Pseudo-task Augmentation: From Deep Multitask Learning to Intratask Sharing—and Back

Elliot Meyerson [1 2]   **Risto Miikkulainen** [1 2]

Abstract

Deep multitask learning boosts performance by sharing learned structure across related tasks. This paper adapts ideas from deep multitask learning to the setting where only a single task is available. The method is formalized as *pseudo-task augmentation*, in which models are trained with multiple decoders for each task. Pseudo-tasks simulate the effect of training towards closely-related tasks drawn from the same universe. In a suite of experiments, pseudo-task augmentation improves performance on single-task learning problems. When combined with multitask learning, further improvements are achieved, including state-of-the-art performance on the CelebA dataset, showing that pseudo-task augmentation and multitask learning have complementary value. All in all, pseudo-task augmentation is a broadly applicable and efficient way to boost performance in deep learning systems.

1. Introduction

Multitask learning (MTL) (Caruana, 1998) improves performance by leveraging relationships between distinct learning problems. In recent years, MTL has been extended to deep learning, in which it has improved performance in applications such as vision (Zhang et al., 2014; Bilen & Vedaldi, 2016; Misra et al., 2016; Rudd et al., 2016; Lu et al., 2017; Rebuffi et al., 2017; Yang & Hospedales, 2017), natural language (Collobert & Weston, 2008; Dong et al., 2015; Liu et al., 2015a; Luong et al., 2016; Hashimoto et al., 2017), speech (Huang et al., 2013; Seltzer & Droppo, 2013; Huang et al., 2015; Wu et al., 2015), reinforcement learning (Devin et al., 2016; Jaderberg et al., 2017b; Teh et al., 2017), and even seemingly unrelated tasks from disparate domains (Kaiser et al., 2017; Meyerson & Miikkulainen,

2018). Deep MTL relies on training signals from multiple datasets to train deep structure that is shared across tasks. Since the shared structure must support solving multiple problems, it is inherently more general, which leads to better generalization to holdout data.

This paper adapts ideas from deep MTL to the single-task learning (STL) case, i.e., when only a single task is available for training. The method is formalized as *pseudo-task augmentation* (PTA), in which a single task has multiple distinct decoders projecting the output of the shared structure to task predictions. By training the shared structure to *solve the same problem in multiple ways*, PTA simulates the effect of training towards distinct but closely-related tasks drawn from the same universe. Theoretical justification shows how training dynamics with multiple pseudo-tasks strictly subsumes training with just one, and a class of algorithms is introduced for controlling pseudo-tasks in practice.

In an array of experiments, PTA is shown to significantly improve performance in single-task settings. Although different variants of PTA traverse the space of pseudo-tasks in qualitatively different ways, they all demonstrate substantial gains. Experiments also show that when PTA is combined with MTL, further improvements are achieved, including state-of-the-art performance on the CelebA dataset. In other words, although PTA can be seen as a base case of MTL, PTA and MTL have complementary value in learning more generalizable models. The conclusion is that pseudo-task augmentation is an efficient, reliable, and broadly applicable method for boosting performance in deep learning systems.

The remainder of the paper is organized as follows: Section 2 covers background on deep learning methods that train multiple models; Section 3 introduces the pseudo-task augmentation framework and practical implementations; Section 4 describes experimental setups and results; Sections 5 and 6 discuss future work and overall implications.

2. Training Multiple Deep Models

There is a broad range of methods that exploit synergies across multiple deep models. This section reviews these methods by classifying them into three types: (1) methods that jointly train a model for multiple tasks; (2) methods

---

[1]The University of Texas at Austin [2]Sentient Technologies, Inc.. Correspondence to: Elliot Meyerson <ekm@cs.utexas.edu>.

that train multiple models separately for a single task; and (3) methods that jointly train multiple models for a single task. This review motivates the development of methods in (3) that unify the advantages of (1) and (2).

2.1. Joint training of models for multiple tasks

There are many real-world scenarios where harnessing data from multiple related tasks can improve overall performance. In general, there are T tasks $\{\{\boldsymbol{x}_{ti}, \boldsymbol{y}_{ti}\}_{i=1}^{N_t}\}_{t=1}^{T}$, where N_t is the number of samples for the tth task. Note that it is possible that for $t_1 \neq t_2$, $N_{t_1} \neq N_{t_2}$, $\dim(\boldsymbol{x}_{t_1}) \neq \dim(\boldsymbol{x}_{t_2})$, and/or $\dim(\boldsymbol{y}_{t_1}) \neq \dim(\boldsymbol{y}_{t_2})$. The only requirement for multitask learning to be useful is that there is some amount of information shared across tasks, and, in theory, this is always the case (Mahmud & Ray, 2008; Mahmud, 2009).

Joint training of neural network models for multiple tasks was proposed decades ago (Caruana, 1998). Modern approaches have extended this early work to deep learning. Though more sophisticated methods now exist, the most common approach is still based on the original work, in which a joint model is decomposed into an underlying model $\mathcal{F}$ (parameterized by $\theta_{\mathcal{F}}$) that is shared across all tasks, and task-specific decoders $\mathcal{D}_t$ (parameterized by $\theta_{\mathcal{D}_t}$) for each task. The model for the tth task is then defined as

$$\hat{\boldsymbol{y}}_{ti} = \mathcal{D}_t(\mathcal{F}(\boldsymbol{x}_{ti}; \theta_{\mathcal{F}}); \theta_{\mathcal{D}_t}). \qquad (1)$$

Given a fixed model architecture for all $\mathcal{D}_t$ and $\mathcal{F}$, the joint model is completely defined by the parameters $\theta = (\{\theta_{\mathcal{D}_t}\}_{t=1}^{T}, \theta_{\mathcal{F}})$. To maximize overall performance, the goal is to find optimal parameters θ^* such that

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\boldsymbol{y}_{ti}, \hat{\boldsymbol{y}}_{ti}) \qquad (2)$$

for a suitable sample-wise loss function $\mathcal{L}$, e.g., mean squared error or cross-entropy loss. More sophisticated deep MTL approaches can be characterized by the design decision of *how* learned structure is shared across tasks. For example, some methods supervise different tasks at different depths of the shared structure (Zhang & Weiss, 2016; Hashimoto et al., 2017; Toshniwal et al., 2017); other methods duplicate the shared structure into columns and define mechanisms for sharing information across columns (Jou & Chang, 2016; Misra et al., 2016; Long et al., 2017; Yang & Hospedales, 2017). More detailed characterizations of deep MTL methods can be found in previous work (Ruder, 2017; Meyerson & Miikkulainen, 2018).

MTL has also been explored extensively outside of deep learning. Many such techniques take a similar approach of having shared structure with a separate linear decoder for each task, while enforcing regularization of shared convex structure (Evgeniou & Pontil, 2004; Argyriou et al., 2008;

Kang et al., 2011; Kumar & Daumé, 2012). Overall, by requiring models to fit multiple real world datasets simultaneously, MTL is a promising approach to learning more realistic, and thus more generalizable, models.

2.2. Separate training of multiple models for STL

How to construct and train deep a neural network effectively is an open-ended design problem even in the case of a single task. A range of methods have been developed that aim at overcoming this problem by training multiple models separately for a single task. One class of methods searches for optimal fixed designs, e.g., by automatically optimizing learning hyperparameters (Bergstra et al., 2011; Snoek et al., 2012) or more open-ended network topologies (Miikkulainen et al., 2017; Real et al., 2017; Zoph & Le, 2017). The multiple models synergize by providing complementary information about different areas of the search space, and, over time, the results of past models can be used to generate better models. Population-based training takes this one step further, by copying the weights of successful models to new models (Jaderberg et al., 2017a). This weight-copying is similar to methods that transfer learned behavior across a sequence of pre-defined architectures (Hinton et al., 2015; Chen et al., 2016; Wei et al., 2016). The synergy of multiple models can also be exploited via ensembling (Dietterich, 2000). Overall, the widespread success of the above methods have shown the value of training multiple models separately, both sequentially and in parallel.

2.3. Joint training of multiple models for STL

Some existing methods can be viewed as jointly training multiple models for a single task. For instance, to improve training of deep models, deep supervision includes loss layers at multiple depths (Lee et al., 2015). As a by-product, this approach yields a distinct model for the task at each such depth, though only the deepest model is ever evaluated. As another example, dropout (Srivastava et al., 2014), and pseudo-ensembles more generally (Bachman et al., 2014), can be seen as implicitly training many relatively weak models that are combined during evaluation. Also, Path-Net (Fernando et al., 2017) jointly trains multiple networks induced by various paths through a set of shared modules. However, the goal is not to improve single task performance, but discover structure that can be effectively reused by future tasks. Although these existing methods jointly train multiple models for a single task, they do not perform joint training in the MTL sense. Ideally, the benefits of the methods in Sections 2.1 and 2.2 could be combined, yielding methods that train multiple models that share underlying parameters *and* sample complementary high-performing areas of the model space. This paper takes first steps in that direction, showing that such methods are indeed promising. The specific approach, PTA, is introduced in the next section.

3. Pseudo-task Augmentation (PTA)

This section introduces the PTA method. First, the classical deep MTL approach is extended to the case of multiple decoders per task. Then, the concept of a pseudo-task is introduced, and increased training dynamics under multiple pseudo-tasks is demonstrated. Finally, practical methods for controlling pseudo-tasks during training are described, which will be compared empirically in Section 4.

3.1. A Classical Approach

The most common approach to deep MTL is still the "classical" approach (Eq. 1), in which all layers are shared across all tasks up to a high level, after which each task learns a distinct decoder that maps high-level points to its task-specific output space (Caruana, 1998; Ranjan et al., 2016; Lu et al., 2017). Even when more sophisticated methods are developed, the classical approach is often used as a baseline for comparison. The classical approach is also computationally efficient, in that the only additional parameters beyond a single task model are in the additional decoders. Thus, when applying ideas from deep MTL to single-task multi-model learning, the classical approach is a natural starting point.

Consider again the case where there are T distinct *true* tasks, but now let there be D decoders for each task. Then, the model for the dth decoder of the tth task is given by

$$\hat{\boldsymbol{y}}_{tdi} = \mathcal{D}_{td}(\mathcal{F}(\boldsymbol{x}_{ti}; \theta_{\mathcal{F}}); \theta_{\mathcal{D}_{td}}), \qquad (3)$$

and the overall loss for the joint model from Eq. 2 becomes

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{TD} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \sum_{d=1}^{D} \mathcal{L}(\boldsymbol{y}_{ti}, \hat{\boldsymbol{y}}_{tdi}), \qquad (4)$$

where $\theta = (\{\{\theta_{\mathcal{D}_{td}}\}_{d=1}^{D}\}_{t=1}^{T}, \theta_{\mathcal{F}})$. In the same way as the classical approach to MTL encourages $\mathcal{F}$ to be more general and robust by requiring it to support multiple tasks, here $\mathcal{F}$ is required to support *solving the same task in multiple ways*. A visualization of a resulting joint model is shown in Figure 1. A theme in MTL is that models for related tasks will have similar decoders, as implemented by explicit regularization (Evgeniou & Pontil, 2004; Kumar & Daumé, 2012; Long et al., 2017; Yang & Hospedales, 2017). Similarly, in Eq. 4, through training, two decoders for the same task will instantiate similar models, and, as long as they do not converge completely to equality, they will simulate the effect of training with multiple closely-related tasks.

Notice that the innermost summation in Eq. 4 is over decoders. This calculation is computationally efficient: because each decoder for a given task takes the same input, $\mathcal{F}(\boldsymbol{x}_{ti})$ (usually the most expensive part of the model) need only be computed once per sample (and only once over all tasks if all tasks share $\boldsymbol{x}_{ti}$). However, when evaluating the
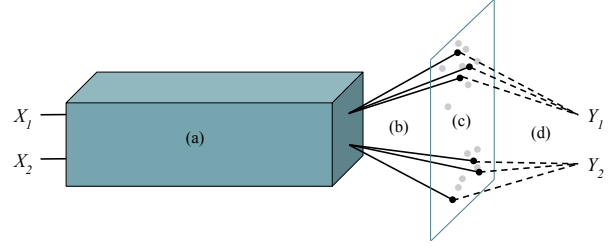

Figure 1. General setup for pseudo-task augmentation with two tasks. (a) *Underlying model.* All task inputs are embedded through an underlying model that is completely shared; (b) *Multiple decoders.* Each task has multiple decoders (solid black lines) each projecting the embedding to a distinct classification layer; (c) *Parallel traversal of model space.* The underlying model coupled with a decoder defines a task model. Task models populate a model space, with current models shown as black dots and previous models shown as gray dots; (d) *Multiple loss signals.* Each current task model receives a distinct loss to compute its distinct gradient. A task coupled with a decoder and its parameters defines a pseudo-task for the underlying model.

performance of a model, since each decoder induces a distinct model for a task, what matters is not the average over decoders, but the best performing decoder for each task, i.e.,

$$\theta_{\text{eval}}^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{N_t} \underset{d \in 1..D}{\operatorname{argmin}} \sum_{i=1}^{N_t} \mathcal{L}(\boldsymbol{y}_{ti}, \hat{\boldsymbol{y}}_{tdi}). \quad (5)$$

Eq. 4 is used in training because it is smoother; Equation 5 is used for model validation, and to select the best performing decoder for each task from the final joint model. This decoder is then applied to future data, e.g., a holdout set. Once the models are trained, in principle they form a set of distinct and equally powerful models for each task. It may therefore be tempting to ensemble them for evaluation, i.e.,

$$\theta_{\text{ens}}^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}\left(\boldsymbol{y}_{ti}, \frac{1}{D} \sum_{d=1}^{D} \hat{\boldsymbol{y}}_{tdi}\right). \quad (6)$$

However, with linear decoders, *training* with Eq. 6 is equivalent to training with a single decoder for each task, while training with Eq. 4 with multiple decoders yields more expressive training dynamics. These ideas are developed more fully in the next section.

3.2. Pseudo-tasks

Following the intuition that training $\mathcal{F}$ with multiple decoders amounts to solving the task in multiple ways, each "way" is defined by a *pseudo-task*

$$(\mathcal{D}_{td}, \theta_{td}, \{\boldsymbol{x}_{ti}, \boldsymbol{y}_{ti}\}_{i=1}^{N_t}) \qquad (7)$$

of the *true* underlying task $\{\boldsymbol{x}_{ti}, \boldsymbol{y}_{ti}\}_{i=1}^{N_t}$. It is termed a pseudo-task because it derives from a true task, but has no

fixed labels. That is, for any fixed θ_{td}, there are potentially many optimal outputs for $\mathcal{F}(\boldsymbol{x}_{ti})$. When $D > 1$, training $\mathcal{F}$ amounts to training each task with multiple pseudo-tasks for each task at each gradient update step. This process is the essence of PTA.

As a first step, this paper considers linear decoders, i.e. each $\mathcal{D}_{td}$ consists of a single dense layer of weights (any following nonlinearity can be considered part of the loss function). ==Prior work has assumed that models for closely-related tasks differ only by a linear transformation== (Evgeniou & Pontil, 2004; Kang et al., 2011; Argyriou et al., 2008). Similarly, with linear decoders, distinct pseudo-tasks for the same task simulate multiple closely-related tasks. When θ_{td} are considered fixed, the learning problem (Eq. 4) reduces to

$$\theta_{\mathcal{F}}^* = \operatorname*{argmin}_{\theta_{\mathcal{F}}} \frac{1}{TD} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \sum_{d=1}^{D} \mathcal{L}(\boldsymbol{y}_{ti}, \hat{\boldsymbol{y}}_{tdi}). \quad (8)$$

In other words, although the overall goal is to learn models for T tasks, $\mathcal{F}$ is at each step optimized towards DT pseudo-tasks. ==Thus, training with multiple decoders may yield positive effects similar to training with multiple true tasks.==

After training, the best model for a given task is selected from the final joint model, and used as the final model for that task (Eq. 5). Of course, using multiple decoders with identical architectures for a single task does not make the final learned predictive models more expressive. It is therefore natural to ask whether including additional decoders has any fundamental effect on learning dynamics. ==It turns out that even in the case of linear decoders, the training dynamics of using multiple pseudo-tasks strictly subsumes using just one.==

Definition 1 (Pseudo-task Simulation). *A set of pseudo-tasks S_1 simulates another S_2 on $\mathcal{F}$ if for all $\theta_{\mathcal{F}}$ the gradient update to $\theta_{\mathcal{F}}$ when trained with S_1 is equal to that with S_2.*

Theorem 1 (Augmented Training Dynamics). *There exist differentiable functions $\mathcal{F}$ and sets of pseudo-tasks of a single task that cannot be simulated by a single pseudo-task of that task, even when all decoders are linear.*

Proof. Consider a task with a single sample $(\boldsymbol{x}, y)$, where y is a scalar. Suppose $\mathcal{L}$ (from Eq. 8) computes mean squared error, $\mathcal{F}$ has output dimension M, and all decoders are linear, with bias terms omitted for clarity. $\mathcal{D}_d$ is then completely specified by the vector $\boldsymbol{w}_d = \langle w_d^1, w_d^2, ..., w_d^M \rangle^\top$. Suppose parameter updates are performed by gradient descent. The update rule for $\theta_{\mathcal{F}}$ with fixed decoders $\{\mathcal{D}_d\}_{d=1}^{D}$ and learning rate α is then given by

$$\theta_{\mathcal{F}} := \theta_{\mathcal{F}} - \alpha \sum_{d=1}^{D} \nabla_{\mathcal{F}} \left(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_{\mathcal{F}}) \right)^2. \quad (9)$$

For a single fixed decoder to yield equivalent behavior, it must have equivalent update steps. The goal then is to

choose $(\boldsymbol{x}, y)$, $\mathcal{F}$, $\{\theta_k\}_{k=1}^K$, $\{\boldsymbol{w}_d\}_{d=1}^D$, and $\alpha > 0$, such that there are no $\boldsymbol{w}_o, \gamma > 0$, for which $\forall k$

$$\alpha \sum_{d=1}^{D} \nabla_{\mathcal{F}} (y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k))^2 =$$
$$\gamma \nabla_{\mathcal{F}} (y - \boldsymbol{w}_o^\top \mathcal{F}(\boldsymbol{x}; \theta_k))^2$$
$$\implies \alpha \sum_{d=1}^{D} 2(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) \boldsymbol{w}_d^\top J_{\mathcal{F}}(\boldsymbol{x}; \theta_k) =$$
$$2\gamma (y - \boldsymbol{w}_o^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) \boldsymbol{w}_o^\top J_{\mathcal{F}}(\boldsymbol{x}; \theta_k), \quad (10)$$

where $J_{\mathcal{F}}$ is the Jacobian of $\mathcal{F}$. By choosing $\mathcal{F}$ and $\{\theta_k\}_{k=1}^K$ so that all $J_{\mathcal{F}}(\boldsymbol{x}; \theta_k)$ have full row rank, Eq. 10 reduces to

$$\alpha \sum_{d=1}^{D} (y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_d^i =$$
$$\gamma (y - \boldsymbol{w}_o^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_o^i \quad \forall i \in 1..M. \quad (11)$$

Choosing $\mathcal{F}$, $\{\theta_k\}_{k=1}^K$, $\{\boldsymbol{w}_d\}_{d=1}^D$, and $\alpha > 0$ such that the left hand side of Eq. 11 is never zero, we can safely write

$$\frac{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_d^i}{(y - \boldsymbol{w}_o^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_o^i} =$$
$$\frac{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_d^j}{(y - \boldsymbol{w}_o^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_o^j} \quad \forall (i, j)$$
$$\implies \frac{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_d^i}{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_k)) w_d^j} = \frac{w_o^i}{w_o^j}. \quad (12)$$

Then, since $\boldsymbol{w}_o$ is fixed, it suffices to find $\mathcal{F}(\boldsymbol{x}; \theta_1), \mathcal{F}(\boldsymbol{x}; \theta_2)$ such that for some (i, j)

$$\frac{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_1)) w_d^i}{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_1)) w_d^j} \neq$$
$$\frac{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_2)) w_d^i}{\sum_{d=1}^{D}(y - \boldsymbol{w}_d^\top \mathcal{F}(\boldsymbol{x}; \theta_2)) w_d^j}. \quad (13)$$

For instance, with $D = 2$, choosing $y = 1$, $w_1 = \langle 2, 3 \rangle^\top$, $w_2 = \langle 4, 5 \rangle^\top$, $\mathcal{F}(\boldsymbol{x}; \theta_1) = \langle 6, 7 \rangle^\top$, and $\mathcal{F}(\boldsymbol{x}; \theta_1) = \langle 8, 9 \rangle^\top$ satisfies the inequality. Note $\mathcal{F}(\boldsymbol{x}; \theta_1)$ and $\mathcal{F}(\boldsymbol{x}; \theta_2)$ can be chosen arbitrarily since $\mathcal{F}$ is only required to be differentiable, e.g., implemented by a neural network. $\square$

==Showing that a single pseudo-task can be simulated by D pseudo-tasks for any $D > 1$ is more direct:== For any $\boldsymbol{w}_o$ and γ, choose $\boldsymbol{w}_d = \boldsymbol{w}_o \, \forall \, d \in 1..D$ and $\alpha = \gamma/D$. Further extensions to tasks with more samples, higher dimensional outputs, and cross-entropy loss are straightforward. Note that this result is related to work on the dynamics of deep linear models (Saxe et al., 2014), in that adding additional linear structure complexifies training dynamics. However,

training an ensemble directly, i.e., via Eq. 6, does *not* yield augmented training dynamics, since

$$\frac{1}{D}\sum_{d=1}^{D}\hat{\boldsymbol{y}}_{tdi} = \frac{1}{D}\sum_{d=1}^{D}\boldsymbol{W}_{td}^{\top}\mathcal{F}(\boldsymbol{x}_{ti};\theta_{\mathcal{F}})$$

$$\implies \boldsymbol{W}_{to}^{\top} = \frac{1}{D}\sum_{d=1}^{D}\boldsymbol{W}_{td}^{\top} \text{ and } \beta = \alpha. \quad (14)$$

Now that we know that t<mark>raining with additional pseudo-tasks yields augmented training dynamics that may be exploited</mark>, the question is how to take advantage of these dynamics in practice. The next section introduces methods to address this question.

3.3. Control of Multiple Pseudo-task Trajectories

Given linear decoders, the primary goal is to optimize $\mathcal{F}$; if an optimal $\mathcal{F}$ were found, optimal decoders for each task could be derived analytically. So, <mark>given multiple linear decoders for each task, how should their induced pseudo-tasks be controlled to maximize the benefit to $\mathcal{F}$</mark>? For one, their weights $\{\boldsymbol{W}_{td}\}_{d=1}^{D}$ must not all be equal, otherwise we would have $\boldsymbol{W}_{to} = \boldsymbol{W}_{t1}$ and $\gamma = D\alpha$ in the proof of Theorem 1. Following Eq. 4, decoders can be trained jointly with $\mathcal{F}$ via gradient-based methods, so that they learn to work well with $\mathcal{F}$. Through optimization, a trained decoder induces a trajectory of pseudo-tasks. Going beyond this *implicit* control, Algorithm 1 gives a high-level framework for applying *explicit* control to pseudo-task trajectories.

Algorithm 1 PTA Training Framework

1: Given T tasks $\{\{\boldsymbol{x}_{ti}, \boldsymbol{y}_{ti}\}_{i=1}^{N_t}\}_{t=1}^{T}$, and D decoders per task
2: $\{\{\theta_{\mathcal{D}_{td}}\}_{d=1}^{D}\}_{t=1}^{T} \leftarrow DecInitialize()$
3: Initialize $\theta_{\mathcal{F}}$
4: Initialize decoder costs $c_{td} \leftarrow \infty \; \forall \; (t,d)$
5: **while** not done training **do**
6:    **for** $m = 1$ **to** M **do**   ▷ M is meta-iteration length
7:       Update $\theta_{\mathcal{F}}$ and $\theta_{\mathcal{D}_{td}}$ via a joint gradient step.
8:    **for** $t = 1$ **to** T **do**
9:       **for** $d = 1$ **to** D **do**
10:         $c_{td} \leftarrow \text{evaluate}(\theta_{\mathcal{D}_{td}}, \theta_{\mathcal{F}}, t)$ ▷ e.g., get validation error
11:       **for** $d = 1$ **to** D **do**
12:         $\theta_{\mathcal{D}_{td}} \leftarrow DecUpdate\big(d, \{\theta_{\mathcal{D}_{td_o}}, c_{td_o}\}_{d_o=1}^{D}\big)$
13: **return** $\big(\{\{\theta_{\mathcal{D}_{td}}\}_{d=1}^{D}\}_{t=1}^{T}, \theta_{\mathcal{F}}\big)$

An instance of the algorithm is parameterized by choices for *DecInitialize*, which defines how decoders are initialized; and *DecUpdate*, which defines non-gradient-based updates to decoders every M gradient steps, i.e., every *meta-iteration*, based on the performance of each decoder (*DecUpdate* defaults to no-op). As a first step, several intuitive methods are evaluated in this paper for instantiating Algorithm 1. These methods can be used together in any combination:

Independent Initialization (I) *DecInitialize* randomly initializes all $\theta_{\mathcal{D}_{td}}$ independently. This is the obvious initialization method, and is assumed in all methods below.

Freeze (F) *DecInitialize* freezes all decoder weights except $\theta_{\mathcal{D}_{t1}}$ for each task. Frozen weights do not receive gradient updates in Line 7 of Algorithm 1. Because they cannot adapt to $\mathcal{F}$, constant pseudo-task trajectories provide a stricter constraint on $\mathcal{F}$. One decoder is left unfrozen so that the optimal model for each task can still be learned.

Independent Dropout (D) *DecInitialize* sets up the dropout layers preceding linear decoder layers to drop out values independently for each decoder. Thus, even when the weights of two decoders for a task are equal, their resulting gradient updates to $\mathcal{F}$ and to themselves will be different.

For the next three methods, let $c_t^{\min} = \min(c_{t1}, \ldots, c_{tD})$.

Perturb (P) *DecUpdate* adds noise $\sim \mathcal{N}(\mathbf{0}, \epsilon_p \boldsymbol{I})$ to each $\theta_{\mathcal{D}_{td}}$ for all d where $c_{td} \neq c_t^{\min}$. This method ensures that $\theta_{\mathcal{D}_{td}}$ are sufficiently distinct before each training period.

Hyperperturb (H) Like *Perturb*, except *DecUpdate* updates the hyperparameters of each decoder other than the best for each task, by adding noise $\sim \mathcal{N}(0, \epsilon_h)$. In this paper, each decoder has only one hyperparameter: the dropout rate of any *Independent Dropout* layer, because adapting dropout rates can be beneficial (Ba & Frey, 2013; Li et al., 2016; Jaderberg et al., 2017a).

Greedy (G) For each task, let $\theta_t^{\min}$ be the weights of a decoder with cost $c_t^{\min}$. *DecUpdate* updates all $\theta_{td} := \theta_t^{\min}$, including hyperparameters. This biases training to explore the highest-performing areas of the pseudo-task space. When combined with any of the previous three methods, decoder weights are still ensured to be distinct through training.

Combinations of these six methods induce an initial class of PTA training algorithms PTA-* for the case of linear decoders. The next section evaluates eight representative combinations of these methods, i.e., PTA-I, PTA-F, PTA-P, PTA-D, PTA-FP, PTA-GP, PTA-GD, and PTA-HGD, in various experimental settings. Note that H and G are related to methods that copy the weights of the entire network (Jaderberg et al., 2017a). Also note that, in a possible future extension to the nonlinear case, the space of possible PTA control methods becomes much more broad, as will be discussed in Section 5.

4. Experiments

In this section, PTA methods are evaluated and shown to excel in a range of settings: (1) single-task character recognition; (2) multitask character recognition; (3) single-task sentiment classification; and (4) multitask visual attribute classification. All experiments are implemented using the
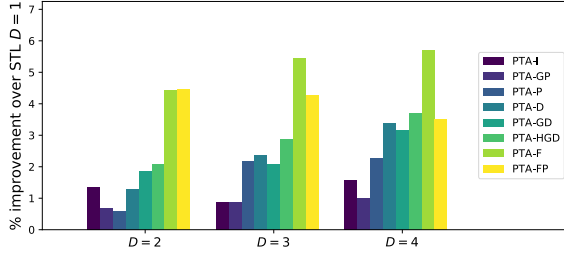
Figure 2. **Omniglot single-task learning results.** For each number of decoders D, mean improvement (absolute % decrease in error) over $D = 1$ is plotted for each setup, averaged across all tasks. All setups outperform the baseline. PTA-F and PTA-FP performs best, as this problem benefits from strong regularization. The mean improvement across all methods also increases with D: 1.86% for $D = 2$; 2.33% for $D = 3$; and 2.70% for $D = 4$.

Figure 3. **Omniglot multitask learning results.** For each number of decoders D, mean improvement (absolute % decrease in error) across all tasks is plotted over STL with $D = 1$. All setups outperform the STL baseline, and all except PTA-I with $D = 2$ outperform the MTL baseline. Again, PTA-F and PTA-FP perform best, and the mean improvement across all methods increases with D: 3.63% for $D = 2$; 4.07% for $D = 3$; and 4.37% for $D = 4$.

Keras framework (Chollet et al., 2015). For PTA-P and PTA-GP, $\epsilon_p = 0.01$; for PTA-HGD, $\epsilon_h = 0.1$ and dropout rates range from 0.2 to 0.8. A dropout layer with dropout rate initialized to 0.5 precedes each decoder.

4.1. Omniglot Character Recognition

This section evaluates and compares the various PTA methods on Omniglot character recognition (Lake et al., 2015). The Omniglot dataset consists of 50 alphabets of handwritten characters, each of which induces its own character recognition task. Each character instance is a 105×105 black-and-white image, and each character has 20 instances, each drawn by a different individual. To reduce variance and improve reproducibility of experiments, a fixed random 50/20/30% train/validation/test split was used for each task. (These splits will be released with the paper.) Methods are evaluated with respect to all 50 tasks as well as a subset consisting of the first 20 tasks in a fixed random ordering of alphabets used in previous work (Meyerson & Miikkulainen, 2018). The underlying model $\mathcal{F}$ for all setups is a simple four layer convolutional network that has been shown to yield good performance on Omniglot (Meyerson & Miikkulainen, 2018). This model has four convolutional layers each with 53 filters and 3×3 kernels, and each followed by a 2×2 max-pooling layer and dropout layer with 0.5 dropout probability. At each meta-iteration, 250 gradient updates are performed via Adam (Kingma & Ba, 2014); each setup is trained for 100 meta-iterations.

4.1.1. OMNIGLOT: SINGLE-TASK LEARNING

The single-task learning case is considered first. For each of the 20 initial Omniglot tasks, the eight PTA methods were applied to the task with 2, 3, and 4 decoders. At least three trials were run with each setup; the mean performance averaged across trials and tasks is shown in Figure 2. Every PTA setup outperforms the baseline, i.e., training with a single
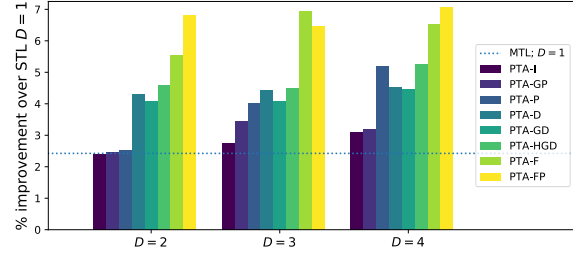
decoder. The methods that use decoder freezing, PTA-F and PTA-FP, perform best, showing how this problem can benefit from strong regularization. Notably, the mean improvement across all methods increases with D: 1.86% for $D = 2$; 2.33% for $D = 3$; and 2.70% for $D = 4$. Like MTL can benefit from adding more tasks (Caruana, 1998; Hashimoto et al., 2017; Jaderberg et al., 2017b), single-task learning can benefit from adding more pseudo-tasks.

4.1.2. OMNIGLOT: MULTITASK LEARNING

Omniglot models have also been shown to benefit from MTL (Maclaurin et al., 2015; Rebuffi et al., 2017; Yang & Hospedales, 2017; Meyerson & Miikkulainen, 2018). This section extends the experiments in Section 4.1.1 to MTL. The setup is exactly the same, except now the underlying convolutional model is fully shared across all tasks for each method. The results are shown in Figure 3. All setups outperform the STL baseline, and all, except for PTA-I with $D = 2$, outperform the MTL baseline. Again, PTA-F and PTA-FP perform best, and the mean improvement across all methods increases with D. The results show that although PTA implements behavior similar to MTL, when combined, their positive effects are complementary. Finally, to test the scalability of these results, three diverse PTA methods with $D = 4$ and $D = 10$ were applied to the complete 50-task dataset: PTA-I, because it is the baseline PTA method; PTA-F, because it is simple and high-performing; and PTA-HGD, because it is the most different from PTA-F, but also relatively high-performing. The results are given in Table 1. The results agree with the 20-task results, with all methods improving upon the baseline, and performance overall improving as D is increased.

4.2. IMDB Sentiment Analysis

The experiments in this section apply PTA to LSTM models in the IMDB sentiment classification problem (Maas et al.,

Table 1. **Omniglot 50-task results.** Test error averaged across all tasks for each setup is shown. Overall, the performance gains from MTL complement those from PTA, with PTA-F again the highest-performing and most robust method.

Method	Single-task Learning		Multitask Learning	
Baseline	35.49		29.02	
	$D = 4$	$D = 10$	$D = 4$	$D = 10$
PTA-I	31.72	32.56	27.26	24.50
PTA-HGD	31.63	30.39	25.77	26.55
PTA-F	**29.37**	**28.48**	**23.45**	**23.36**
PTA-Mean	30.91	30.48	25.49	24.80

Table 2. **IMDB Results.** All PTA methods outperform the LSTM baseline. The best performance is achieved by PTA-HGD with $D = 10$. This method receives a substantial boost from increasing the number of decoders from 4 to 10, as the greedy algorithm gets to perform broader search. On the other hand, PTA-I and PTA-F do not improve with the additional decoders, suggesting that, without careful control, too many decoders can overconstrain $\mathcal{F}$.

Method	Test Accuracy %	
LSTM Baseline ($D = 1$)	82.75 ($\pm$0.13)	
	$D = 4$	$D = 10$
PTA-I	83.20 ($\pm$0.07)	83.02 ($\pm$0.11)
PTA-HGD	83.22 ($\pm$0.05)	**83.51** ($\pm$0.08)
PTA-F	**83.30** ($\pm$0.12)	83.30 ($\pm$0.08)

2011). The dataset consists of 50K natural-language movie reviews, 25K for training and 25K for testing. There is a single binary classification task: whether a review is positive or negative. As in previous work, 2500 of the training reviews are withheld for validation (McCann et al., 2017). The underlying model $\mathcal{F}$ is the off-the-shelf LSTM model for IMDB provided by Keras, with no parameters or preprocessing changed. In particular, the vocabulary is capped at 20K words, the LSTM layer has 128 units and dropout rate 0.2, and each meta-iteration consists of one epoch of training with Adam (Kingma & Ba, 2014). This is not a state-of-the-art model, but it is a very different architecture from that used in Omniglot, and therefore serves to demonstrate the broad applicability of PTA.

The final three PTA methods from Section 4.1 were evaluated with 4 and 10 decoders (Table 2). As in Section 4.1, all PTA methods outperform the baseline. In this case, however, PTA-HGD with $D = 10$ performs best. Notably, PTA-I and PTA-F do not improve from $D = 4$ to $D = 10$, suggesting that underlying models have a critical point after which, without careful control, too many decoders can be overconstraining. To contrast PTA with standard regularization, additional Baseline experiments were run with dropout rates $[0.3, 0.4, ..., 0.9]$. At 0.5 the best accuracy was achieved:

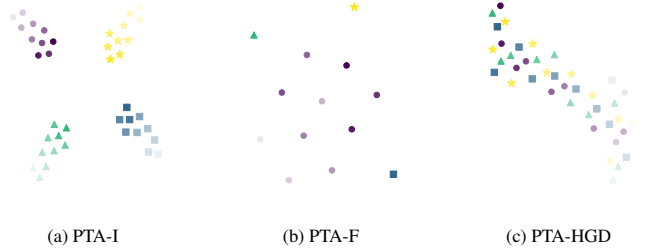(a) PTA-I   (b) PTA-F   (c) PTA-HGD

Figure 4. **Pseudo-task Trajectories.** t-SNE (van der Maaten & Hinton, 2008) projections of pseudo-task trajectories, for runs of PTA-I, PTA-F, and PTA-HGD on IMDB. Each shape corresponds to a particular decoder; each point is a projection of the length-129 weight vector at the end of an epoch, with opacity increasing by epoch. The behavior matches our intuition for what should be happening in each case: (a) When decoders are only initialized independently, their pseudo-tasks gradually converge; (b) when all but one decoder is frozen, the unfrozen one settles between the others; (c) when a greedy method is used, decoders perform local exploration as they traverse the pseudo-task space together.

83.14 ($\pm$0.05), which is less than all PTA variants except PTA-I with $D = 10$, thus confirming that PTA adds value. To help understand what each PTA method is actually doing, snapshots of decoder parameters taken every epoch are visualized in Figure 4 with t-SNE (van der Maaten & Hinton, 2008) using cosine distance. The behavior matches our intuition for what should be happening in each case: When decoders are only initialized independently, their pseudo-tasks gradually converge; when all but one decoder is frozen, the unfrozen one settles between the others; and when a greedy method is used, decoders perform local exploration as they traverse the pseudo-task space together.

4.3. CelebA Facial Attribute Recognition

To further test applicability and scalability, PTA was evaluated on CelebA large-scale facial attribute recognition (Liu et al., 2015b). The dataset consists of $\approx$200K 178×218 color images. Each image has binary labels for 40 facial attributes; each attribute induces a binary classification task. Facial attributes are related at a high level that deep models can exploit, making CelebA a popular deep MTL benchmark. Thus, this experiment focuses on the MTL setting.

The underlying model was Inception-ResNet-v2 (Szegedy et al., 2016), with weights initialized from training on ImageNet (Russakovsky et al., 2015). Due to computational constraints, only one PTA method was evaluated: PTA-HGD with $D = 10$. PTA-HGD was chosen because of its superior performance on IMDB, and because CelebA is a large-scale problem that may require extended pseudo-task exploration; Figure 4 shows how PTA-HGD may support such exploration above other methods. Each meta-iteration consists

Table 3. **CelebA results.** Comparison of PTA against state-of-the-art methods for CelebA, with and without ensembling. Test error is averaged across all attributes. PTA-HGD outperforms all other methods, establishing a new state-of-the-art in this benchmark.

| MTL Method	% Error
Single Task (He et al., 2017)	10.37
MOON (Rudd et al., 2016)	9.06
Adaptive Sharing (Lu et al., 2017)	8.74
MCNN-AUX (Hand & Chellappa, 2017)	8.71
Soft Order (Meyerson & Miikkulainen, 2018)	8.64
VGG-16 MTL (Lu et al., 2017)	8.56
Adaptive Weighting (He et al., 2017)	8.20
AFFACT (Günther et al., 2017) (best of 3)	8.16
MTL Baseline (Ours; mean of 3)	8.14
PTA-HGD, $D = 10$ (mean of 3)	**8.10**
Ensemble of 3: AFFACT (Günther et al., 2017)	8.00
Ensemble of 3: PTA-HGD, $D = 10$	**7.94**
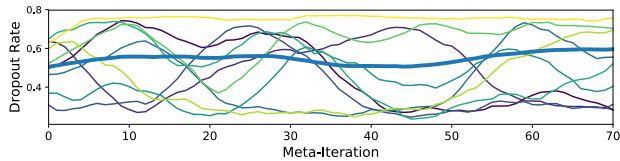
Figure 5. **CelebA dropout schedules.** The thick blue line shows the mean dropout schedule across all 400 pseudo-tasks in a run of PTA-HGD. Each of the remaining lines shows the schedule of a particular task, averaged across their 10 pseudo-tasks. All lines are plotted with a simple moving average of length 10. The diversity of schedules shows that the system is taking advantage of PTA-HGD's ability to adapt task-specific hyperparameter schedules.

of 250 gradient updates with batch size 32. The optimizer schedule is co-opted from previous work (Günther et al., 2017): RMSprop is initialized with a learning rate of 10^{-4}, which is decreased to 10^{-5} and 10^{-6} when the model converges. PTA-HGD and the MTL baseline were each trained three times. The computational overhead of PTA-HGD is marginal, since the underlying model has 54M parameters, while each decoder has only 1.5K. Table 3 shows the results. PTA-HGD outperforms all other methods, thus establishing a new state-of-the-art in CelebA. Figure 5 shows resulting dropout schedules for PTA-HGD. No one type of schedule dominates; PTA-HGD gives each task the flexibility to adapt its own schedule via the performance of its pseudo-tasks.

5. Discussion and Future Work

The experiments in this paper demonstrated that PTA is broadly applicable, and that it can boost performance in a variety of single-task and multitask problems. Training with multiple decoders for a single task allows a broader set of models to be visited. If these decoders are diverse and perform well, then the shared structure has learned to

solve the same problem in diverse ways, which is a hallmark of robust intelligence. In the MTL setting, controlling each task's pseudo-tasks independently makes it possible to discover diverse task-specific learning dynamics (Figure 5). Increasing the number of decoders can also increase the chance that pairs of decoders align well across tasks.

The crux of PTA is the method for controlling pseudo-task trajectories. Experiments showed that the amount of improvement from PTA is dependent on the choice of control method. Different methods exhibit highly structured but different behavior (Figure 4). The success of initial methods indicates that developing more sophisticated methods is a promising avenue of future work. In particular, methods from Section 2.2 can be co-opted to control pseudo-task trajectories more effectively. Consider, for instance, the most involved method evaluated in this paper: PTA-HGD. This online decoder search method could be replaced by methods that generate new models more intelligently (Bergstra et al., 2011; Snoek et al., 2012; Miikkulainen et al., 2017; Real et al., 2017; Zoph & Le, 2017). Such methods will be especially useful in extending PTA beyond the linear case considered in this paper, to complex nonlinear decoders. For example, since a set of decoders is being trained in parallel, it could be natural to use neural architecture search methods (Miikkulainen et al., 2017; Real et al., 2017; Zoph & Le, 2017) to search for optimal decoder architectures. While ensembling separate PTA models is useful (Table 3), in preliminary tests naïvely ensembling decoders for evaluation (Eq. 6) did not yield remarkable improvements over the single best (Eq. 5). In a further preliminary test with IMDB, when $\mathcal{F}$ was *not* shared, PTA-I outperformed PTA-HGD and PTA-F, indicating that the latter two methods address dynamics that arise in joint training but not naïve ensemble training. Developing PTA training methods for generating a more complementary set of decoders, coupled with effective methods for ensembling this set, could push performance even further, especially when decoders are more complex.

6. Conclusion

This paper has introduced *pseudo-task augmentation*, a method that makes it possible to apply ideas from deep MTL to single-task learning. By training shared structure to solve the same task in multiple ways, pseudo-task augmentation simulates training with multiple closely-related tasks, yielding performance improvements similar to those in MTL. However, the methods are complementary: combining pseudo-task augmentation with MTL results in further performance gains. Broadly applicable, pseudo-task augmentation is thus a promising method for improving deep learning performance. Overall, this paper has taken first steps towards a future class of efficient model search algorithms that exploit intratask parameter sharing.

Acknowledgements

References

Argyriou, A., Evgeniou, T., and Pontil, M. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, Dec 2008.

Ba, J. and Frey, B. Adaptive dropout for training deep neural networks. In *NIPS*, pp. 3084–3092. 2013.

Bachman, P., Alsharif, O., and Precup, D. Learning with pseudo-ensembles. In *NIPS*, pp. 3365–3373. 2014.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pp. 2546–2554. 2011.

Bilen, H. and Vedaldi, A. Integrated perception with recurrent multi-task neural networks. In *NIPS*, pp. 235–243. 2016.

Caruana, R. Multitask learning. In *Learning to learn*, pp. 95–133. Springer US, 1998.

Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. In *Proc. of ICLR*, 2016.

Chollet, F. et al. Keras, 2015.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. of ICML*, pp. 160–167, 2008.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016.

Dietterich, T. G. Ensemble methods in machine learning. *International workshop on multiple classifier systems*, pp. 1–15, 2000.

Dong, D., Wu, H., He, W., Yu, D., and Wang, H. Multi-task learning for multiple language translation. In *Proc. of ACL*, pp. 1723–1732, 2015.

Evgeniou, T. and Pontil, M. Regularized multi–task learning. In *Proc. of KDD*, pp. 109–117, 2004.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017.

Günther, M., Rozsa, A., and Boult, T. E. AFFACT - alignment free facial attribute classification technique. *CoRR*, abs/1611.06158v2, 2017.

Hand, E. M. and Chellappa, R. Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification. In *Proc. of AAAI*, pp. 4068–4074, 2017.

Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. A joint many-task model: Growing a neural network for multiple NLP tasks. In *Proc. of EMNLP*, pp. 1923–1933, 2017.

He, K., Wang, Z., Fu, Y., Feng, R., Jiang, Y.-G., and Xue, X. Adaptively weighted multi-task deep network for person attribute classification. 2017.

Hinton, G., Vinyals, O., and Dean, J. Distilling the Knowledge in a Neural Network. *ArXiv e-prints*, 2015.

Huang, J. T., Li, J., Yu, D., Deng, L., and Gong, Y. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Proc. of ICASSP*, pp. 7304–7308, 2013.

Huang, Z., Li, J., Siniscalchi, S. M., Chen, I.-F., Wu, J., and Lee, C.-H. Rapid adaptation for deep neural networks through multi-task learning. In *Proc. of Interspeech*, 2015.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017a.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. In *Proc. of ICLR*, 2017b.

Jou, B. and Chang, S.-F. Deep cross residual learning for multitask visual recognition. In *Proc. of MM*, pp. 998–1007, 2016.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. One model to learn them all. *CoRR*, abs/1706.05137, 2017.

Kang, Z., Grauman, K., and Sha, F. Learning with whom to share in multi-task feature learning. In *Proc. of ICML*, pp. 521–528, 2011.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Kumar, A. and Daumé, III, H. Learning task grouping and overlap in multi-task learning. In *Proc. of ICML*, pp. 1723–1730, 2012.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., and Tu, Z. Deeply-Supervised Nets. In *Proc. of AISTATS*, pp. 562–570, 2015.

Li, Z., Gong, B., and Yang, T. Improved dropout for shallow and deep learning. In *NIPS*, pp. 2523–2531. 2016.

Liu, X., Gao, J., He, X., Deng, L., Duh, K., and Wang, Y. Y. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Proc. of NAACL*, pp. 912–921, 2015a.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proc. of ICCV*, 2015b.

Long, M., Cao, Z., Wang, J., and Yu, P. S. Learning multiple tasks with multilinear relationship networks. In *NIPS*, pp. 1593–1602. 2017.

Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., and Feris, R. S. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *Proc. of CVPR*, 2017.

Luong, M. T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. Multi-task sequence to sequence learning. In *Proc. ICLR*, 2016.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proc. of ACL: HLT*, pp. 142–150, 2011.

Maclaurin, D., Duvenaud, D., and Adams, R. Gradient-based hyperparameter optimization through reversible learning. In *Proc. of ICML*, pp. 2113–2122, 2015.

Mahmud, M. M. and Ray, S. Transfer learning using Kolmogorov complexity: Basic theory and empirical evaluations. In *NIPS*, pp. 985–992. 2008.

Mahmud, M. M. H. On universal transfer learning. *Theoretical Computer Science*, 410(19):1826 – 1846, 2009.

McCann, B., Bradbury, J., Xiong, C., and Socher, R. Learned in translation: Contextualized word vectors. In *NIPS*, pp. 6297–6308. 2017.

Meyerson, E. and Miikkulainen, R. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *Proc. of ICLR*, 2018.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. Cross-stitch networks for multi-task learning. In *Proc. of CVPR*, 2016.

Ranjan, R., Patel, V. M., and Chellappa, R. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *CoRR*, abs/1603.01249, 2016.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *Proc. of ICML*, pp. 2902–2911, 2017.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. Learning multiple visual domains with residual adapters. In *NIPS*, pp. 506–516. 2017.

Rudd, E. M., Günther, M., and Boult, T. E. MOON: A mixed objective optimization network for the recognition of facial attributes. In *Proc. of ECCV*, pp. 19–35, 2016.

Ruder, S. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proc. of ICLR*, 2014.

Seltzer, M. L. and Droppo, J. Multi-task learning in deep neural networks for improved phoneme recognition. In *Proc. of ICASSP*, pp. 6965–6969, 2013.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pp. 2951–2959. 2012.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR*, 15(1):1929–1958, 2014.

Szegedy, C., Ioffe, S., and Vanhoucke, V. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *NIPS*, pp. 4499–4509. 2017.

Toshniwal, S., Tang, H., Lu, L., and Livescu, K. Multitask Learning with Low-Level Auxiliary Tasks for Encoder-Decoder Based Speech Recognition. *CoRR*, abs/1704.01631, 2017.

van der Maaten, L. and Hinton, G. *JMLR*, 9:2579–2605, Nov 2008.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *Proc. of ICML*, pp. 564–572, 2016.

Wu, Z., Valentini-Botinhao, C., Watts, O., and King, S. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *Proc. of ICASSP*, pp. 4460–4464, 2015.

Yang, Y. and Hospedales, T. Deep multi-task representation learning: A tensor factorisation approach. In *Proc. of ICLR*, 2017.

Zhang, Y. and Weiss, D. Stack-propagation: Improved representation learning for syntax. pp. 1557–1566, 2016.

Zhang, Z., Ping, L., Chen, L. C., and Xiaoou, T. Facial landmark detection by deep multi-task learning. In *Proc. of ECCV*, pp. 94–108, 2014.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *Proc. of ICLR*, 2017.