Paper Reading Assignment – 3

Sanyam Jain (P20QC001)

Indian Institute of Technology, Jodhpur

November, 2021

Meta-Learning with Implicit Gradients

Paper Summary

The motivation of meta-learning comes from being able to learn from small data. From previous research, we have already known that supervised ML models are data-hungry. If you give large datasets to train large models, you learn a brilliant decision boundary (broad generalization). Meta-learning (or learning to learn) uses bi-level optimization criteria. In the multiple time scales of iterative optimization, the meta-learning can be divided into 2 parts, inner loop, and outer loop. Inner loop may optimize some cost function however, outer loop optimizes environment. For example, in an individual's lifetime of learning, whatever learns from the outside world is the outer loop and whatever learns inside that adapts to the outer environment is the inner loop. Traditional ML algorithms start to fall when situations tend to change fast and your model runs out of training data. Essentially we want our model to classify with large prior experience and small training knowledge because you don't have the luxury of having huge training data. In other words, can we explicitly learn priors from previous experiences that lead to an efficient entirely data-driven approach to learning? You create sets of your dataset that resembles each set for each task. However, each of these sets contains additional data that is part of other sources where the labels are different from your required task. These tasks are structurally similar. Whatever we see in the meta-dataset or the additional dataset we store it in a variable such that you throw away the dataset however keep the learned parameters intact within the variable. In simpler words, you come up with a task that has less training data. Now you say, I have a more structurally similar dataset, and let's give a round to that dataset too, such that I learn only the underlying pattern of that meta-dataset of different tasks. Now that you have learned your few sampled dataset along with similar data, throw away the dataset and just keep the learned parameters to solve other tasks. We use a meta-train dataset to learn theta. Now theta contains all the parameters that are required to solve new tasks. In simple words, collectively we can say, use plentiful data from other previously seen tasks to learn how to learn and then learn new tasks efficiently from small amounts of data.

Recasting meta-learning as a supervised learning problem at another level is a standard way to solve meta-learning formulations and tasks. One-shot (case of few-shot learning) learning paradigms where you have only one example from each class, the task is to distinguish between all the classes. The question to ask here is can you come up with a classifier that can distinguish between the classes and you will only be given one example from each class. To answer this question, let's have an X-shot learning paradigm with supervision such that we come up with a meta dataset that already consists of many different dataset examples with structural similarity where each subset is part of a different learning task. Not only each subset is part of training a learning algorithm but also tests these learning algorithms with the test cases that contain images from their respective tasks. Once you learn these few shot meta-training sets (that originally contain subsets of learning tasks of different subsets of the dataset), the learned meta-algorithm is presented with a meta-test such that classes are different from training sets. This is where adaptation comes into the picture, as the model was trained on some distribution of tasks that were presented with their labels in meta training. Now the meta-model is expected to learn quickly with the meta-test dataset on how to classify these new images keeping the structure of tasks similar.

To tell the differences between general learning and meta-learning, we can say that Generic Learning can be viewed as the process of taking some loss function and some training set (X_train, Y_train) and then recovering the best model parameters. In simpler words, generic learning is nothing but optimizing the loss function (maybe using GD) while training the dataset such that your model generalizes well on unseen examples (X_test). Generic Meta-Learning, can be considered as optimizing multiple tasks and minimizing the error/risk of those tasks on each subset of the dataset wrt some parameter that carries learned function.

Meta-Learning till now is divided into 3 kinds of methodologies that are in practice:

1. Black-Box Meta-Learning (Recasting Meta-Learning as supervised with sequence models and memory-based architectures such that the model could read the entire training set in a few shots)
2. Non-Parametric Meta-Learning - The inner loop / actual learning of multiple datasets is parametric, however, the outer loop that is an adaptation is non-parametric (kind of nearest neighbor) to learn representations.
3. **Gradient-Based Meta-Learning** - Highly parallel. The fundamental way of adapting to a new task in gradient-based meta-learning is based on fine-tuning Gradient Descent. Adaptation to the environment (outer loop) is based on gradient descent whereas the inner

loop (meta training) tries to maintain the parameters such that your outer loop (gradient descent) works well.

Firstly, *Transfer Learning* helps in fine-tuning the parameters for individual tasks. You have a giant dataset (say imagenet). Now you train a neural network on top of this big dataset. Now you have learned parameters and stored them. Now in Transfer Learning, you adapt your previously learned model on top of another task to speed up your classification process. In other words, you initialize your new task's neural network with previous parameters and finetune it to task-specific parameters. Secondly, Multi-Task Learning has one shared single neural network that performs different tasks with a single model with multiple heads and outputs. In other words, you want to learn the features from one class that could help build the similarities and differences between multiple classes. In addition, you have to a tradeoff with the class-wise accuracies. One neural network is not only performing classification for one task rather multiple tasks, thus limiting accuracy. However, meta-learning can be taken as analog to Transfer Learning in the sense that you are not having the giant dataset. Rather you just start with a random guess of initial parameters. Now the interesting part comes when (let's say in a neural network) you backpropagate the error from the end of the outer loop (environment) to the root of the inner loop (meta-train). In simpler words, let's say you have *theta_ 0* as your random initial parameters for your dataset. Now you go forward and train a learner based on individual tasks with their respective individual datasets. Once each of these tasks gets trained on their training datasets, then evaluate on their testing datasets or validation dataset. Once the evaluation is done they report back a number. This number is backpropagated to the learner of the task in the form of a generalization error for that particular task. All such errors are collected and updates the initial random guess *theta_ 0* on a collective basis. We measure that how easy is to adapt initial parameters to the tasks on their dataset. Now, this updation is iterative such that you reach a threshold for *theta_ 0*. To incorporate new tasks for the learning algorithm, will be very easy. Because you just have to update your guessing parameter (even if it is trained earlier for previous tasks) *theta_ 0*. The new model is expected to be more robust and accurate than the previous model as it would be able to adapt quickly the initial parameters to its dataset. And the model will be good in terms of the previous model that had the task just trained on its small dataset from scratch. And all of this optimization is optimized under Gradient Descent.

Earlier meta-learning with gradients was performed under **MAML** (Model Agnmostic Meta-Learning) scheme. In this kind of training procedure, we start with a random initial parameters guess. Now you want that initial guess to optimize such that you super-converge with your tasks. In simpler words, you don't know the exact gradient initially and come up with a gradient such that it directs an average pointer from the initial random parameter for all your tasks. In order to find that better guess parameter to update your initial random guess, you have to calculate the loss. Globe Loss function is an average of all the loss functions that are cumulated across tasks individually. That is how the gradient is the sum of all the gradients of these loss functions wrt original parameter. In other words, you start with that global random guess parameter *theta* that undergoes task-specific gradient descent and produces a task-specific loss. The gradient from the adapted task-specific neural network has to be backpropagated but the question is how would you do that along with the same learning procedure? The solution to this problem is to answer, how can I adjust my initial parameters *theta_0* such that the task-specific loss function final parameters go in the same direction. However, this is very tricky because the procedure is highly non-linear and we have to undergo iterative optimization for this particular task. That is why we have to perform backpropagation through the same SGD optimization procedure with the same k number of steps which is highly expensive. There is **First order MAML** that follows the same procedure as MAML however it performs the aggregation/average of all the task-specific gradients and points the initial random guess *theta_0* to the average direction of all the task-specific gradients. However, in this work, **implicit gradient descent** puts the learning algorithm to bypass the necessity of backpropagating the gradients to the same path. Authors find the relationship between the final gradient is related to initial parameters with the help of a quadratic regularizer. In order to keep the summary short, I will write about the problem formulation and intuition of why this method works. The search of best meta-learning parameters is completed by gradient descent to minimize the cost function *argmin:x [F(x)]* that is the outer level. The inner level F(x) is the average of the loss function such that your task-specific testing dataset and prediction of the neural network are close to each other. The neural network is trained on the task-specific training dataset while updating the global parameter *theta* (initial random parameter). Note that there is no dependence on task for the meta parameters because it remains the same for all tasks. In simpler words, you keep the global parameter the same (*theta*) and keep on iterative optimization of the same *theta* while training the learning algorithm on task-specific training data wrt predictions of the task-specific neural network and task-specific test data. And this loss has to be minimal to converge. The meta parameter and gradient step on the training dataset is the inner level of iMAML. The algorithm component (inner level) starts from meta parameters and runs gradient steps on the training dataset gradient (loss). This is the first step for global parameters and then further steps task-specific parameters are updated.

$$\overbrace{\boldsymbol{\theta}^*_{\mathrm{ML}} := \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} F(\boldsymbol{\theta})}^{\text{outer}-\text{level}}, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\Big(\overbrace{\mathcal{A}lg\big(\boldsymbol{\theta}, \mathcal{D}_i^{\mathrm{tr}}\big)}^{\text{inner}-\text{level}},\ \mathcal{D}_i^{\mathrm{test}}\Big).$$

The intuition behind the working of the implicit gradients is because of the quadratic regularizer. The search space can be imagined around the starting point such that a strong connection is created between the ending gradient and the initial gradient. The space of training loss (where you will find your optimal loss function) is present in the SGD space. Now SGD will go over the training loss space and find out the optimal loss such that there is a tradeoff between the quadratic regularizer and the training loss space (region). Two forces on regularizer namely one pointing to the training loss and the other towards inside the quadratic (for animation visit here - https://imgur.com/a/fiq88Ba).

Thanks for Reading!