Towards Automatically-Tuned Neural Networks

Hector Mendoza
MENDOZAH@CS.UNI-FREIBURG.DE
Aaron Klein
KLEINAA@CS.UNI-FREIBURG.DE
Matthias Feurer
FEURERM@CS.UNI-FREIBURG.DE
Jost Tobias Springenberg
SPRINGJ@CS.UNI-FREIBURG.DE
Frank Hutter
FH@CS.UNI-FREIBURG.DE

Department of Computer Science, University of Freiburg

Abstract

Recent advances in AutoML have led to automated tools that can compete with machine learning experts on supervised learning tasks. However, current AutoML tools do not yet support modern neural networks effectively. In this work, we present a first version of Auto-Net, which provides automatically-tuned feed-forward neural networks without any human intervention. We report results on datasets from the recent AutoML challenge showing that ensembling Auto-Net with Auto-sklearn often performs better than either alone and report the first results on winning competition datasets against human experts with automatically-tuned neural networks.

Keywords: Automated Machine Learning, Bayesian Optimization, Neural Networks

1. Introduction

Deep neural networks have improved the state of the art on a variety of benchmarks significantly during the last years and opened new promising research avenues (Krizhevsky et al., 2012; Sutskever et al., 2014; Taigman et al., 2014; Mnih et al., 2015; Silver et al., 2016). However, deep neural networks are not easy to use for non-experts since their performance crucially depends on proper settings of a large set of hyperparameters. 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. (2013) 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.

Current approaches for finding solutions to this CASH problem view evaluating the loss of a machine learning pipeline w.r.t. its hyperparameters as an expensive blackbox function and use Bayesian optimization to search for its minimizer. While Bayesian optimization typically uses Gaussian processes (Rasmussen and Williams, 2006), they cannot handle the specal 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). Bayesian optimization methods using tree-based models (Hutter et al., 2011; Bergstra et al., 2011) have been shown to work best in this setting (Thornton et al., 2013; Eggensperger et al., 2013).

Two prominent AutoML systems are Auto-WEKA (Thornton et al., 2013) and Auto-sklearn (Feurer et al., 2015a). Both of these use the random-forest-based Bayesian optimization method SMAC (Hutter et al., 2011) to tackle the CASH problem, to find the best instantiation of classifiers in WEKA (Hall et al., 2009) and scikit-learn (Pedregosa et al., 2011), respectively. On top of tackling the CASH problem with SMAC, Auto-sklearn also employs meta-learning (Brazdil et al., 2008) to start Bayesian optimization from good configurations (Feurer et al., 2015b) evaluated on previous similar datasets. Furthermore, Auto-sklearn constructs ensembles of the models evaluated by SMAC with the ensemble selection technique by Caruana et al. (2004) to reduce the risk of overfitting. Even though both Auto-WEKA and Auto-sklearn include a wide range of supervised learning methods, neither includes modern neural networks.

Here, we introduce a first version of Auto-Net to fill this gap, a system that automatically configures neural networks with SMAC by following the same AutoML approach as Auto-WEKA and Auto-sklearn. With an early version of Auto-Net, we achieved the best performance on two datasets in the human expert track of the recent ChaLearn AutoML Challenge (Guyon et al., 2015). To the best of our knowledge, this is the first time that a fully-automatically-tuned neural network won a competition dataset. We describe the configuration space and implementation of our neural network in Section 2, evaluate its performance in Section 3, and conclude in Section 4.

## 2. Auto-Net

We now introduce Auto-Net and describe its implementation. For this first version of Auto-Net, we have chosen to implement Auto-Net within Auto-sklearn (Feurer et al., 2015a) 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 consider the extension to other types of neural networks, such as convolutional or recurrent neural networks, as future work. To have access to state-of-the-art deep learning techniques we use the Python deep learning library Lasagne (Dieleman et al., 2015), which is built around Theano (Theano Development Team, 2016). However, we note that in general our approach is independent of the neural network implementation.

Following Bergstra et al. (2011) and Domhan et al. (2015), 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 1), using the same configuration space for all types of supervised learning (binary, multiclass and multilabel classification, as well as regression).

The per-layer hyperparameters of layer $k$ are conditionally dependent on the number of layers being at least $k$. In order to prevent that the configuration space becomes too big, we constrain the number of layers to be between 1 and 6 layers; each layer has the following
Towards Automatically-Tuned Neural Networks

per-layer hyperparameters: number of units, amount of dropout, type of activation function, and weight initialization.

The most common way to optimize the internal weights of neural networks (back propagation) is via stochastic gradient descent (SGD). 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 AutoNet: vanilla stochastic gradient descent (SGD), stochastic gradient descent with momentum (Momentum), Adam (Kingma and Ba, 2014), Adadelta (Zeiler, 2012), Nesterov momentum (Nesterov, 1983) and Adagrad (Duchi et al., 2011). Additionally, we used a variant of the vSGD optimizer from Schaul et al. (2014), 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 \( \alpha \) and an own set of hyperparameters, for example Adam’s momentum vectors \( \beta_1 \) and \( \beta_2 \). Each solver’s hyperparameter(s) are only active if the corresponding solver is chosen.

As it is common in deep learning, we also decay the learning rate \( \alpha \) over time to prevent jumping around a local optimum and to obtain better convergence (Goodfellow et al., 2016). We implemented different policies which multiply the initial learning rate by a factor \( \alpha_{\text{decay}} \) after each epoch \( t = 0 \ldots T \):

- Fixed: \( \alpha_{\text{decay}} = 1 \)
- Inv: \( \alpha_{\text{decay}} = (1 + \gamma t)^{(-k)} \)
- Exp: \( \alpha_{\text{decay}} = \gamma^t \)
- Step: \( \alpha_{\text{decay}} = \gamma^{[t/s]} \)

Here, the hyperparameters \( k, s \) and \( \gamma \) are conditionally dependent on the choice of the policy.

In contrast to other methods, neural networks cannot handle datasets in sparse representation out of the box. For that, we transform the data to a dense representation by replacing the sparse entries with zero values and then pass it to the network.

3. Experiments

We now empirically evaluate our methods. Since our implementation of Auto-Net is built on top of Theano, we can run it on both CPUs and GPUs. As neural networks heavily employ matrix operations, we expect them to run much faster on GPUs. The CPU-based experiments were run on a compute cluster, each node of which has two Intel Xeon E5-2650 v2 CPUs with 8 cores each, running at 2.6GHz, and a shared memory of 64GB. The GPU-based experiments were run on a compute cluster, each node of which has four GeForce GTX TITAN X GPUs.

3.1. Baseline evaluation of Auto-Net and Auto-sklearn

In our first experiment, we compare different instantiations of Auto-Net on the five datasets of phase 0 in 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
### Table 1: Hyperparameter configuration space of Auto-Net. The configuration space for the preprocessing methods can be found in Feurer et al. (2015a).

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Default</th>
<th>log scale</th>
<th>Type</th>
<th>Conditional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network hyperparameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>batch size</td>
<td>[32, 4096]</td>
<td>32</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>number of updates</td>
<td>[50, 2,000]</td>
<td>200</td>
<td>✓</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>number of layers</td>
<td>[1, 6]</td>
<td>1</td>
<td>-</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>learning rate</td>
<td>[10^{-5}, 1.0]</td>
<td>10^{-2}</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>L2 regularization</td>
<td>[10^{-7}, 10^{-2}]</td>
<td>10^{-4}</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>dropout output layer</td>
<td>[0.0, 0.99]</td>
<td>0.5</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>solver type</td>
<td>(SGD, Momentum, Adam, AdaDelta, Adagrad, smorm, Nesterov)</td>
<td>smorm3s</td>
<td>-</td>
<td>cat</td>
<td></td>
</tr>
<tr>
<td>lr-policy</td>
<td>(Fixed, Inv, Exp, Step)</td>
<td>fixed</td>
<td>-</td>
<td>cat</td>
<td></td>
</tr>
<tr>
<td><strong>Conditioned on solver type</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>β1</td>
<td>[10^{-4}, 10^{-1}]</td>
<td>10^{-1}</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>β2</td>
<td>[10^{-4}, 10^{-1}]</td>
<td>10^{-1}</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>ρ</td>
<td>[0.05, 0.99]</td>
<td>0.95</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>momentum</td>
<td>[0.3, 0.999]</td>
<td>0.9</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>γ</td>
<td>[10^{-3}, 10^{-1}]</td>
<td>10^{-2}</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>[0.0, 1.0]</td>
<td>0.5</td>
<td>-</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>[2, 20]</td>
<td>2</td>
<td>-</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td><strong>Per-layer hyperparameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>activation-type</td>
<td>{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}</td>
<td>ReLU</td>
<td>-</td>
<td>cat</td>
<td></td>
</tr>
<tr>
<td>number of units</td>
<td>[64, 4096]</td>
<td>128</td>
<td>✓</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>dropout in layer</td>
<td>[0.0, 0.99]</td>
<td>0.5</td>
<td>-</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>weight initialization</td>
<td>{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}</td>
<td>He-Normal</td>
<td>-</td>
<td>cat</td>
<td></td>
</tr>
<tr>
<td>std. normal init.</td>
<td>[10^{-7}, 0.4]</td>
<td>0.0005</td>
<td>-</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>leakiness</td>
<td>[0.01, 0.99]</td>
<td>(\frac{1}{2})</td>
<td>-</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>tanh scale in</td>
<td>[0.5, 1.0]</td>
<td>2/3</td>
<td>-</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>tanh scale out</td>
<td>[1, 3.0]</td>
<td>1.7159</td>
<td>✓</td>
<td>float</td>
<td></td>
</tr>
</tbody>
</table>

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 Feurer et al. (2015a) we constructed an ensemble from the models it had evaluated so 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 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 1), tied with Auto-sklearn on 3 of the other datasets, and performed worse in 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 in the dorothea dataset in Figure 1.

### 3.2. Results for AutoML competition datasets

Having developed Auto-Net during the AutoML challenge, we used a combination of Auto-sklearn and GPU-based Auto-Net for the last 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 4th phase and 3/5 datasets in the 5th 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.
Towards Automatically-Tuned Neural Networks

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 ± 1/4 standard deviations over the 10 runs of each method.

Figure 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 ± 1/4 standard deviations over the 10 runs of each method.

Figure 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 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 dataset were part of the 4th phase (“expert phase”) of the challenge. For yolanda, we ran Auto-Net for 48 hours on 256 CPUs and automatically constructed an ensemble of five neural networks, achieving a close third place. For tania, we ran Auto-Net for 48 hours on 256 CPUs along with Auto-sklearn 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 3.1. Figure 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.

4. Conclusion

We presented Auto-Net, which provides automatically-tuned feed-forward neural networks without any human intervention. Even though neural networks show superior performance on some datasets, for traditional data sets with manually-defined features they do not
Figure 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.

Figure 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 ± 1/4 standard deviations over the 10 runs of each method.

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 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.

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

References


