Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace

Yoonho Lee 1  Seungjin Choi 1

Abstract
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.

1. Introduction
While recent deep learning methods achieve superhuman performance on various tasks including image classification (Krizhevsky et al., 2012) or playing games (Mnih et al., 2015), 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 (Schmidhuber, 1987; Schmidhuber et al., 1997; Thrun & Pratt, 1998) 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) (Finn et al., 2017) 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 can 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.

2. 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 \{T_1, T_2, T_3, \ldots\}. Each task provides a training set and a test set \{D_{T_{train}}, D_{T_{test}}\}. We assume here that the training set \{T_{train}\} has k examples per class, hence the name k-shot learning. A particular task \mathcal{T} \in \{T_1, T_2, T_3, \ldots\}
is assumed to be drawn from the distribution of tasks \( p(T) \). Given a task \( T \sim p(T) \), the task-specific model \( f_{\theta_T} \) (our work considers a feedforward neural network) is trained using the dataset \( D_{T,\text{train}} \) and its corresponding loss \( L_T(\theta_T, D_{T,\text{train}}) \). Denote by \( \theta_T \) parameters obtained by optimizing \( L_T(\theta_T, D_{T,\text{train}}) \). Then, the meta-learner \( f_\theta \) is updated using the feedback from the collection of losses \( \left\{ L_T(\tilde{\theta}_T, D_{T,\text{test}}) \right\}_{T \sim p(T)} \), where the loss of each task is evaluated using the test data \( D_{T,\text{test}} \). Given a new task \( T_{\text{new}} \) (not considered during meta-training), the meta-learner helps the model \( f_{\theta_{T_{\text{new}}}} \) to quickly adapt to the new task \( T_{\text{new}} \), by warm-starting the gradient updates.

### 2.2. Model-Agnostic Meta-Learning

We briefly review model-agnostic meta-learning (MAML) (Finn et al., 2017), 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 \( \theta \). MAML alternates between the two updates (1) and (2) to determine initial parameters \( \theta \) 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 (1) using the loss evaluated with the training data \( \left\{ D_{T,\text{train}} \right\} \). The meta-optimization across tasks (2) is performed such that the parameters \( \theta \) are updated using the loss evaluated with \( \left\{ D_{T,\text{test}} \right\} \). Note that during meta-optimization (2), the gradient is computed with respect to initial parameters \( \theta \) but the test loss is computed with respect to task-specific parameters \( \tilde{\theta}_T \).

\[
\tilde{\theta}_T \leftarrow \theta - \alpha \nabla_\theta L_T(\theta, D_{T,\text{train}}) \tag{1}
\]

\[
\theta \leftarrow \theta - \beta \nabla_\theta \left( \sum_{T \sim p(T)} L_T(\tilde{\theta}_T, D_{T,\text{test}}) \right) \tag{2}
\]

where \( \alpha > 0 \) and \( \beta > 0 \) are learning rates and the summation in (2) is computed using minibatches of tasks sampled from \( p(T) \).

Intuitively, a well-learned initial parameter \( \theta \) is close to some local optimum for every task \( T \sim p(T) \). Furthermore, the update (1) is sensitive to task identity in the sense that \( \tilde{\theta}_{T_1} \) and \( \tilde{\theta}_{T_2} \) have different behaviors for different tasks \( T_1, T_2 \sim p(T) \).

Recent work has shown that gradient-based optimization is a universal learning algorithm (Finn & Levine, 2017), 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 (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.
3. Meta-Learning Models

We present our two models in this section: 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 task-specific parameters \( \tilde{\theta}_W, \tau \) 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

We consider a model \( f_\theta(\cdot) \) with parameters \( \theta \). This model consists of \( L \) cells, where each cell is parameterized* as \( TW \):

\[
\begin{align*}
\theta & = \left\{ W^1, \ldots, W^L, T^1, \ldots, T^L \right\},
\end{align*}
\]

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

Given a task \( T \), each \( W \) is adjusted with the gradient update

\[
\begin{align*}
\tilde{W}_T & \leftarrow W - \alpha \nabla_W L_T (\tilde{\theta}_W, \theta_T, D_{T,train}).
\end{align*}
\]

Again, \( \tilde{\theta}_W, \tau \) is defined as \( \{ \tilde{W}^1, \ldots, \tilde{W}^L \} \). Using the task-specific learner \( \tilde{\theta}_W, \tau \), the meta-learner improves itself with the gradient update

\[
\begin{align*}
\theta & \leftarrow \theta - \beta \nabla_\theta \sum_{T \sim p(T)} L_T \left( \tilde{\theta}_W, \theta_T, D_{T,train} \right).
\end{align*}
\]

*For convolutional cells, \( W \) is a convolutional layer with some size and stride and and \( T \) is a \( 1 \times 1 \) convolution that doesn’t change the number of channels.

\begin{algorithm}[H]
\caption{Transformation Networks (T-net)}
\begin{algorithmic}[1]
\Require \( p(T) \)
\Require \( \alpha, \beta \)
\State randomly initialize \( \theta \)
\While{not done}
\State Sample batch of tasks \( T_i \sim p(T) \)
\For{all \( T_j \)}
\For{\( i = 1, \ldots, L \)}
\State Compute \( W_{T_j} \) according to (4)
\EndFor
\State \( \tilde{\theta}_{W,T_j} = \{ \tilde{W}^1_{T_j}, \ldots, \tilde{W}^L_{T_j} \} \)
\EndFor
\State \( \theta \leftarrow \theta - \beta \nabla_\theta \sum_{T \sim p(T)} L_T (\tilde{\theta}_W, \theta_T, D_{T,train}) \)
\EndWhile
\end{algorithmic}
\end{algorithm}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{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.}
\end{figure}
We approximately differentiate through the Bernoulli sampling of masks using the Gumbel-Softmax estimator (Jang et al., 2017; Maddison et al., 2017):

\[
g_1, g_2 \sim \text{Gumbel}(0, 1),
\]

\[
m_j^\top \leftarrow \frac{\exp \left( \frac{g_j + g_2}{\alpha} \right)}{\exp \left( \frac{g_j + g_1}{\alpha} \right) + \exp \left( \frac{g_2}{\alpha} \right)} 1^\top,
\]

where \( \alpha \) is a temperature hyperparameter. This reparameterization allows us to directly backpropagate through the mask. At the limit of \( \alpha \to 0 \), (11) follows the behavior of (9).

As in T-nets, we denote the collection of altered weights as \( \theta_{W,T} = \{W_1^T, \ldots, W_L^T\} \). The meta-learner learns all parameters \( \theta \):

\[
\theta = \left\{ \begin{array}{c}
W_1^T, \ldots, W_L^T, \\
\theta_\omega, \\
\theta_T, \\
\theta_D, \\
\theta_\zeta,
\end{array} \right. \tag{12}
\]

As in a T-net, the meta-learner performs stochastic gradient descent on \( \mathcal{L}_T \left( \theta_{W,T}, \theta_T, \theta_\zeta, D_{T,test} \right) \):

\[
\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_T \mathcal{L}_T \left( \theta_{W,T}, \theta_T, \theta_\zeta, D_{T,test} \right).
\]

The full algorithm is shown in Algorithm 2.

We emphasize that the binary mask used for task-specific learning (M) depends on meta-learned parameter weights (\( \zeta \)). Since the meta-learner optimizes the loss in a task after a gradient step (8), the matrix M gets assigned a high probability of having value 1 for weights that are meant to encode task-specific information. Furthermore, since we update M along with model parameters 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.

### 4. Analysis

In this section, 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 \( W \) encodes a learned curvature matrix. By using such an analysis to reason about a whole network consisting of several cells, we are implicitly 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 (Heskes, 2000; Martens & Grosse, 2015; Desjardins et al., 2015) 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 \( y \) is given by

\[
y = TWx = Ax, \quad \text{(14)}
\]

where \( A = TW \) and \( x \) is the input to the cell. We omit superscripts throughout this section.

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

\[
y^{\text{new}} = (A - \alpha \nabla_A L_T)x = y - \alpha \nabla_A L_Tx. \quad \text{(15)}
\]

The update of a T-net (4) results in the following new value of \( y \):

\[
y^{\text{new}} = T(T^{-1}A - \alpha \nabla_{T^{-1}A} L_T)x = y - \alpha (TT^\top) \nabla_A L_Tx, \quad \text{(16)}
\]

where \( T \) is determined by the meta-learner. Thus, in a T-net, the incremental change of \( y \) is proportional to the negative of the gradient \( (TT^\top) \nabla_A L_T \), while the standard feedforward net resorts to a step proportional to the negative of \( \nabla_A L_T \). 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 \( T \). This metric \( (TT^\top)^{-1} \) 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 (8) means from the viewpoint of \( y = TWx = Ax \).

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

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

**Proof.** See Appendix B.

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

In addition to learning a subspace that we project gradients onto \( (U) \), 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 (16):

\[
y^{\text{new}} = T((T^{-1}A - \alpha M \odot \nabla_{T^{-1}A} L_T)x) = y - \alpha TM(T^{-1} \nabla_A L_T)x = y - \alpha (T \odot M_T)(TT^\top) \nabla_A L_Tx. \quad \text{(17)}
\]

Where \( M_T \) is an \( m \times m \) matrix which has the same columns as \( M \). Let’s denote \( T_M = T_M \odot T^\top \). We see that the update of a task-specific learner in an MT-net performs the update \( T_M^\top T_M \nabla_A L_T \). Note that \( T_M^\top T_M \) is an \( n \times n \) matrix that only has nonzero elements in rows and columns where \( m \) is 1. By setting appropriate \( \zeta \), we can view \( T_M^\top T_M \) as a full-rank \( d \times d \) metric tensor.

This observation can be formally stated as:

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

**Proof.** See Appendix B.

Therefore, not only can MT-nets project gradients of task-specific learners onto a subspace of the pre-activation \( (y) \) 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 (8) is gradient descent in this low-dimensional embedding, so the meta-objective shown in (13) is minimized when gradient descent in this embedding requires few steps to converge and is sensitive to task identity.
5. Related Work

A successful line of research in few-shot learning uses feed-forward neural networks as learners. These approaches learn update rules (Ravi & Larochelle, 2017; Li & Malik, 2016; Andrychowicz et al., 2016) or directly generate weights (Ha et al., 2016). A related research direction is to learn initial parameters (Finn et al., 2017) while fixing the learning rule to gradient descent, or additionally learning learning rates for each weight (Li et al., 2017). (Grant et al., 2018) interprets such gradient-based meta-learning as hierarchical bayesian inference, and (Finn & Levine, 2017) states that such methods are expressive enough to approximate any learning algorithm.

Our work is closely related to this line of research. Unlike previous work, MT-nets learn how many degrees of freedom the task-specific learner should have at meta-test time. Additionally, while MT-nets learn update rules, these update rules are directly embedded in the network itself instead of being stored in a separate model.

Distance metric learning (Xing et al., 2003; Weinberger et al., 2006) methods learn a distance function between datapoints. Similarly, MT-nets learn a full metric matrix. 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 (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017), 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. Our method applies to a larger class of problems including regression and reinforcement learning, since all MT-nets require is a differentiable loss function.

Another line of research in few-shot learning is to use a recurrent neural network (RNN) as a learner (Santoro et al., 2016; Munkhdalai & Yu, 2017). 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 (Mishra et al., 2018) is also closely related to this line of research.

6. Experiments

We performed experiments to answer:

- Do our novel components (TW, M etc) improve meta-learning performance? (6.1)
- Is applying a mask M row-wise actually better than applying one parameter-wise? (6.1)
- To what degree does 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 (Finn et al., 2017), 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 (Finn et al., 2017; Li et al., 2017). The details of our regression task are the same as (Li et al., 2017). Each individual task is to regress from the input x to the output y of a sine function

\[ y(x) = A \sin(wx + b) \]  

(18)

For each task, A, w, b are sampled uniformly from \([0.1, 5.0], [0.8, 1.2], [0, \pi]\), 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 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 (Kingma & Ba, 2015) as our meta-optimizer with a learning rate of \(\beta = 10^{-3}\). Task-specific learners used step size \(\alpha = 10^{-2}\). We initialize all \(\zeta\) to 0, all T as identity matrices, and all 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 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 \( \alpha \). This indicates that as shown in section 4.2, the matrix \( T \) is capable of warping the parameter space to recover from suboptimal step size \( \alpha \).

### 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 \( n \) \( \left( \sum_{i=0}^{n} c_i x^i \right) \), we uniformly sampled \( c_0, \ldots, c_n \) from \([-1, 1]\). We used the same network architecture and hyperparameters as in Section 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 \( \frac{\sum_{i=0}^{n} |c_i|}{\sum_{i=0}^{n} \sum_{j=0}^{n} c_i c_j} \) over all logits \( \zeta \).

We show results in Figure 4, and additional results in Appendix C. 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 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 (Lake et al., 2015) and MinilImagenet (Ravi & Larochelle, 2017) datasets. We used the minilma-
Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace

<table>
<thead>
<tr>
<th>Models</th>
<th>5-way 1-shot acc. (%)</th>
<th>20-way 1-shot acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching Networks</td>
<td>98.1</td>
<td>93.8</td>
</tr>
<tr>
<td>Prototypical Networks</td>
<td>97.4</td>
<td>92.0</td>
</tr>
<tr>
<td>mAP-SSVM</td>
<td>98.6</td>
<td>95.4</td>
</tr>
<tr>
<td>MAML</td>
<td>98.7 ± 0.4</td>
<td>95.8 ± 0.3</td>
</tr>
<tr>
<td>Meta-SGD</td>
<td>99.53 ± 0.26</td>
<td>95.93 ± 0.38</td>
</tr>
<tr>
<td>T-net (ours)</td>
<td>99.4 ± 0.3</td>
<td>96.1 ± 0.3</td>
</tr>
<tr>
<td>MT-net (ours)</td>
<td>99.5 ± 0.3</td>
<td>96.2 ± 0.4</td>
</tr>
</tbody>
</table>

Table 3. Few-shot classification accuracy on (top) held-out Omniglot characters and (bottom) test split of MiniImagenet. ± represents 95% confidence intervals.

1 Reported by (Ravi & Larochelle, 2017).
2 Reported results for 5-way 1-shot.

Our CNN model uses the same architecture as (Finn et al., 2017). The model has 4 modules: each has $3 \times 3$ convolutions and 64 filters, followed by batch normalization (Ioffe & Szegedy, 2015). As in (Finn et al., 2017), we used 32 filters per layer in miniImagenet. Convolutions have stride $2 \times 2$ on Omniglot, and $2 \times 2$ 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 (Mishra et al., 2018; Munkhdalai & Yu, 2017; Sung et al., 2017) 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 (Vinyals et al., 2016). Not controlling for network expressivity, the highest reported accuracy so far on 5-way 1-shot miniImagenet classification is 57.02 (Sung et al., 2017).

7. 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 we think MT-nets are a gradient-based meta-learning method, our analysis has shown that it has some interesting commonalities with optimizer learning methods such as (Ravi & Larochelle, 2017). 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.
Acknowledgements

SC was supported by Samsung DS Software Center, Samsung Research, and Naver.

References


Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace


