JMLR: Workshop and Conference Proceedings 64:58–65, 2016 ICML 2016 AutoML Workshop

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 can perform better than either approach 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

Neural networks have improved the state of the art on a variety of benchmarks significantly during the last years and opened many 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, 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.

The currently best-performing approaches treat this cross-validation performance as an expensive blackbox function and use Bayesian optimization (Brochu et al., 2010; Shahriari et al., 2016) to search for its optimizer. While Bayesian optimization typically uses Gaussian processes (GPs; Rasmussen and Williams, 2006), 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 (Swersky et al., 2013; Wang et al., 2016), but so far Bayesian optimization methods using tree-based models (Hutter et al., 2011; Bergstra et al., 2011) work best in the CASH 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. Auto-sklearn employs two additional methods to boost performance. Firstly, it uses meta-learning (Brazdil et al., 2008) based on experience on previous datasets to start SMAC from good configurations (Feurer et al., 2015b). 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 by Caruana et al. (2004). 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 against human experts.

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 modern neural network 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). 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.

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 (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 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.

As it is common in the training of neural networks, we also decay the learning rate α 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 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.

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, 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.

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

	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 1: Hyperparameter configuration space of Auto-Net. The configuration space for the preprocessing methods can be found in Feurer et al. (2015a).

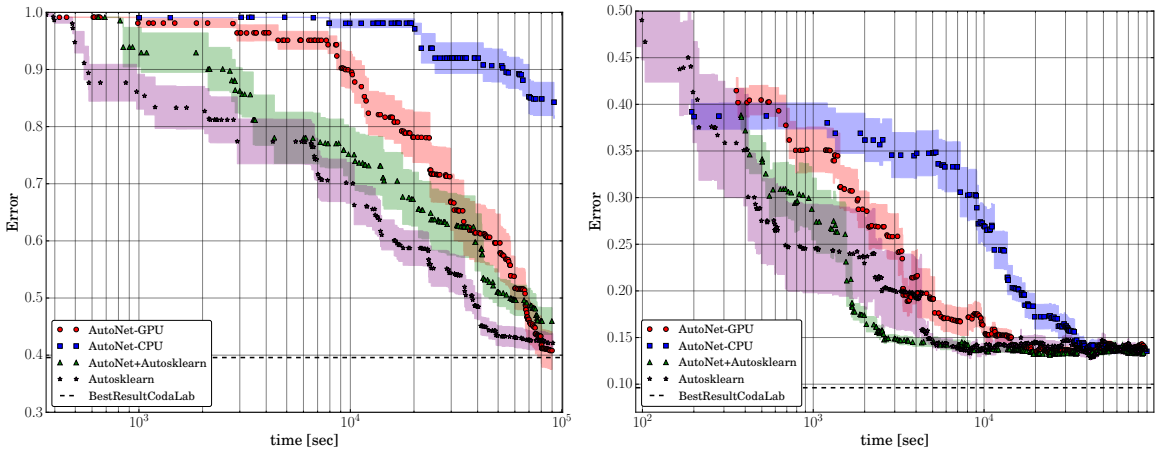
3.1. Baseline evaluation of Auto-Net and Auto-sklearn

In our first experiment, we compare different instantiations of Auto-Net on the five datasets of phase 0 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 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(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 1(b).

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 two phases to win the respective human



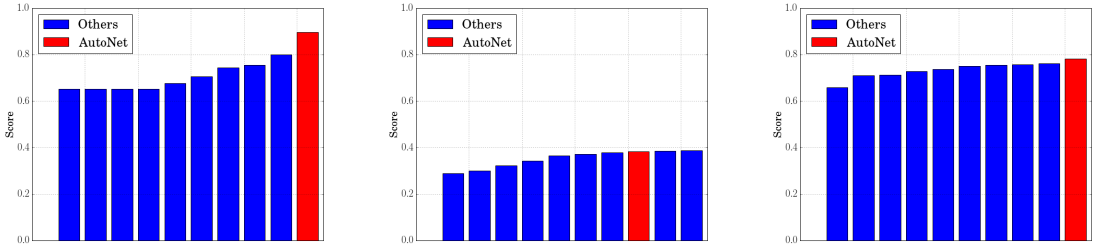
(*a*) `newsgroups` dataset (*b*) `dorothea` dataset

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

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 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 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 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.



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

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.

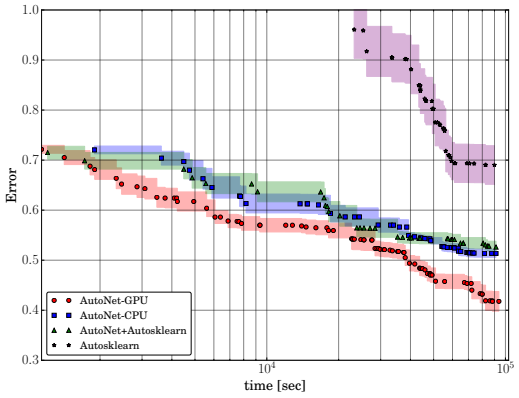


Figure 3: Performance on the `tania` dataset over time. We show cross-validation performance on the training set since the true labels for the competition's validation or test set are not available. To avoid clutter, we plot mean error $\pm$ 1/4 standard deviations over the 10 runs of each method.

4. Conclusion

We presented Auto-Net, which provides automatically-tuned feed-forward neural networks without any human intervention. Even though neural networks show superior performace on some 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.

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

References

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*, pages 2546–2554, 2011.

P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008.

E. Brochu, V. Cora, and N. de Freitas. 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.

R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *In Proceedings of the 21st International Conference on Machine Learning*, pages 137–144. ACM Press, 2004.

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

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

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

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

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

M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, pages 1128–1135. AAAI Press, 2015b.

I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 chalearn automl challenge. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. doi: 10.1109/IJCNN.2015.7280767.

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

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

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, 2014.

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

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

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

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

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

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

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

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

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. Osborne. 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.

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. 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)*, pages 1701–1708. IEEE Computer Society Press, 2014.

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

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

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

M. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.