Side-Tuning: Network Adaptation via Additive Side Networks

Jeffrey O. Zhang[1*] Alexander Sax[1*] Amir Zamir[1,2]
Leonidas Guibas[2,3] Jitendra Malik[1,3]
[1]University of California, Berkeley [2]Stanford University [3]Facebook AI Research
http://jozhang97.github.io/sidetuning

Abstract

When training a neural network for a desired task, one may prefer to adapt a pretrained *network rather than a randomly initialized one – due to lacking enough training data, performing lifelong learning where the system has to learn a new task while being previously trained for other tasks, or wishing to encode priors in the network via preset weights. The most commonly employed approaches for network adaptation are* fine-tuning *and using the pre-trained network as a fixed* feature extractor, *among others.*

In this paper we propose a straightforward alternative: Side-Tuning*. Side-tuning adapts a pre-trained network by training a lightweight "side" network that is fused with the (unchanged) pre-trained network using a simple additive process. This simple method works as well as or better than existing solutions while it resolves some of the basic issues with fine-tuning, fixed features, and several other common baselines. In particular, side-tuning is less prone to overfitting when little training data is available, yields better results than using a fixed feature extractor, and doesn't suffer from catastrophic forgetting in lifelong learning. We demonstrate the performance of side-tuning under a diverse set of scenarios, including lifelong learning (iCIFAR, Taskonomy), reinforcement learning, imitation learning (visual navigation in Habitat), NLP question-answering (SQuAD v2), and single-task transfer learning (Taskonomy), with consistently promising results.*

1. Introduction

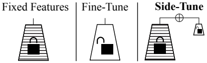

Figure 1. **The *side-tuning* framework vs the common alternatives *fine-tuning* and *fixed features*.** Given a pre-trained network that should be adapted to a new task, *fine-tuning* re-trains the pretrained network's weights and *fixed feature extraction* trains a readout function with no re-training of the pre-trained weights. In contrast, *Side-tuning* adapts the pre-trained network by training a lightweight conditioned "side" network that is fused with the (unchanged) pre-trained network using a simple additive process.

The goal of *side-tuning* is to capitalize on a pretrained model to better learn one or more novel tasks. By design, *side-tuning* does so without degrading performance of the base model. The framework is straightforward: it assumes access to the frozen base model $B : \mathbb{X} \to \mathbb{Y}$ that maps inputs into some representation space that is shared between the base task and the current (target) task. This representation space is flexible and could either be a latent space (e.g. in $\mathbb{R}^N$) or actual model predictions. *Side-tuning* then *learns* a side model $S : \mathbb{X} \to \mathbb{Y}$, so that the representations for the target task are

$$R(x) \triangleq B(x) \oplus S(x),$$

for some combining operation $\oplus$. We use a learned alpha-blending, $a \oplus b \triangleq \alpha a + (1 - \alpha)b$ for this purpose (other options are discussed in Section 3.1).

Certain pre-set curricula of α reduce the side-tuning framework to: fine-tuning, feature extration, and stage-wise training (see Fig. 2, right). Hence those can be viewed as special cases of the general side-tuning framework. Other curricula suggest (e.g.) a *maximum a posteriori* estimator that integrates the $B(x)$ prior with the evidence from $S(x)$.

Side-tuning is an example of an *additive learning* approach as it adds (strategically placed) parameters for each new task. Fixed feature extraction would be a simple example of an additive approach with *zero* new parameters. As a result, fixed features are don't adapt the base network over the lifetime of the agent. A number of existing approaches address this by learning new parameters (the number of which scales with the size of the base network) for each new task (e.g. [31]). Unlike these approaches, *side-tuning* places no constraints on the structure of the side network, allowing the parameters to be strategically allocated. In particular, *side-tuning* can use tiny networks when the base requires minor updates. By adding fewer parameters per task, *side-tuning* can learn more tasks before the model grows large enough to require parameter consolidation.

Method	1 Target Task Low Data	 High Data	> 1 Target Tasks (Lifelong)
Fixed features	✓	✗ (Info Loss)	✗ (Info Loss)
Fine-tuning	✗ (Overfit)	✓	✗ (Forgetting)
Side-tuning	✓	✓	✓

Table 1. **Advantages of *side-tuning* vs. alternatives**. Fixed features has no learnable parameters and thus cannot adapt to new information. On the other hand, fine-tuning has a large number of learnable parameters which leads to overfitting. *Side-tuning* is a simple method that addresses these limitations by strategically placing a small number of learnable parameters.

These approaches stand in contrast to most existing methods for incremental learning, which do not increase the number of parameters over time and instead gradually fill up the capacity of a large base model. For example, fine-tuning updates all the parameters. A large body of constraint-based methods focus on how to regularize these updates in order to prevent inter-task interference [4]. *Side-tuning* does not require such regularization since the *additive* structure means inter-task interference is not possible.

We compare *side-tuning* to alternative approaches on both the iCIFAR and Taskonomy datasets. iCIFAR consists of ten distinct 10-class image classification problems. Taskonomy covers multiple tasks of varied complexity from across computer vision (surface normal and depth estimation, edge detection, image 1000-way classification, etc.). On these datasets, *side-tuning* uses side networks that are much smaller than the base. Consequently, even without consolidation, *side-tuning* uses fewer learnable parameters than the alternative methods.

This simple approach deals with the key challenges of incremental learning. Namely, it does not suffer from either:

- **Catastrophic forgetting**: tendency of a network to abruptly lose previously learned knowledge upon learning new information. Discussed in Sec. 4.2.1.
- **Rigidity**: where networks become increasingly unable to adapt to new problems as they accrue constraints from previous problems. Discussed in Sec. 4.2.2.

Side-tuning avoids these problems while remaining highly performant, which we demonstrate in Section 4.2.3. Moreover, it is significantly simpler than most existing lifelong learning approaches.

2. Related Work

Broadly speaking, network adaptation methods either overwrite existing knowledge (*substitutive* methods) or save it and add new parameters (*additive* learning). In incremental (lifelong) learning, substitutive methods like fine-tuning are at risk of forgetting early tasks. To this end, existing methods constrain the learning procedure, leading to undesirable trade-offs. By design, additive approaches are able to circumvent forgetting, reuse knowledge, and scale to more tasks. *Side-tuning* is one of the simplest consistent additive methods which takes this family of methods and captures a small yet core component that makes them work. We show this experimentally on various tasks and datasets, including iCIFAR [27], Habitat [32], SQuAD v2 [25], and Taskonomy [36]. In the remainder of this section we overview *sidetuning's* connection to related fields.

Incremental learning's objective is to learn a sequence of tasks $\mathcal{T}_1, ..., \mathcal{T}_m$ and perform well on the entire set at the end of training. Two problems arise from this sequential presentation: *catastrophic forgetting* (see Sec. 4.2.1) and *learning speed* (see Sec. 4.2.2). Incremental learning methods fall under two paradigms: substitutive and additive.

Substitutive methods modify an existing network to solve a new task by updating some or all of the network weights (simplest approach being fine-tuning). There are many alternatives that attempt to avoid catastrophic forgetting. [16] adds constraints on how the parameters are updated and [34, 16, 18] add a parameter regularization term per task. These approaches tend to impose constraints which slow down learning on later tasks (see Sec. 4.2.2 on rigidity, [3]). [4] relegates each task to approximately orthogonal subspaces but is unable to transfer information across tasks.

Additive methods circumvent the aforementioned problems by freezing the weights and adding a small number of new parameters per task (simplest approach being fixed features). One economical approach is to use off-the-shelf-features with one or more readout layers [26]. [20, 30] modulate the output by applying learned weight masks. [2, 29] learn across different tasks with additional task-specific parameters. Perhaps the most comparable work to *side-tuning* is Progressive Neural Networks (PNN) [31] which, for each new task, learn a new network utilizing dense lateral connections from neural networks of earlier tasks. *Side-tuning* does not require these lateral connections, making it significantly simpler and applicable on a larger variety of problems. Furthermore, the results suggest that *side-tuning* offers similar or better performance to these methods.

Meta-learning seeks to create agents that rapidly adapt to new problems by first training on tasks sampled from a standing distribution of tasks. *Side-tuning* is fundamentally compatible with this formulation and with existing approaches (e.g. [8]). Moreover, recent work suggests that these approaches work primarily by feature adaptation rather than rapid learning [24], and feature adaptation is also the motivation for our method.

Representation learning in other fields. Developing infants are hypothesized to learn separate, discontinuous, and context-dependent perception systems [1, 17]. Adults are able to rapidly learn new affordances, but only when

---

[*]Equal Contribution

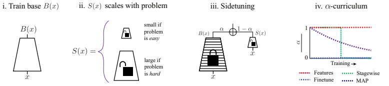

Figure 2. **Mechanics of *side-tuning*.** (i) *Side-tuning* takes some core network (B) and adapts it to a new task by (ii) adapting a side network. (iii) Shows the connectivity structure when using *side-tuning* along with alpha-blending. (iv) Existing adaptation methods turn out to be special cases of *side-tuning*. In particular: fine-tuning, feature extraction, and other approaches are *side-tuning* with a fixed curriculum on the blending parameter α.

those are minor updates to familiar, well-practiced systems [5]. On a more fine-grained scale, there are areas of functional specificity within the brain [14], including wholly separate pathways where output is mutually conditioned on one another [33].

Sensor Fusion . Many problems are easier to solve when using a suite of sensors providing complementary sources of information [7]. We combine both a base and a residual side network via a simple additive mechanism. This has been successfully used in computer vision (ResNets [10]) and in robotics, where residual RL [12, 35] learns a single task by combining a coarse policy (e.g. hand-coded optimal control) and with a learned residual network. Alternatively, multiplication can be though of as a measure of agreement among sensors, and multiplication for fusion has been explored in [4, 20]. FiLM [22] defines a combination operation that combines both multiplicative and additive transformations.

3. The Side-tuning Framework

Side-tuning learns a side model $S(X)$ and combines this with a base model $B(x)$ so that the representations for the target task are computed as $R(x) \triangleq B(x) \oplus S(x)$.

3.1. Architectural Elements

Base Model. The base model $B(x)$ provides some core cognition or perception, and we put no restrictions on how $B(x)$ is computed. We never update $B(x)$, and in our approach it has zero learnable parameters. In general $B(x)$ could be nonparametric, and it might not be optimized for any particular task. We consider several choices for $B(x)$ in Section 4.4, but the simplest choice is just a pretrained network.

Side Model. Unlike the base model, the side network $S(x)$ is updated during training; learning a residual that we apply on top of the base encoding. Iteratively learning residuals for a *single* task is known as *gradient boosting* (see Section 4.4 for a comparison). *Side-tuning* is instead focused on learning *multiple* tasks.

One crucial component of the framework is that *the complexity of the side network can scale to the difficulty of the problem at hand*. When the base is *relevant* and requires only a minor update, a very small network can suffice. Section 4.4 explores the effect of network size, how that changes with the choice of base and target tasks.

While the side network can be initialized using a variety of methods, we initialize the side network with a copy of the base network. When the forms of the base and side networks differ, we initialize the side network with weights distilled from the base network using knowledge distillation [11]. We test alternatives in Section 4.4.

Combining Base and Side Representations. The final *side-tuning* representation is a combination, $B(x) \oplus S(x)$. What should $\oplus$ be?

Side-tuning admits several options for this combination operator. We compare several operators in 4.4. Two simple examples are concatenation and summation. We observe that alpha blending, $a \oplus b \triangleq \alpha a + (1-\alpha)b$, works well in practice. Alpha blending preserves the dimensions of the inputs and is simpler than concatenation. In fact, concatenation followed by a channel-collapsing operation (e.g. 1x1 convolution) is a strict generalization of alpha-blending.

While simple, alpha blending is expressive enough that it encompasses several common transfer learning approaches. As shown in Figure 2 and when the side network is the same as the base, *side-tuning* is equivalent to feature extraction when $\alpha = 1$. When $\alpha = 0$, *side-tuning* is instead equivalent to fine-tuning. If we allow α to vary during training (which we generally do), then switching α from 1 to 0 is equivalent to the common (stage-wise) training curriculum in RL where a policy is trained on top of some fixed features that are unlocked partway through training.

Another notable curriculum is $\alpha(N) = \frac{k}{k+N}$ for $k > 0$ (hyperbolic decay). In this curriculum, α controls the weighting of the prior ($B(x)$) with the learned estimate ($S(x)$), and the weight of the evidence scales with the amount of data. This curriculum is suggestive of a *maximum a posteriori* estimate and, like the MAP estimate, it converges to the MLE (fine-tuning, $\alpha = 0$).

Finally, α can treated as a learnable parameter that determines how heavily to weight the base model. In practice, the value of α correlates with task relevance (see Sec. 4.4).

3.2. Asymptotic Consistency and Bias/Variance

When minimizing estimation error there is often a tradeoff between the bias and variance contributions [9]. Choosing between feature extraction or fine-tuning exemplifies this dilemma. Feature extraction ($\alpha = 0$) locks the weights and corresponds to a point-mass prior that, unless the weights are already optimal, yields a biased estimator. In fact, the estimator allows no adaptation to new evidence and is *asymptotically inconsistent*. On the other hand, fine-tuning ($\alpha = 1$), is an uninformative prior yielding a low-bias high-variance estimator. With enough data, fine-tuning can produce better estimates, but this usually takes more data than feature extraction. *Side-tuning* addresses both the these problems. It reduces variance by including the fixed features in the representation, and it is *consistent* because it allows updating via the residual side network.

3.3. Perceptual Regularization and Catastrophic Forgetting

While α provides a way to control the importance of the prior, another natural approach for enforcing a prior is to penalize deviations from the original feature representation. Typically, it is easier to specify meaningful explicit priors on outputs (e.g. L2 for pixels) than on the latent representations, which can be difficult if not impossible to interpret. As long as the decoder $D : \mathbb{Y} \rightarrow \mathbb{A}$ is differentiable, any distance measure on the outputs can be *pulled back* through the decoder and into the latent space. This induced distance d_D on the latent representations is called the *pullback metric* in differential geometry, and in deep learning it is called the *perceptual loss* [13]. This may be a useful method for knowledge transfer when (i) the previous task is *relevant* to the new task and (ii) there is limited training data. A recent successful application of this approach would be the auxiliary losses in *GPT* [23], though we did not find it effective.

Perceptual regularization is often used to dampen *catastrophic forgetting*. For example, Elastic Weight Consolidation uses a diagonalized second-order Taylor expansion of the expectation of the pullback metric. *Learning Without Forgetting* uses a decoder-based approach that can be interpreted as jointly updating both the base network and the pullback metric. We show that such regularization does not fully address the problem of catastrophic forgetting (Section 4.2.1). *Side-tuning* avoids catastrophic forgetting by design (as the base network is never updated).

3.4. Sidetuning for Continual Learning

Network adaptability is the sole criterion only if we care we solely about raw performance on a *single* target task. In reality we often care about the performance on both the current and previous tasks. This is the case for *incremental learning*, where we want an agent that can learn a sequence of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_m$ and, at the end, is capable of reasonable performance across the entire set. Thus, catastrophic forgetting becomes a major issue.

In our experiments we dedicate one new side network to each new task and train it independently of the earlier side networks. In principle, learning of new tasks can benefit from all the side networks learned in previous tasks (i.e. the nth task can use all $n-1$ previous tasks). Since we do not make use of this available information, our results should be considered as a lower bound on *side-tuning* performance.

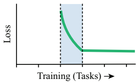

Figure 4. **Theoretical learning curve of *side-tuning*.** The model learns during task-specific training and those weights are immediately frozen, preserving performance.

We show that this simple approach provides a strong baseline for incremental learning, outperforming existing approaches in the literature while using fewer parameters on more tasks (in Section 4.2).

Side-tuning takes an additive learning approach to incremental learning, which means that already-learned components are never updated and performance across the whole set can only increase as the agent sees more tasks. This monotonicity is the key property of the *additive* family of algorithms. It is worth repeating that there is *No Catastrophic Forgetting in Additive Learning* and a typical learning curve for one of the tasks is shown in Figure 4. Furthermore, because our implementation of *side-tuning* treats problems independently of their order in the sequence (always using the fixed base and one side network), *side-tuning* incurs no rigidity during training. We show this in Section 4.2.2.

Side-tuning naturally handles other continuous learning scenarios besides incremental learning. A related problem is that of continuous adaptation, where the agent needs to perform well (e.g. minimizing regret) on a stream of tasks with undefined boundaries and where there might very little data per task and no task repeats. As we show in Section 4.2, inflexibility becomes a serious problem for constraint-based methods and task-specific performance declines after learning more than a handful of tasks. Moreover, continuous adaptation requires an online method as task boundaries must be detected and data cannot be replayed (e.g. to generate constraints for EWC).

Side-tuning could be applied to continuous adaptation by

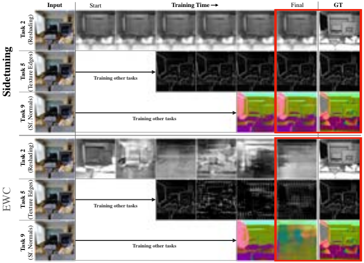

Figure 3. ***Side-tuning* does not forget in incremental learning.** Qualitative results for incremental learning on Taskonomy with additive learning (*side-tuning*, top 3 rows) and constraint-based learning (EWC, bottom 3 rows). Each row contains results for one task and columns show how predictions change over the course of training. Predictions from EWC quickly degrade over time, showing that EWC still catastrophically forgets. Predictions from *side-tuning* do not degrade, and the initial quality is better in later tasks (e.g. compare the table in surface normals). We provide additional comparisons (including for PSP) in the supplementary.

keeping a small working memory of cheap side networks that constantly adapt the base network to the input task. These side networks are small, easy to train, and when one of the networks begins performing poorly (e.g. signaling a distribution shift) that network can simply be discarded. This is an online approach, and online adaptation with small cheap networks has found recent success in (e.g. [21]).

4. Experiments

In the first section we show that *side-tuning* compares favorably to existing incremental learning approaches on both iCIFAR and the more challenging Taskonomy dataset. We then extend to multiple domains (computer vision, RL, imitation learning, NLP) in the simplified (transfer learning) scenario for $N = 2$ tasks. Finally, we interpret *side-tuning* in a series of analysis experiments.

4.1. Baselines

We provide comparisons of *side-tuning* against the following methods:

Scratch: The network is initialized with appropriate random weights and trained using minibatch SGD with Adam [15].

Feature extraction (features): The pretrained base network is used as-is and is not updated during training.

Fine-tuning: An umbrella term that encompasses a variety of techniques, we consider a more narrow definition where pretrained weights are used as initialization and then training proceeds as in *scratch*.

Elastic Weight Consolidation (EWC). A constraint-based substitutive approach from [16]. We use the formulation from [34] which scales better, giving an advantage to EWC since otherwise we could use a larger *side-tune* network and maintain parameter parity.

Parameter Superposition (PSP): A parameter-masking substitutive approach from [4] which attempts to make tasks independent from one another by mapping the weights to approximately orthogonal spaces. This approach is unable to transfer across tasks.

Progressive Neural Network (PNN): An additive approach from [31] which utilizes many lateral connections between the base and side networks. This requires the base model to be a neural network and the architecture of the base and side networks to be the same or similar.

Independent: Each task uses a network trained independently for the target task. This method uses far more learnable parameters than all the alternatives (e.g. saving a separate ResNet-50 for each task) and achieves very strong performance. Due to the scaling, it is generally not considered a viable incremental learning method.

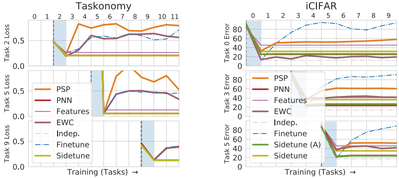

Figure 5. **Incremental Learning on Taskonomy and iCIFAR**. The above curves show loss and error on incremental learning experiments for three tasks on Taskonomy (left) and iCIFAR dataset (right). The fact that *side-tuning* losses are flat after training (as we go right) shows that it does not forget previously learned tasks. Similarly, the performance remains consistent even on later tasks (as we go down), showing that *side-tuning* does not become rigid. Alternative methods clearly forget (e.g. PSP) and/or become rigid (e.g. EWC). In Taskonomy, PNN and Independent are hidden under Sidetune. In iCIFAR, Sidetune (A) merges base and side information with a multilayer perceptron (adapter).

4.2. Incremental Learning: No Catastrophic Forgetting in Additive Learning

On both the Taskonomy dataset [36] and incremental CIFAR (iCIFAR, [28]), *side-tuning* outperforms existing incremental learning approaches while using fewer parameters[1]. Moreover, the performance gap is larger on more challenging datasets.

Taskonomy includes labels for multiple computer vision tasks including 2D (e.g. edge detection), 3D (e.g. surface normal estimation), and semantic (e.g. object classification) tasks. We first selected the twelve tasks that make predictions from a single RGB image, and then created an incremental learning setup by selecting a random order in which to learn these tasks (starting with *curvature*). As images are 256x256 we use a ResNet-50 for the base network and a 5-layer convolutional network for the *side-tuning* side network. The number of learnable network parameters used across all tasks is 24.6M for EWC and PSP, and 11.0M for *side-tuning*[2].

iCIFAR. First, we pretrain the base network (ResNet-44) on CIFAR-10. Then the 10 subsequent tasks are formed by partitioning CIFAR-100 classes into 10 disjoint sets of 10-classes each. We train on each subtask for 20k steps before moving to the next one. Our state-of-the-art substitutive baselines (EWC and PSP) update the base network for each task (683K parameters), while *side-tuning* updates a four layer convolutional network per task (259K parameters after 10 tasks).

4.2.1 Catastrophic Forgetting

As expected, there is no catastrophic forgetting in *side-tuning*. Figure 5 shows that the error for *side-tuning* does not increase after training (blue shaded region), while it increases sharply for the other methods on both Taskonomy and iCIFAR.

The difference is meaningful, and Figure 3 shows sample predictions from *side-tuning* vs. EWC for a few tasks during and after training. As is evident from the bottom rows, EWC exhibits catastrophic forgetting on all tasks (worse image quality as we move right). In contrast, *side-tuning*

---

[1] Full experimental details (e.g. learning rate and architecture) provided in the supplementary.

[2] All numbers not counting readout parameters, which are common between all methods.

(top) shows no forgetting and the final predictions are significantly closer to the ground-truth (boxed red).

4.2.2 Rigidity

Side-tuning learns later tasks as easily as the first, while constraint-based methods such as EWC stagnate. The predictions for later tasks such as *surface normals* (in Figure 3) are significantly better using *side-tuning*—**even immediately after training and before any forgetting can occur**.

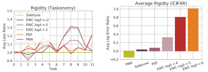

Figure 6. **Rigidity on Taskonomy and iCIFAR**. *Side-tuning* always learns new tasks easily; EWC becomes increasingly unable to learn new tasks as training progresses. The same trend holds on iCIFAR (right), and the average rigidity is zero for *side-tuning* (and almost zero for PSP).

Figure 6 quantifies this slowdown. We measure rigidity as the log-ratio of the *actual loss* or the ith task over the loss when that task is *instead trained first* in the sequence. As expected, *side-tuning* experiences effectively zero slowdown on both datasets. For EWC, the increasing constraints make learning new tasks increasingly difficult—and the log-ratio increases with the number of tasks (Taskonomy, left). It is too rigid (log-ratio > 0) even in iCIFAR, where the later tasks are similar to earlier ones.

4.2.3 Final Performance

Overall, *side-tuning* significantly outperforms the other methods while using fewer than half the number of trainable parameters of the other methods. When the other methods use smaller networks, their performance decreases further. On both iCIFAR and Taskonomy, *side-tuning* achieves the best average rank (1.13 of 4 on Taskonomy, while the next best is 1.88 (PNN), see Fig. 2).

This is a direct result of the fact (shown above) that *side-tuning* does not suffer from catastrophic forgetting or rigidity. It is not due to the fact that the sidetuning structure is specially designed for these types of image tasks; it is not (we show in Sec. 4.3 that it performs well on other domains). In fact, the much larger networks used in EWC and PSP *should* achieve better performance on any single task. For example, EWC produces sharper images early on in training, before it has had a chance to accumulate too many constraints (e.g. *reshading* in Fig. 3). But this factor was outweighed by *side-tuning's* immunity from the effects of catastrophic forgetting and creeping rigidity.

Method	Avg. Rank (↓) Taskonomy	 iCIFAR
Fine-tune	4.13	4.60
EWC ($\lambda = 10^0, 10^5$)	4.00	2.80
PSP	3.88	4.10
PNN	1.88	**1.20**
Side-tune	**1.13**	2.30

Table 2. **Average rank on Taskonomy and iCIFAR.** While being remarkably simpler, *side-tuning* generally achieved a better average rank than other approaches. The difference increases on the more challenging Taskonomy dataset, where *side-tuning* significantly outperformed all tested alternatives.

4.3. Universality of Side-Tuning

In order to address the possibility that *side-tuning* is somehow domain- or task-specific, we provide results showing that it is well-behaved in other settings. As the concern with *additive learning* is mainly that it is too inflexible to learn new tasks, we compare with fine-tuning (which outperforms other lifelong learning tasks when forgetting is not an issue). For extremely limited amounts of data, feature extraction can outperform fine-tuning. We show that *side-tuning* generally performs as well as features or fine-tuning–whichever is better.

Transfer learning in Taskonomy. We trained networks to perform one of three target tasks (object classification, surface normal estimation, and curvature estimation) on the Taskonomy dataset [36] and varied the size of the training set $N \in \{100, 4 \times 10^6\}$. In each scenario, the base network was trained (from scratch) to predict one of the non-target tasks. The side network was a copy of the original base network. We experimented with a version of fine-tuning that updated both the base and side networks; the results were similar to standard fine-tuning [3]. In all scenarios, *side-tuning* successfully matched the adaptiveness of fine-tuning, and significantly outperformed learning from scratch, as shown in Figure 3a. The additional structure of the frozen base did not constrain performance with large amounts of data (4M images), and *side-tuning* performed as well as (and sometimes slightly better than) fine-tuning.

Question-Answering in SQuAD v2. We also evaluated *side-tuning* on a question-answering task (SQuAD v2 [25]) using a non-convolutional architecture. We use a pretrained BERT [6] model for our base, and a second for the side networkdefer. Unlike in the previous experiments, BERT uses attention and no convolutions. Still, *side-tuning* adapts to the new task just as well as fine-tuning, outperforming features and scratch (Figure 3b).

Imitation Learning for Navigation in Habitat. We trained an agent to navigate to a target coordinate in the Habitat environment. The agent is provided with both RGB input image and also an occupancy map of previous locations. The map does not contain any information about the environment—just previous locations. In this section we use Behavior Cloning to train an agent to imitate experts following the shortest path on 49k trajectories in 72 buildings. The agents are evaluated in 14 held-out validation buildings. Depending on the what the base network was trained on, the source task might be useful (*Curvature*) or harmful (*Denoising*) for imitating the expert and this determines whether features or learning from scratch performs best. Figure 3c shows that regardless of the which approach worked best, *side-tuning* consistently matched or beat it.

Method	**Transfer Learning in Taskonomy** From Curvature (100/4M ims.) Normals (MSE ↓)	 Obj. Cls. (Acc. ↑)	**QA on SQuAD** Match (↑) Exact	 F1	**Navigation (IL)** Nav. Rew. (↑) Curvature	 Denoise	**Navigation (RL)** Nav. Rew. (↑) Curvature	 Denoise
Fine-tune	**0.200 / 0.094**	**24.6 / 62.8**	**79.0**	**82.2**	10.5	9.2	10.7	**10.0**
Features	**0.204** / 0.117	24.4 / 45.4	49.4	49.5	**11.2**	8.2	**11.9**	8.3
Scratch	0.323 / **0.095**	19.1 / 62.3	0.98	4.65	9.4	**9.4**	7.5	7.5
Side-tune	**0.199 / 0.095**	**24.8 / 63.3**	**79.6**	**82.7**	**11.1**	**9.5**	**11.8**	**10.4**
	(a)		**(b)**		**(c)**		**(d)**	

Table 3. **Side-tuning comparisons in other domains.** Sidetuning matched the adaptability of fine-tuning on large datasets, while performing as well or better than the best competing method in each domain: **(a)** In Taskonomy, performing either *Normal Estimation* or *Object Classification* using a base trained for *Curvatures* and either 100 or 4M images for transfer. Results using *Obj. Cls.* base are similar and provided in the appendix. **(b)** In SQuAD v2 question-answering, using BERT instead of a convolutional architecture. **(c)** In Habitat, learning to navigate by imitating expert navigation policies, using inputs based on either *Curvature* or *Denoising*. Finetuning does not perform as well in this domain. **(d)** Using RL (PPO) and direct interaction instead of supervised learning.

Reinforcement Learning for Navigation in Habitat. Using a different learning algorithm (PPO) and using direct interaction instead of expert trajectories, we observe identical trends. We trained agents directly in Habitat (74 buildings). Fig. 3d shows performance in 14 held-out buildings after 10M frames of training. *Side-tuning* performs comparably to the max of competing approaches.

4.4. Learning Mechanics in Side-Tuning

Task relevance predicts alpha α. In our experiments, we treat α as a learnable parameter and find that the relative values of α are predictive of emprical performance. In imitation learning (Fig. 4.3), *curvature* ($\alpha = 0.557$) outperformed *denoising* ($\alpha = 0.252$). In Taskonomy, the α values from training on just 100 images predicted the actual transfer performance to normals in [36], (e.g. *curvature* ($\alpha = 0.56$) outperformed *object classification* ($\alpha = 0.50$)). For small datasets, usually $\alpha \approx 0.5$ and the relative order, rather than the actual value is important.

Benefits for intermediate amounts of data. We showed in the previous section that *side-tuning* performs like the best of {features, fine-tuning, scratch} in domains with abundant or scant data.

In order to test whether *side-tuning* could profitably synthesize the features with *intermediate* amounts of data, we evaluated each approach's ability to learn to navigate using 49, 490, 4900, or 49k expert trajectories and pretrained *denoising* features. *Side-tuning* was always the best-performing approach and, on intermediate amounts of data (e.g. 4.9k trajectories), outperformed the other techniques (*side-tune* 9.3 vs. fine-tune: 7.5, features: 6.7, scratch: 6.6), Figure 7).

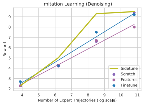

Figure 7. ***Side-tuning* outperformed alternatives on intermediate amounts of data.**

Network size. Does network size matter? We find (i) If the target problem benefits from a large network (e.g. classification tasks), then performance is sensitive to side network size but not size of the base. (ii) The base network can usually be distilled to a smaller network and *sidetuning* will still offer advantages over alternatives. In the supplementary material we provide supporting experiments from Taskonomy using both high- and low-data settings (curvature → {obj. class, normals}, obj. class → normals), and in Habitat (RL using {curvature, denoise} → navigation).

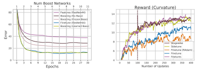

Figure 8. **Boosting and Gradient Variance**. (Left) Deeper networks outperform many shallow learners. (Right) Features and *Side-tuning* do more than reduce gradient variance.

Not Boosting. Since the side network learns a resid-

---

[3] We defer remaining experimental details (learning rate, full architecture, etc.) to the appendix. See provided code for full details.

ual on top of the base network, we ask: what benefits we could glean by extending *side-tuning* to do boosting? Although network boosting this does improve performance on iCIFAR (Figure 8 Left), if catastrophic forgetting is not a concern then the parameters would've been better used in a deeper network rather than many shallow networks.

Initialization. A good side network initialization can yield a minor boost in performance. We found that initializing from the base network slightly outperforms a low-energy initialization[4], which slightly outperforms Xavier initialization. However, we found that these differences were not statistically significant across tasks (H_0 : *pretrained* = *xavier*; $p = 0.07$, Wilcoxon signed-rank test). We suspect that initialization might be more important on harder problems. We test this by repeating the analysis without the simple texture-based tasks (2D keypoint + edge detection and autoencoding) and find the difference in initialization is now significant ($p = 0.01$).

More than just stable updates. In RL, fine-tuning often fails to improve performance. One common rationalization is that the early updates in RL are 'high variance'. The usual solution is to first train using fixed features and then unfreeze the weights at some point in training (via a hyperparameter to be set). We found that this stage-wise approach performs as well (but no better than) keeping the features fixed–and *side-tuning* performed as well as both while being simpler than stage-wise (Fig. 8 Right). We tested the 'high-variance update' theory by fine-tuning with both gradient clipping and an optimizer designed to prevent such high-variance updates by adaptively warming up the learning rate (RAdam, [19]). This provided no benefits over vanilla fine-tuning, suggesting that the benefits of side-tuning are not solely due to gradient stabilization early in training.

Merge methods. Section 2 and 3.1 described different ways to merge the base and side networks. Figure 4 evaluates a few of these approaches on the Taskonomy dataset. Element-wise product and addition via alpha-blending are two of the simplest approaches and have little overhead in terms of compute and parameter count. [31] uses $a \oplus b = F_\theta(a) + b$ where F_θ is an a Multi-Layer Perceptron (MLP). FiLM [22] adds additional compute by defining $a \oplus b = \gamma_\theta(a) \odot b + \beta_\theta(a)$ where γ_θ and β_θ is an MLP with two heads. Figure 4 shows that all three approaches are roughly comparable, though the MLP-based methods achieve marginally better average rank on the Taskonomy dataset. Nonetheless, we chose to use the simplest feature-wise transformation (alpha-blending) since it adds no parameters and achieves similar performance.

Ablating Base and Side Elements. *Side-tuning* uses two streams of information - one from the base model and one from the side model. Are both streams necessary? Figure 5 shows that on the Taskonomy experiment performance improves when using both models.

Method	Avg. Rank (↓) Taskonomy
Product (Element-wise)	3.64
Addition (Alpha Blending)	2.27
MLP (as in [31])	2.18
FiLM [22]	1.91

Table 4. **Average rank of various merge methods.** There are multiple types of feature-wise transformation to choose from. We decided to pick a simple yet highly competitive one.

Method	Avg. Rank (↓) Taskonomy
Base-Only	2.55
Side-Only	2.10
Side-tuning	1.36

Table 5. **Average rank on Taskonomy when ablating base and side models.** Performance improved when using *both the base and side model in* side-tuning.

5. Discussion

We have introduced the *side-tuning* framework, a simple yet effective approach for *additive learning*. Since it does not suffer from catastrophic forgetting or rigidity, it is naturally suited to incremental learning. The theoretical advantages are reflected in empirical results, and *side-tuning* outperforms existing approaches in challenging contexts and with state-of-the-art neural networks. We further demonstrated that the approach is effective in multiple domains and with various network types. Overall, we found *side-tuning* to perform on-par-with or better-than many current lifelong-learning approaches, while being significantly simpler.

6. Limitations

The naïve approach to incremental learning used in this paper made a number of design decisions. These decisions could be analyzed and subsequently relaxed. In particular:

Flexible parameterizations for side networks: Our incremental learning experiments used the same side network architecture for all subtasks. A method for automatically adapting the networks to the subtask at hand could make more efficient use of the computation and supervision. *Better forward transfer*: Our experiments used only a single base and single side network. Leveraging the already previously trained side networks could yield better performance on later tasks. *Learning when to deploy side networks*: Like most incremental learning setups, we assume that the tasks are presented in a sequence and that task identities are known. Using several active side networks in tandem would provide a natural way to detecting distribution shift. *Using side-tuning to measure task relevance*: We noted that α tracked task relevance, but a more rigorous treatment of the interaction between the base network, side network, α and final performance could yield insight into how tasks relate to one another.

Acknowledgements: This material is based upon work supported by ONR MURI (N00014-14-1-0671), Google Cloud, NSF (IIS-1763268), NVIDIA NGC beta, and TRI. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

References

[1] Karen E Adolph, Sarah E Berger, and Andrew J Leo. Developmental continuity? crawling, cruising, and walking. *Developmental science*, 14(2):306–318, 2011. 2

[2] Hakan Bilen and Andrea Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017. 2

[3] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. *CoRR*, abs/1812.00420, 2018. 2

[4] Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno A. Olshausen. Superposition of many models into one. *CoRR*, abs/1902.05522, 2019. 2, 3, 5

[5] Whitney G Cole, Gladys LY Chan, Beatrix Vereijken, and Karen E Adolph. Perceiving affordances for different motor skills. *Experimental brain research*, 225(3):309–319, 2013. 3

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. 7

[7] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations. 3

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. 2

[9] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992. 4

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3

[12] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. *CoRR*, abs/1812.03201, 2018. 3

[13] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. 4

[14] Nancy Kanwisher. Functional specificity in the human brain: a window into the functional architecture of the mind. *Proceedings of the National Academy of Sciences*, 107(25):11163–11170, 2010. 3

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015. 5

[16] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. 2, 5

[17] Kari S Kretch and Karen E Adolph. Cliff or step? posture-specific learning at the edge of a drop-off. *Child development*, 84(1):226–240, 2013. 2

[18] Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016. 2

[19] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019. 9

[20] Arun Mallya and Svetlana Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018. 2, 3

[21] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. *CoRR*, abs/1812.02699, 2018. 5

[22] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3, 9

[23] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018. 4

[24] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. *arXiv e-prints*, page arXiv:1909.09157, Sep 2019. 2

[25] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. 2, 7

[26] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014. 2

[27] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016. 2

---

[4] Where the side network is trained so that it does not impact the output. Full details in the supplementary.