasks to be considered, i.e. a tau identification task, or there is room to extend the object types and topology, e.g. b-jets, photon or leptons. We plan to integrate such models and objects for MSNAS to be more practical in the future.

6 Conclusions

The usefulness and value of Multi-step ML are presented in this paper. The re-optimization after connecting multiple ML models gives better performance than the no re-optimization case. The selection of a single ML model has been performed by using the idea of DARTS and SPOS-NAS, where the construction of a loss function is improved to keep all ML models smoothly learning. Using DARTS and SPOS-NAS as an optimization and selection as well as the connecting for multi-step machine learning systems, we find that (1) such system can quickly and successfully select highly performant model combinations, and (2) the selected models are consistent with baseline algorithms such as grid search and their outputs are well controlled. Our idea has been tested for one specific problem so that we need more problems to test if our results are general.

Acknowledgments

We thank Dr. Michael Kagan (SLAC) and Dr. Lukas Heinrich (CERN) for the useful discussion and guidance for addressing ML techniques. This research was partially supported by Institute for AI and Beyond of the University of Tokyo.

Code availability

Our codes for the framework of Multi-step ML [31] and for the implementation for this application [32] are available.

References

[1] F. Alamri, N. Pugeault, IEEE Transactions on Cognitive and Developmental Systems pp. 1–1 (2020)

[2] T. Kishimoto, M. Saito, J. Tanaka, Y. Iiyama, R. Sawada, K. Terashi, *An improvement of object detection performance using multi-step machine learnings* (2021), 2101.07571

[3] H. Liu, K. Simonyan, Y. Yang, *Darts: Differentiable architecture search* (2019), 1806.09055

[4] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, J. Sun, *Single path one-shot neural architecture search with uniform sampling* (2020), 1904.00420

[5] B.P. Roe, H.J. Yang, J. Zhu, Y. Liu, I. Stancu, G. McGregor, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **543**, 577 (2005)

[6] P. Baldi, P. Sadowski, D. Whiteson, Nature Communications **5** (2014)

[7] G. Kasieczka, T. Plehn, A. Butter, K. Cranmer, D. Debnath, B.M. Dillon, M. Fairbairn, D.A. Faroughy, W. Fedorko, C. Gay et al., SciPost Physics **7** (2019)

[8] F. Hutter, L. Kotthoff, J. Vanschoren, eds., *Automated Machine Learning: Methods, Systems, Challenges* (Springer, 2018), in press, available at http://automl.org/book.

[9] P. Ren, Y. Xiao, X. Chang, P.Y. Huang, Z. Li, X. Chen, X. Wang, *A comprehensive survey of neural architecture search: Challenges and solutions* (2020), 2006.02903

[10] T. Elsken, J.H. Metzen, F. Hutter, *Neural architecture search: A survey* (2019), 1808.05377

[11] B. Zoph, Q.V. Le, *Neural architecture search with reinforcement learning* (2017), 1611.01578

[12] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, *Learning transferable architectures for scalable image recognition* (2018), 1707.07012

[13] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, *Regularized evolution for image classifier architecture search* (2019), 1802.01548

[14] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, J. Sun, *Detnas: Backbone search for object detection* (2019), 1903.10979

[15] A. Brock, T. Lim, J.M. Ritchie, N. Weston, *Smash: One-shot model architecture search through hypernetworks* (2017), 1708.05344

[16] G. Bender, P.J. Kindermans, B. Zoph, V. Vasudevan, Q. Le, *Understanding and Simplifying One-Shot Architecture Search*, in *Proceedings of the 35th International Conference on Machine Learning*, edited by J. Dy, A. Krause (PMLR, Stockholmsmassan, Stockholm Sweden, 2018), Vol. 80 of *Proceedings of Machine Learning Research*, pp. 550–559, http://proceedings.mlr.press/v80/bender18a.html

[17] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, J. Dean, *Efficient neural architecture search via parameter sharing* (2018), 1802.03268

[18] X. Chen, L. Xie, J. Wu, Q. Tian, *Progressive differentiable architecture search: Bridging the depth gap between search and evaluation* (2019), 1904.12760

[19] X. Dong, Y. Yang, *Searching for a robust neural architecture in four gpu hours* (2019), 1910.04465

[20] H. Cai, L. Zhu, S. Han, *Proxylessnas: Direct neural architecture search on target task and hardware* (2019), 1812.00332

[21] S. Xie, H. Zheng, C. Liu, L. Lin, *Snas: Stochastic neural architecture search* (2020), 1812.09926

[22] T. Sjostrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, P.Z. Rasmussen, Christine O. a nd Skands, Computer Physics Communications **191**, 159 (2015)

[23] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaitre, A. Mertens, M. Selvaggi, Journal of High Energy Physics **2014** (2014)

[24] K. He, X. Zhang, S. Ren, J. Sun, *Deep Residual Learning for Image Recognition*, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778

[25] R. Cipolla, Y. Gal, A. Kendall, *Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics*, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 7482–7491

[26] Z. Chen, V. Badrinarayanan, C.Y. Lee, A. Rabinovich, *GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks*, in *Proceedings of the 35th International Conference on Machine Learning*, edited by J. Dy, A. Krause (PMLR, 2018), Vol. 80 of *Proceedings of Machine Learning Research*, pp. 794–803, http://proceedings.mlr.press/v80/chen18a.html

[27] S. Liu, E. Johns, A.J. Davison, *End-To-End Multi-Task Learning With Attention*, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 1871–1880

[28] M. Guo, A. Haque, D.A. Huang, S. Yeung, L. Fei-Fei, *Dynamic Task Prioritization for Multitask Learning*, in *Computer Vision – ECCV 2018*, edited by V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss (Springer International Publishing, Cham, 2018), pp. 282–299, ISBN 978-3-030-01270-0

[29] O. Sener, V. Koltun, *Multi-Task Learning as Multi-Objective Optimization*, in *Advances in Neural Information Processing Systems*, edited by S. Bengio, H. Wal-

lach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Curran Associates, Inc., 2018), Vol. 31, https://proceedings.neurips.cc/paper/2018/file/432aca3a1e345e339f35a30c8f65edce-Paper.pdf

[30] X. Lin, H.L. Zhen, Z. Li, Q.F. Zhang, S. Kwong, *Pareto Multi-Task Learning*, in *Advances in Neural Information Processing Systems*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Curran Associates, Inc., 2019), Vol. 32, https://proceedings.neurips.cc/paper/2019/file/685bfde03eb646c27ed565881917c71c-Paper.pdf

[31] *Multiml framework*, https://github.com/UTokyo-ICEPP/multiml

[32] *Multiml htautau*, https://github.com/UTokyo-ICEPP/multiml_htautau

Table 1. Input and output format for Task 1

	Name	Shape
Input	Reconstructed jet 4 vector (p_T, η, ϕ, m)	(4,)
	Tracker p_T distribution	(16, 16)
	Electromagnetic (EM) calorimeter E_T distribution	(16, 16)
	Hadronic calorimeter E_T distribution	(16, 16)
Output	Tau 4 vector (p_T, η, ϕ)[8]	(3,)

Table 2. Input and output format for Task 2

	Name	Shape
Input	Tau 4 vector $(p_T, \eta, \phi) \times 2$	(6,)
Output	Higgs classification[9]	(1,)

A Details of MSNAS algorithm

Algorithm 1 DARTS for Multi-step ML

Notation
$\boldsymbol{x}$: input/output data
$\{M_i^j\}$: j-th model for i-th task
$\boldsymbol{w}_i^j$: model weights of M_i^j
α_i^j: model architecture weight of M_i^j
$\mathcal{L}^i$: loss function for i-th task
$\mathcal{L}$: loss function for multi-steps ML

Pre-training:
for each model (M_i^j) **do**
  while not converged **do**
    Update $\boldsymbol{w}_i^j$ by descending $\nabla_{\boldsymbol{w}_i^j} \mathcal{L}^i(\boldsymbol{w}_i^j | \boldsymbol{x}_{\text{train}})$
  end while
end for

Weights determination:
Build a multi-step model parameterized by α_i^j for M_i^j
while not converged **do**
  Update α by descending $\nabla_\alpha \mathcal{L}(\boldsymbol{w}^*(\alpha), \alpha | \boldsymbol{x}_{\text{valid}})$
  Update $\boldsymbol{w}$ by descending $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \alpha | \boldsymbol{x}_{\text{train}})$
end while
$M_i^{\text{final}} \leftarrow M_i^{j_{\text{best}}}$, $j_{\text{best}} = \arg\max_{j \in \text{models}} \alpha_i^j$
Build a multi-step model using $\{M_i^{\text{final}}\}$

Post-training:
while not converged **do**
  Update $\boldsymbol{w}_i^{\text{final}}$ by descending $\nabla_{\boldsymbol{w}_i^j} \mathcal{L}^i(\boldsymbol{w}_i^j | \boldsymbol{x}_{\text{train}})$
end while

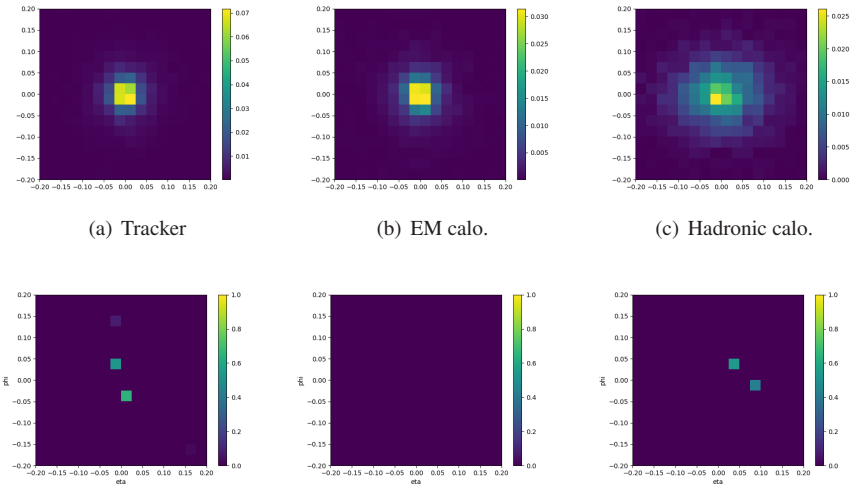
Algorithm 2 SPOS-NAS for Multi-step ML

Same Notation, Pre-training and Post-training as DARTS

Weights determination:
while not converged **do**
  Build a multi-step model using uniformly sampled models $(\{M_i^j\})$
  Update $\boldsymbol{w}$ by descending $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w} | \boldsymbol{x}_{\text{train}})$ for sampled model
end while
while $\{M_i^j\}$ in all model combination **do**
  if $\mathcal{L}(\{M_i^j\}) < \mathcal{L}(\{M_i^{\text{final}}\})$ **then**
    $\{M_i^{\text{final}}\} \leftarrow \{M_i^j\}$
  end if
end while
Build a multi-step model using $\{M_i^{\text{final}}\}$

B Details of Dataset

The input/output formats of Task 1 and Task 2 is summarized in Tables 1 and 2, respectively. Calorimeter/tracker information is given as 16x16 pixel images as shown in Figs. 8.

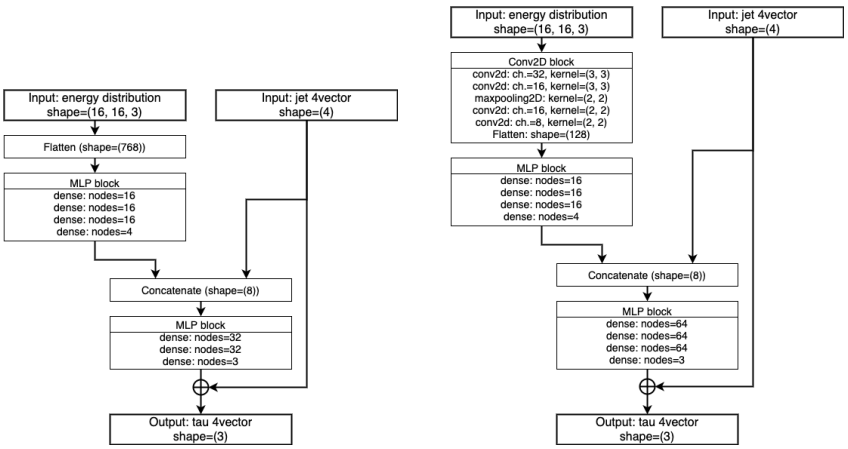


(a) Tracker (b) EM calo. (c) Hadronic calo.

(d) Tracker (e) EM calo. (f) Hadronic calo.

Figure 8. Distributions of input data in Task 1: (a-c) average over all events and (d-f) a single event.

C Details of Task 1 Models



(a) MLP for Task 1 (b) CNN for Task 1

Figure 9. Models architecture of (a) MLP model and (b) CNN model used in Task 1. Both models consist of two blocks: image feature extraction and correction factor evaluation.

[8]The energy and 3-momentum $\vec{p}$ of a particle form a Lorentz vector (E, p_x, p_y, p_z) and can be converted into $p_T = \sqrt{p_x^2 + p_y^2}$, $\eta = -0.5 \ln(1 - \cos\theta)/(1 + \cos\theta)$, $\cos\theta = p_z/|\vec{p}|$, $\phi = \texttt{atan2}(p_y, p_x)$, $m = \sqrt{E^2 - p_x^2 - p_y^2 - p_z^2}$

[9]This variable is close to $+\infty$ for H but $-\infty$ for Z.