Chapter 7

Towards Automatically-Tuned Deep Neural Networks

Hector Mendoza and Aaron Klein and Matthias Feurer and Jost Tobias Springenberg and Matthias Urban and Michael Burkart and Max Dippel and Marius Lindauer and Frank Hutter

Abstract

Recent advances in AutoML have led to automated tools that can compete with machine learning experts on supervised learning tasks. In this work, we present two versions of Auto-Net, which provide automatically-tuned deep neural networks without any human intervention. The first version, Auto-Net 1.0, builds upon ideas from the competition-winning system Auto-sklearn by using the Bayesian Optimization method SMAC and uses Theano as the underlying deep learning (DL) framework. The more recent Auto-Net 2.0 builds upon a recent combination of Bayesian Optimization and HyperBand, called BOHB, and uses PyTorch as DL framework. To the best of our knowledge, Auto-Net 1.0 was the first automatically-tuned neural network to win competition datasets against human experts (as part of the first AutoML challenge). Further empirical results show that ensembling Auto-Net 1.0 with Auto-sklearn can perform better than either approach alone, and that Auto-Net 2.0 can perform better yet.

7.1 Introduction

Neural networks have significantly improved the state of the art on a variety of benchmarks in recent years and opened many new promising research avenues [20, 34, 36, 24, 31]. However, neural networks are not easy to use for non-experts

since their performance crucially depends on proper settings of a large set of hyperparameters (e.g., learning rate and weight decay) and architecture choices (e.g., number of layers and type of activation functions). Here, we present work towards effective off-the-shelf neural networks based on approaches from automated machine learning (AutoML).

AutoML aims to provide effective off-the-shelf learning systems to free experts and non-experts alike from the tedious and time-consuming tasks of selecting the right algorithm for a dataset at hand, along with the right preprocessing method and the various hyperparameters of all involved components. Thornton et al [38] phrased this AutoML problem as a combined algorithm selection and hyperparameter optimization (CASH) problem, which aims to identify the combination of algorithm components with the best (cross-)validation performance.

One powerful approach for solving this CASH problem treats this cross-validation performance as an expensive blackbox function and uses Bayesian optimization [4, 30] to search for its optimizer. While Bayesian optimization typically uses Gaussian processes [28], these tend to have problems with the special characteristics of the CASH problem (high dimensionality; both categorical and continuous hyperparameters; many conditional hyperparameters, which are only relevant for some instantiations of other hyperparameters). Adapting GPs to handle these characteristics is an active field of research [35, 39], but so far Bayesian optimization methods using tree-based models [16, 2] work best in the CASH setting [38, 9].

Auto-Net is modelled after the two prominent AutoML systems Auto-WEKA [38] and Auto-sklearn [11], discussed in Chapters 4 and 6 of this book, respectively. Both of these use the random-forest-based Bayesian optimization method SMAC [16] to tackle the CASH problem – to find the best instantiation of classifiers in WEKA [15] and scikit-learn [27], respectively. Auto-sklearn employs two additional methods to boost performance. Firstly, it uses meta-learning [3] based on experience on previous datasets to start SMAC from good configurations [12]. Secondly, since the eventual goal is to make the best predictions, it is wasteful to try out dozens of machine learning models and then only use the single best model; instead, Auto-sklearn saves all models evaluated by SMAC and constructs an ensemble of these with the ensemble selection technique [5]. Even though both Auto-WEKA and Auto-sklearn include a wide range of supervised learning methods, neither includes modern neural networks.

Here, we introduce two versions of a system we dub *Auto-Net* to fill this gap. Auto-Net 1.0 is based on Theano and has a relatively simple search space, while the more recent Auto-Net 2.0 is implemented in PyTorch and uses a more complex space and more recent advances in DL. A further difference lies in their respective search procedure: Auto-Net 1.0 automatically configures neural networks with SMAC [16], following the same AutoML approach as Auto-WEKA and Auto-sklearn, while Auto-Net 2.0 builds upon BOHB [10], a combination of Bayesian Optimization (BO) and efficient racing strategies via HyperBand (HB) [21].

Auto-Net 1.0 achieved the best performance on two datasets in the human expert track of the recent *ChaLearn AutoML Challenge* [14]. To the best of

our knowledge, this is the first time that a fully-automatically-tuned neural network won a competition dataset against human experts. Auto-Net 2.0 further improves upon Auto-Net 1.0 on large data sets, showing recent progress in the field.

We describe the configuration space and implementation of Auto-Net 1.0 in Section 7.2 and of Auto-Net 2.0 in Section 7.3. We then study their performance empirically in Section 7.4 and conclude in Section 7.5. We omit a thorough discussion of related work and refer to Chapter 3 of this book for an overview on the extremely active field of neural architecture search. Nevertheless, we note that several other recent tools follow Auto-Net's goal of automating deep learning, such as Auto-Keras [18], Photon-AI, H2O.ai, DEvol or Google's Cloud AutoML service.

7.2 Auto-Net 1.0

We now introduce Auto-Net 1.0 and describe its implementation. We chose to implement this first version of Auto-Net as an extension of Auto-sklearn [11] by adding a new classification (and regression) component; the reason for this choice was that it allows us to leverage existing parts of the machine learning pipeline: feature preprocessing, data preprocessing and ensemble construction. Here, we limit Auto-Net to fully-connected feed-forward neural networks, since they apply to a wide range of different datasets; we defer the extension to other types of neural networks, such as convolutional or recurrent neural networks, to future work. To have access to neural network techniques we use the Python deep learning library Lasagne [6], which is built around Theano [37]. However, we note that in general our approach is independent of the neural network implementation.

Following [2] and [7], we distinguish between layer-independent *network hyperparameters* that control the architecture and training procedure and *per-layer hyperparameters* that are set for each layer. In total, we optimize 63 hyperparameters (see Table 7.1), using the same configuration space for all types of supervised learning (binary, multiclass and multilabel classification, as well as regression). Sparse datasets also share the same configuration space. (Since neural networks cannot handle datasets in sparse representation out of the box, we transform the data into a dense representation on a per-batch basis prior to feeding it to the neural network.)

The per-layer hyperparameters of layer k are conditionally dependent on the number of layers being at least k. For practical reasons, we constrain the number of layers to be between one and six: firstly, we aim to keep the training time of a single configuration low[1], and secondly each layer adds eight per-layer hyperparameters to the configuration space, such that allowing additional layers would further complicate the configuration process.

[1]We aimed to be able to afford the evaluation of several dozens of configurations within a time budget of two days on a single CPU.

	Name	Range	Default	log scale	Type	Conditional
Network hyperparameters	batch size	$[32, 4096]$	32	✓	float	-
	number of updates	$[50, 2500]$	200	✓	int	-
	number of layers	$[1, 6]$	1	-	int	-
	learning rate	$[10^{-6}, 1.0]$	10^{-2}	✓	float	-
	L_2 regularization	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	-
	dropout output layer	$[0.0, 0.99]$	0.5	✓	float	-
	solver type	{SGD, Momentum, Adam, Adadelta, Adagrad, smorm, Nesterov }	smorm3s	-	cat	-
	lr-policy	{Fixed, Inv, Exp, Step}	fixed	-	cat	-
Conditioned on solver type	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	$[0.05, 0.99]$	0.95	✓	float	✓
	momentum	$[0.3, 0.999]$	0.9	✓	float	✓
Conditioned on lr-policy	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	$[0.0, 1.0]$	0.5	-	float	✓
	s	$[2, 20]$	2	-	int	✓
Per-layer hyperparameters	activation-type	{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}	ReLU	-	cat	✓
	number of units	$[64, 4096]$	128	✓	int	✓
	dropout in layer	$[0.0, 0.99]$	0.5	-	float	✓
	weight initialization	{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}	He-Normal	-	cat	✓
	std. normal init.	$[10^{-7}, 0.1]$	0.0005	-	float	✓
	leakiness	$[0.01, 0.99]$	$\frac{1}{3}$	-	float	✓
	tanh scale in	$[0.5, 1.0]$	$2/3$	-	float	✓
	tanh scale out	$[1.1, 3.0]$	1.7159	✓	float	✓

Table 7.1: Configuration space of Auto-Net. The configuration space for the preprocessing methods can be found in [11].

The most common way to optimize the internal weights of neural networks is via stochastic gradient descent (SGD) using partial derivatives calculated with backpropagation. Standard SGD crucially depends on the correct setting of the learning rate hyperparameter. To lessen this dependency, various algorithms (solvers) for stochastic gradient descent have been proposed. We include the following well-known methods from the literature in the configuration space of Auto-Net: vanilla stochastic gradient descent (SGD), stochastic gradient descent with momentum (Momentum), Adam [19], Adadelta [42], Nesterov momentum [25] and Adagrad [8]. Additionally, we used a variant of the vSGD optimizer from [29], dubbed "smorm", in which the estimate of the Hessian is replaced by an estimate of the squared gradient (calculated as in the RMSprop procedure). Each of these methods comes with a learning rate α and an own set of hyperparameters, for example Adam's momentum vectors β_1 and β_2. Each solver's hyperparameter(s) are only active if the corresponding solver is chosen.

We also decay the learning rate α over time, using the following policies (which multiply the initial learning rate by a factor α_{decay} after each epoch $t = 0 \dots T$):

- Fixed: $\alpha_{decay} = 1$
- Inv: $\alpha_{decay} = (1 + \gamma t)^{(-k)}$
- Exp: $\alpha_{decay} = \gamma^t$
- Step: $\alpha_{decay} = \gamma^{\lfloor t/s \rfloor}$

Here, the hyperparameters k, s and γ are conditionally dependent on the choice of the policy.

To search for a strong instantiation in this conditional search space of Auto-Net 1.0, as in Auto-WEKA and Auto-sklearn, we used the random-forest based

Bayesian optimization method SMAC [16]. SMAC is an anytime approach that keeps track of the best configuration seen so far and outputs this when terminated.

7.3 Auto-Net 2.0

AutoNet 2.0 differs from AutoNet 1.0 mainly in the following three aspects:

- it uses PyTorch [26] instead of Lasagne as a deep learning framework
- it uses a larger configuration space including up-to-date deep learning techniques, modern architectures (such as ResNets) and includes more compact representations of the search space, and
- it applies BOHB [10] instead of SMAC to obtain a well-performing neural network more efficiently.

In the following, we will discuss these points in more detail.

Since the development and maintenance of Theano ended last year, we chose a different Python framework for Auto-Net 2.0. The most popular deep learning frameworks right now are PyTorch [26] and Tensorflow [1]. These come with quite similar features and mostly differ in the level of detail they give insight into. For example, PyTorch offers the user the possibility to trace all computations during training. While there are advantages and disadvantages for each of these frameworks, we decided to use PyTorch because of its ability to dynamically construct computational graphs.

The search space of AutoNet 2.0 includes both hyperparameters for module selection (e.g. scheduler type, network architecture) and hyperparameters for each of the specific modules. It supports different deep learning modules, such as network type, learning rate scheduler, optimizer and regularization technique, as described below. Auto-Net 2.0 is also designed to be easily extended; users can add their own modules to the ones listed below.

Auto-Net 2.0 currently offers four different network types:

Multi-Layer Perceptrons This is a standard implementation of conventional MLPs extended by dropout layers [33]. Similar as in AutoNet 1.0, each layer of the MLP is parameterized (e.g., number of units and dropout rate).

Residual Neural Networks Following [41], these are deep neural networks that learn residual functions, with the difference that we use fully connected layers instead of convolutional ones. As is standard with ResNets, the architecture consists of M groups, each of which stacks N residual blocks in sequence. While the architecture of each block is fixed, the number M of groups, the number of blocks N per group, as well as the width of each group is determined by hyperparameters, as shown in Table 7.2.

Shaped Multi-Layer Perceptrons To avoid that every layer has its own hyperparameters (which is an inefficient representation to search), in shaped MLPs the overall shape of the layers is predetermined, e.g. as a funnel, long funnel, diamond, hexagon, brick, or triangle. We followed the shapes from `https://mikkokotila.github.io/slate/#shapes`; Ilya Loshchilov also proposed parameterization by such shapes to us before [23].

Shaped Residual Networks A ResNet where the overall shape of the layers is predetermined (e.g. funnel, long funnel, diamond, hexagon, brick, triangle).

The network types of ResNets and ShapedResNets can also use any of the regularization methods of Shake-Shake [13] and ShakeDrop [40]. MixUp [43] can be used for all networks.

The optimizers currently supported in Auto-Net 2.0 are Adam [19] and SGD with momentum. Moreover, Auto-Net 2.0 currently offers five different schedulers that change the optimizer's learning rate over time (as a function of the number of epochs):

Exponential This multiplies the learning rate with a constant factor in each epoch.

Step This decays the learning rate by a multiplicative factor after a constant number of steps.

Cyclic This modifies the learning rate in a certain range, alternating between increasing and decreasing [32].

Cosine Annealing with Warm Restarts [22] This learning rate schedule implements multiple phases of convergence. It cools down the learning rate to zero following a cosine decay [22], and after each convergence phase heats it up to start a next phase of convergence, often to a better optimum. The network weights are not modified when heating up the learning rate, such that the next phase of convergence is warm-started.

OnPlateau This scheduler[2] changes the learning rate whenever a metric stops improving; specifically, it multiplies the current learning rate with a factor γ if there was no improvement after p epochs.

In contrast to Auto-Net 1.0, Auto-Net 2.0 does not (yet) search over preprocessing techniques and does not build an ensemble at the end. Users can, however, specify the pre-processing techniques to be used and can also choose between different balancing and normalization strategies (for balancing strategies, resampling and weighting the loss are available, and for normalization strategies, min-max normalization and standardization are supported). In future versions of Auto-Net 2.0, we will again include a search over pre-processors. If there are more than 500 features we use univariate feature selection to reduce

[2]Implemented by PyTorch

the number of features to 500, using χ^2 if all values in the feature matrix are non-negative and mutual information otherwise.

All hyperparameters of Auto-Net 2.0 with their respective ranges and default values can be found in Table 7.2.

	Name	Range	Default	log scale	Type	Conditional
Worker	batch size	$[32, 500]$	32	✓	int	-
	use mixup	{True, False}	True	-	bool	-
	mixup alpha	$[0.0, 1.0]$	1.0	-	float	-
	network	{MLP, ResNet, ShapedMLP, ShapedResNet}	MLP	-	cat	-
	optimizer	{Adam, SGD}	Adam	-	cat	-
	learning rate scheduler	{Step, Exponential, OnPlateau, Cyclic, CosineAnnealing}	Step	-	cat	-
Networks						
MLP	activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	-	cat	✓
	num layers	$[1, 15]$	9	-	int	✓
	num units (for layer i)	$[10, 1024]$	100	✓	int	✓
	dropout (for layer i)	$[0.0, 0.5]$	0.25	-	int	✓
ResNet	activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	-	cat	✓
	residual block groups	$[1, 9]$	4	-	int	✓
	blocks per group	$[1, 4]$	2	-	int	✓
	num units (for group i)	$[128, 1024]$	200	✓	int	✓
	use dropout	{True, False}	True	-	bool	✓
	dropout (for group i)	$[0.0, 0.9]$	0.5	-	int	✓
	use shake drop	{True, False}	True	-	bool	✓
	use shake shake	{True, False}	True	-	bool	✓
	shake drop β_{max}	$[0.0, 1.0]$	0.5	-	float	✓
ShapedMLP	activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	-	cat	✓
	num layers	$[3, 15]$	9	-	int	✓
	max units per layer	$[10, 1024]$	200	✓	int	✓
	network shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	-	cat	✓
	max dropout per layer	$[0.0, 0.6$	0.2	-	float	✓
	dropout shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	-	cat	✓
Shaped ResNet	activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	-	cat	✓
	num layers	$[3, 9]$	4	-	int	✓
	blocks per layer	$[1, 4]$	2	-	int	✓
	use dropout	{True, False}	True	-	bool	✓
	max units per layer	$[10, 1024]$	200	✓	int	✓
	network shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	-	cat	✓
	max dropout per layer	$[0.0, 0.6$	0.2	-	float	✓
	dropout shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	-	cat	✓
	use shake drop	{True, False}	True	-	bool	✓
	use shake shake	{True, False}	True	-	bool	✓
	shake drop β_{max}	$[0.0, 1.0]$	0.5	-	float	✓
Optimizers						
Adam	learning rate	$[0.0001, 0.1]$	0.003	✓	float	✓
	weight decay	$[0.0001, 0.1]$	0.05	-	float	✓
SGD	learning rate	$[0.0001, 0.1]$	0.003	✓	float	✓
	weight decay	$[0.0001, 0.1]$	0.05	-	float	✓
	momentum	$[0.1, 0.9]$	0.3	✓	float	✓
Schedulers						
Step	γ	$[0.001, 0.9]$	0.4505	-	float	✓
	step size	$[1, 10]$	6	-	int	✓
Exponential	γ	$[0.8, 0.9999]$	0.89995	-	float	✓
OnPlateau	γ	$[0.05, 0.5]$	0.275	-	float	✓
	patience	$[3, 10]$	6	-	int	✓
Cyclic	cycle length	$[3, 10]$	6	-	int	✓
	max factor	$[1.0, 2.0]$	1.5	-	float	✓
	min factor	$[0.001, 1.0]$	0.5	-	float	✓
Cosine Annealing	T_0	$[1, 20]$	10	-	int	✓
	T_{mult}	$[1.0, 2.0]$	1.5	-	float	✓

Table 7.2: Configuration space of Auto-Net 2.0.

As optimizer for this highly conditional space, we used BOHB (**B**ayesian **O**ptimization with **H**yper**B**and) [10], which combines conventional Bayesian optimization with the bandit-based strategy Hyperband [21] to substantially improve its efficiency. Like Hyperband, BOHB uses repeated runs of Successive Halving [17] to invest most runtime in promising neural networks and stops training neural networks with poor performance early. Like in Bayesian optimization, BOHB learns which kinds of neural networks yield good results.

Algorithm 2 Example Usage of Auto-Net 2.0

```
from autonet import AutoNetClassification

cls = AutoNetClassification(min_budget=5, max_budget=20,
max_runtime=120)
cls.fit(X_train, Y_train)
predictions = cls.predict(X_test)
```

Specifically, like the BO method TPE [2], BOHB uses a kernel density estimator (KDE) to describe regions of high performance in the space of neural networks (architectures and hyperparameter settings) and trades off exploration versus exploitation using this KDE. One of the advantages of BOHB is that it is easily parallelizable, achieving almost linear speedups with an increasing number of workers [10].

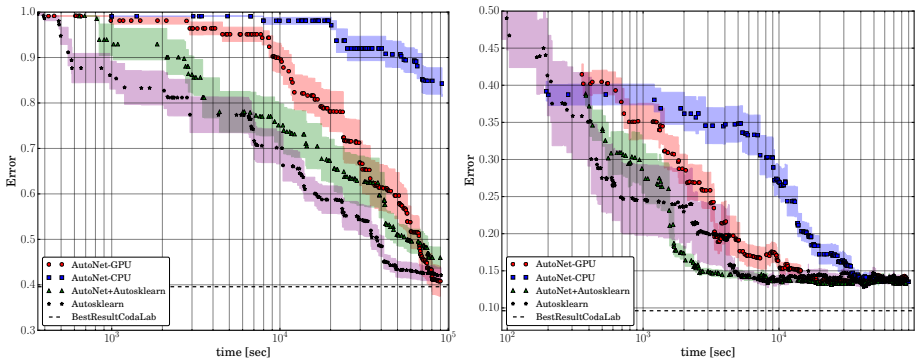
As budget for BOHB we can either handle epochs or (wallclock) time in minutes. While by default we use runtime, the user can freely choose the different budget parameters. An example usage is shown in Algorithm 2. Similar to Auto-sklearn, Auto-Net is build as a plugin estimator for scikit-learn. Users have to provide a training set and a performance metric (e.g., accuracy). Optionally, they might specify a validation and testset. The validation set is used during training to get a measure for the performance of the network and to train the KDE models of BOHB.

7.4 Experiments

We now empirically evaluate our methods. Our implementations of Auto-Net run on both CPUs and GPUs, but since neural networks heavily employ matrix operations they run much faster on GPUs. Our CPU-based experiments were run on a compute cluster, each node of which has two eight-core Intel Xeon E5-2650 v2 CPUs, running at 2.6GHz, and a shared memory of 64GB. Our GPU-based experiments were run on a compute cluster, each node of which has four GeForce GTX TITAN X GPUs.

7.4.1 Baseline Evaluation of Auto-Net 1.0 and Auto-sklearn

In our first experiment, we compare different instantiations of Auto-Net 1.0 on the five datasets of phase 0 of the AutoML challenge. First, we use the CPU-based and GPU-based versions to study the difference of running NNs on different hardware. Second, we allow the combination of neural networks with the models from Auto-sklearn. Third, we also run Auto-sklearn without neural networks as a baseline. On each dataset, we performed 10 one-day runs of each method, allowing up to 100 minutes for the evaluation of a single configuration by 5-fold cross-validation on the training set. For each time step of each run, following [11] we constructed an ensemble from the models it had evaluated so



(a) `newsgroups` dataset (b) `dorothea` dataset

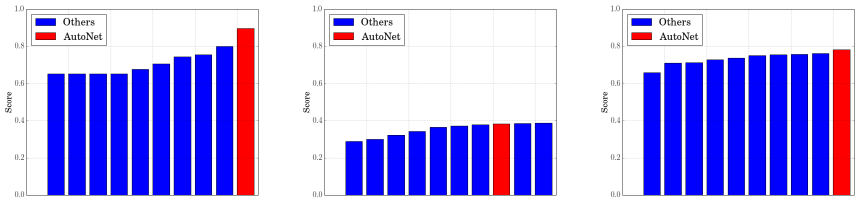
Figure 7.1: Results for the 4 methods on 2 datasets from *Tweakathon0* of the AutoML challenge. We show errors on the competition's validation set (not the test set since its true labels are not available), with our methods only having access to the training set. To avoid clutter, we plot mean error ± 1/4 standard deviations over the 10 runs of each method.

far and plot the test error of that ensemble over time. In practice, we would either use a separate process to calculate the ensembles in parallel or compute them after the optimization process.

Figure 7.1 shows the results on two of the five datasets. First, we note that the GPU-based version of Auto-Net was consistently about an order of magnitude faster than the CPU-based version. Within the given fixed compute budget, the CPU-based version consistently performed worst, whereas the GPU-based version performed best on the `newsgroups` dataset (see Figure 7.1(a)), tied with Auto-sklearn on 3 of the other datasets, and performed worse on one. Despite the fact that the CPU-based Auto-Net was very slow, in 3/5 cases the combination of Auto-sklearn and CPU-based Auto-Net still improved over Auto-sklearn; this can, for example, be observed for the `dorothea` dataset in Figure 7.1(b).

7.4.2 Results for AutoML Competition Datasets

Having developed Auto-Net 1.0 during the first AutoML challenge, we used a combination of Auto-sklearn and GPU-based Auto-Net for the last two phases to win the respective human expert tracks. Auto-sklearn has been developed for much longer and is much more robust than Auto-Net, so for 4/5 datasets in the 3rd phase and 3/5 datasets in the 4th phase Auto-sklearn performed best by itself and we only submitted its results. Here, we discuss the three datasets for which we used Auto-Net. Figure 7.2 shows the official AutoML human expert track competition results for the three datasets for which we used Auto-Net. The `alexis` dataset was part of the 3rd phase ("advanced phase") of the



(a) `alexis` dataset (b) `yolanda` dataset (c) `tania` dataset

Figure 7.2: Official AutoML human expert track competition results for the three datasets for which we used Auto-Net. We only show the top 10 entries.

challenge. For this, we ran Auto-Net on five GPUs in parallel (using SMAC in shared-model mode) for 18 hours. Our submission included an automatically-constructed ensemble of 39 models and clearly outperformed all human experts, reaching an AUC score of 90%, while the best human competitor (Ideal Intel Analytics) only reached 80%. To our best knowledge, this is the first time an automatically-constructed neural network won a competition dataset. The `yolanda` and `tania` datasets were part of the 4th phase ("expert phase") of the challenge. For `yolanda`, we ran Auto-Net for 48 hours on eight GPUs and automatically constructed an ensemble of five neural networks, achieving a close third place. For `tania`, we ran Auto-Net for 48 hours on eight GPUs along with Auto-sklearn on 25 CPUs, and in the end our automated ensembling script constructed an ensemble of eight 1-layer neural networks, two 2-layer neural networks, and one logistic regression model trained with SGD. This ensemble won the first place on the `tania` dataset.

For the `tania` dataset, we also repeated the experiments from Section 7.4.1. Figure 7.3 shows that for this dataset Auto-Net performed clearly better than Auto-sklearn, even when only running on CPUs. The GPU-based variant of Auto-Net performed best.

7.4.3 Comparing AutoNet 1.0 and 2.0

Finally, we show an illustrative comparison between Auto-Net 1.0 and 2.0. We note that Auto-Net 2.0 has a much more comprehensive search space than Auto-Net 1.0, and we therefore expect it to perform better on large datasets given enough time. We also expect that searching the larger space is harder than searching Auto-Net 1.0's smaller space; however, since Auto-Net 2.0 uses the efficient multi-fidelity optimizer BOHB to terminate poorly-performing neural networks early on, it may nevertheless obtain strong anytime performance. On the other hand, Auto-Net 2.0 so far does not implement ensembling, and due to this missing regularization component and its larger hypothesis space, it may be more prone to overfitting than Auto-Net 1.0.

In order to test these expectations about performance on different-sized

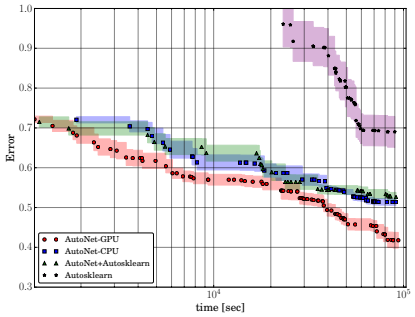


Figure 7.3: Performance on the `tania` dataset over time. We show cross-validation performance on the training set since the true labels for the competition's validation or test set are not available. To avoid clutter, we plot mean error ± 1/4 standard deviations over the 10 runs of each method.

	`newsgroups`			`dorothea`		
	10^3 sec	10^4 sec	1 day	10^3 sec	10^4 sec	1 day
Auto-Net 1.0	0.99	0.98	0.85	0.38	0.30	0.13
Auto-sklearn + Auto-Net 1.0	0.94	0.76	0.47	0.29	0.13	0.13
Auto-Net 2.0: 1 worker	1.0	0.67	0.55	0.22	0.18	0.22
Auto-Net 2.0: 4 workers	0.89	0.57	0.44	0.18	0.22	0.22

Table 7.3: Error metric of different Auto-Net versions, run for different times, all on CPU. We compare Auto-Net 1.0, ensembles of Auto-Net 1.0 and Auto-sklearn, Auto-Net 2.0 with one worker, and Auto-Net 2.0 with four workers. All results are means across 10 runs of each system. We show errors on the competition's validation set (not the test set since its true labels are not available), with our methods only having access to the training set.

datasets, we used a medium-sized dataset (`newsgroups`, with 13k training data points) and a small one (`dorothea`, with 800 training data points). The results are presented in Table 7.3.

On the medium-sized dataset `newsgroups`, Auto-Net 2.0 performed much better than Auto-Net 1.0, and using four workers also led to strong speedups on top of this, making Auto-Net 2.0 competitive to the ensemble of Auto-sklearn and Auto-Net 1.0. We found that despite Auto-Net 2.0's larger search space its anytime performance (using the multi-fidelity method BOHB) was better than that of Auto-Net 1.0 (using the blackbox optimization method SMAC).

On the small dataset `dorothea`, Auto-Net 2.0 also performed better than Auto-Net 1.0 early on, but in contrast to Auto-Net 1.0, its performance did not improve over time but rather even worsened somewhat. We observed that the performance criterion optimized by BOHB (5-fold cross-validation on the training set, not shown) consistently improved over time, but this result did not

generalize to new data. This experiment illustrates that Auto-Net 2.0 is indeed more prone to overfitting than Auto-Net 1.0; we attribute this to its larger search space and lack of ensembling. One of the next steps for Auto-Net 2.0 will therefore be to study further regularization techniques to avoid over-fitting on small data sets.

7.5 Conclusion

We presented Auto-Net, which provides automatically-tuned deep neural networks without any human intervention. Even though neural networks show superior performace on many datasets, for traditional data sets with manually-defined features they do not always perform best. However, we showed that, even in cases where other methods perform better, combining Auto-Net with Auto-sklearn to an ensemble often leads to an equal or better performance than either approach alone.

Finally, we reported results on three datasets from the AutoML challenge's human expert track, for which Auto-Net won one third place and two first places. We showed that ensembles out of Auto-sklearn and Auto-Net can get users the best of both worlds and quite often improve over the individual tools. First experiments on the new Auto-Net 2.0 showed that using a more comprehensive search space, combined with BOHB as an optimizer yields promising results. However, Auto-Net 2.0 does not yet use ensembling, making it more prone to over-fitting on small data sets.

In future work, we aim to extend Auto-Net to more general neural network architectures, including convolutional and recurrent neural networks.

Acknowledgements

This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 265–283 (2016), `https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf`

[2] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P.,

Pereira, F., Weinberger, K. (eds.) Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11). pp. 2546–2554 (2011)

[3] Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer Publishing Company, Incorporated, 1 edn. (2008)

[4] Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Computing Research Repository (CoRR) abs/1012.2599 (2010)

[5] Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: In Proceedings of the 21st International Conference on Machine Learning. pp. 137–144. ACM Press (2004)

[6] Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., Fauw, J.D., Heilman, M., diogo149, McFee, B., Weideman, H., takacsg84, peterderivaz, Jon, instagibbs, Rasul, K., CongLiu, Britefury, Degrave, J.: Lasagne: First release. (Aug 2015), `http://dx.doi.org/10.5281/zenodo.27878`

[7] Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 3460–3468 (2015)

[8] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12, 2121–2159 (Jul 2011)

[9] Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013)

[10] Falkner, S., Klein, A., Hutter, F.: Combining hyperband and bayesian optimization. In: NIPS 2017 Bayesian Optimization Workshop (Dec 2017)

[11] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NIPS'15) (2015)

[12] Feurer, M., Springenberg, T., Hutter, F.: Initializing Bayesian hyperparameter optimization via meta-learning. In: Bonet, B., Koenig, S. (eds.)

Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15). pp. 1128–1135. AAAI Press (2015)

[13] Gastaldi, X.: Shake-shake regularization. CoRR abs/1705.07485 (2017)

[14] Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macià, N., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: Design of the 2015 chalearn automl challenge. In: 2015 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (July 2015)

[15] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)

[16] Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer-Verlag (2011)

[17] Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton, A., Robert, C. (eds.) Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS. JMLR Workshop and Conference Proceedings, vol. 51, pp. 240–248. JMLR.org (2016)

[18] Jin, H., Song, Q., Hu, X.: Efficient neural architecture search with network morphism. CoRR abs/1806.10282 (2018)

[19] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations (2015)

[20] Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12). pp. 1097–1105 (2012)

[21] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research 18, 185:1–185:52 (2017)

[22] Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: International Conference on Learning Representations (ICLR) 2017 Conference Track (2017)

[23] Loshchilov, I.: Personal communication (2017)

[24] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015)

[25] Nesterov, Y.: A method of solving a convex programming problem with convergence rate O(1/sqr(k)). Soviet Mathematics Doklady 27, 372–376 (1983)

[26] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)

[27] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)

[28] Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. The MIT Press (2006)

[29] Schaul, T., Zhang, S., LeCun, Y.: No More Pesky Learning Rates. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML'13). Omnipress (2014)

[30] Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A Review of Bayesian Optimization. Proc. of the IEEE (1) (12/2015 2016)

[31] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature 529, 484–503 (2016)

[32] Smith, L.N.: Cyclical learning rates for training neural networks. In: Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on. pp. 464–472. IEEE (2017)

[33] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)

[34] Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. CoRR abs/1409.3215 (2014), `http://arxiv.org/abs/1409.3215`

[35] Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In: NIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013)

[36] Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'14). pp. 1701–1708. IEEE Computer Society Press (2014)

[37] Theano Development Team: Theano: A Python framework for fast computation of mathematical expressions. Computing Research Repository (CoRR) abs/1605.02688 (may 2016)

[38] Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: I.Dhillon, Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R. (eds.) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13). pp. 847–855. ACM Press (2013)

[39] Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Feitas, N.: Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research 55, 361–387 (2016)

[40] Yamada, Y., Iwamura, M., Kise, K.: Shakedrop regularization. CoRR abs/1802.02375 (2018)

[41] Zagoruyko, S., Komodakis, N.: Wide residual networks. CoRR abs/1605.07146 (2016)

[42] Zeiler, M.: ADADELTA: an adaptive learning rate method. CoRR abs/1212.5701 (2012), http://arxiv.org/abs/1212.5701

[43] Zhang, H., Cissé, M., Dauphin, Y., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. CoRR abs/1710.09412 (2017)