Master's Thesis

Gradient-based Meta-learning with Learned Layerwise Metric and Subspace

Yoonho Lee (이 윤 호)

Department of Computer Science and Engineering

Pohang University of Science and Technology

2018

학습된 거리함수와 부분공간을 이용한 경사도 기반 메타러닝

Gradient-based Meta-learning with Learned Layerwise Metric and Subspace

Gradient-based Meta-learning with Learned Layerwise Metric and Subspace

by

Yoonho Lee
Department of Computer Science and Engineering
Pohang University of Science and Technology

A thesis submitted to the faculty of the Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Computer Science and Engineering

Pohang, Korea
06. 12. 2018
Approved by
Seungjin Choi
Academic advisor

Gradient-based Meta-learning with Learned Layerwise Metric and Subspace

Yoonho Lee

The undersigned have examined this thesis and hereby certify that it is worthy of acceptance for a master's degree from POSTECH

06. 12. 2018

Committee Chair	Seungjin Choi
Member	Minsu Cho
Member	Suha Kwak

MCSE 이 윤 호. Yoonho Lee
20162782 Gradient-based Meta-learning with Learned Layerwise Metric and Subspace,
학습된 거리함수와 부분공간을 이용한 경사도 기반 메타러닝
Department of Computer Science and Engineering , 2018, 27p, Advisor : Seungjin Choi. Text in English.

ABSTRACT

Deep learning has been tremendously successful in many difficult tasks including image classification and game-playing. However, deep networks require copious amounts of data in order to achieve such performance. Meta-learning methods have recently gained attention in this context as a way to remedy this dependence on large datasets.

Gradient-based meta-learning methods leverage gradient descent to learn the commonalities among various tasks. While previous such methods have been successful in meta-learning tasks, they resort to simple gradient descent during meta-testing. Our primary contribution is the *MT-net*, which enables the meta-learner to learn on each layer's activation space a subspace that the task-specific learner performs gradient descent on. Additionally, a task-specific learner of an *MT-net* performs gradient descent with respect to a meta-learned distance metric, which warps the activation space to be more sensitive to task identity. We demonstrate that the dimension of this learned subspace reflects the complexity of the task-specific learner's adaptation task, and also that our model is less sensitive to the choice of initial learning rates than previous gradient-based meta-learning methods. Our method achieves state-of-the-art or comparable performance on few-shot classification and regression tasks.

Contents

I. Introduction

While recent deep learning methods achieve superhuman performance on various tasks including image classification [1] or playing games [2], they can only do so using copious amounts of data and computational resources. In many problems of interest, learners may not have such luxuries. *Meta-learning* [3, 4, 5] methods are a potential solution to this problem; these methods leverage information gathered from prior learning experience to learn more effectively in novel tasks. This line of research typically casts learning as a two-level process, each with a different scope. The *meta-learner* operates on the level of tasks, gathering information from several instances of task-specific learners. A *task-specific learner*, on the other hand, operates on the level of datapoints, and incorporates the meta-learner's knowledge in its learning process.

Model-agnostic meta-learning (MAML) [6] is a meta-learning method that directly optimizes the gradient descent procedure of task-specific learners. All task-specific learners of MAML share initial parameters, and a meta-learner optimizes these initial parameters such that gradient descent starting from such initial parameters quickly yields good performance. An implicit assumption in having the meta-learner operate in the same space as task-specific learners is that the two different scopes of learning require equal degrees of freedom.

Our primary contribution is the MT-net (Figure 1), a neural network architecture and task-specific learning procedure. An MT-net differs from previous gradient-based meta-learning methods in that the meta-learner determines a subspace and a corresponding metric that task-specific learners can learn in, thus setting the degrees of freedom of task-specific learners to an appropriate amount. Note that the activation space of the cell shown in Fig.1(b) is 3-dimensional. Because the task-specific learners can only change weights that affect two of the three intermediate activations, task-specific learning only happens on a subspace with 2 degrees of freedom. Additionally, meta-learned parameters $\mathbf{T}$ alter the geometry of the activation space (Fig.1(c)) of task-specific parameters so that task-specific learners are more sensitive to change in task.

II. Background

2.1 Problem Setup

We briefly explain the meta-learning problem setup which we apply to few-shot tasks.

The problems of k-shot regression and classification are as follows. In the training phase for a meta-learner, we are given a (possibly infinite) set of tasks $\{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots\}$. Each task provides a training set and a test set $\{\mathcal{D}_{\mathcal{T}_i,train}, \mathcal{D}_{\mathcal{T}_i,test}\}$. We assume here that the training set $\mathcal{D}_{\mathcal{T}_i,train}$ has k examples per class, hence the name k-shot learning. A particular task $\mathcal{T} \in \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots\}$ is assumed to be drawn from the distribution of tasks $p(\mathcal{T})$. Given a task $\mathcal{T} \sim p(\mathcal{T})$, the task-specific model $f_{\theta_{\mathcal{T}}}$ (our work considers a feedforward neural network) is trained using the dataset $\mathcal{D}_{\mathcal{T},train}$ and its corresponding loss $\mathcal{L}_{\mathcal{T}}(\theta_{\mathcal{T}}, \mathcal{D}_{\mathcal{T},train})$. Denote by $\widetilde{\theta}_{\mathcal{T}}$ parameters obtained by optimizing $\mathcal{L}_{\mathcal{T}}(\theta_{\mathcal{T}}, \mathcal{D}_{\mathcal{T},train})$. Then, the meta-learner f_θ is updated using the feedback from the collection of losses $\left\{\mathcal{L}_{\mathcal{T}}(\widetilde{\theta}_{\mathcal{T}}, \mathcal{D}_{\mathcal{T},test})\right\}_{\mathcal{T}\sim p(\mathcal{T})}$, where the loss of each task is evaluated using the test data $\mathcal{D}_{\mathcal{T},test}$. Given a new task $\mathcal{T}_{new}$ (not considered during meta-training), the meta-learner helps the model $f_{\theta_{\mathcal{T}_{new}}}$ to quickly adapt to the new task $\mathcal{T}_{new}$, by warm-starting the gradient updates.

2.2 Model-Agnostic Meta-Learning

We briefly review model-agnostic meta-learning (MAML) [6], emphasizing commonalities and differences between MAML and our method. MAML is a meta-learning method that can be used on any model that learns using gradient descent. This method is loosely inspired by fine-tuning, and it learns initial parameters of a network such that the network's loss after a few (usually $1 \sim 5$) gradient steps is minimized.

Consider a model with parameters θ. MAML alternates between the two updates (2.1) and (2.2) to determine initial parameters θ for task-specific learners to warm-start the gradient descent updates, such that new tasks can be solved using a small number of examples. Each task-specific learner updates its parameters by gradient descent (2.1) using the loss evaluated with the training data $\{\mathcal{D}_{\mathcal{T},train}\}$. The meta-optimization across tasks (2.2) is performed such that the parameters θ are updated using the loss evaluated with $\{\mathcal{D}_{\mathcal{T},test}\}$. Note that during meta-optimization (2.2), the gradient is computed with respect to initial parameters θ but the test loss is computed with respect to task-specific parameters $\widetilde{\theta}_{\mathcal{T}}$.

$$\begin{aligned} \widetilde{\theta}_{\mathcal{T}} &\leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}}\left(\theta, \mathcal{D}_{\mathcal{T},train}\right) && (2.1) \\ \theta &\leftarrow \theta - \beta \nabla_\theta \left(\sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}}\left(\widetilde{\theta}_{\mathcal{T}}, \mathcal{D}_{\mathcal{T},test}\right) \right), && (2.2) \end{aligned}$$

where $\alpha > 0$ and $\beta > 0$ are learning rates and the summation in (2.2) is computed using minibatches of tasks sampled from $p(\mathcal{T})$.

Intuitively, a well-learned initial parameter θ is close to some local optimum for every task $\mathcal{T} \sim p(\mathcal{T})$. Furthermore, the update (2.1) is sensitive to task identity in the sense that $\widetilde{\theta}_{\mathcal{T}_1}$ and $\widetilde{\theta}_{\mathcal{T}_2}$ have different behaviors for different tasks $\mathcal{T}_1, \mathcal{T}_2 \sim p(\mathcal{T})$.

Recent work has shown that gradient-based optimization is a universal learning algorithm [7], meaning that any learning algorithm can be approximated up to arbitrary accuracy using some parameterized model and gradient descent. Thus, no expressiveness is lost by only considering gradient-based learners as in (2.1). Note that since MAML operates using a single fixed model, one may have to go through trial and error to find such a good model.

Our method is similar to MAML in that our method also differentiates through gradient update steps to optimize performance after fine-tuning. However, while MAML assumes a fixed model, our method actually chooses a subset of its weights to fine-tune. In other words, it (meta-)learns which model is most suitable for the task at hand. Furthermore, whereas MAML learns with standard gradient descent, a subset of our method's parameters effectively 'warp' the parameter space of the parameters to be learned during meta-testing to enable faster learning.

III. Meta-Learning Models

We present our two models in this chapter: Transformation Networks (T-net) and Mask Transformation Networks (MT-net), both of which are trained with gradient-based meta-learning. A T-net learns a metric in its activation space; this metric informs each task-specific learner's update direction and step size. An MT-net additionally learns which subset of its weights to update for task-specific learning. Therefore, an MT-net learns to automatically assign one of two roles (task-specific or task-mutual) to each of its weights.

3.1 T-net

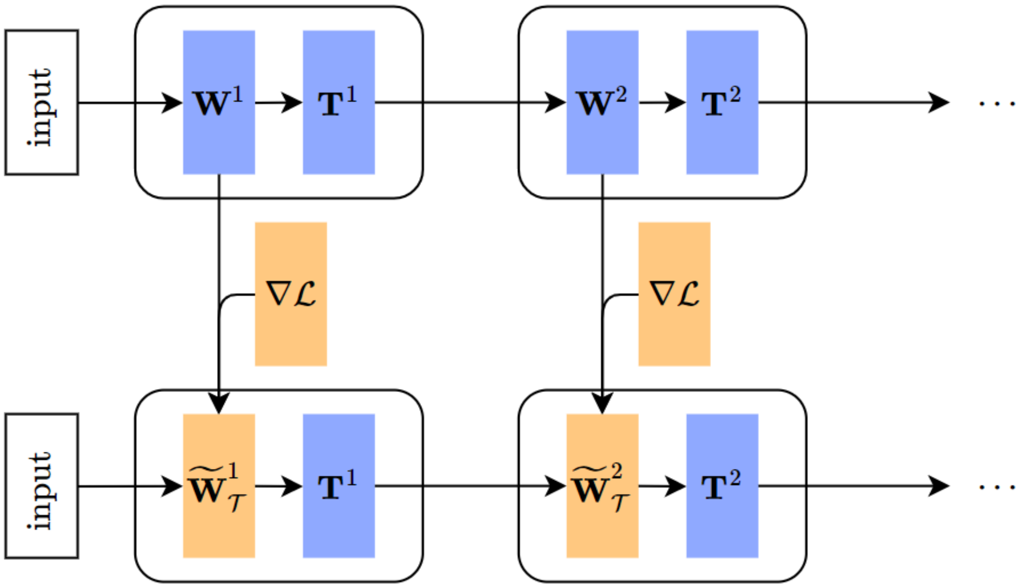

Figure 3.1: A diagram of the adaptation process of a Transformation Network (T-net). Blue values are meta-learned and shared across all tasks. Orange values are different for each task.

We consider a model $f_\theta(\cdot)$ with paramaters θ. This model consists of L cells,

Algorithm 1 Transformation Networks (T-net)

```
Require: $p(\mathcal{T})$
Require: $\alpha$, $\beta$
 1: randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_j$ do
 5:         for $i = 1, \cdots, L$ do
 6:             Compute $\widetilde{\mathbf{W}}_{\mathcal{T}}$ according to (3.2)
 7:         end for
 8:         $\widetilde{\theta}_{\mathbf{W},\mathcal{T}_j} = \{\widetilde{\mathbf{W}}^1_{\mathcal{T}_j}, \cdots \widetilde{\mathbf{W}}^L_{\mathcal{T}_j}\}$
 9:     end for
10:     $\theta \leftarrow \theta - \beta \nabla_\theta \sum_j \mathcal{L}_{\mathcal{T}}(\widetilde{\theta}_{\mathbf{W},\mathcal{T}_j}, \theta_{\mathbf{T}}, \mathcal{D}_{\mathcal{T}_j,test})$
11: end while
```

where each cell is parameterized* as $\mathbf{TW}$:

$$\begin{aligned} & f_\theta(\mathbf{x}) \\ & \quad = \mathbf{T}^L\mathbf{W}^L\left(\sigma\left(\mathbf{T}^{L-1}\mathbf{W}^{L-1}\left(\ldots\sigma\left(\mathbf{T}^1\mathbf{W}^1\mathbf{x}\right)\right)\right)\right), \end{aligned} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^D$ is an input, and $\sigma(\cdot)$ is a nonlinear activation function. T-nets get their name from transformation matrices ($\mathbf{T}$) because the linear transformation defined by a $\mathbf{T}$ plays a crucial role in meta-learning. Note that a cell has the same expressive power as a linear layer. Model parameters θ are therefore a collection of $\mathbf{W}$'s and $\mathbf{T}$'s, i.e.,

$$\theta = \left\{ \underbrace{\mathbf{W}^1, \ldots, \mathbf{W}^L}_{\theta_{\mathbf{W}}}, \underbrace{\mathbf{T}^1, \ldots, \mathbf{T}^L}_{\theta_{\mathbf{T}}} \right\}.$$

Transformation parameters $\theta_{\mathbf{T}}$, which are shared across task-specific models, are determined by the meta-learner. All task-specific learners share the same initial $\theta_{\mathbf{W}}$ but update to different values since each uses their corresponding train set $\mathcal{D}_{\mathcal{T},train}$. Thus we denote such (adjusted) parameters for task $\mathcal{T}$ as $\widetilde{\theta}_{\mathbf{W},\mathcal{T}}$. Though they may look similar, $\mathcal{T}$ denotes a task while $\mathbf{T}$ denotes a transformation matrix.

*For convolutional cells, $\mathbf{W}$ is a convolutional layer with some size and stride and and $\mathbf{T}$ is a 1×1 convolution that doesn't change the number of channels

Given a task $\mathcal{T}$, each $\mathbf{W}$ is adjusted with the gradient update

$$\widetilde{\mathbf{W}}_{\mathcal{T}} \leftarrow \mathbf{W} - \alpha \nabla_{\mathbf{W}} \mathcal{L}_{\mathcal{T}} \left(\theta_{\mathbf{W}}, \theta_{\mathbf{T}}, \mathcal{D}_{\mathcal{T},train} \right). \tag{3.2}$$

Again, $\widetilde{\theta}_{\mathbf{W},\mathcal{T}}$ is defined as $\{\widetilde{\mathbf{W}}^1_{\mathcal{T}}, \ldots, \widetilde{\mathbf{W}}^L_{\mathcal{T}}\}$. Using the task-specific learner $\widetilde{\theta}_{\mathbf{W},\mathcal{T}}$, the meta-learner improves itself with the gradient update

$$\theta \leftarrow \theta - \beta \nabla_\theta \left(\sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}} \left(\widetilde{\theta}_{\mathbf{W},\mathcal{T}}, \theta_{\mathbf{T}}, \mathcal{D}_{\mathcal{T},test} \right) \right). \tag{3.3}$$

$\alpha > 0$ and $\beta > 0$ are learning rate hyperparameters. We show our full algorithm in Algorithm 1.

To evaluate on a new task $\mathcal{T}_*$, we do the following. We compute task-specific parameters $\widetilde{\theta}_{\mathbf{W},\mathcal{T}_*}$ using (3.2), starting from the meta-learned initial value $\theta_{\mathbf{W}}$. We report the loss of task-specific parameters $\widetilde{\theta}_{\mathbf{W},\mathcal{T}_*}$ on the test set $\mathcal{D}_{\mathcal{T}_*,test}$.

We now briefly examine a single cell:

$$\mathbf{y} = \mathbf{T}\mathbf{W}\mathbf{x},$$

where $\mathbf{x}$ is the input to the cell and $\mathbf{y}$ its output. The squared length of a change in output $\Delta\mathbf{y} = \mathbf{y}^* - \mathbf{y}_0$ is calculated as

$$\|\Delta\mathbf{y}\|^2 = ((\Delta\mathbf{W})\mathbf{x})^\top \left(\mathbf{T}^\top \mathbf{T} \right) ((\Delta\mathbf{W})\mathbf{x}), \tag{3.4}$$

where $\Delta\mathbf{W}$ is similarly defined as $\mathbf{W}^* - \mathbf{W}_0$. We see here that the magnitude of $\Delta\mathbf{y}$ is determined by the interaction between $(\Delta\mathbf{W})\mathbf{x}$ and $\mathbf{T}^\top\mathbf{T}$. Since a task-specific learner performs gradient descent only on $\mathbf{W}$ and not $\mathbf{T}$, the change in $\mathbf{y}$ resulting from (3.2) is guided by the meta-learned value $\mathbf{T}^\top\mathbf{T}$. We provide a more precise analysis of this behavior in Chapter IV.

3.2 MT-net

The MT-net is built on the same feedforward model (3.1) as the T-net:

$$\begin{aligned} &f_\theta(\mathbf{x}) \\ &\quad = \mathbf{T}^L \mathbf{W}^L \left(\sigma \left(\mathbf{T}^{L-1} \mathbf{W}^{L-1} \left(\ldots \sigma \left(\mathbf{T}^1 \mathbf{W}^1 \mathbf{x} \right) \right) \right) \right). \end{aligned} \tag{3.5}$$

The MT-net differs from the T-net in the binary mask applied to the gradient update to determine which parameters are to be updated. The update rule for task-specific parameters $\widetilde{\mathbf{W}}_{\mathcal{T}}$ is given by

$$\widetilde{\mathbf{W}}_{\mathcal{T}} \leftarrow \mathbf{W} - \alpha \mathbf{M} \odot \nabla_{\mathbf{W}} \mathcal{L}(\theta_{\mathbf{W}}, \theta_{\mathbf{T}}, \mathcal{D}_{\mathcal{T},train}), \tag{3.6}$$

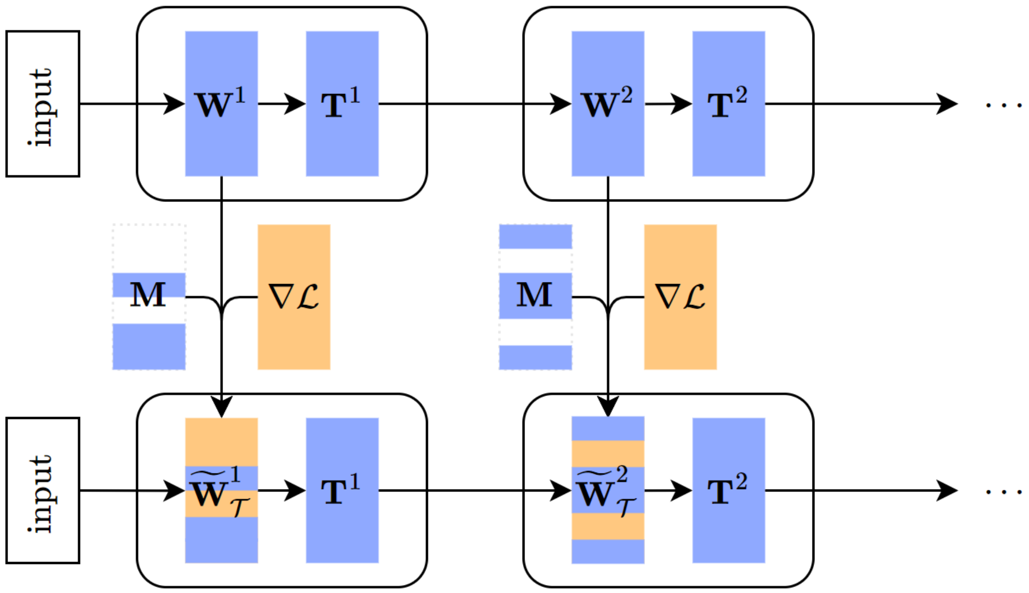

Figure 3.2: A diagram of the adaptation process of a Mask Transformation Network (MT-net). Blue values are meta-learned and shared across all tasks. Orange values are different for each task.

where $\odot$ is the Hadamard (elementwise) product between matrices of the same dimension. $\mathbf{M}$ is a binary gradient mask which is sampled each time the task-specific learner encounters a new task. Each row of $\mathbf{M}$ is either an all-ones vector $\mathbf{1}$ or an all-zeros vector $\mathbf{0}$. We parameterize the probability of row j in $\mathbf{M}$ being $\mathbf{1}$ with a scalar variable ζ_j:

$$\begin{aligned}\mathbf{M} &= [\mathbf{m}_1, \ldots, \mathbf{m}_n]^\top, \\ \mathbf{m}_j^\top &\sim \text{Bern}\left(\frac{\exp(\zeta_j)}{\exp(\zeta_j)+1}\right)\mathbf{1}^\top, \end{aligned} \tag{3.7}$$

where Bern$(\cdot)$ denotes the Bernoulli distribution. Each logit $\boldsymbol{\zeta}$ acts on a row of a weight matrix $\mathbf{W}$, so weights that contribute to the same immediate activation are updated or not updated together.

We approximately differentiate through the Bernoulli sampling of masks using the Gumbel-Softmax estimator [8, 9]:

$$g_1, g_2 \sim \text{Gumbel}(0,1), \tag{3.8}$$

$$\mathbf{m}_j^\top \leftarrow \frac{\exp\left(\frac{\zeta_j+g_1}{\mathbf{c}}\right)}{\exp\left(\frac{\zeta_j+g_1}{\mathbf{c}}\right)+\exp\left(\frac{g_2}{\mathbf{c}}\right)}\mathbf{1}^\top, \tag{3.9}$$

where $\mathbf{c}$ is a temperature hyperparameter. This reparameterization allows us to

Algorithm 2 Mask Transformation Networks (MT-net)

```
Require: $p(\mathcal{T})$
Require: $\alpha$, $\beta$
 1: randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_j$ do
 5:         for $i = 1, \cdots, L$ do
 6:             Sample binary mask $\mathbf{M}^i$ according to (3.9)
 7:             Compute $\widetilde{\mathbf{W}}^i_{\mathcal{T}_j}$ according to (3.6)
 8:         end for
 9:         $\widetilde{\theta}_{\mathbf{W},\mathcal{T}_j} = \{\widetilde{\mathbf{W}}^1_{\mathcal{T}_j}, \cdots \widetilde{\mathbf{W}}^L_{\mathcal{T}_j}\}$
10:     end for
11:     $\theta \leftarrow \theta - \beta \nabla_\theta \sum_j \mathcal{L}_{\mathcal{T}} \left(\widetilde{\theta}_{\mathbf{W},\mathcal{T}}, \theta_{\mathbf{T}}, \theta_{\boldsymbol{\zeta}}, \mathcal{D}_{\mathcal{T},test}\right)$
12: end while
```

directly backpropagate through the mask. At the limit of $\mathbf{c} \to 0$, (3.9) follows the behavior of (3.7).

As in T-nets, we denote the collection of altered weights as $\widetilde{\theta}_{\mathbf{W},\mathcal{T}} = \{\widetilde{\mathbf{W}}^1_{\mathcal{T}}, \ldots, \widetilde{\mathbf{W}}^L_{\mathcal{T}}\}$. The meta-learner learns all parameters θ:

$$\theta = \left\{ \underbrace{\mathbf{W}^1, \ldots, \mathbf{W}^L}_{\theta_{\mathbf{W}}}, \underbrace{\mathbf{T}^1, \ldots, \mathbf{T}^L}_{\theta_{\mathbf{T}}}, \underbrace{\boldsymbol{\zeta}^1, \ldots, \boldsymbol{\zeta}^L,}_{\theta_{\boldsymbol{\zeta}}} \right\}. \tag{3.10}$$

As in a T-net, the meta-learner performs stochastic gradient descent on $\mathcal{L}_{\mathcal{T}} \left(\widetilde{\theta}_{\mathbf{W},\mathcal{T}}, \theta_{\mathbf{T}}, \theta_{\boldsymbol{\zeta}}, \mathcal{D}_{\mathcal{T},test}\right)$:

$$\theta \leftarrow \theta - \beta \nabla_\theta \left(\sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}} \left(\widetilde{\theta}_{\mathbf{W},\mathcal{T}}, \theta_{\mathbf{T}}, \theta_{\boldsymbol{\zeta}}, \mathcal{D}_{\mathcal{T},test}\right) \right). \tag{3.11}$$

The full algorithm is shown in Algorithm 2.

We emphasize that the binary mask used for task-specific learning ($\mathbf{M}$) depends on meta-learned parameter weights ($\boldsymbol{\zeta}$). Since the meta-learner optimizes the loss in a task after a gradient step (3.6), the matrix $\mathbf{M}$ gets assigned a high probability of having value 1 for weights that are meant to encode task-specific information. Furthermore, since we update $\mathbf{M}$ along with model parameters $\mathbf{W}$

and **T**, the meta-learner is incentivized to learn configurations of **W** and **T** in which there exists a clear divide between task-specific and task-mutual neurons.

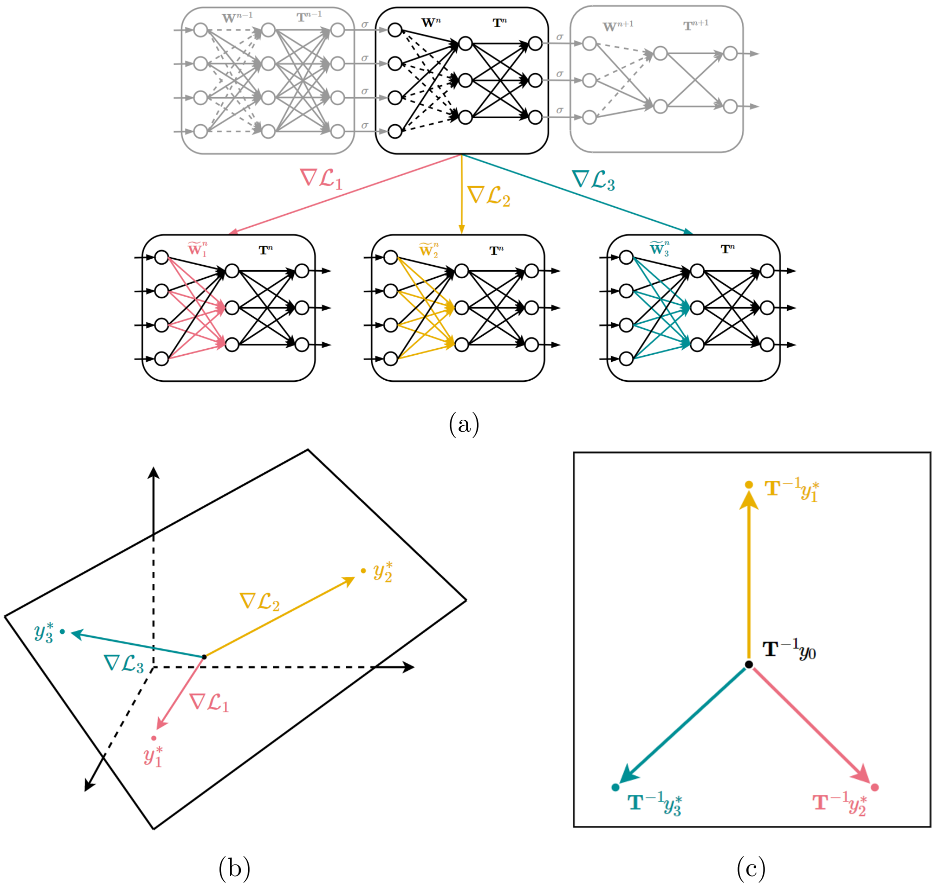

Figure 3.3: Task-specific learning in an MT-net. (a) A cell (rounded rectangle) consists of two layers. In addition to initial weights (black), the meta-learner specifies weights to be changed (dotted lines) by task-specific learners (colored). (b) Activation of this cell has 3 dimensions, but activation of task-specific learners only change within a subspace (white plane). (c) The value of $\mathbf{T}$ affects task-specific learning so that gradients of $\mathbf{W}$ are sensitive to task identity. Best seen in color.

IV. Analysis

In this chapter, we provide further analysis of the update schemes of T-nets and MT-nets.

We analyse how the activation space of a single cell of a T-net or MT-net behaves during task-specific learning. More specifically, we make precise how $\mathbf{W}$ encodes a learned curvature matrix. By using such an analysis to reason about a whole network consisting of several cells, we are impliticly approximating the full curvature matrix of the network by a block-diagonal curvature matrix. In this approximation, second-order interactions only occur among weights in the same layer (or cell). Previous works [10, 11, 12] have used such an approximation of the curvature of a neural network.

4.1 T-nets Learn a Metric in Activation Space

We consider a cell in a T-net where the pre-activation value $\mathbf{y}$ is given by

$$\mathbf{y} = \mathbf{TWx} = \mathbf{Ax}, \tag{4.1}$$

where $\mathbf{A} = \mathbf{TW}$ and $\mathbf{x}$ is the input to the cell. We omit superscripts throughout this chapter.

A standard feedforward network resorts to the gradient of a loss function $\mathcal{L}_{\mathcal{T}}$ (which involves a particular task $\mathcal{T} \sim p(\mathcal{T})$) with respect to the parameter matrix $\mathbf{A}$, to update model parameters. In such a case, a single gradient step yields

$$\begin{aligned} \mathbf{y}^{\text{new}} &= (\mathbf{A} - \alpha \nabla_{\mathbf{A}} \mathcal{L}_{\mathcal{T}})\mathbf{x} \\ &= \mathbf{y} - \alpha \nabla_{\mathbf{A}} \mathcal{L}_{\mathcal{T}} \mathbf{x}. \end{aligned} \tag{4.2}$$

The update of a T-net (3.2) results in the following new value of $\mathbf{y}$:

$$\begin{aligned} \mathbf{y}^{\text{new}} &= \mathbf{T}\left(\mathbf{T}^{-1}\mathbf{A} - \alpha \nabla_{\mathbf{T}^{-1}\mathbf{A}} \mathcal{L}_{\mathcal{T}}\right)\mathbf{x} \\ &= \mathbf{y} - \alpha \left(\mathbf{T}\mathbf{T}^{\top}\right) \nabla_{\mathbf{A}} \mathcal{L}_{\mathcal{T}} \mathbf{x}, \end{aligned} \tag{4.3}$$

where $\mathbf{T}$ is determined by the meta-learner. Thus, in a T-net, the incremental change of $\mathbf{y}$ is proportional to the negative of the gradient $\left(\mathbf{T}\mathbf{T}^{\top}\right) \nabla_{\mathbf{A}} \mathcal{L}_{\mathcal{T}}$, while the standard feedforward net resorts to a step proportional to the negative of

$\nabla_{\mathbf{A}}\mathcal{L}_{\mathcal{T}}$. Task-specific learning in the T-net is guided by a full rank metric in each cell's activation space, which is determined by each cell's transformation matrix $\mathbf{T}$. This metric $(\mathbf{T}\mathbf{T}^\top)^{-1}$ warps (scaling, rotation, etc.) the activation space of the model so that in this warped space, a single gradient step with respect to the loss of a new task yields parameters that are well suited for that task.

4.2 MT-nets Learn a Subspace with a Metric

We now consider MT-nets and analyze what their update (3.6) means from the viewpoint of $\mathbf{y} = \mathbf{T}\mathbf{W}\mathbf{x} = \mathbf{A}\mathbf{x}$.

MT-nets can restrict its task-specific learner to any subspace of its gradient space:

Proposition 1. *Fix* $\mathbf{x}$ *and* $\mathbf{A}$. *Let* $\mathbf{y} = \mathbf{T}\mathbf{W}\mathbf{x}$ *be a cell in an MT-net and let* $\boldsymbol{\zeta}$ *be its corresponding mask parameters. Let* $\mathbf{U}$ *be a* d*-dimensional subspace of* $\mathbb{R}^n$ $(d \leq n)$*. There exist configurations of* $\mathbf{T}, \mathbf{W}$*, and* $\boldsymbol{\zeta}$ *such that the span of* $\mathbf{y}^{new} - \mathbf{y}$ *is* $\mathbf{U}$ *while satisfying* $\mathbf{A} = \mathbf{T}\mathbf{W}$.

Proof. We show by construction that Proposition 1 is true.

Suppose that $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is a basis of $\mathbb{R}^n$ such that $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d\}$ is a basis of $\mathbf{U}$. Let $\mathbf{T}$ be the $n \times n$ matrix $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$. $\mathbf{T}$ is invertible since it consists of linearly independent columns. Let $\mathbf{W} = \mathbf{T}^{-1}\mathbf{A}$ and let $\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_d \to \infty$ and $\boldsymbol{\zeta}_{d+1}, \ldots, \boldsymbol{\zeta}_n \to -\infty$. The resulting mask $\mathbf{M}$ that $\boldsymbol{\zeta}$ generates is a matrix with only ones in the first d rows and zeroes elsewhere.

$$\begin{aligned}\mathbf{y}^{\text{new}} - \mathbf{y} &= \mathbf{T}(\mathbf{W}^{\text{new}} - \mathbf{W})\mathbf{x} \\ &= \mathbf{T}(\mathbf{M} \odot \nabla_{\mathbf{W}}\mathcal{L}_{\mathcal{T}})\mathbf{x} \qquad (4.4)\end{aligned}$$

Since all but the first d rows of $\mathbf{M}$ are $\mathbf{0}$, $(\mathbf{M} \odot \nabla_{\mathbf{W}}\mathcal{L}_{\mathcal{T}})\mathbf{x}$ is an n-dimensional vector in which nonzero elements can only appear in the first d dimensions. Therefore, the vector $\mathbf{T}(\mathbf{M} \odot \nabla_{\mathbf{W}}\mathcal{L}_{\mathcal{T}})\mathbf{x}$ is a linear combination of $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d\}$. Thus the span of $\mathbf{y}^{\text{new}} - \mathbf{y}$ is $\mathbf{U}$. □

This proposition states that $\mathbf{W}, \mathbf{T}$, and $\boldsymbol{\zeta}$ have sufficient expressive power to restrict updates of $\mathbf{y}$ to any subspace. Note that this construction is only possible because of the transformation $\mathbf{T}$; if we only had binary masks $\mathbf{M}$, we would only be able to restrict gradients to axis-aligned subspaces.

In addition to learning a subspace that we project gradients onto ($\mathbf{U}$), we are also learning a metric in this subspace. We first provide an intuitive exposition of this idea.

We unroll the update of an MT-net as we did with T-nets in (4.3):

$$\begin{aligned}\mathbf{y}^{\text{new}} =& \mathbf{T}((\mathbf{T}^{-1}\mathbf{A} - \alpha\mathbf{M} \odot \nabla_{\mathbf{T}^{-1}\mathbf{A}}\mathcal{L}_{\mathcal{T}})\mathbf{x}) \\ =& \mathbf{y} - \alpha\mathbf{T}(\mathbf{M} \odot (\mathbf{T}^{\top}\nabla_{\mathbf{A}}\mathcal{L}_{\mathcal{T}}))\mathbf{x} \\ =& \mathbf{y} - \alpha\mathbf{T}(\mathbf{M}_{\mathbf{T}} \odot \mathbf{T}^{\top})\nabla_{\mathbf{A}}\mathcal{L}_{\mathcal{T}}\mathbf{x} \\ =& \mathbf{y} - \alpha(\mathbf{T} \odot \mathbf{M}_{\mathbf{T}}^{\top})(\mathbf{M}_{\mathbf{T}} \odot \mathbf{T}^{\top})\nabla_{\mathbf{A}}\mathcal{L}_{\mathcal{T}}\mathbf{x}. \end{aligned} \tag{4.5}$$

Where $\mathbf{M}_{\mathbf{T}}$ is an $m \times m$ matrix which has the same columns as $\mathbf{M}$. Let's denote $\mathbf{T}_M = \mathbf{M}_{\mathbf{T}} \odot \mathbf{T}^{\top}$. We see that the update of a task-specific learner in an MT-net performs the update $\mathbf{T}_M^{\top}\mathbf{T}_M\nabla_{\mathbf{A}}\mathcal{L}_{\mathcal{T}}$. Note that $\mathbf{T}_M^{\top}\mathbf{T}_M$ is an $n \times n$ matrix that only has nonzero elements in rows and columns where $\mathbf{m}$ is 1. By setting appropriate $\boldsymbol{\zeta}$, we can view $\mathbf{T}_M^{\top}\mathbf{T}_M$ as a full-rank $d \times d$ metric tensor.

This observation can be formally stated as:

Proposition 2. *Fix $\mathbf{x}$, $\mathbf{A}$, and a loss function $\mathcal{L}_{\mathcal{T}}$. Let $\mathbf{y} = \mathbf{TWx}$ be a cell in an MT-net and let $\boldsymbol{\zeta}$ be its corresponding mask parameters. Let $\mathbf{U}$ be a d-dimensional subspace of $\mathbb{R}^n$, and $g(\cdot,\cdot)$ a metric tensor on $\mathbf{U}$. There exist configurations of $\mathbf{T}, \mathbf{W}$, and $\boldsymbol{\zeta}$ such that the vector $\mathbf{y}^{new} - \mathbf{y}$ is in the steepest direction of descent on $\mathcal{L}_{\mathcal{T}}$ with respect to the metric $g(\cdot,\cdot)$.*

Proof. We show Proposition 2 is true by construction as well.

We begin by constructing a representation for the arbitrary metric tensor $g(\cdot,\cdot)$. Let $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$ be a basis of $\mathbb{R}^n$ such that $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d\}$ is a basis of $\mathbf{U}$. Vectors $\mathbf{u}_1, \mathbf{u}_2 \in \mathbf{U}$ can be expressed as $\mathbf{u}_1 = \sum_{i=0}^{d} c_{1i}\mathbf{v}_i$ and $\mathbf{u}_2 = \sum_{i=0}^{d} c_{2i}\mathbf{v}_i$. We can express any metric tensor $g(\cdot,\cdot)$ using such coefficients c:

$$g(\mathbf{u}_1, \mathbf{u}_2) = \underbrace{\begin{bmatrix} c_{11} & \dots & c_{1d} \end{bmatrix}}_{\mathbf{c}_1^{\top}} \underbrace{\begin{bmatrix} g_{11} & \cdots & g_{1d} \\ \vdots & \ddots & \vdots \\ g_{d1} & \cdots & g_{dd} \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} c_{21} \\ \vdots \\ c_{2d} \end{bmatrix}}_{\mathbf{c}_2}, \tag{4.6}$$

where $\mathbf{G}$ is a positive definite matrix. Because of this, there exists an invertible $d \times d$ matrix $\mathbf{H}$ such that $\mathbf{G} = \mathbf{H}^{\top}\mathbf{H}$. Note that $g(\mathbf{u}_1, \mathbf{u}_2) = (\mathbf{Hc}_1)^{\top}(\mathbf{Hc}_2)$: the metric $g(\cdot,\cdot)$ is equal to the inner product after multiplying $\mathbf{H}$ to given vectors $\mathbf{c}$.

Using $\mathbf{H}$, we can alternatively parameterize vectors in $\mathbf{U}$ as

$$\mathbf{u}_1 = \underbrace{\begin{bmatrix}\mathbf{v}_1 & \ldots & \mathbf{v}_d\end{bmatrix}}_{\mathbf{V}} \mathbf{c}_1 \tag{4.7}$$

$$= \mathbf{V}\mathbf{H}^{-1}\left(\mathbf{H}\mathbf{c}_1\right). \tag{4.8}$$

Here, we are using $\mathbf{H}\mathbf{c}_1$ as a d-dimensional parameterization and the columns of the $n \times d$ matrix $\mathbf{V}\mathbf{H}^{-1}$ as an alternative basis of $\mathbf{U}$.

Let $\mathbf{v}_1^H, \ldots, \mathbf{v}_d^H$ be the columns of $\mathbf{V}\mathbf{H}^{-1}$, and set $\mathbf{T} = [\mathbf{v}_1^{\mathbf{H}}, \ldots, \mathbf{v}_d^{\mathbf{H}}, \mathbf{v}_{d+1}, \ldots, \mathbf{v}_n]$. Since $\mathbf{H}$ is invertible, $\{\mathbf{v}_1^{\mathbf{H}}, \ldots, \mathbf{v}_d^{\mathbf{H}}\}$ is a basis of $\mathbf{U}$ and thus $\mathbf{T}$ is an invertible matrix. As in Proposition 1, set $\mathbf{W} = \mathbf{T}^{-1}\mathbf{A}$, $\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_d \to \infty$, and $\boldsymbol{\zeta}_{d+1}, \ldots, \boldsymbol{\zeta}_n \to -\infty$. Note that this configuration of $\boldsymbol{\zeta}$ generates a mask $\mathbf{M}$ that projects gradients onto the first d rows, which will later be multiplied by the vectors $\{\mathbf{v}_1^{\mathbf{H}}, \ldots, \mathbf{v}_d^{\mathbf{H}}\}$.

We can express $\mathbf{y}$ as $\mathbf{y} = V\mathbf{c}_\mathbf{y} = \mathbf{V}\mathbf{H}^{-1}(\mathbf{H}\mathbf{c}_\mathbf{y})$, where $\mathbf{c}_\mathbf{y}$ is again a d-dimensional vector. Note that $\mathbf{V}\mathbf{H}^{-1}$ is constant in the network and change in $\mathbf{W}$ only affects $\mathbf{H}\mathbf{c}_\mathbf{y}$. Since $\nabla_\mathbf{W}\mathcal{L}_\mathcal{T} = (\nabla_{\mathbf{W}\mathbf{x}}\mathcal{L}_\mathcal{T})\mathbf{x}^\top$, the task-specific update is in the direction of steepest descent of $\mathcal{L}_\mathcal{T}$ in the space of $\mathbf{H}\mathbf{c}_y$ (with the Euclidean metric). This is exactly the direction of steepest descent of $\mathcal{L}_\mathcal{T}$ in $\mathbf{U}$ with respect to the metric $g(\cdot,\cdot)$. □

Therefore, not only can MT-nets project gradients of task-specific learners onto a subspace of the pre-activation ($\mathbf{y}$) space, they can also learn a metric in that subspace and thereby learning a low-dimensional linear embedding of the activation space. The MT-net update (3.6) is gradient descent in this low-dimensional embedding, so the meta-objective shown in (3.11) is minimized when gradient descent in this embedding requires few steps to converge and is sensitive to task identity.

V. Related Work

A successful line of research in few-shot learning uses feedforward neural networks as learners. With a fixed feedforward neural network, one can learn learn update rules given gradients[13, 14, 15] or directly generate weights when encountered with a new task [16]. A related research direction is to learn initial parameters [6] while fixing the learning rule to gradient descent, or additionally learning learning rates for each weight [17]. [18] interprets such gradient-based meta-learning as hierarchical bayesian inference, and [7] states that such methods are expressive enough to approximate any learning algorithm.

Our proposed method is closely related to this line of research. Like as in [13, 14, 15, 16, 6, 17, 18, 7], we fix a parametric model and meta-learn a quantity that aids in learning using this model. Unlike previous work, MT-nets learn how many degrees of freedom the task-specific learner should have at meta-test time. While the meta-learner of an MT-net uses a fixed model, task-specific learning occurs in a smaller model made of a learned subset of the full set of weights. Additionally, while MT-nets do learn update rules, these update rules are directly embedded in the network itself instead of being stored in a separate model as in [13, 14, 15].

Distance metric learning [19, 20] methods learn a distance function between datapoints. Similarly, MT-nets learn a full metric matrix which corresponds to a distance function for each activation. Whereas those methods required constrained optimization techniques to enforce that the learned matrix represents a metric, our parameterization allows us to directly learn such a metric using gradient descent. Recently, neural networks have been used to learn a metric between images[21, 22, 23], achieving state-of-the-art performance on few-shot classification benchmarks. Unlike these methods, we learn a metric in feature space instead of input space. Additionally, while those methods use the learned metric directly to classify images, we use our learned metric to perform gradient descent on. Compared to [21, 22, 23], our method applies to a larger class of problems including regression and reinforcement learning, since MT-nets only require a differentiable loss function.

Another line of research in few-shot learning is to use a recurrent neural network (RNN) as a learner [24, 25]. Here, the meta-learning algorithm is gradient descent on an RNN, and the learning algorithm is the update of hidden cells. The (meta-learned) weights of the RNN specify a learning strategy, which processes training data and uses the resulting hidden state vector to make decisions about test data. A recent work that uses temporal convolutions for meta-learning[26] is also closely related to this line of research.

VI. Experiments

We performed experiments to answer:

- Do our novel components ($\mathbf{TW}, \mathbf{M}$ etc) improve meta-learning performance? (6.1)
- Is applying masks $\mathbf{M}$ row-wise actually better than applying them parameter-wise? (6.1)
- To what degree does $\mathbf{T}$ alleviate the need for careful tuning of step size α? (6.2)
- In MT-nets, does learned subspace dimension reflect the difficulty of tasks? (6.3)
- Can T-nets and MT-nets scale to large-scale meta-learning problems? (6.4)

Most of our experiments were performed by modifying the code accompanying [6], and we follow their experimental protocol and hyperparameters unless specified otherwise.

6.1 Toy Regression Problem

We start with a K-shot regression problem and compare results to previous meta-learning methods [6, 17]. The details of our regression task are the same as [17]. Each individual task is to regress from the input x to the output y of a sine function

$$y(x) = A \sin(wx + b) \tag{6.1}$$

For each task, A, w, b are sampled uniformly from $[0.1, 5.0], [0.8, 1.2]$, and $[0, \pi]$, respectively. Each task consists of $K \in \{5, 10, 20\}$ training examples and 10 testing examples. We sample x uniformly from $[-5.0, 5.0]$ for both train and test sets. Our regressor architecture has two hidden cells each with activation size 40. After every $\mathbf{T}$ is a ReLU nonlinearity. The loss function is the mean squared error (MSE) between the regressor's prediction $f(x)$ and the true value $y(x)$. We used Adam [27] as our meta-optimizer with a learning rate of $\beta = 10^{-3}$. Task-specifc

Models	5-shot	10-shot	20-shot
MAML[1]	1.07 ± 0.11	0.71 ± 0.07	0.50 ± 0.05
Meta-SGD[1]	0.88 ± 0.14	0.53 ± 0.09	0.35 ± 0.06
M-net-full	0.91 ± 0.09	0.63 ± 0.07	0.38 ± 0.04
M-net	0.88 ± 0.09	0.60 ± 0.06	0.41 ± 0.04
T-net	0.83 ± 0.08	0.56 ± 0.06	0.38 ± 0.04
MT-net-full	0.81 ± 0.08	0.51 ± 0.05	0.35 ± 0.04
MT-net	$\mathbf{0.76 \pm 0.09}$	$\mathbf{0.49 \pm 0.05}$	$\mathbf{0.33 \pm 0.04}$

Table 6.1: Loss on sine wave regression. Networks were meta-trained using 10-shot regression tasks. Reported losses were calculated after adaptation using various numbers of examples.

[1] Reported by [17].

learners used step size $\alpha = 10^{-2}$. We initialize all $\boldsymbol{\zeta}$ to 0, all $\mathbf{T}$ as identity matrices, and all $\mathbf{W}$ as truncated normal matrices with standard deviation 10^{-2}. While we trained our meta-learner with $K = 10$ examples, we tested using various numbers of examples ($K \in \{5, 10, 20\}$).

We show results in Table 1. To see if each of our novel components increased meta-learning performance, we also performed the same experiments with variations of MT-nets. An M-net uses a mask $\mathbf{M}$ like an MT-net, but each cell consists of a single matrix $\mathbf{W}$ instead of $\mathbf{TW}$. A model with "-full" at the end of its name learns a separate mask parameter for each weight of $\mathbf{W}$ instead of sharing a mask among weights that contribute to the same activation. For example, if $\mathbf{W}$ has size 5×10, the corresponding $\boldsymbol{\zeta}$ in an MT-net would be of dimension 5, but in MT-net-full, the dimension of $\boldsymbol{\zeta}$ would be 50. MT-nets outperform MAML, meta-SGD, and all variations of MT-nets.

6.2 Robustness to learning rate change

The transformation $\mathbf{T}$ of our method can change the effective step size α. We performed experiments to see how robust our method is to variations in α. We perform the same sinusoid experiment as in chapter 6.1, but with various step sizes ($\alpha \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$). We evaluate on $K = 10$ training examples, and all other settings are identical to the experiments in chapter 6.1.

α	MAML	T-net	MT-net
10	171.92 $\pm$ 25.04	**4.08 $\pm$ 0.30**	4.18 $\pm$ 0.30
1	5.81 $\pm$ 0.49	4.15 $\pm$ 0.30	**0.61 $\pm$ 0.07**
0.1	1.05 $\pm$ 0.11	0.68 $\pm$ 0.06	**0.54 $\pm$ 0.05**
0.01	0.71 $\pm$ 0.07	0.56 $\pm$ 0.06	**0.49 $\pm$ 0.05**
0.001	0.82 $\pm$ 0.08	**0.59 $\pm$ 0.06**	**0.59 $\pm$ 0.06**
0.0001	2.54 $\pm$ 0.19	**0.62 $\pm$ 0.06**	0.72 $\pm$ 0.07

Table 6.2: Loss on 10-shot sine wave regression. T-nets and MT-nets are bost robust to change in step size α. This is due to the meta-learned matrix $\mathbf{T}$ inside each cell, which alters the effective step size.

We show losses after adaptation of both MAML and MT-nets in Table 2. We can see that MT-nets are more robust to change in step size. This indicates that as shown in chapter 4.2, the matrix $\mathbf{T}$ is capable of warping the parameter space to recover from suboptimal step size α.

6.3 Task Complexity and Subspace Dimension

We performed this experiment to see whether the dimension of the learned subspace of MT-nets reflect the underlying complexity of its given set of tasks.

We consider 10-shot regression tasks in which the target function is a polynomial. A polynomial regression meta-task consists of polynomials of the same order with various coefficients. To generate a polynomial of order n ($\sum_{i=0}^{n} c_i x^i$), we uniformly sampled $c_0, \ldots, c_n$ from $[-1, 1]$. We used the same network architecture and hyperparameters as in chapter 6.1 and performed 10-shot regression for polynomial orders $n \in \{0, 1, 2\}$. Since the number of free parameters is proportional to the order of the polynomial, we expect higher-order polynomials to require more parameters to adapt to. The fraction of parameters that task-specific learners change is calculated as the expected value of $\frac{e^{-\boldsymbol{\zeta}}}{e^{-\boldsymbol{\zeta}}+1}$ over all logits $\boldsymbol{\zeta}$.

We show results in Figure 4. The number of weights that the meta-learner of an MT-net sets to be altered increases as the task gets more complex. We interpret this as the meta-learner of MT-nets having an effect akin to Occam's razor: it selects a task-specific model of just enough complexity to learn in a set of tasks. This behavior emerges even though we do not introduce any additional

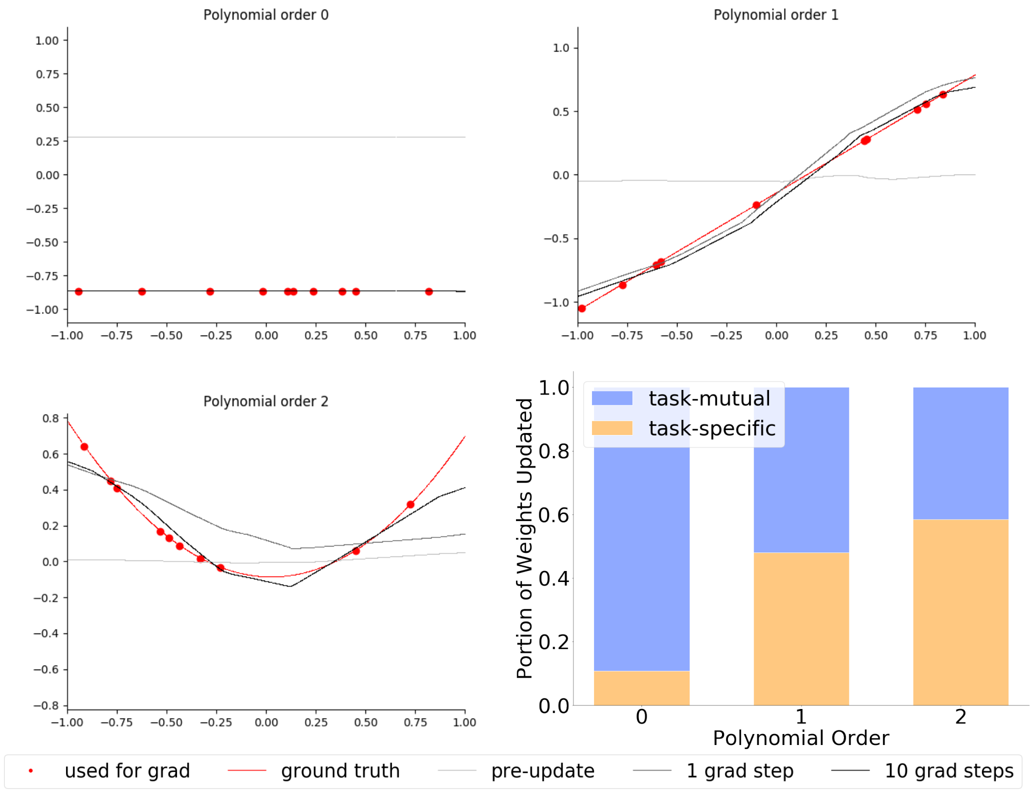

Figure 6.1: 10-shot regression tasks to sets of polynomials of various degrees. MT-nets choose to update a larger fraction of weights as the set of tasks gets more complex.

loss terms to encourage such behavior. We think this is caused by the noise inherent in stochastic gradient descent. Since the meta-learner of an MT-net can choose whether or not to perform gradient descent in a particular direction, it is incentivized not to do so in directions that are not model-specific, because doing so would introduce more noise into the network parameters and thus (in expectation) suffer more loss.

6.4 Classification

To compare the performance of MT-nets to prior work in meta-learning, we evaluate our method on few-shot classification on the Omniglot [29] and MiniImagenet [13] datasets. We used the miniImagenet splits proposed by [13] in our experiments.

Our CNN model uses the same architecture as [6]. The model has 4 modules:

each has 3×3 convolutions and 64 filters, followed by batch normalization [30]. As in [6], we used 32 filters per layer in miniImagenet. Convolutions have stride 2×2 on Omniglot, and 2×2 max-pooling is used after batch normalization instead of strided convolutions on MiniImagenet. We evaluate with 3, 5, and 10 gradient steps for Omniglot 5-way, Omniglot 20-way, and miniImagenet 5-way, respectively.

Results are shown in Table 3. MT-nets achieve state-of-the-art or comparable performance on both problems. Several works [26, 25, 31] have reported improved performance on MiniImagenet using a significantly more expressive architecture. We only report methods that have equal or comparable expressiveness to the model first described in [22]. Not controlling for network expressivity, the highest reported accuracy so far on 5-way 1-shot miniImagenet classification is 57.02 [31].

Models	5-way 1-shot acc. (%)	20-way 1-shot acc. (%)
Matching Networks[22]	98.1	93.8
Prototypical Networks[23]	97.4	92.0
mAP-SSVM[28]	98.6	95.4
MAML[6]	98.7 ± 0.4	95.8 ± 0.3
Meta-SGD[17]	$\mathbf{99.53 \pm 0.26}$	95.93 ± 0.38
T-net (ours)	99.4 ± 0.3	96.1 ± 0.3
MT-net (ours)	$\mathbf{99.5 \pm 0.3}$	$\mathbf{96.2 \pm 0.4}$

Models	5-way 1-shot acc. (%)
Matching Networks[22][1]	43.56 ± 0.84
Prototypical Networks[23][2]	46.61 ± 0.78
mAP-SSVM[28]	50.32 ± 0.80
Fine-tune baseline[1]	28.86 ± 0.54
Nearest Neighbor baseline[1]	41.08 ± 0.70
meta-learner LSTM[13]	43.44 ± 0.77
MAML[6]	48.70 ± 1.84
L-MAML[18]	49.40 ± 1.83
Meta-SGD[17]	50.47 ± 1.87
T-net (ours)	50.86 ± 1.82
MT-net (ours)	$\mathbf{51.70 \pm 1.84}$

Table 6.3: Few-shot classification accuracy on (top) held-out Omniglot characters and (bottom) test split of MiniImagenet. ± represents 95% confidence intervals.

[1] Reported by [13].
[2] Reported results for 5-way 1-shot.

VII. Conclusion

We introduced T-nets and MT-nets. One can transform any feedforward neural network into an MT-net, so any future architectural advances can take advantage of our method. Experiments showed that our method alleviates the need for careful tuning of the learning rate in few-shot learning problems and that the mask **M** reflects the complexity of the set of tasks it is learning to adapt in. MT-nets also showed state-of-the-art performance in a challenging few-shot classification benchmark (MiniImagenet).

While MT-nets are a gradient-based meta-learning method, our analysis has shown that it also has interesting commonalities with optimizer learning methods such as [13]. Unlike other optimizer learning methods which constructed a separate neural network (typically an RNN) to encode a learning rule, having such a rule arise from the interaction between gradient descent and the structure of the network seems to be a more biologically plausible alternative. We will investigate this connection between two seemingly disparate approaches to meta-learning in future work.

One of the biggest weaknesses of deep networks is that they are very data intensive. By learning what to learn when a new task is encountered, we can train networks with high capacity using a small amount of data. We believe that designing effective gradient-based meta-learners will be beneficial not just for the few-shot learning setting, but also machine learning problems in general.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

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

[3] J. Schmidhuber. *Evolutionary Principles in Self-Referential Learning.* PhD thesis, Technical University of Munich, 1987.

[4] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.

[5] S. Thrun and L. Pratt. *Learning to Learn.* Kluwer Academic Publishers, 1998.

[6] C. Finn, P. Abbeel., and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, Sydney, Australia, 2017.

[7] C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm, 2017. *Preprint arXiv:1710.11622.*

[8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[9] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[10] Tom Heskes. On “natural” learning and pruning in multilayered perceptrons. *Neural Computation*, 2000.

[11] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. 2015.

[12] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. Natural neural networks. 2015.

[13] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.

[14] Ke Li and Jitendra Malik. Learning to Optimize. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[15] Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, Nando De Freitas, and Google Deepmind. Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[16] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[17] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to Learn Quickly for Few Shot Learning, 2017. *Preprint arXiv:1707.09835.*

[18] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[19] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russel. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems (NIPS)*, volume 15. MIT Press, 2003.

[20] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems (NIPS)*, volume 18, 2006.

[21] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*, Lille, France, 2015.

[22] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, 2016.

[23] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, 2017.

[24] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lilicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, New York, NY, USA, 2016.

[25] T. Munkhdalai and H. Yu. Meta networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, Sydney, Australia, 2017.

[26] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[27] D. P. Kingma and J. L. Ba. ADAM: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

[28] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-shot learning through an information retrieval lens. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[29] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

[30] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[31] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to Compare: Relation Network for Few-Shot Learning, 2017. *Preprint arXiv:1711.06025.*

[32] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[33] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[34] Harrison Edwards and Amos Storkey. Towards a Neural Statistician. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[35] Lukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to Remember Rare Events. *ICLR*, 2017.

[36] V. Garcia and J. Bruna. Few-Shot Learning with Graph Neural Networks, 2017. *Preprint arXiv:1711.04043.*

[37] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Learning to Generalize: Meta-Learning for Domain Generalization. *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI)*, 2017.

[38] Y.-H. H. Tsai and R. Salakhutdinov. Improving One-Shot Learning through Fusing Side Information, 2017. *Preprint arXiv:1710.08347.*

[39] S. Fort. Gaussian Prototypical Networks for Few-Shot Learning on Omniglot, 2017. *Preprint arXiv:1708.02735.*

[40] Mengye Ren, Sachin Ravi, Eleni Triantafillou, Jake Snell, Kevin Swersky, Josh B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[41] Bharath Hariharan and Ross B. Girshick. Low-shot visual object recognition by shrinking and hallucinating features. *Proceedings of the International Conference on Computer Vision (ICCV)*, 2016.

요 약 문

본 논문은 데이터가 많지 않은 문제에서의 효율적인 경사도 기반(gradient-based) 학습을 위해 신경망의 각 층 안에서 학습이 진행될 부분공간(subspace)와 그 부분공간에서 어떤 속도로 움직일 지를 나타내는 거리함수(metric)을 학습하는 메타러닝 기법을 제안한다.

Acknowledgements

지난 2년 동안 최선을 다해 지도해주신 최승진 교수님께 감사드립니다. 제가 연구 때문에 힘들어할 때마다 교수님께서는 필요한 조언을 해주셨고, 그런 조언들 덕분에 석사학위를 잘 끝마칠 수 있었다고 생각합니다. 연구자가 가져야 할 자세, 자신의 아이디어를 남들에게 전달하는 방법 등 많은 것을 교수님으로부터 배웠습니다.

저를 믿어주시면서 연구자로서 어떻게 해야 하는지를 가르쳐주신 하승열 교수님, 제 연구에 대해서 아낌없이 조언을 해 주셨던 한보형 교수님, 그리고 논문심사를 해주시면서 발표와 관련된 조언을 해주신 조민수 교수님과 곽수하 교수님께 감사드립니다.

연구실에서 많은 시간을 함께 보낸 연구실 동료들에게도 고마운 마음을 느낍니다. 본곤이형, 세훈이형, 주호형, 정택이형, 지유형, 민섭이형, 인혁이형, 원빈이형, 영남이형, 영석이, 그리고 나영이 모두, 힘든 일도 많은 연구실 생활이었지만 함께하는 사람들이 있어서 행복했습니다.

멀리 살지만 힘들 때 가장 먼저 연락하고 싶은 친구 동현이와 소희, 같은 시기에 대학원 생활을 하면서 기쁨과 슬픔을 진심으로 함께해 준 단일이, 그리고 부족한 저와 계속 함께해준 고등학교 친구들, 분반 친구들, 과 친구들 모두에게 감사합니다.

무엇보다도 항상 믿고 응원해주시는 부모님께 감사하고 사랑한다고 말씀드리고 싶습니다. 무슨 일이 있을지라도 부모님께서는 저를 사랑해주실 거라는 믿음이 있었기에 묵묵히 연구를 할 수 있었습니다.

Curriculum Vitae

Name	: Yoonho Lee

Education

2012. 3. – 2016. 8.	Department of Mathematics, Pohang University of Science and Technology (B.S.)
2016. 9. – 2018. 8.	Department of Computer Science and Engineering, Pohang University of Science and Technology (M.S.)

Publications

Yoonho Lee and Seungjin Choi, Gradient-based Meta-Learning with Learned Layerwise Metric and Subspace, 35th International Conference on Machine Learning (ICML 2018)