Script for Paper 3 - AML

Slide 1: Hi Sir and Maam. So in this presentation I am going to describe about meta learning with implicit gradients. This is the original work of Aravind Rajeswaran and co-authors. And I am going to present this work here.

Slide 2: As described in the assignment task list here are the 4 contents of this presentation. Motivation, Key algorithm, Limitations and key takeaways.

Slide 3: The techniques that help us in shotage of data, As we have already studied Transfer learning and multi task learning, though for the purpose of completensess, 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, Now *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

Slide 4: Earlier meta-learning with gradients was performed under MAML (Model Agnmostic MetaLearning) 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.

Slide 5: 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.

Slide 6:
step 1 says general pre reqs of the learning algorithm, step 2 says run this loop until convergence. In that loop, step 3 says that just like in meta learning we have tasks coupled in the outer level of the learning algorithm here also we are pulling the tasks to learn in our algorithm from the pool of tasks. (or more precisely say distribution of tasks as in meta learning this is what you do right, training 2 learning criterion, first one is the task specific learner and another is trained globally where you update the theta values using task specific loss function) Step 4 and 5 says that compute the meta gradient for each task. and then in step 7 take a global average of those gradients for the outer paramter (the global paramter the one you started with) and then in step 8 do the gradient descent.

Now, in the algorithm Implicit meta gradient computation, the core problem is to find the best guess of the parameter theta such that your GD converges faster in outer level. Here particularly in the inner level the paramter that you had initialized in the beginning with random guess called meta paramter, has to be updated iteratively. As step 3 in algorithm 2 says that obtain task paramters using iterative optimization solver such that there exist an upper bound delta for the difference between task specific parameters and global level parameters. To prevent from overfit and generalize better the author has put an upper bound of delta such that you do not memorise by really completely optmising the problem but can be closer to the optimal solution to become sub-optimal convergence.

No there is one computationally expensive task we are dealing here to calculate the exact optimal procedure to propogate the gradient back to the initial random paramter to udpate it. However, with highly non linear subspaces it will be expensive. Authors suggest that just dont follow the path backwards to reach the intial parameters but instead try to use the gradient that is calculated as vector v_i that is the gradient at the end of your task specific optimization procedure, by multiplying it with the matrix. Now there comes the important part of the contribution of the paper because what the standard procedure has been doing is inverting the matrix blindly that became highly computationally expensive. The authors say that dont do that, instead multiply the gradient wiht your invertible matrix and calculate something (g_i) that is close to the second term in the equation which si upper bounded by some delta dash.

Slide 7: Unless experimented we cannot say exactly about using the SGD as author has not covered the usage of SGD procesure or using Stochasticity in the optimization procedure. It is expected that introducing SGD may cause the learning procedure to converge speedly because rather than doing iterative optimization you will reach your threshold delta may be very quickly than your standard GD procedure and could help in updating the initial random guessing paramters to update faster. In a Bayesian interpretation of iMAML, one can think of the anchor point θ_0 as the mean of a prior distribution over the neural network's weights. The inner loop of the algorithm, or Alg(fi,θ_0) then finds the maximum-a-posteriori (MAP) approximation to the posterior over θ given the dataset in question. This is assuming that the loss is a log likelihood of some kind. The question is, how should one update the meta-parameter θ_0?
While we studied different gradient-based optimization methods in the inner loop, iMAML can in principle be used with a variety of inner loop algorithms, including dynamic programming methods such as Q-learning, two-player adversarial games such as GANs, energy-based models, and actor-critic RL methods, and higher-order model-based trajectory optimization methods. This significantly expands the kinds of problems that optimizationbased meta-learning can be applied to.

Slide 8: repeat same from slide 8