Transfer Learning with Neural AutoML

Catherine Wong
Stanford University
catwong@cs.stanford.edu

Neil Houlsby
Google
neilhoulsby@google.com

Yifeng Wu
Google
yifenglu@google.com

Andrea Gesmundo
Google
agesmundo@google.com

Abstract

We reduce the computational cost of Neural AutoML with transfer learning. AutoML relieves human effort by automating the design of ML algorithms. Neural AutoML has become popular for the design of deep learning architectures, however, this method has a high computation cost.To address this we propose Transfer Neural AutoML that uses knowledge from prior tasks to speed up network design. We extend RL-based architecture search methods to support parallel training on multiple tasks and then transfer the search strategy to new tasks. On language and image classification data, Transfer Neural AutoML reduces convergence time over single-task training by over an order of magnitude on many tasks.

1 Introduction

Automatic Machine Learning (AutoML) aims to find the best performing learning algorithms with minimal human intervention. Many AutoML methods exist, including random search [1], performance modelling [2, 3], Bayesian optimization [4], genetic algorithms [5, 6] and RL [7, 8]. We focus on neural AutoML that uses deep RL to optimize architectures. These methods have shown promising results. For example, Neural Architecture Search has discovered novel networks that rival the best human-designed architectures on challenging image classification tasks [9, 10].

However, neural AutoML is expensive because it requires training many networks during architecture search. This may require vast computations resources; Zoph and Le [8] report 800 concurrent GPUs to train on Cifar-10. Further, training needs to be repeated for every new task. Some methods have been proposed to address this cost, such as using a progressive search space [11], or by sharing weights among generated networks [12]. We propose a complementary solution when one has multiple ML tasks to solve. Humans can tune networks based on knowledge gained from prior tasks. We aim to leverage the same information using transfer learning.

We exploit the fact that deep RL-based AutoML algorithms learn an explicit parameterization of the distribution over performant models. We present Transfer Neural AutoML to accelerate network design on new tasks based on priors learned on previous tasks. To do this we design a network to perform neural AutoML on multiple tasks simultaneously. Our method for multitask neural AutoML learns both hyperparameter choices common to multiple tasks and specific choices for individual tasks. We then transfer this controller to new tasks and leverage the learned priors over performant models. We reduce the time to converge in both text and image domains by over an order of magnitude in most tasks. In our experiments we save 10s of CPU hours for every task that we transfer to.

Preprint. Work in progress.

2 Methods

Our Transfer Neural AutoML algorithm is based on Neural Architecture Search (NAS) [8]. We first review NAS, then we present our method for multitask training, and its application to transfer learning.

2.1 Neural Architecture Search

NAS uses deep reinforcement learning to generate models that maximize performance on a given task. The framework consists of two components: a controller model and child models. The controller generates network configurations, seeking to maximize performance on a particular ML task. These configurations define the architecture of the child models. At every iteration of training, the child models are trained and evaluated on the ML task at hand. The performance of the child network on the validation set is used as a reward to update the controller via a policy gradient algorithm.

To generate network configurations the controller's architecture is an RNN. The controller RNN generates a sequence of discrete actions. Each action specifies a design choice; for example, if the child models are CNNs, these choices could include the filter heights, widths, and strides. The model is autoregressive, like a language model: the action taken at each time step is fed into the RNN as input for the next time step. The recurrent state of the RNN maintains a history of the design choices taken so far. The use of an RNN allows dependencies between the design choices to be learned, and variable length sequences of decisions to be made when appropriate.

The original NAS performed a search over a space of strictly architectural parameters. Other hyperparameters, such as those for the controller of the learning algorithm, were chosen by hand. In this work we use the controller to choose the architecture, optimization hyperparameters and make other design decisions such as which pre-trained embedding layers to use.

2.2 Multitask Training

We next describe our approach to Multitask Neural AutoML, which allows simultaneous model search for multiple tasks. We first define a generic search space that is shared across all tasks. Many deep learning models require the same common design decisions, such as choice of network depth, learning rate, and number of training iterations. By defining a generic search space that contains many architecture and hyperparameter choices, the controller can generate a wide range of models applicable to many common problems. Multitask training allows the controller to learn broadly applicable prior and dependencies in the search space observing shared behaviour across tasks.

We propose a controller capable of simultaneous multitask training through two key features.

Learned task representations The multitask AutoML controller is trained simultaneously on a set of tasks. Figure 1 shows the architecture of the controller at each time step. To generate different network configurations for each task, we associate a unique embedding vector to each. We then condition the generation of network configurations on the task by feeding the task embedding into the RNN controller at every time step. These task-embeddings are analogous to word-embeddings in NLP, where each word is associated to a trainable vector [13]. In the single task training of NAS, the action at each time step is fed into the next time step via an embedding layer. In multitask training, we also concatenate the task embedding to the embedding of the previous action which we feed to the RNN. We also add a skip connection across the RNN in order to learn action marginal distributions more easily.

During training, at each trial (i.e. sample from the controller) we sample a task uniformly. We generate a model conditioned on that task's embedding and the child model is trained and evaluated on that task. The task embeddings are the only task-specific parameters; they are initialized randomly and trained jointly with the controller.

Task-specific advantage normalization We train the controller using policy gradient. Each task defines a different performance metric which we use as reward. The reward affects the amplitude of the gradients applied to update the controller's policy, π . To train effectively on differing tasks, we need to ensure that the distribution of each task's rewards are scaled to have same mean and variance.

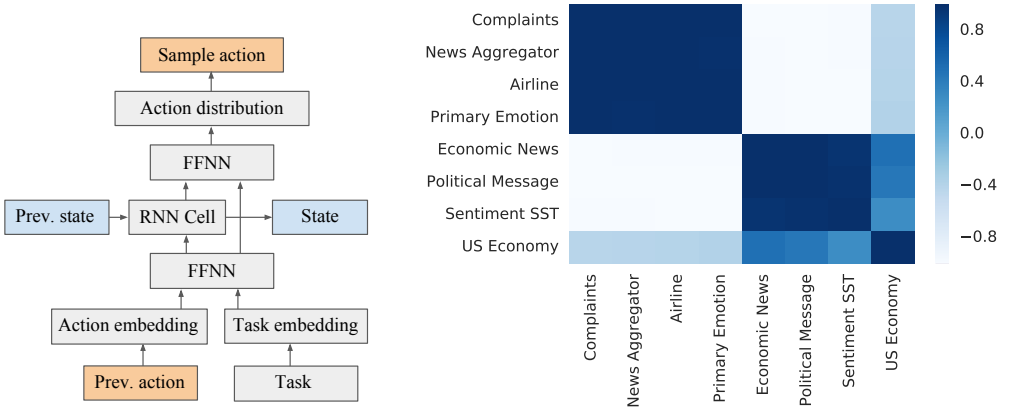

Figure 1: *Left:*A single time step of the recurrent multitask AutoML controller, in which a single action is taken. The task embedding is concatenated with the embedding of the action sampled at the previous timestep and passed into the controller RNN. All parameters, other than the task embeddings, are shared across tasks. *Right:* Cosine similarity between the task embeddings learned by the multitask neural AutoML model.

The mean of each task's reward distribution is centered on zero by subtracting the expected reward for the given task. The centered reward, or advantage, $A_\tau(m)$, of a model, m, applied to a task, τ, is defined as the difference between the reward obtained by the model, $R_\tau(m)$, and the expected reward for the given task, $b_\tau = \mathbb{E}_{m\sim\pi}[R_\tau(m)]$: $A_\tau(m) = R_\tau(m) - b_\tau$. b_τ is commonly referred to as the baseline. Subtracting a baseline is a standard technique in policy gradient algorithms used to reduce the variance of the parameter updates [14].

The variance of each task's reward distribution is normalized by dividing the advantage by the standard deviation of the reward: $A'_\tau(m) = (R_\tau(m) - b_\tau)\sigma_\tau^{-1}$. Where $\sigma_\tau = \sqrt{\mathbb{E}_{m\sim\pi}[(R_\tau(m) - b_\tau)^2]}$. We refer to A' as the normalized advantage.

In policy gradient, the gradient update to the parameters to the policy θ is the product of the advantage and expected derivative of the log probability of sampling an action, $A'_\tau(m)\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(m)]$. The normalizing the advantage may also be seen as adapting the learning rate for each task, since σ_τ directly scales the gradient.

We estimate a different baseline and standard deviation for each task. In practice, we compute b_τ and σ_τ using exponential moving averages over the sequence of rewards: $b_\tau^t = (1-\alpha)b_\tau^{t-1} + \alpha R_\tau(m)$, $\sigma_\tau^{2,t} = (1-\alpha)\sigma_\tau^{2,t-1} + \alpha(R_\tau(m) - b_\tau^t)^2$, where t indexes the trial, and $\alpha = 0.01$ is the decay factor. Using the normalized advantage to scale the gradients allows us to train on tasks whose performance may have different ranges and scales, while maintaining balanced gradient updates.

2.3 Transfer Learning

Given the multitask neural AutoML controller, transfer can be performed in a straightforward manner. The pre-trained controller learns a prior over generic architectural and parameter choices, along with task-specific decisions encoded in the task embeddings. Given a new task, we significantly speed up exploration by leveraging the learned priors over which architectures or hyperparameters worked well. By learning an embedding for the new tasks, the controller can then learn a representation that biases towards actions that performed well on similar tasks.

To perform transfer we simply to initialize the generic parameters of the controller for the new task with the pre-trained multitask AML controller. We add a new randomly initialized task embedding for the new task. The controller weights and the new task embedding are then updated jointly with policy gradient as before.

3 Related Work

A variety of optimization methods have been proposed to search over architectures, hyperparameters, and learning algorithms. These include random search [1], parameter modeling [3], meta-learned hyperparameter initialization [15], deep-learning based tree searches over a predefined model-specification language [16], and learning of gradient descent optimizers [17, 18]. An emerging body of neuro-evolution research has adapted genetic algorithms for these complex optimization problems [19], including to set the parameters of existing deep networks [20], evolve image classifiers [5], and evolve generic deep neural networks [6].

Our work relates closest to NAS [8]. NAS was applied to construct CNNs for the CIFAR-10 task and RNNs for the Penn Treebank tasks. Subsequent work attempts to reduce the computational cost for more challenging tasks [10]. To engineer an architecture for ImageNet classification, the authors train the NAS controller on the simpler CIFAR-10 task and then transfer the child architecture to ImageNet by stacking it. However, they did not transfer the controller model itself, relying instead on the intuition that additional depth is necessary for the more challenging task. Other works applied RL to automate architecture generation and also try to reduce the computation cost. MetaQNN sequentially chooses CNN layers using Q-learning [21]. MetaQNN uses an aggressive exploration to reduce search time, though it can cause the resulting architectures to underperform. Cai et al. [22] transform existing architectures incrementally to avoid generating entire networks from scratch. Liu et al. [11] reduce search time by progressively increasing architecture complexity, and [12] propose child-model weight sharing to reduce child training time.

Transfer learning has achieved excellent results as an initialization method for deep networks, including for models trained using RL [23, 24, 25]. Recent meta-learning research has broadened this concept to learn generalizable representations across classes of tasks [26, 27]. Simultaneous multitask training can facilitate learning between tasks with a common structure, though retaining knowledge effectively across tasks is still an active area of research [28, 29]. There is also prior research on transfer of optimizers for AutoML; Sequential Model-based Optimizers have been transferred across tasks to improve hyperparameter tuning [30, 31]. We aim to do the same for neural methods.

4 Experiments

Our main result shows that we substantially reduce the convergence time of neural AutoML. Transfer Neural AutoML achieves an equivalent reward equal to single-task in an order of magnitude fewer trials on many datasets, and generates models with higher accuracy given a fixed budget.

Child models Constructing the space needs human input, but we choose very wide parameter ranges to minimize injected domain expertise. Our search space for child models contains two-tower feedforward neural networks (FFNN), similar to the wide and deep models in Cheng et al. [32]. One tower is a deep FFNN, containing an embedding module, fully connected layers and a softmax classification layer. This tower is regularized with an L2 loss. The other is a shallow layer that directly connects the one-hot token encodings to the softmax classification layer with a linear projection. This tower is regularized with a sparse L1 loss. The shallow tower allows the model to learn task-specific biases for each token directly.

The controller selects among pre-trained embedding modules for the input to the model. This has two benefits: first, the quality of the child models on smaller datasets improves, and second, it decreases convergence time of the child models. The pretrained modules are shared publicly[1].

The single search space for all tasks is defined by the following sequence of choices: 1) The pre-trained input embedding module. 2) Whether to fine tune the input embedding module. 3) The number of hidden layers. 4) The hidden layers size. 5) The hidden layers activation function. 6) Which layer normalization scheme to use (if any). 7) Dropout rate for the hidden layers. 8) The learning rate for the deep network. 9) Regularization strength for the deep layers. 10) The learning rate for the shallow network. 11) Regularization strength for the shallow layer. 12) The number of training steps. The Appendix contains the exact specification. The search space is much larger than the number of possible trials, containing 1.1B configurations. All models are trained using Proximal Adagrad with batch size 100.

[1] Location anonymized for review

Controller model The controller is a 2-layer LSTM with 50 units. The action and task embeddings have size 25. The controller and embedding weights are initialized uniformly at random, yielding an approximate uniform initial distribution over actions. The learning rate is set to 10^{-4} and it receives gradient updates after every child completes (batch size 1). We tried four variants of policy gradient to train the controller: REINFORCE [33], TRPO [34], UREX [35] and PPO [36]. On a pilot study on four NLP tasks we found REINFORCE and TRPO to perform best and selected REINFORCE for the following experiments.

Metrics To evaluate the ability of AutoML to find good configurations we compute the accuracy of the best child models generated during the search. To reduce noise introduced during training and evaluation of the child models we average the performance of the best N models. These best N models are selected according to accuracy on validation set. We refer to the validation/test accuracy of these N models as 'validation/test accuracy-topN', respectively.

We assess convergence rates with two metrics: first, accuracy-topN given a fixed budget of trials, and second, the number of trials required to attain a certain level of performance. The latter can only be used with validation accuracy-topN since test accuracy-topN does not necessarily increase monotonically with the number of trials.

4.1 Natural Language Processing

Data We evaluate using 21 text classification tasks with varied statistics. The dataset sizes range from 500 to $420k$ datapoints. The number of classes range from 2 to 157, and the mean length of the texts, in characters, range from 19 to $20k$. The Appendix contains full statistics and links.

During each trial, the child model is trained on the training set. The accuracy on the validation set is used as reward for the controller. After search is finished, the child model with the best validation accuracy is evaluated on the test set. For the datasets that do come with a pre-defined train/validation/test split, we split randomly 80/10/10.

Experiment Setup We randomly sampled 8 of the 21 tasks (Airline, Complaints, Economic News, News Aggregator, Political Message, Primary Emotion, Sentiment SST, US Economy) to pre-train a multitask model. We then transfer from this model to each of the remaining 13 tasks (20 Newsgroups, Brown Corpus, Customer Reviews, Corp Messaging, Disasters, Emotion, Global Warming, MPQA Opinion, Progressive Opinion, Sentiment Cine, Sentiment IMDB, SMS Spam, Subj Movie). We compare the performance of Transfer Neural AutoML (T-NAML) to the single-task Neural (NAML). We also benchmark against random search (RS).

Results To assess the algorithms' ability to optimize the reward (validation set accuracy) we compute the speed-up versus the baseline, RS. We first compute accuracy-top10 on the validation set for RS given a fixed budget of B trials. We use $B = 5000$, except for the Brown Corpus and 20 Newsgroups where we can only use a $B = 500, 3500$, respectively, because these datasets were slower to train. We then report the number of trials required by AutoML and T-AutoML to achieve the same validation accuracy-top10 as RS with B trials. Table 1 (left) shows the results. Note, RS may exhibit fewer than $B = 5000$ trials if it converged earlier. Table 1 shows that T-AutoML is effective at optimizing validation accuracy, offering a large reduction in time to attain a fixed reward. In 12 of the 13 datasets T-AutoML achieves the desired reward fastest, and in 10 cases achieves over an order of magnitude speed-up.

Next, we assess the quality of the models on the test set. Table 1 (right) shows test accuracy-top10 with a budget of 2000 trials (500 for Brown Corpus). The table shows that within this budget T-AutoML performs best, or joint best on all but one dataset. T-AutoML outperforms single task AutoML on 10 out of the 13 datasets, ties on one, and loses on two. On the datasets where T-AutoML does not produce the best final model at 2000 trials, it often produces better model at earlier iterations. Figure 2 shows the full learning curves of test set accuracy-top10 versus number of trials. Figure 2 shows that in most cases the controller with transfer starts with a much better prior over good models. On some datasets the quality is improved with further training e.g. Emotion, Corp Messaging, but in others the initial configurations learned from the multitask model are not improved upon.

For reference, we put the learning curves for the initial multitask training phase in the Appendix. We also ran RS and single task AutoML on these datasets. Slightly disappointingly, multitask training

Number of trials needed to attain a validation accuracy-top10 equal to the best achieved by random search with 5000 trials (250/2500 for Brown and 20 Newsgroups, respectively).

Dataset	RS	NAML	T-NAML
20 Newsgroups	2470	1870	**435**
Brown Corpus	245	235	**10**
SMS Spam	4815	3390	**70**
Corp Messaging	3850	1510	**80**
Disasters	4970	2730	**25**
Emotion	4995	1645	**195**
Global Warming	4985	1935	**90**
Prog Opinion	4200	3620	**60**
Customer Reviews	4895	925	**15**
MPQA Opinion	4965	1510	**15**
Sentiment Cine	4520	3225	**535**
Sentiment IMDB	4760	**630**	690
Subj Movie	4745	1600	**105**

Accuracy-top10 on the test set given at a fixed budget B of 500 trials ($B = 250$ for Brown). Error bars show ± 2 s.e.m. computed across the top 10 models. Similar s.e.m. values are observed for all methods.

Dataset	RS	NAML	T-NAML
20 Newsgroups	87.5	87.4	**88.1 ± 0.4**
Brown Corpus	37.0	38.2	**53.4 ± 3.3**
SMS Spam	97.9	97.8	**98.1 ± 0.1**
Corp Messaging	**90.0**	**90.2**	**90.2 ± 0.3**
Disasters	81.7	81.5	**82.1 ± 0.3**
Emotion	33.9	33.7	**35.3 ± 0.3**
Global Warming	82.4	**82.8**	**82.9 ± 0.3**
Prog Opinion	68.9	66.3	**70.3 ± 0.9**
Customer Reviews	77.8	79.0	**81.4 ± 0.5**
MPQA Opinion	87.9	87.9	**88.6 ± 0.3**
Sentiment Cine	73.2	**76.3**	75.4 ± 0.4
Sentiment IMDB	85.8	87.3	**88.1 ± 0.1**
Subj Movie	92.6	93.2	**93.4 ± 0.2**

Table 1: Performance of Random Search (RS, Neural AutoML (NAML) and Transfer Neural AutoML (T-NAML). *Left*: convergence rate of the optimization, measured using time to fixed reward. *Right*: performance on the test set, measured using test accuracy at a fixed budget. Bolding indicates the best performing algorithm or those within 2 s.e.m. of the best.

did not in itself yield substantial improvements over single task, although it attains a higher accuracy on two datasets, and in similar on the other six.

We aim to to attain good performance automatically with fewest possible trials. We do not seek to beat state of the art all datasets because first, although our search space is large, it does not contain all high-performing components (e.g. convolutions). We leave broadening the search space to future work. Second, we use embedding modules pre-trained on large datasets which makes the results incomparable to those that only use the tasks' training data.

However, to confirm that Neural AutoML generates good models we compare to some previous published results where available. Overall we find that Transfer AutoML with the search space described above yields models competitive with the state-of-the-art. For example, Almeida et al. [37] use classical ML classifiers (Logistic Regression, SVMs, etc.) on SMS Spam and report best accuracy of 97.59%. Transfer AutoML gets accuracy-top10 of 98.1%. Le and Mikolov [38] report 92.58% accuracy on Sentiment IMDB which is greater than Transfer AutoML with more complex architectures. Li et al. [39] report 86.8% accuracy using an ensemble of weighted neural BOWs on MPQA. Transfer AutoML achieve accuracy-top10 of 88.6%. Li et al. [39] also evaluate their ensemble of weighted neural BOW models on Customer Reviews, and achieve 82.5% best accuracy, though the best accuracy of any single model is 81.1%. Comparably, T-AutoML gets an accuracy-top10 of 81.4%. Barnes et al. [40] compare many algorithms and report best accuracy on Sentiment-SST of 83.1% using LSTMs. Multitask AutoML gets an accuracy-Top10 of 83.4%. The best performance achieved with a more complex architecture that is not in our search space is: 87.8% [38]. Maas et al. [41] report 88.1% on Movie Subj, Transfer AutoML gets accuracy-top10 of 93.4%.

Computational Cost and Savings The median cost to perform a single trial across all 21 datasets in our experiments is $T = 268s$.[2] If we run B trials with a speedup factor of S, we save $BT(1 - S^{-1})/3600$CPU-h per task to attain a fixed reward (validation accuracy-top10). Estimating the speedup factors from Table 1 (left) for transfer over single-task, we attain a median computational saving of 30CPU-h per task when performing $B = 500$ trials. The mean is 89CPU-h, but this is heavily influenced by the slow Brown Corpus. If we do not need the models found on the M tasks used to train the multitask controller, then we must run $> (1 - 1/S)^{-1}M$ new tasks to amortize this cost. For the median speedup in our experiments $S = 22$ that is $> 1.05M$ new tasks.

[2]The mean, 666s, is heavily influenced by the task Brown Corpus, which is an outlier with very long texts.

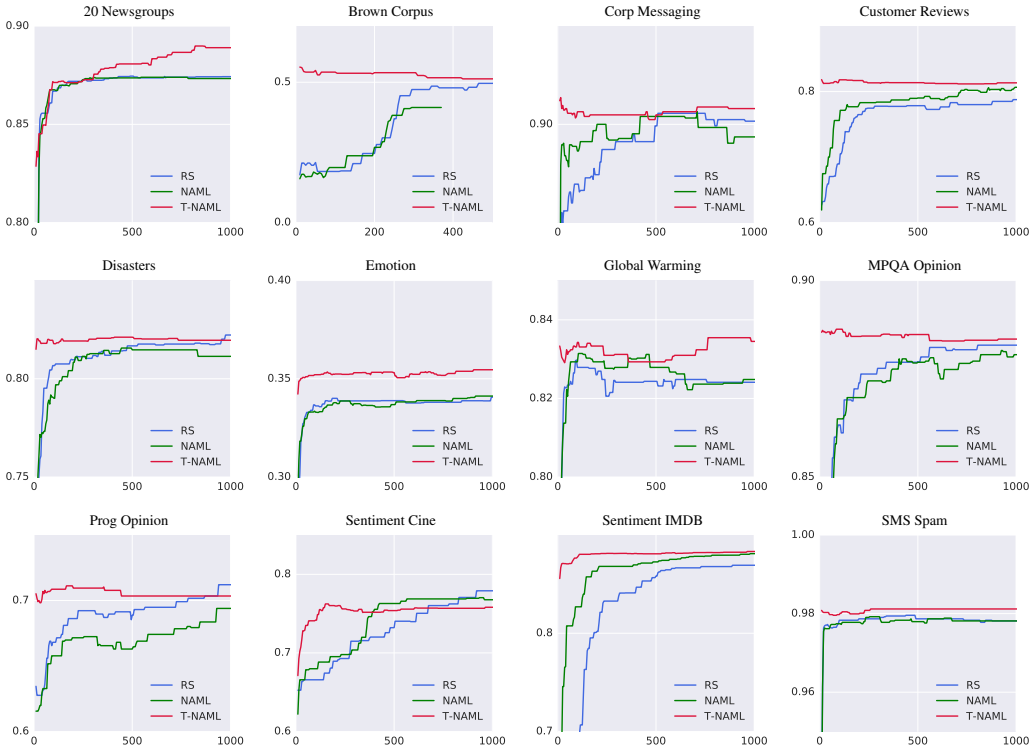

Figure 2: Learning curves for transfer learning. *x-axis*: Number of trials (child model evaluations). *y-axis*: Average test set accuracy of the 10 models with best validation accuracy (test accuracy-top10) found up to each trial.

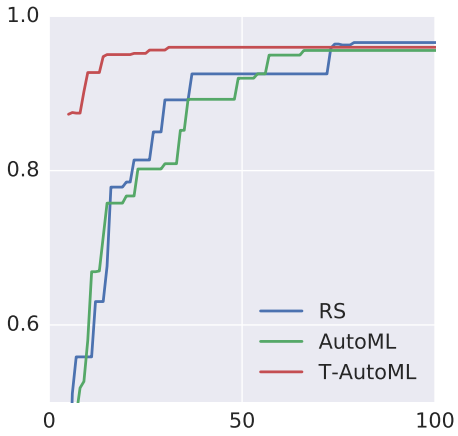

Figure 3: Comparison on an image classification task, Cifar 10. Mean test accuracy of the top 10 models chosen on the validation set.

4.2 Image classification

To validate the generality of our approach we evaluate on image classification task: Cifar-10. We conmpare 3 controllers: RS, AutoML trained from scratch, and AutoML pre-train on MNIST and Flowers[3]. Figure 3 shows the mean accuracy-top-10 on the test set. The transferred controller attains an accuracy-top-10 of 96.5%, similar to the other methods, but converges much faster as in the NLP tasks. The best models embed images with a finetuned Inception v3 network, pre-trained on ImageNet. Relu activations are preferred over Swish [42] and the dropout rate of converges to 0.3.

[3] goo.gl/tpzfR1

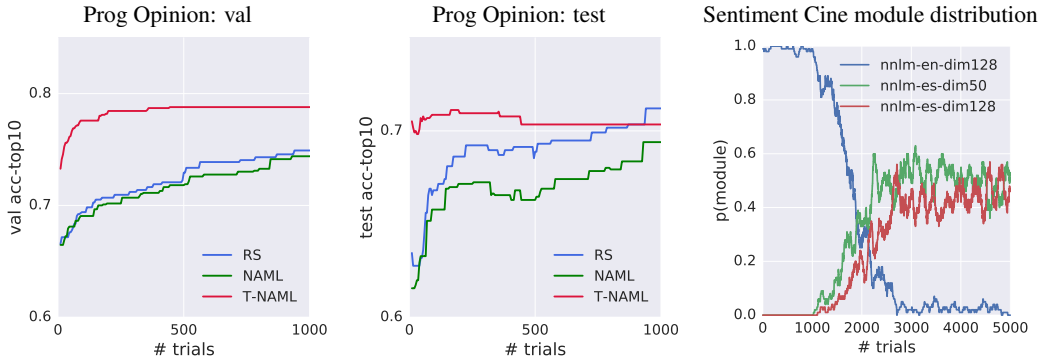

Figure 4: *Left, Center:* Learning curves on the validation (left) and test sets (center) for the Prog Opinion dataset. *Right:* Evolution of the choice of pre-trained embedding module for transfer to the Spanish Corpus-Cine task. y-axis indicates the probability of sampling each table. This probability is estimated from the samples using a sliding window of width 100.

4.3 Analysis

Meta overfitting The controller is trained on the tasks' validation sets. Overfitting of AutoML to the validation set is not often addressed. This type of overfitting may seem unlikely because each trial is expensive, and many trials may be required to overfit. However, we observe it in some cases.

Figure 4 (left, center) shows the accuracy-top10 on the validation and test sets on the Prog Opinion dataset. Transfer Neural AutoML attains good solutions in the first few trials, but afterwards its validation performance grows while test performance does not. The generalization gap between the validation and test accuracy increases over time. This is the most extreme case we observed, but some other datasets exhibit some generalization gap also (see Appendix for all validation curves). This effect is largest on Prog Opinion because the validation set is tiny, with only 116 examples.

Overfitting arises from bias due to selecting the best models on the validation set. Child evaluation contains randomness due to the stochastic training procedure. Therefore, over time we see an improved validation score, even after convergence, due to lucky evaluations. However, those apparent improvements are not reflected on the test set. Transfer AutoML exhibits more overfitting than single task because it converges earlier. We confirmed this effect; if we 'cheat' and select models by their test-set performance, we observe the same artificial improvement on the test score as on the validation score. Other than entropy regularization, we do not combat overfitting extensively. Here, we simply emphasize that because our Transfer Neural AutoML model observes many trials in total, meta-overfitting becomes a bigger issue. We leave combatting this effect to future research.

Distant transfer: across languages The more distant the tasks, the harder it is to perform transfer learning. The Sentiment Cine task is an outlier because it is the only Spanish task. Figure 2 and Table 1 show poorer performance of transfer on this task.

The most language-sensitive parameters are the pre-trained word embeddings. The controller selects from eight pretrained embeddings (see Appendix) of which only two are Spanish. In the first 1500 iterationsn the transferred controller chooses English embeddings, limiting the achievable performance. However, after further training, the controller switches to Spanish tables Figure 4 (right). At trial 2000, T-AutoML attains a test accuracy-top10 of 79.8%, approximately equal to that or random search with 79.4%, and greater than single task with 78.1%. This indicates that although transfer works best on similar tasks, the controller is still able to adapt to outliers given sufficient training time.

Task representations and learned models We inspect the learned task similarities via the embeddings. Figure 1 shows the cosine similarity between the task embeddings learned during multitask training. The model assigns most tasks to two clusters. It is hard to guess *a priori* which tasks require similar models; the dataset sizes, number of classes and text lengths differ greatly. However, the controller assigns the same model to tasks within the same cluster. At convergence, the cluster {Complaints, New Agg, Airline, Primary Emotion} is assigned (with high probability) a 1-layer networks with 256 units, Swish activation function, wide-layer learning rate 0.01, and dropout rate

0.2. The cluster {Economic News, Political Emotion, Sentiment SST} is assigned 2-layer networks with 64 units, Relu activation, wide-layer learning rate 0.003, and dropout rate 0.3.

Other choices follow similar distributions for each cluster. For example, the same 128D word embeddings, trained using a Neural Language Model are chosen. The controller also always chooses to fine-tune these embeddings. The controller may remove either the deep or shallow tower by setting the regularization very high, but in all cases it chooses to keep both active.

5 Conclusion

Neural AutoML, whilst becoming popular, comes with a high computational cost. To address this we propose transfer learning and show a large reductions in convergence time across many datasets. Extensions to this work include: Broadening the search space to contain more models classes. Attempting transfer across modalities; some priors over hyperparameter combinations learned on NLP tasks may be useful for images or other domains. Robustifying the controller to evaluation noise, and addressing potential to meta overfit on small datasets.

Acknowledgments

We are very grateful to Quentin de Laroussilhe, Andrey Khorlin, Quoc Le, Sylvain Gelly, the Tensorflow Hub team and the Google Brain team Zurich for developing software frameworks and many useful discussions.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 2012.

[2] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

[3] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*, 2013.

[4] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

[5] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

[6] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017.

[7] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[8] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[9] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. Practical network blocks design with q-learning. In *AAAI*, 2018.

[10] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

[11] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017.

[12] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[14] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *JMLR*, 2004.

[15] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, 2015.

[16] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[17] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *arXiv preprint arXiv:1703.04813*, 2017.

[18] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In *ICML*, 2017.

[19] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

[20] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[21] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[22] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017.

[23] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

[24] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR workshops*, 2014.

[25] Yusen Zhan and Matthew E Taylor. Online transfer learning in reinforcement learning domains. *arXiv preprint arXiv:1507.00436*, 2015.

[26] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017.

[27] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *NIPS 2017 Workshop on Meta-Learning*, 2017.

[28] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 2017.

[29] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

[30] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *ICML*, pages 199–207, 2013.

[31] Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *AISTATS*, pages 1077–1085, 2014.

[32] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016.

[33] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer, 1992.

[34] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[35] Ofir Nachum, Mohammad Norouzi, and Dale Schuurmans. Improving policy gradient by exploring under-appreciated rewards. In *ICLR*, 2017.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[37] Tiago Almeida, José María Gómez Hidalgo, and Tiago Pasqualini Silva. Towards sms spam filtering: Results under a new dataset. *International Journal of Information Security Science*, 2013.

[38] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, ICML'14, 2014.

[39] Bofang Li, Zhe Zhao, Tao Liu, Puwei Wang, and Xiaoyong Du. Weighted neural bag-of-n-grams model: New baselines for text classification. In *COLING*, 2016.

[40] Jeremy Barnes, Roman Klinger, and Sabine Schulte im Walde. Assessing state-of-the-art sentiment models on state-of-the-art sentiment datasets. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. ACL, 2017.

[41] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL: Human Language Technologies*. ACL, 2011.

[42] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 2017.

Supplementary Material for Transfer Learning with Neural AutoML

This document contains a description of the search space used in our experiments (Table 1), details of the pretrained modules for embedding text and images (Tables 2 and 3), and statistics for the datasets used (Table 4 and 5). It also contains the learning curves for Transfer Neural AutoML (Figures 1 and 2) and Multitask Neural AutoML (Figures 3 and 4) on the validation and test sets.

Table 1: The search space for our AML models.

Parameter	**Search Space**
1) Input embedding modules	Text input: refer to Table 2. Image input: refer to Table 3.
2) Fine-tune input embedding module	{True, False}
3) Number of hidden layers	{1, 2, 3, 5, 7}
4) Hidden layers size	{8, 16, 32, 64, 128, 256}
5) Hidden layers activation	{relu, swish}
6) Hidden layers normalization	{none, batch norm, layer norm}
7) Hidden layers dropout rate	{0.0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6}
8) Deep tower learning rate	{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0}
9) Deep tower regularization weight	{0.0, 0.00001, 0.0001, 0.001, 0.01, 0.1, disable deep tower}
10) Wide tower learning rate	{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0}
11) Wide tower regularization weight	{0.0, 0.00001, 0.0001, 0.001, 0.01, 0.1, disable wide tower}
12) Number of training samples	{1000, 3000, 10000, 30000, 100000, 300000, 1000000}

Table 2: Options for text input embedding modules. These are pre-trained text embedding tables, trained on datasets with different languages and size. The text input to these modules is tokenized according to the module dictionary and normalized by lower-casing and stripping rare characters. The embeddings of each token are aggregated with a mean BOW approach. We provide the handle for the modules that are publicly distributed.

Language/ID	**Dataset size** (tokens)	**Embed dim.**	**Vocab. size**	**Training algorithm**	**Link**
Spanish-small	50B	50	995k	Lang. model	Anonymous
Spanish-big	50B	128	995k	Lang. model	Anonymous
English-small	7B	50	982k	Lang. model	Anonymous
English-big	200B	128	999k	Lang. model	Anonymous
English-wiki-small	4B	250	1M	Skipgram	Anonymous
English-wiki-big	4B	500	1M	Skipgram	Anonymous
English-news-small	90B	100	5.9M	CBOW	
English-news-big	90B	500	5.9M	CBOW	

Table 3: Options for image input embedding modules. To map an image, the controller can choose among state of the art architectures pre-trained on ImageNet. The module consists in the pre-trained model up to the final layer of logits. We provide the handle for the modules that are publicly distributed.

Architecture	**Dataset**	**Reference**	**Link**
MobileNet v1	Imagenet	(Howard et al., 2017)	Anonymous
Inception v2	Imagenet	(Ioffe & Szegedy, 2015)	Anonymous
Inception v3	Imagenet	(Szegedy et al., 2015)	Anonymous
Resnet v1.101	Imagenet	(He et al., 2015)	Anonymous
Resnet v1.50	Imagenet	(He et al., 2015)	Anonymous

Table 4: Statistics and references for the NLP classification tasks.

Dataset	**Train samples**	**Valid. samples**	**Test samples**	**Classes**	**Lang (chars)**	**Len**	**Reference**
20 Newsgroups	15,076	1,885	1,885	20	En	2,000	(Lang, 1995)
Airline	11,712	1,464	1,464	3	En	104	crowdflower.com
Brown Corpus	400	50	50	15	En	20,000	(Francis & Kuera, 1982)
Complaints	146,667	18,333	18,334	157	En	1,000	catalog.data.gov
Corp Messaging	2,494	312	312	4	En	121	crowdflower.com
Customer Reviews	3,044	378	378	2	En	100	(Hu & Liu, 2004)
Disasters	8,688	1,086	1,086	2	En	101	crowdflower.com
Economic News	6,392	799	800	2	En	1,400	crowdflower.com
Emotion	32,000	4,000	4,000	13	En	73	crowdflower.com
Global Warming	3,380	422	423	2	En	112	crowdflower.com
MPQA Opinion	8,547	1,025	1,034	2	En	19	(Deng & Wiebe, 2015)
News Aggregator	338,349	42,294	42,294	4	En	57	(Lichman, 2013)
Political Message	4,000	500	500	9	En	205	crowdflower.com
Primary Emotions	2,019	252	253	18	En	87	crowdflower.com
Prog Opinion	927	116	116	3	En	102	crowdflower.com
Sentiment Cine	3119	382	377	2	Spanish	2,760	(Cruz et al., 2008)
Sentiment IMDB	19946	5054	25000	2	En	1,360	(Maas et al., 2011)
Sentiment SST	67,349	872	1,821	2	En	105	(Socher et al., 2013)
SMS Spam	4,459	557	557	2	En	81	(Almeida et al., 2011)
Subj Movie	8052	972	976	2	En	127	(Pang et al., 2002)
US Economy	3,961	495	495	2	En	305	crowdflower.com

Table 5: Statistics and references for the Image classification tasks.

Dataset	**Train samples**	**Valid. samples**	**Test samples**	**Classes**	**Image size**	**Reference**
Cifar 10	45000	5000	10000	10	32x32x3	(Krizhevsky et al.)
Mnist	55000	5000	10000	10	28x28x1	(LeCun & Cortes, 2010)
Flowers	2018	552	550	5	variablex3	goo.gl/tpzfR1

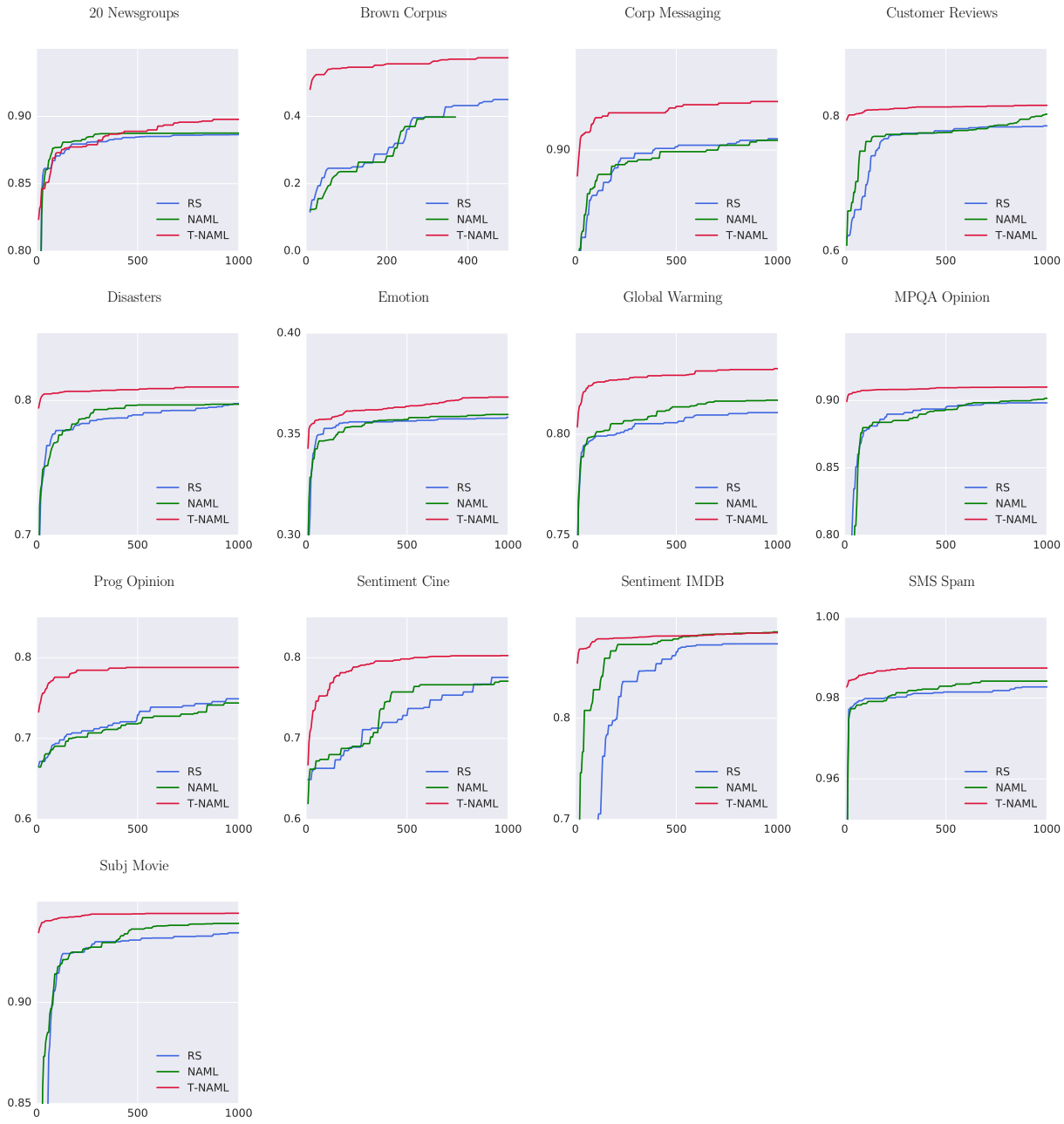

Figure 1: Learning curves for transfer learning. X-axis depicts number of trials (T) performed for each task. Y-axis depicts the mean validation accuracy of the 10 models achieving top validation accuracy (validation accuracy-top10).

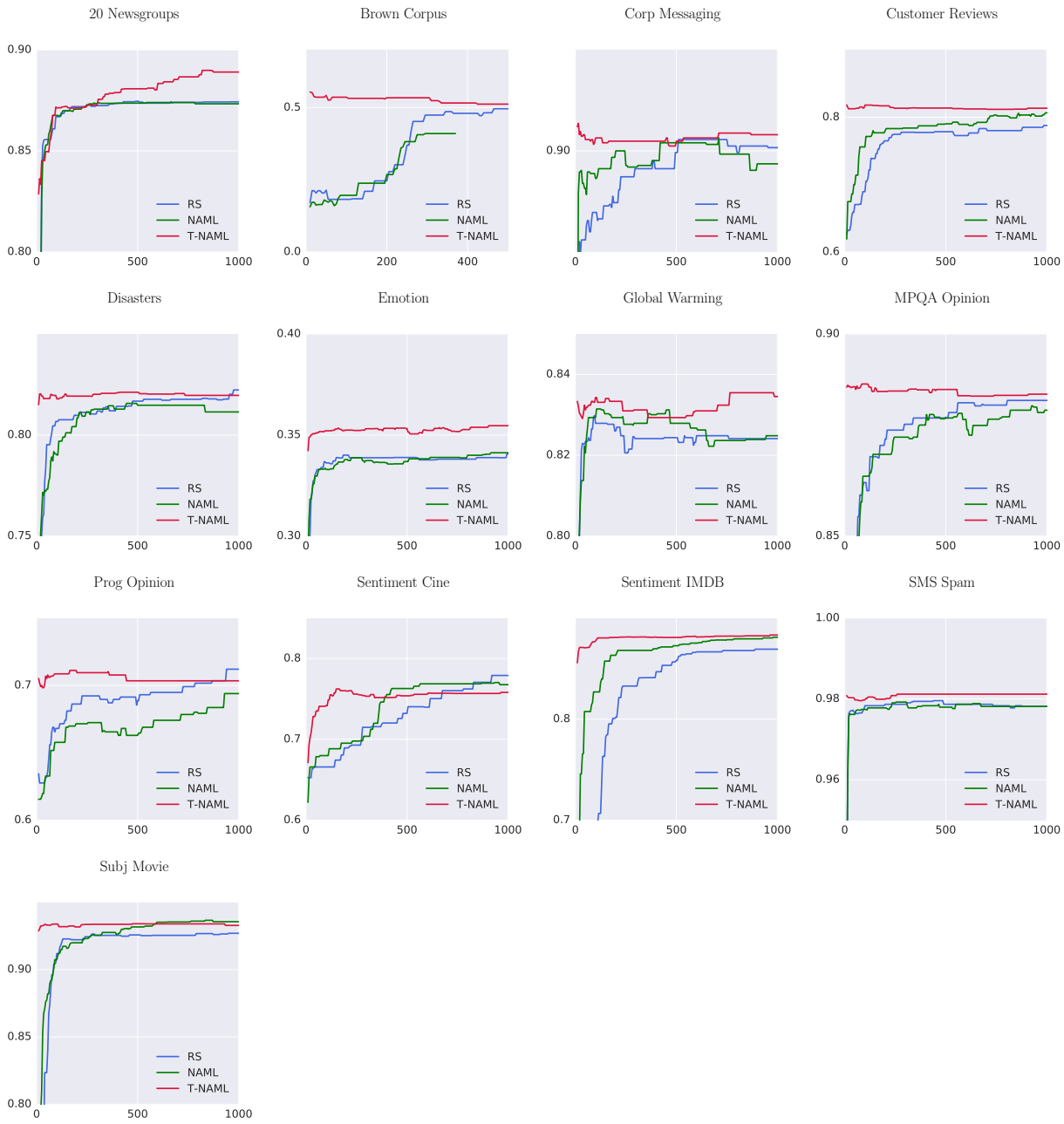

Figure 2: Learning curves for transfer learning. X-axis depicts number of trials (T) performed for each task. Y-axis depicts the mean test accuracy of the 10 models achieving top validation accuracy (test accuracy-top10).

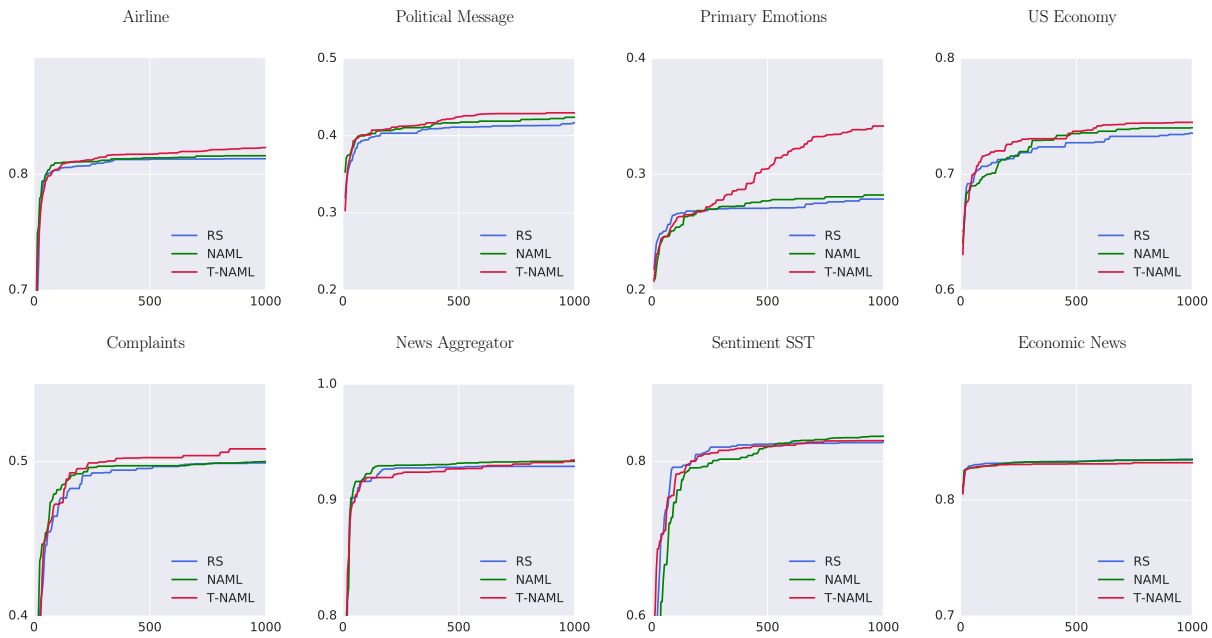

Figure 3: Learning curves for multitask training. X-axis depicts number of trials (T) performed for each task. Y-axis depicts the mean validation accuracy of the 10 models achieving top validation accuracy (validation accuracy-top10).

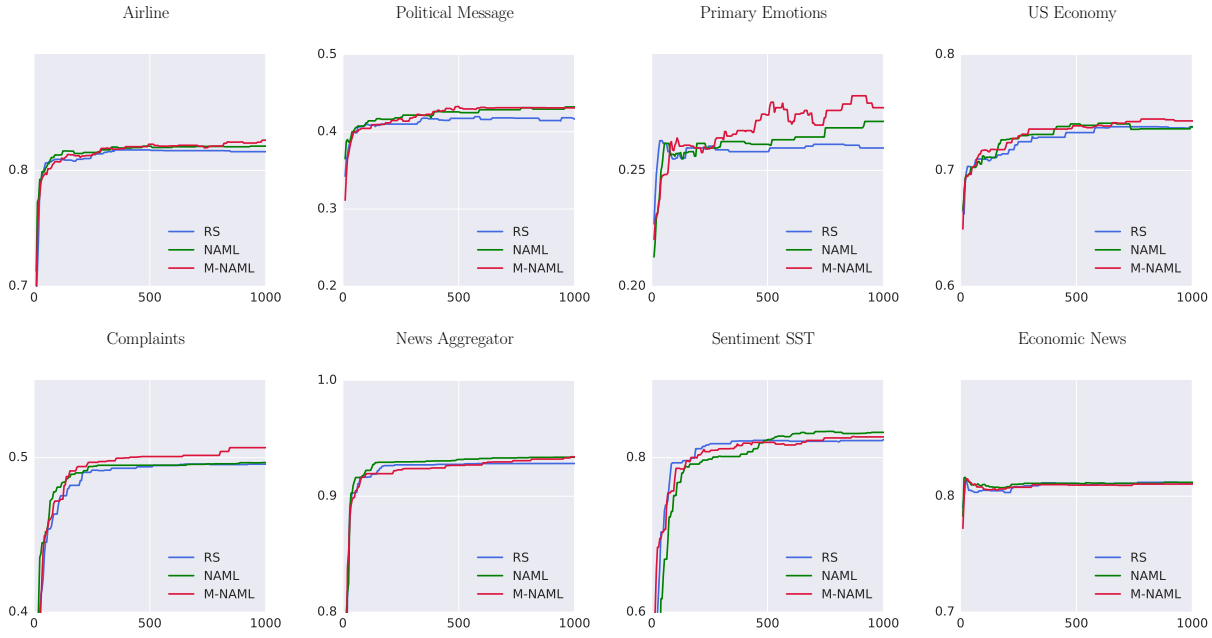

Figure 4: Learning curves for multitask training. X-axis depicts number of trials (T) performed for each task. Y-axis depicts the mean test accuracy of the 10 models achieving top validation accuracy (test accuracy-top10).

References

Almeida, Tiago A., Hidalgo, José María G., and Yamakami, Akebo. Contributions to the study of sms spam filtering: New collection and results. In *Proceedings of the 11th ACM Symposium on Document Engineering*, DocEng '11, New York, NY, USA, 2011. ACM.

Cruz, Fermın L, Troyano, Jose A, Enriquez, Fernando, and Ortega, Javier. Clasificación de documentos basada en la opinión: experimentos con un corpus de crıticas de cine en espanol. *Procesamiento del lenguaje natural*, 2008.

Deng, Lingjia and Wiebe, Janyce. Mpqa 3.0: Entity/event-level sentiment corpus. In *ACL*, Denver, Colorado, USA, 2015. Sociedad Española para el Procesamiento del Lenguaje Natural.

Francis, W. Nelson and Kuera, Henry. Frequency analysis of english usage. lexicon and grammar. In *Houghton Mifflin*, 1982.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

Howard, Andrew G., Zhu, Menglong, Chen, Bo, Kalenichenko, Dmitry, Wang, Weijun, Weyand, Tobias, Andreetto, Marco, and Adam, Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL `http://arxiv.org/abs/1704.04861`.

Hu, Minqing and Liu, Bing. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, New York, NY, USA, 2004. ACM.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

Krizhevsky, Alex, Nair, Vinod, and Hinton, Geoffrey. Cifar-10 (canadian institute for advanced research).

Lang, Ken. Newsweeder: Learning to filter netnews. In *ICML*, 1995.

LeCun, Yann and Cortes, Corinna. MNIST handwritten digit database. 2010.

Lichman, M. UCI machine learning repository, 2013.

Maas, Andrew L., Daly, Raymond E., Pham, Peter T., Huang, Dan, Ng, Andrew Y., and Potts, Christopher. Learning word vectors for sentiment analysis. In *ACL: Human Language Technologies*. ACL, 2011.

Pang, Bo, Lee, Lillian, and Vaithyanathan, Shivakumar. Thumbs up?: Sentiment classification using machine learning techniques. In *EMNLP*, Stroudsburg, PA, USA, 2002. ACL.

Socher, Richard, Perelygin, Alex, Wu, Jean, Chuang, Jason, Manning, Christopher D., Ng, Andrew, and Potts, Christopher. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*. ACL, 2013.

Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jonathon, and Wojna, Zbigniew. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.