ATTENTIVE WEIGHTS GENERATION FOR FEW SHOT LEARNING VIA INFORMATION MAXIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Few shot image classification aims at learning a classifier from limited labeled data. Generating the classification weights has been applied in many meta-learning approaches for few shot image classification due to its simplicity and effectiveness. However, we argue that it is difficult to generate the exact and universal classification weights for all the diverse query samples from very few training samples. In this work, we introduce **A**ttentive **W**eights **G**eneration for few shot learning via **I**nformation **M**aximization (AWGIM), which addresses current issues by two novel contributions. i) AWGIM generates different classification weights for different query samples by letting each of query samples attends to the whole support set. ii) To guarantee the generated weights adaptive to different query sample, we re-formulate the problem to maximize the lower bound of mutual information between generated weights and query as well as support data. As far as we can see, this is the first attempt to unify information maximization into few shot learning. Both two contributions are proved to be effective in the extensive experiments and we show that AWGIM is able to achieve state-of-the-art performance on benchmark datasets.

1 INTRODUCTION

While deep learning methods achieve great success in domains such as computer vision (He et al., 2016), natural language processing (Devlin et al., 2018), reinforcement learning (Silver et al., 2018), their hunger for large amount of labeled data limits the application scenarios where only a few data are available for training. Humans, in contrast, are able to learn from limited data, which is desirable for deep learning methods. Few shot learning is thus proposed to enable deep models to learn from very few samples (Fei-Fei et al., 2006).

Meta learning is by far the most popular and promising approach for few shot problems (Vinyals et al., 2016; Finn et al., 2017; Snell et al., 2017; Ravi & Larochelle, 2016; Rusu et al., 2019). In meta learning approaches, the model extracts high level knowledge across different tasks so that it can adapt itself quickly to a new-coming task (Schmidhuber, 1987; Andrychowicz et al., 2016). There are several kinds of meta learning methods for few shot learning, such as gradient-based (Finn et al., 2017; Ravi & Larochelle, 2016) and metric-based (Snell et al., 2017; Sung et al., 2018). Weights generation, among these different methods, has shown effectiveness with simple formulation (Qi et al., 2018; Qiao et al., 2018; Gidaris & Komodakis, 2018; 2019). In general, weights generation methods learn to generate the classification weights for different tasks conditioned on the limited labeled data. However, fixed classification weights for different query samples within one task might be sub-optimal, due to the few shot challenge.

We introduce **A**ttentive **W**eights **G**eneration for few shot learning via **I**nformation **M**aximization (AWGIM) in this work to address these limitations. In AWGIM, the classification weights are generated for each query sample specifically. This is done by two encoding paths where the query sample attends to the task context. However, we show in experiments that simple cross attention between query samples and support set fails to guarantee classification weights fitted to diverse query data since the query-specific information is lost during weights generation. Therefore, we propose to maximize the lower bound of mutual information between generated weights and query, support data. As far as we know, AWGIM is the first work introducing Variational Information Maximization in few shot learning. The induced computational overhead is minimal due to the

nature of few shot problems. Furthermore, by maximizing the lower bound of mutual information, AWGIM gets rid of inner update without compromising performance.

AWGIM is evaluated on two benchmark datasets and shows state-of-the-art performance. We also conducted detailed analysis to validate the contribution of each component in AWGIM.

2 Related Works

2.1 Few shot learning

Learning from few labeled training data has received growing attentions recently. Most successful existing methods apply meta learning to solve this problem and can be divided into several categories. In the gradient-based approaches, an optimal initialization for all tasks is learned (Finn et al., 2017). Ravi & Larochelle (2016) learned a meta-learner LSTM directly to optimize the given few-shot classification task. Sun et al. (2019) learned the transformation for activations of each layer by gradients to better suit the current task.

In the metric-based methods, a similarity metric between query and support samples is learned. (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017; Sung et al., 2018; Li et al., 2019a). Spatial information or local image descriptors are also considered in some works to compute richer similarities (Lifchitz et al., 2019; Li et al., 2019b; Wertheimer & Hariharan, 2019).

Generating the classification weights directly has been explored by some works. Gidaris & Komodakis (2018) generated classification weights as linear combinations of weights for base and novel classes. Similarly, Qiao et al. (2018) and Qi et al. (2018) both generated the classification weights from activations of a trained feature extractor. Graph neural network denoising auto-encoders are used in (Gidaris & Komodakis, 2019). Munkhdalai & Yu (2017) proposed to generate "fast weights" from the loss gradient for each task. All these methods do not consider generating different weights for different query examples, nor maximizing the mutual information.

There are some other methods for few-shot classification. Generative models are used to generate or hallucinate more data in (Zhang et al., 2018; Wang et al., 2018; Chen et al., 2019). Bertinetto et al. (2019) and Lee et al. (2019) used the closed-form solutions directly for few shot classification. Liu et al. (2019) integrated label propagation on a transductive graph to predict the query class label.

2.2 Attention

Attention mechanism shows great success in computer vision (Xu et al., 2015; Parmar et al., 2018) and natural language processing (Bahdanau et al., 2015; Vaswani et al., 2017). It is effective in modeling the interaction between queries and key-value pairs from certain context. Based on the fact that keys and queries point to the same entities or not, people refer to attention as *self attention* or *cross attention*. In this work, we use both types of attention to encode the task and query-task information. The work most similar to ours is Attentive Neural Processes (Kim et al., 2019), which also employs self and cross attention. However, we are using attention for few-shot image classification via maximizing the mutual information. In stark contrast, Kim et al. (2019) worked on regression from the perspective of a stochastic process and the variational objective is optimized.

2.3 Mutual Information

Given two random variables x and y, mutual information $I(\mathrm{x};\mathrm{y})$ measures the decrease of uncertainty in one random variable when another is known. It is defined as the Kullback-Leibler divergence between joint distribution $p(\mathrm{x},\mathrm{y})$ and product of marginal distributions $p(\mathrm{x}) \otimes p(\mathrm{y})$,

$$I(\mathrm{x};\mathrm{y}) = D_{\mathrm{KL}}(p(\mathrm{x},\mathrm{y}) \| p(\mathrm{x}) \otimes p(\mathrm{y})). \tag{1}$$

When x and y are independent, $p(\mathrm{x},\mathrm{y}) = p(\mathrm{x}) \otimes p(\mathrm{y})$ so that $I(\mathrm{x},\mathrm{y}) = 0$, indicating that knowing x does not reveal any information about y. When y is a deterministic function of x, $I(\mathrm{x},\mathrm{y})$ achieves its maximum value. Mutual information has been widely applied in applications such as Generative Adversarial Networks(Chen et al., 2016), self-supervised learning(Hjelm et al., 2019), visual question generation Krishna et al. (2019) and so on.

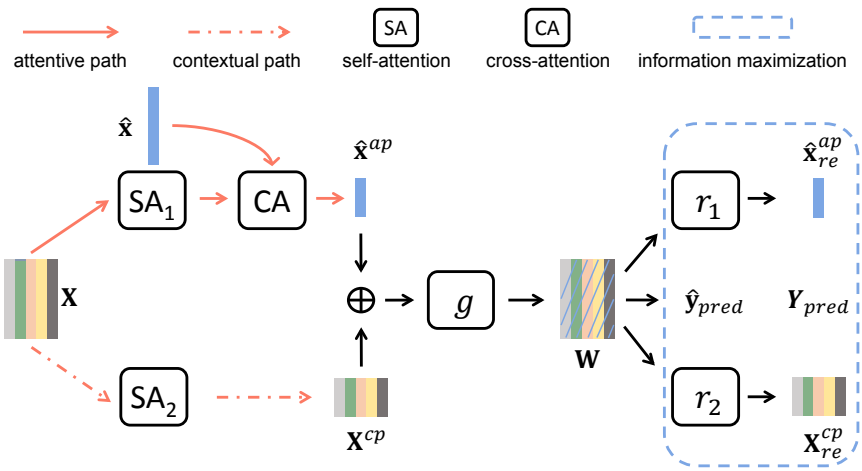

Figure 1: The overview of our proposed AWGIM. The input task is 5-way 1-shot with $\mathbf{X}$ as support set and $\hat{\mathbf{x}}$ as one query example. Different colors of the data in support set indicate different categories. The encoding process in contextual path produces context-aware support representations $\mathbf{X}^{cp}$. Similarly, the attentive path enables the query sample $\hat{\mathbf{x}}$ to be equipped with task knowledge. Both paths are achieved by attention mechanism. $\hat{\mathbf{x}}^{ap}$ is repeated to concatenate with $\mathbf{X}^{cp}$. The weight generator g takes these concatenated representations as input to generate classification weights $\mathbf{W}$ specific for $\hat{\mathbf{x}}$, denoted by the colorful matrix with slash. It can be used to predict the class label for $\hat{\mathbf{x}}$ and $\mathbf{X}$. $\mathbf{W}$ is also used to reconstruct the inputs of the generator g by two networks r_1 and r_2. In this way, the lower bound of mutual information is maximized and g is forced to generate classification weights sensitive to different query samples.

3 Proposed Method

In this section, we provide the problem formulation first. Then the proposed model is described in Sec. 3.3. The objective function, which maximizes the mutual information between certain variables, and theoretical analysis are provided in Sec. 3.4.

3.1 Problem Formulation

Following many popular meta-learning methods for few shot classification, we formulate the problem under episodic training paradigm (Vinyals et al., 2016; Finn et al., 2017). One N-way K-shot task sampled from an unknown task distribution $P(\mathcal{T})$ includes support set and query set:

$$\mathcal{T} = (\mathcal{S}, \mathcal{Q}), \tag{2}$$

where $\mathcal{S} = \{(\mathbf{x}^{c_n;k}, \mathbf{y}^{c_n;k})|k = 1, ..., K; n = 1, ..., N\}$, $\mathcal{Q} = \{(\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_{|\mathcal{Q}|})\}$. Support set $\mathcal{S}$ contains NK labeled samples. Query set $\mathcal{Q}$ includes $\hat{\mathbf{x}}$ and we need to predict label $\hat{\mathbf{y}}$ for $\hat{\mathbf{x}}$ based on $\mathcal{S}$. During meta-training, the meta-loss is estimated on $\mathcal{Q}$ to optimize the model. During meta-testing, the performance of meta-learning method is evaluated on $\mathcal{Q}$, provided the labeled $\mathcal{S}$. The classes used in meta-training and meta-testing are disjoint so that the meta-learned model needs to learn the knowledge transferable across tasks and adapt itself quickly to novel tasks.

Our proposed approach follows the general framework to generate the classification weights (Qi et al., 2018; Qiao et al., 2018; Rusu et al., 2019; Gidaris & Komodakis, 2018; 2019). In this framework, there is a feature extractor to output image feature embeddings. The meta-learner needs to generate the classification weights for different tasks.

3.2 Latent Embedding Optimization

Latent Embedding Optimization (LEO) (Rusu et al., 2019) is one of the weights generation methods that is most related to our work. In LEO, the latent code $\boldsymbol{z}$ is generated by h conditioned on support set $\mathcal{S}$, described as $\boldsymbol{z} = h(\mathcal{S})$. h is instantiated as relation networks (Santoro et al., 2017). Classification weights $\boldsymbol{w}$ can be decoded from $\boldsymbol{z}$ with l, $\boldsymbol{w} = l(\boldsymbol{z})$. In the inner loop, we use $\boldsymbol{w}$ to compute the loss (usually cross entropy) on the support set and then update $\boldsymbol{z}$:

$$\boldsymbol{z}' = \boldsymbol{z} - \eta \nabla_{\boldsymbol{z}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{w}), \tag{3}$$

where $\mathcal{L}_{\mathcal{S}}$ indicates that the loss is evaluated on $\mathcal{S}$ only. The updated latent code $\boldsymbol{z}'$ is used to decode new classification weights $\boldsymbol{w}'$ with generating function l. $\boldsymbol{w}'$ is adopted in the outer loop for query set $\mathcal{Q}$ and the objective function of LEO then can be written as

$$\min_{\theta} \mathcal{L}_{\mathcal{Q}}(\boldsymbol{w}'). \tag{4}$$

Here θ stands for the parameters of h and l and we omit the regularization terms for clarity. LEO avoids updating high-dimensional $\boldsymbol{w}$ in the inner loop by learning a lower-dimensional latent space, from which sampled $\boldsymbol{z}$ can be used to generate $\boldsymbol{w}$. The most significant difference between LEO and AWGIM is that we do not need inner updates to adapt the model. Instead, AWGIM is a feedforward network trained to maximize the mutual information so that it fits to different tasks well. On the other hand, AWGIM learns to generate optimal classification weights for each query sample while LEO generates fixed weights conditioned on the support set within one task. In Section 3.4 we will show LEO can be casted as a special case of AWGIM under certain conditions.

3.3 Attentive Weights Generation

The framework of our proposed method is shown in Figure 1. Assume that we have a feature extractor, which can be a simple 4-layer Convnet or a deeper Resnet. All the images included in the sampled task $\mathcal{T}$ are processed by this feature extractor and represented as d-dimensional vectors afterwards, i.e., $\mathbf{x}^{c_n;k}, \hat{\mathbf{x}} \in \mathbb{R}^d$. There are two paths to encode the task context and the individual query sample respectively, which are called contextual path and attentive path. The outputs of both paths are concatenated together as input to the generator for classification weights. Generated classification weights are used to not only predict the label of $\hat{\mathbf{x}}$, but also maximize the lower bound of mutual information between itself and other variables, which will be discussed in the following section 3.4.

3.3.1 contextual and attentive paths

The encoding process includes two paths, namely the contextual path and attentive path. The contextual path aims at learning representations for only the support set with a multi-head self-attention network f_{sa}^{cp} (Vaswani et al., 2017). The outputs of contextual path $\mathbf{X}^{cp} \in \mathbb{R}^{NK \times d_h}$ [1] thus contain richer information about the task and can be used later for weights generation.

Existing weights generation methods generate the classification weights conditioned on the support set only, which is equivalent to using contextual path. However, the classification weights generated in this way might be sub-optimal. This is because estimating the exact and universal classification weights from very few labeled data in the support set is difficult and sometimes impossible. The generated weights are usually in lack of adaptation to different query samples. We address this issue by introducing attentive path, where the individual query example attends to the task context and then is used to generate the classification weights. Therefore, the classification weights are adaptive to different query samples and aware of the task context as well.

In the attentive path, a new multi-head self-attention network f_{sa}^{ap} on the support set is employed to encode the global task information. f_{sa}^{ap} is different from f_{sa}^{cp} in contextual path because the self-attention network in contextual path emphasizes on generating the classification weights. On the contrary, outputs of self-attention here plays the role of providing the $Value$ context for different query samples to attend in the following cross attention. Sharing the same self-attention networks might limit the expressiveness of learned representations in both paths. The cross attention network f_{ca}^{ap} applied on each query sample and task-aware support set is followed to produce $\hat{\mathbf{X}}^{ap} \in \mathbb{R}^{|\mathcal{Q}| \times d_h}$.

We use multi-head attention with h heads in both paths. In one attention block, we produce h different sets of queries, keys and values. Multi-head attention is claimed to be able to learn more comprehensive and expressive representations from h different subspaces (Vaswani et al., 2017; Voita et al., 2019). More details of these two paths can be found in A.2.

3.3.2 weights generator

We replicate $\mathbf{X}^{cp} \in \mathbb{R}^{NK \times d_h}$ and $\hat{\mathbf{X}}^{ap} \in \mathbb{R}^{|\mathcal{Q}| \times d_h}$ for $|\mathcal{Q}|$ and NK times respectively and reshape them afterwards. Then we have $\mathbf{X}^{cp} \in \mathbb{R}^{|\mathcal{Q}| \times NK \times d_h}$ and $\hat{\mathbf{X}}^{ap} \in \mathbb{R}^{|\mathcal{Q}| \times NK \times d_h}$. These two tensors

[1] $d_h < d$ is the hidden dimension. We use matrix form here to be consistent with the description in 3.3.2.

are concatenated to become $\mathbf{X}^{cp\oplus ap} \in \mathbb{R}^{|\mathcal{Q}|\times NK\times 2d_h}$. $\mathbf{X}^{cp\oplus ap}$ can be interpreted that each query sample has its own latent representations for support set to generate specific classification weights, which are both aware of the task-context and adaptive to individual query sample.

$\mathbf{X}^{cp\oplus ap}$is decoded by the weights generator $g : \mathbb{R}^{2d_h} \rightarrow \mathbb{R}^{2d}$. We assume that the classification weights follow Gaussian distribution with diagonal covariance. g outputs the distribution parameters and we sample the weights from learned distribution during meta-training. The sampled classification weights are represented as $\mathbf{W} \in \mathbb{R}^{|\mathcal{Q}|\times NK\times d}$. To reduce complexity, we compute the mean value on K classification weights for each class to have $\mathbf{W}^{final} \in \mathbb{R}^{|\mathcal{Q}|\times N\times d}$. Therefore, ith query sample has its specific classification weight matrix $\mathbf{W}^{final}_{i,:,:} \in \mathbb{R}^{N\times d}$. The prediction for query data can be computed by $\hat{\mathbf{X}}\mathbf{W}^{final\mathbf{T}}$. The support data $\mathbf{X}$ is replicated for $|\mathcal{Q}|$ times and reshaped as $\mathbf{X}_s \in \mathbb{R}^{|\mathcal{Q}|\times NK\times d}$. So the prediction for support data can also be computed as $\mathbf{X}_s\mathbf{W}^{final\mathbf{T}}$.

Besides the weights generator g, we have another two decoders $r_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{d_h}$ and $r_2 : \mathbb{R}^d \rightarrow \mathbb{R}^{d_h}$. They both take the generated weights $\mathbf{W}$ as inputs and learn to reconstruct $\mathbf{X}^{cp}$ and $\hat{\mathbf{X}}^{ap}$ respectively. The outputs of r_1 and r_2 are denoted as $\mathbf{X}^{cp}_{re}, \hat{\mathbf{X}}^{ap}_{re} \in \mathbb{R}^{|\mathcal{Q}|\times NK\times d_h}$. The reason we are using reconstruction as auxiliary tasks will be discussed in following Sec. 3.4.

3.4 Information Maximization

In this section, we perform the analysis for one query sample without loss of generality. The subscripts for classification weights are omitted for clarity. In general, we use $(\mathbf{x}, \mathbf{y})$ and $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ to represent support and query samples respectively.

Since the classification weights $\mathbf{w}$ generated from g are encoded with attentive path and contextual path, it is expected that we can directly have the query-specific weights. However, we show in the experiments that simply doing this does not outperform a weight generator conditioned only on the $\mathcal{S}$ significantly, which implies that the generated classification weights from two paths are not sensitive to different query samples. In other words, the information from attentive path is not kept well during the weights generation.

To address this limitation, we propose to maximize the mutual information between generated weights $\mathbf{w}$ and support as well as query data. The objective function can be described as

$$\max I((\hat{\mathbf{x}}, \hat{\mathbf{y}}); \mathbf{w}) + \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{S}} I((\mathbf{x}, \mathbf{y}); \mathbf{w}) \tag{5}$$

According to the chain rule of mutual information, we have

$$I((\hat{\mathbf{x}}, \hat{\mathbf{y}}); \mathbf{w}) = I(\hat{\mathbf{x}}; \mathbf{w}) + I(\hat{\mathbf{y}}; \mathbf{w}|\hat{\mathbf{x}}). \tag{6}$$

Equation 6 stands for both terms in 5. So the objective function can be written as

$$\max I(\hat{\mathbf{x}}; \mathbf{w}) + I(\hat{\mathbf{y}}; \mathbf{w}|\hat{\mathbf{x}}) + \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{S}} [I(\mathbf{x}; \mathbf{w}) + I(\mathbf{y}; \mathbf{w}|\mathbf{x})]. \tag{7}$$

Directly computing the mutual information in Equation 7 is intractable since the true posteriori distributions like $p(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w})$, $p(\hat{\mathbf{x}}|\mathbf{w})$ are still unknown. Therefore, we use Variational Information Maximization (Barber & Agakov, 2003; Chen et al., 2016) to compute the lower bound of Equation 5. We use $p_\theta(\hat{\mathbf{x}}|\mathbf{w})$ to approximate the true posteriori distribution, where θ represents the model parameters. As a result, we have

$$\begin{aligned}
I(\hat{\mathbf{x}}; \mathbf{w}) &= H(\hat{\mathbf{x}}) - H(\hat{\mathbf{x}}|\mathbf{w}) && (8)\\
&= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\hat{\mathbf{x}},\mathcal{S})}[\mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w})}[\log p(\hat{\mathbf{x}}|\mathbf{w})]] && (9)\\
&= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\hat{\mathbf{x}},\mathcal{S})}[D_{\mathrm{KL}}(p(\hat{\mathbf{x}}|\mathbf{w})\|p_\theta(\hat{\mathbf{x}}|\mathbf{w})) + \mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w})}[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w})]] && (10)\\
&\geq H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\hat{\mathbf{x}},\mathcal{S})}[\mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w})}[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w})]] && (11)
\end{aligned}$$

$H(\cdot)$ is the entropy of a random variable. $H(\hat{\mathbf{x}})$ is a constant value for given data. We can maximize this lower bound as the proxy for the true mutual information.

Similar to $I(\hat{\mathbf{x}}; \mathbf{w})$,

$$I(\hat{\mathbf{y}}; \mathbf{w}|\hat{\mathbf{x}}) \geq H(\hat{\mathbf{y}}|\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\hat{\mathbf{x}},\mathcal{S})}[\mathbb{E}_{\hat{\mathbf{y}}\sim p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\mathbf{w})}[\log p_\theta(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w})]], \tag{12}$$

$$\sum_{(\mathbf{x},\mathrm{y})\in\mathcal{S}} I((\mathbf{x},\mathrm{y});\mathbf{w}) \geq \sum_{(\mathbf{x},\mathrm{y})\in\mathcal{S}} H((\mathbf{x},\mathrm{y}))+\mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\hat{\mathbf{x}},\mathcal{S})}[\mathbb{E}_{(\mathbf{x},\mathrm{y})\sim p((\mathbf{x},\mathrm{y})|\mathbf{w})}[\log p_\theta(\mathbf{x}|\mathbf{w})+\log p_\theta(\mathrm{y}|\mathbf{x},\mathbf{w})]]. \tag{13}$$

$p_\theta(\hat{\mathbf{x}}|\mathbf{w})$, $p_\theta(\mathbf{x},\mathrm{y}|\mathbf{w})$ are used to approximate the true posteriori distribution $p(\hat{\mathbf{x}}|\mathbf{w})$ and $p(\mathbf{x},\mathrm{y}|\mathbf{w})$.

Put the lower bounds back into Equation 7. Omit the constant entropy terms and the expectation subscripts for clarity, we have the new objective function as

$$\max_\theta \mathbb{E}[\log p_\theta(\hat{\mathbf{y}}|\hat{\mathbf{x}},\mathbf{w}) + \log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{w}) + \log p_\theta(\mathbf{x}|\mathbf{w}) + \log p_\theta(\hat{\mathbf{x}}|\mathbf{w})]. \tag{14}$$

The first two terms are maximizing the log likelihood of label for both support and query data with respective to the network parameters, given the generated classification weights. This is equivalent to minimizing the cross entropy between prediction and ground-truth. We assume that $p_\theta(\hat{\mathbf{x}}|\mathbf{w})$ and $p_\theta(\mathbf{x}|\mathbf{w})$ are Gaussian distributions. r_1 and r_2 are used to approximate the mean of these two Gaussian distributions. Therefore maximizing the log likelihood is equivalent to reconstruct $\mathbf{x}^{cp}$ and $\hat{\mathbf{x}}^{ap}$ with $L2$ loss. Thus the loss function to train the network can be written as

$$L = \mathrm{CE}(\hat{\mathbf{y}}_{pred},\hat{\mathbf{y}}) + \lambda_1 \sum_{\mathbf{y}\in\mathcal{S}} \mathrm{CE}(\mathbf{y}_{pred},\mathbf{y}) + \lambda_2 \sum_{\mathbf{x}^{cp}\in\mathcal{S}} ||\mathbf{x}^{cp} - \mathbf{x}^{cp}_{re}||_2 + \lambda_3||\hat{\mathbf{x}}^{ap} - \hat{\mathbf{x}}^{ap}_{re}||_2. \tag{15}$$

CE here stands for cross entropy. $\mathbf{x}^{cp}$ and $\hat{\mathbf{x}}^{ap}$ are the inputs to weights generator g. $\mathbf{x}^{cp}_{re} \sim p_\theta(\mathbf{x}|\mathbf{w})$ and $\hat{\mathbf{x}}^{ap}_{re} \sim p_\theta(\hat{\mathbf{x}}|\mathbf{w})$ are the reconstruction of $\mathbf{x}^{cp}$ and $\hat{\mathbf{x}}^{ap}$. Since we convert the log likelihood in Equation 14 to mean square error or cross entropy in Equation 15 to optimize, the value of each term in Equation 15 is not equal to real log likelihood and we have to decide the weightage for each one. $\lambda_1, \lambda_2, \lambda_3$ are thus hyper-parameters for trade-off of different terms. With the help of last three terms, the generated classification weights are forced to carry information about the support data and the specific query sample.

In LEO (Rusu et al., 2019), the inner update loss is computed as cross entropy on support data. If we merge the inner update into outer loop, then the loss becomes the summation of first two terms in Equation 15. However, the weight generation in LEO does not involve specific query samples, thus making reconstructing $\hat{\mathbf{x}}^{ap}$ impossible. In this sense, LEO can be regarded as a special case of our proposed method, where (1) only contextual path exits and (2) $\lambda_2 = \lambda_3 = 0$.

3.5 Complexity analysis

The encoding process in contextual path results in computational complexity $O((NK)^2)$ due to self-attention. Similarly, the computational complexity of attentive path is $O((NK)^2 + |\mathcal{Q}|(NK))$. In total, the complexity is $O((NK)^2 + |\mathcal{Q}|(NK))$. However, because of the nature of few-shot learning problem, the value of $(NK)^2$ is usually negligible. The value of $|\mathcal{Q}|$ depends on the setting and the cross attention can be implemented parallelly via matrix multiplication. Therefore, the induced computational overhead will be negligible. AWGIM avoids the inner update without compromising the performance, which furthers reduces both training and inference time significantly. The empirical evaluation is presented in A.3.4.

4 Experiments

4.1 Datasets and protocols

We conduct experiments on ***mini*ImageNet** (Vinyals et al., 2016) and ***tiered*ImageNet** (Ren et al., 2018), two commonly used benchmark datasets, to compare with other methods and analyze our model. Both datasets are subsets of ILSVRC-12 dataset (Russakovsky et al., 2015). *mini*ImageNet contains 100 randomly sampled classes with 600 images per class. We follow the train/test split in (Ravi & Larochelle, 2016), where 64 classes are used for meta-training, 16 for meta-validation and 20 for meta-testing. *tiered*ImageNet is a larger dataset compared to *mini*ImageNet. There are 608 classes and 779,165 images in total. They are selected from 34 higher level nodes in ImageNet (Deng et al., 2009) hierarchy. 351 classes from 20 high level nodes are used for meta-training, 97 from 6 nodes for meta-validation and 160 from 8 nodes for meta-testing.

We use the image features in LEO (Rusu et al., 2019) provided by the authors [2]. They trained a 28-layer Wide Residual Network (Zagoruyko & Komodakis, 2016) on the meta-training set. Each image then is represented by a 640 dimensional vector, which is used as the input to our model.

For N-way K-shot experiments, we randomly sample N classes from meta-training set and each of them contains K samples as the support set and 15 as query set. Similar to other works, we train 5-way 1-shot and 5-shot models on two dataset. During meta-testing, 600 N-way K-shot tasks are sampled from meta-testing set and the average accuracy for query set is reported with 95% confidence interval, as done in recent works (Finn et al., 2017; Snell et al., 2017; Rusu et al., 2019).

4.2 Implementation details

We use TensorFlow (Abadi et al., 2016) to implement our method and the code will be made available. $d = 640$ is the dimension of feature embeddings. d_h is set to be 128. The number of heads h in attention module is set to be 4. g, r_1 and r_2 are 2-layer MLPs with 256 hidden units. We decide $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = 0.001$ by meta-validation performance.

Table 1: Accuracy comparison with other approaches on *mini*ImageNet. The results are averaged on 600 tasks from meta-testing set with 95% confidence interval. Best results are highlighted.

Model	Feature Extractor	5-way 1-shot	5-way 5-shot
Matching Networks (Vinyals et al., 2016)	Conv-4	46.60	60.00
MAML(Finn et al., 2017)	Conv-4	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
Meta LSTM (Ravi & Larochelle, 2016)	Conv-4	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
Prototypical Nets (Snell et al., 2017)	Conv-4	$49.42 \pm 0.78\%$	$68.20 \pm 0.66\%$
Relation Nets (Sung et al., 2018)	Conv-4	$50.44 \pm 0.82\%$	$65.32 \pm 0.70\%$
SNAIL (Mishra et al., 2018)	Resnets-12	$55.71 \pm 0.99\%$	$68.88 \pm 0.92\%$
TPN (Liu et al., 2019)	Resnets-12	59.46	75.65
MTL (Sun et al., 2019)	Resnets-12	$61.20 \pm 1.80\%$	75.50 ± 0.80
Dynamic (Gidaris & Komodakis, 2018)	WRN-28-10	$60.06 \pm 0.14\%$	$76.39 \pm 0.11\%$
Prediction (Qiao et al., 2018)	WRN-28-10	$59.60 \pm 0.41\%$	$73.74 \pm 0.19\%$
DAE-GNN (Gidaris & Komodakis, 2019)	WRN-28-10	$62.96 \pm 0.15\%$	$\mathbf{78.85 \pm 0.10\%}$
LEO (Rusu et al., 2019)	WRN-28-10	$61.76 \pm 0.08\%$	$77.59 \pm 0.12\%$
AWGIM (ours)	WRN-28-10	$\mathbf{63.12 \pm 0.08\%}$	$78.40 \pm 0.11\%$

Table 2: Accuracy comparison with other approaches on *tiered*ImageNet. The results are averaged on 600 tasks from meta-testing set with 95% confidence interval. Best results are highlighted.

Model	Feature Extractor	5-way 1-shot	5-way 5-shot
MAML (Finn et al., 2017)	Conv-4	$51.67 \pm 1.81\%$	$70.30 \pm 1.75\%$
Prototypical Nets (Snell et al., 2017)	Conv-4	$53.31 \pm 0.89\%$	$72.69 \pm 0.74\%$
Relation Nets (Sung et al., 2018)	Conv-4	$54.48 \pm 0.93\%$	$71.32 \pm 0.78\%$
TPN (Liu et al., 2019)	Conv-4	$59.91 \pm 0.96\%$	$72.85 \pm 0.74\%$
MetaOptNet (Lee et al., 2019)	Resnets-12	$65.81 \pm 0.74\%$	$81.75 \pm 0.53\%$
LEO (Rusu et al., 2019)	WRN-28-10	$66.33 \pm 0.05\%$	$81.44 \pm 0.09\%$
AWGIM (ours)	WRN-28-10	$\mathbf{67.69 \pm 0.11\%}$	$\mathbf{82.82 \pm 0.13\%}$

ADAMW Loshchilov & Hutter (2017) is used to optimize the network with weight decay 1×10^{-6}. The initial learning rate is set to 0.0002 for 5-way 1-shot and 0.001 for 5-way 5-shot, which is decayed by 0.2 for every 15,000 iterations. We train the model for 50,000 iterations. Batch size is 64 for 5-way 1-shot and 32 for 5-way 5-shot. Similar to LEO (Rusu et al., 2019), we first train the model on meta-training set and choose the optimal hyper-parameters by validation results. Then we train the model on meta-training and meta-validation sets together using fixed hyper-parameters.

4.3 Comparison with other methods

We compare the performance of our approach AWGIM on two datasets with several state-of-the-art methods proposed in recent years. The results of MAML, Prototypical Nets, Relation Nets on

[2] https://github.com/deepmind/leo

*tiered*ImageNet are evaluated by Liu et al. (2019). The results of Dynamic on *mini*ImageNet with WRN-28-10 as the feature extractor is reported in (Gidaris & Komodakis, 2019). The other results are reported in the corresponding original papers. We also include the backbone network structure of the used feature extractor for reference. The results on *mini*ImageNet and *tiered*ImageNet are shown in Table 1 and 2 respectively.

The top half parts of Table 1 and 2 display the methods belonging with different meta learning categories, such as metric-based(Matching Networks, Prototypical Nets), gradient-based (MAML, MTL), graph-based (TPN). The bottom part shows the classification weights generation approaches including Dynamic, Prediction, DAE-GNN, LEO and our proposed AWGIM.

AWGIM can outperform all the methods in top parts of two table. Comparing with other classification weights generation methods in the bottom part, AWGIM still shows very competitive performance, namely the best on *tiered*ImageNet and close to the state-of-the-art on *mini*ImageNet. We note that all the classification weights generation methods are using WRN-28-10 as backbone network, which makes the comparison fair. In particular, AWGIM can outperform LEO in all settings.

4.4 ANALYSIS

Table 3: Analysis of our proposed AWGIM. In the top half, the attentive path is removed to compare with LEO. In the bottom part, ablation analysis with respective to different components is provided. We also shuffle the generated classification weights randomly to show that they are indeed optimal for different query samples.

Model	*mini*ImageNet		*tiered*ImageNet	
	5-way 1-shot	5-way 5-shot	5-way 1-shot	5-way 5-shot
LEO	61.76 %	77.59 %	66.33%	81.44 %
Generator in LEO	60.33 %	74.53 %	65.17%	78.77 %
Generator conditioned on $\mathcal{S}$ only	61.02%	74.33%	66.22%	79.66%
Generator conditioned on $\mathcal{S}$ with IM	62.04%	77.54%	66.43%	81.73%
MLP encoding, $\lambda_1 = \lambda_2 = \lambda_3 = 0$	58.95%	71.68%	63.92%	75.80%
MLP encoding	62.26%	76.91%	65.84%	79.24%
$\lambda_1 = \lambda_2 = \lambda_3 = 0$	61.61%	74.14%	65.65%	79.93%
$\lambda_1 = \lambda_2 = 0$	62.06%	74.18%	65.85%	80.42%
$\lambda_3 = 0$	62.91%	77.88%	67.27%	81.67%
$\lambda_1 = 0$	62.19%	74.21%	66.82%	80.61%
$\lambda_2 = \lambda_3 = 0$	62.12%	77.65%	66.86%	81.03%
random shuffle in class	62.87%	77.48%	67.52%	82.55%
random shuffle between classes	61.20%	77.48%	66.55%	82.53%
AWGIM (ours)	**63.12%**	**78.40%**	**67.69%**	**82.82%**

We perform detailed analysis on AWGIM, shown in Table 3. We include the results of LEO Rusu et al. (2019) for reference. "Generator in LEO" means that there is no inner update in LEO. In the upper part of the table, we first studied **the effect of attentive path**. We implemented two generators **including only the contextual path** during encoding. "Generator conditioned on $\mathcal{S}$ with IM" indicates that we add the cross entropy loss and reconstruction loss for support set. It can be observed that "Generator conditioned on $\mathcal{S}$ only" is trained with cross entropy on query set, which is similar to "Generator in LEO" without inner update. It is able to achieve similar or slightly better results than "Generator in LEO", which implies that self-attention is no worse than relation networks used in LEO to model task-context. With information maximization, our generator is able to obtain slightly better performance than LEO.

The effect of attention is investigated by replacing the attention modules with 2-layer MLPs, which is shown as "MLP encoding". More specifically, one MLP in contextual path is used for support set and another MLP in attentive path for query samples. We can see that even without attention to encode the task-contextual information, "MLP encoding" can achieve accuracy close to LEO, for the sake of information maximization. However, if we let $\lambda_1 = \lambda_2 = \lambda_3 = 0$ for MLP encoding, the performance drops significantly, which demonstrates the importance of maximizing the information.

We conducted **ablation analysis with respective to λ_1, λ_2 and λ_3** to investigate the effect of information maximization. First, λ_1, λ_2 and λ_3 are all set to be 0. In this case, the accuracy is similar to "generator conditioned on $\mathcal{S}$ only", showing that the generated classification weights are not fitted

for different query samples, even with the attentive path. It can also be observed that maximizing the mutual information between weights and support is more crucial since $\lambda_1 = \lambda_2 = 0$ degrades accuracy significantly, comparing with $\lambda_3 = 0$. We further investigate the relative importance of the classification on support as well as reconstruction. $\lambda_1 = 0$ affects the performance noticeably. We conjecture that the support label prediction is more critical for information maximization.

The classification weights are generated specifically for each query sample in AWGIM. To this point, we shuffle the classification weights between query samples within the same classes and between different classes as well to study **whether the classification weights are adapted for different query samples**. Assume there are T query samples per class in one task. $\mathbf{W}^{final} \in \mathbb{R}^{|\mathcal{Q}| \times N \times d}$ can be reshaped into $\mathbf{W}^{final} \in \mathbb{R}^{N \times T \times N \times d}$. Then we shuffle this weight tensor along the first and second axis randomly. The results are shown as "random shuffle between classes" and "random shuffle in class" in Table 3. For 5-way 1-shot experiments, the random shuffle between classes degrades the accuracy noticeably while the random shuffle in class dose not affect too much. This indicates that when the support data are very limited, the generated weights for query samples from the same class are very similar to each other while distinct for different classes. When there are more labeled data in support set, two kinds of random shuffle show very close or even the same results in 5-way 5-shot experiments, which are both worse than the original ones. This implies that the generated classification weights are more diverse and specific for each query sample in 5-way 5-shot setting. The possible reason is that larger support set provides more knowledge to estimate the optimal classification weights for each query example.

More analysis is provided in Appendix A.3.

5 Conclusion

In this work, we introduce Attentive Weights Generation via Information Maximization (AWGIM) for few shot image classification. AWGIM learns to generate optimal classification weights for each query sample within the task by two encoding paths. To guarantee this, the lower bound of mutual information between generated weights and query, support data is maximized. As far as we know, AWGIM is the first work utilizing mutual information techniques for few shot learning. The effectiveness of AWGIM is demonstrated by state-of-the-art performance on two benchmark datasets and extensive analysis.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

David Barber and Felix V Agakov. The im algorithm: a variational approach to information maximization. In *NeurIPS*, 2003.

Luca Bertinetto, Joao F Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *ICLR*, 2019.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, 2016.

Zitian Chen, Yanwei Fu, Yu-Xiong , Lin Ma, Wei Liu, and Martial Hebert. Image deformation meta-networks for one-shot learning. In *CVPR*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *TPAMI*, 2006.

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

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, 2018.

Spyros Gidaris and Nikos Komodakis. Generating classification weights with gnn denoising autoencoders for few-shot learning. In *CVPR*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *ICLR*, 2019.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *ICLR*, 2019.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, 2015.

Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Information maximizing visual question generation. In *CVPR*, 2019.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, 2019.

Hongyang Li, David Eigen, Samuel Dodge, Matthew Zeiler, and Xiaogang Wang. Finding task-relevant features for few-shot learning by category traversal. In *CVPR*, 2019a.

Wenbin Li, Lei Wang, Jinglin Xu, Jing Huo, Yang Gao, and Jiebo Luo. Revisiting local descriptor based image-to-class measure for few-shot learning. In *CVPR*, 2019b.

Yann Lifchitz, Yannis Avrithis, Sylvaine Picard, and Andrei Bursuc. Dense classification and implanting for few-shot learning. In *CVPR*, 2019.

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *ICLR*, 2019.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 2008.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *ICML*, 2017.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *ICML*, 2018.

Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *CVPR*, 2018.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *CVPR*, 2018.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2016.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. In *ICLR*, 2018.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *NeurIPS*, 2017.

Jürgen Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. *PhD thesis*, 1987.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, 2019.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *CVPR*, 2018.

Davis Wertheimer and Bharath Hariharan. Few-shot learning with localization in realistic settings. In *CVPR*, 2019.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. In *NeurIPS*, 2018.

A APPENDIX

A.1 MUTLI-HEAD ATTENTION IN AWGIM

The multi-head attention can be described as

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_H)W^O, \tag{16}$$

$$head_i(Q^i, K^i, V^i) = Attention(Q^i, K^i, V^i), \tag{17}$$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V), \tag{18}$$

$$Q^i = QW_Q^i, K^i = KW_K^i, V^i = VW_V^i, \tag{19}$$

Here Q, K, V are query, key, value matrices. W_Q^i, W_K^i, W_V^i are the weight matrices for ith head. W^O is the weight matrix for output. d_k is the dimension of keys. Original Q is added to the output of Equation 16 to stabilize the training as residual learning.

A.2 MODEL DETAILS

A.2.1 CONTEXTUAL PATH

The encoding process in contextual path is realized by a simple multi-head self-attention network on support data. First, $\mathbf{x}^{c_n;k}$ are mapped to a lower dimensional hidden space by a MLP $f_1 : \mathbb{R}^d \to \mathbb{R}^{d_h}$ to reduce the computation complexity. Then the low-dimensional representations $\mathbf{x}_{h1}^{c_n;k}$ are processed by the H-head self-attention network $f_{cp}^{sa} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$,

$$\mathbf{X}^{cp} = MultiHeadAttention(Q = \mathbf{X}_{h1}, K = \mathbf{X}_{h1}, V = \mathbf{X}_{h1}). \tag{20}$$

$\mathbf{X}_{h1} \in \mathbb{R}^{NK \times d_h}$ is the matrix where each row stands for one support sample $\mathbf{x}_{h1}^{c_n;k}$. For one N-way K-shot task, the outputs of f_{cp}^{sa} are represented by a matrix $\mathbf{X}^{cp} \in \mathbb{R}^{NK \times d_h}$.

A.2.2 ATTENTIVE PATH

The attentive path is instantiated by attention, similar to contextual path. First, a MLP $f_2 : \mathbb{R}^d \to \mathbb{R}^{d_h}$ is used to map both $\mathbf{x}^{c_n;k}$ and $\hat{\mathbf{x}}$ to $\mathbf{x}_{h2}^{c_n;k}$ and $\hat{\mathbf{x}}_{h2}$. Then we employ another H-head self-attention network $f_{ap}^{sa} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$ on $\mathbf{x}_{h2}^{c_n;k}$ to encode the global task information to each support sample,

$$\mathbf{X}^{ap} = MultiHeadAttention(Q = \mathbf{X}_{h2}, K = \mathbf{X}_{h2}, V = \mathbf{X}_{h2}). \tag{21}$$

The cross attention between query and context-aware support samples are computed as

$$\hat{\mathbf{X}}^{ap} = MultiHeadAttention(Q = \hat{\mathbf{X}}_{h2}, K = \mathbf{X}_{h2}, V = \mathbf{X}^{ap}). \tag{22}$$

Here $\hat{\mathbf{X}}^{ap} \in \mathbb{R}^{|\mathcal{Q}| \times d_h}$ is the matrix form of $\hat{\mathbf{x}}_q$, where each query sample is context-aware.

A.2.3 WEIGHT GENERATOR

Assume $\mathbf{x}^{cp \oplus ap} = \mathbf{X}_{i,j,:}^{cp \oplus ap} \in \mathbb{R}^{2d_h}$, where i, j stands for ith query sample and jth support sample. $\mathbf{x}^{cp \oplus ap}$ is decoded by the weights generator $g : \mathbb{R}^{2d_h} \to \mathbb{R}^{2d}$. We assume that the classification weights follow Gaussian distribution with diagonal covariance and we sample the weights from this distribution during meta-training, shown in Equation 23 and 24.

$$\boldsymbol{\mu}_{\mathbf{w}}, \sigma_{\mathbf{w}} = g(\mathbf{z}) \tag{23}$$

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w}}, \boldsymbol{\Sigma}_{\mathbf{w}}) \tag{24}$$

A.3 EXPERIMENTAL ANALYSIS

A.3.1 FEW SHOT REGRESSION

AWGIM can be applied to few shot regression task by slight modification. During meta-training, we set the number of classes N equal to 1 and adapt the cross entropy loss to mean square error. We use the data points (x, y) as inputs to AWGIM and generate weight as well as bias parameters for a three layer MLP with hidden dimension 40. This is consistent with few shot regression experimental setting in LEO.

The few shot regression tasks are constructed as either sinusoidal or linear regression tasks. For sinusoidal regression tasks, the amplitude range is $[0.1, 5]$, phase range $[0, 2\pi]$, frequency range $[0.5, 2.0]$. For linear regression tasks, the slope range is $[-1, 1]$, intercept range $[-5, 5]$. Input x is randomly sample from $[-5, 5]$. Gaussian noise with standard deviation 0.3 is added to y during meta-training. We show some qualitative results in Figure 2. (a) and (b) are examples that can be tackled easily. For some non-trivial cases such as (c) and (d), AWGIM produces predictions slightly mixing with another regression family, despite that overall results are still faithful.

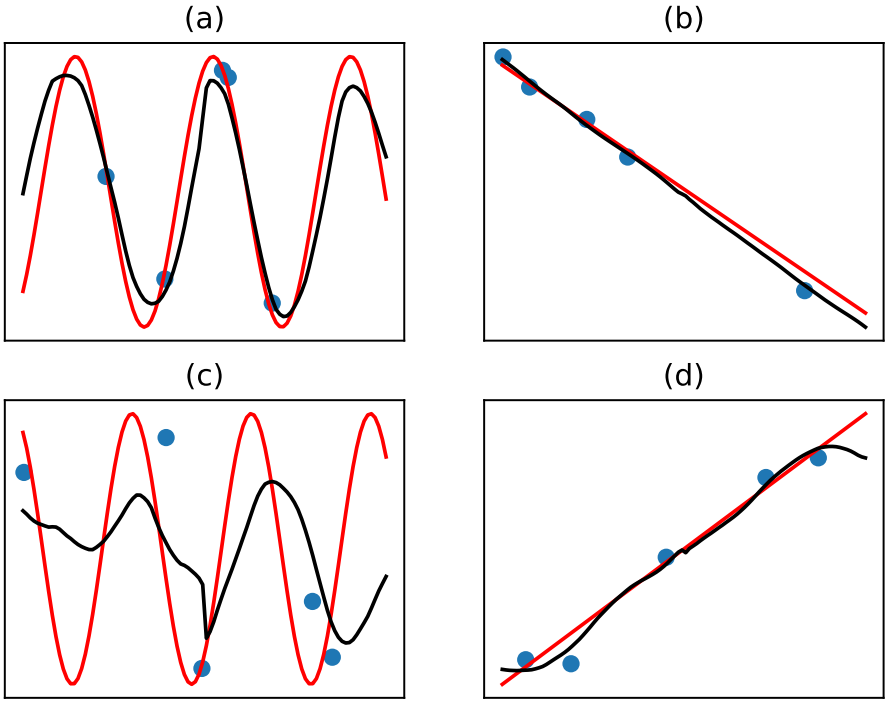

Figure 2: 5-shot regression results for a multi-modal task distribution. Regression targets are plotted in red and prediction in black. 5 training samples per task are plotted with blue solid circles.

A.3.2 EFFECT OF MULTI-HEAD ATTENTION

We replace the multi-head attention in the two paths with single-head attention and conduct the 5-way 1-shot and 5-way 5-shot experiments on *mini*ImageNet dataset. The results are shown in Table 4. We can see clearly that multi-head attention improve the performance. In particular, for 5-way 1-shot experiment, single head attention gives results close to MLP encoding, which indicates that single head attention struggles when data are extremely scarce.

Table 4: Accuracy results on *mini*ImageNet with 4 heads or single head in attention networks.

Method	5-way 1 -shot	5-way 5-shot
4 heads	63.12%	78.40%
single head	62.35%	77.75%

A.3.3 CONVERGENCE

We compare AWGIM with LEO in terms of convergence speed. The batch size is set to be 16 for both methods. We use the hyper-parameters tuned by authors to train LEO. The accuracy of meta-validation set during meta-training on 5-way 1-shot *mini*ImageNet is plotted, shown in Figure 3. we can see clearly that AWGIM converges faster than LEO and outperforms LEO except for the first few iterations.

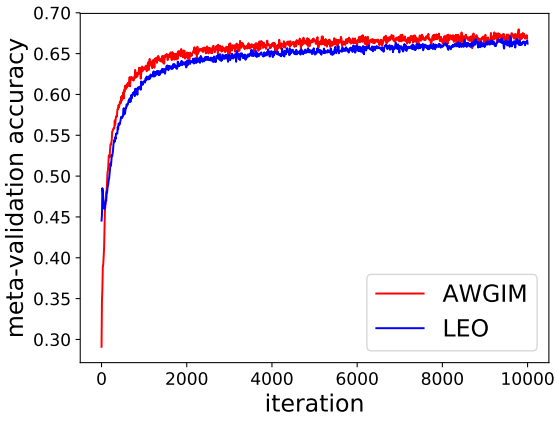

Figure 3: The meta-validation accuracy during meta-training.

A.3.4 INFERENCE TIME

We measure the inference time of AWGIM to show that it induces minimal computational overhead. In comparison, we use "MLP encoding" in two paths, which has time complexity $O(NK + |\mathcal{Q}|)$. We use two set-ups on *mini*ImageNet and the batch size is set to be 64. 100 batches are processed and we report the average consumed time for one batch. All these experiments on done with the same GPU and workstation. The results are shown in Table 5. It can be observed that the usage of self-attention and cross attention in AWGIM occurs negligible overhead, compared with MLP encoding. This is because the values of $N, K, |\mathcal{Q}|$ are all relatively small and matrix multiplication further can be processed very fast by GPU.

Table 5: The comparison of inference time between AWGIM and MLP encoding.

Method	5-way 1 -shot	5-way 5-shot
AWGIM	0.036s	0.093s
MLP encoding	0.033s	0.093s

A.3.5 VISUALIZATION

We visualize the generated classification weights by t-SNE (Maaten & Hinton, 2008). First we sample 400 tasks from meta-validation set of 5-way 1-shot *mini*ImageNet experiment. Each task contains 5 query samples from 5 different classes. Thus in total there are $400 \times 5 \times 5 = 10,000$ weight vectors to visualize. As comparison, inputs to the generator g are also plotted. The visualization results are shown in Figure 4. The inputs to g are displayed in (a, b) and the generated

classification weights in (c, d). From the comparison between (a) and (c), we can see the decoded weights for each class in (c) are clustered closer than (a) in general. Red and blue dots in (b, d) denotes the classification weights for two query samples from two classes within one task. It can be observed that g can generate adapted weights for different query samples. This is consistent with Table 3, where the results of "random shuffle between classes" suggest that query samples from different class have distinct classification weights.

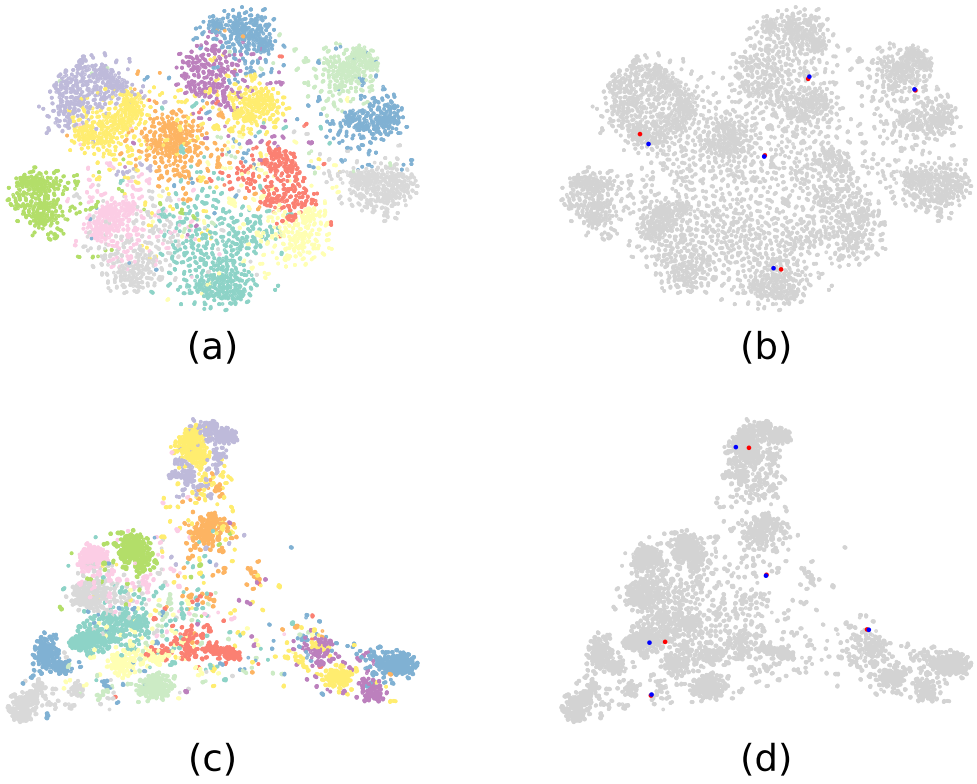

Figure 4: t-SNE visualization of the inputs to g in (a, b) and the generated classification weights in (c, d). Blue and red dots in (b) and (d) are the classification weights for two query samples in the same task.