Meta-Learning Is All You Need

James Le

May 24th, 2020

Abstract

Neural networks have been highly influential in the past decades in the machine learning community, thanks to the rise of compute power, the abundance of unstructured data, and the advancement of algorithmic solutions. However, it is still a long way for researchers to completely use neural networks in real-world settings where the data is scarce and requirements for model accuracy/speed are critical. Meta-learning, also known as learning how to learn, has recently emerged as a potential learning paradigm that can learn information from one task and generalize that information to unseen tasks proficiently. In this report, the key questions that I attempt to answer are: (1) Why do we need meta-learning?, (2) How does the math of meta-learning work?, and (3) What are the different approaches to design a meta-learning algorithm?

1 Motivation For Meta-Learning

Thanks to the advancement in algorithms, data, and compute power in the past decade, deep neural networks have allowed us to handle unstructured data (such as images, text, audio, video, etc.) very well without the need to engineer features by hand. Empirical research has shown that if neural networks can generalize very well if we feed them large and diverse inputs. For example, Transformers [1] and GPT-2 [2] have made the wave in the Natural Language Processing research community last year with their wide applicability in various tasks.

However, there is a catch with using neural networks in the real-world setting where:

- **Large datasets are unavailable**? This issue is common in many domains ranging from classi-

fication of rare diseases to translation of rare languages. It is clearly impractical to learn from scratch for each task in these scenarios.

- **Data has a long tail**? This issue can easily break the standard machine learning paradigm. For example, in the self-driving car setting, an autonomous vehicle can be trained to handle common situations very well, but it often struggles with uncommon situations (such as people jay-walking, animals crossing, traffic lines not working) where humans can easily handle. This can lead to very bad outcomes, such as the Uber's accident in Arizona a few years ago.
- **We want to quickly learn something about a new task without training our model from scratch**? Humans can do this quite easily by leveraging our prior experience. For example, if I know a bit of Spanish, then it should not be too difficult for me to learn Italian, as these two languages are quite similar linguistically.

In this report, I would like to give an introductory overview of **meta-learning**, which is a learning framework that can help our neural network become more effective in the settings mentioned above. In this setup, we want our network to learn a new task more proficiently - assuming that it is given access to data on previous tasks.

Historically, there has been a few papers thinking along this direction.

- Back in 1992, Bengio et al. [3] looked at the possibility of a learning rule that can solve new tasks.
- In 1997, Caruana [4] wrote a survey about multitask learning, which is a variant of meta-learning. He explained how tasks can be learned in parallel using a shared representation between models and also presented a multitask inductive transfer notion that uses back-propagation to handle additional tasks.
- In 1998, Thrun [5] explored the problem of lifelong learning, which is inspired by the ability of humans to exploit experiences that come from related learning tasks to generalize to new tasks.

Right now is an exciting period to study meta-learning because it is increasingly becoming more fundamental in machine learning research. There are many recent works that have leveraged meta-learning algorithms (and their variants) to do well for the given tasks. A few examples include:

- Aharoni et al. [6] expands the number of languages used in a multi-lingual neural machine translation setting from 2 to 102. Their method learns a small number of languages and generalizes

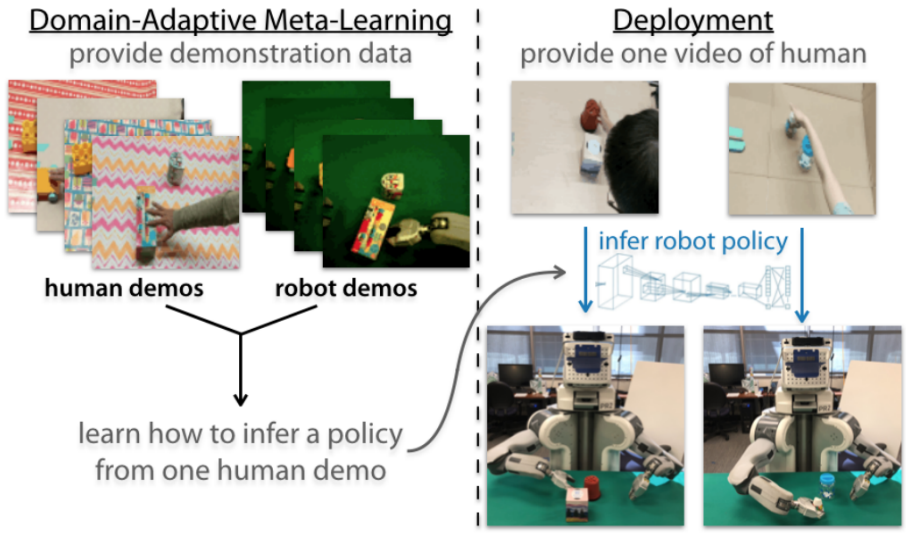

Figure 1: The Domain-Adaptive Meta-Learning system described in [7]

them to a vast amount of other.

- Yu et al. [7] presents **Domain-Adaptive Meta-Learning** (figure 1), a system that allows robots to learn from a single video of a human via prior meta-training data collected from related tasks.
- A recent paper from YouTube [8] shows how their team used multi-task methods to make video recommendations and handle multiple competing ranking objectives.

Forward looking, the development of meta-learning algorithms will help democratize deep learning and solve problems in domains with limited data.

2 Basics of Meta-Learning

In this section, I will cover the basics of meta-learning. Let's start out with the mathematical formulation of supervised meta-learning.

2.1 Formulation

In a standard supervised learning, we want to maximize the likelihood of model parameters ϕ given the training data D:

$$arg\max_{\phi} logp(\phi|D) \tag{1}$$

Equation (1) can be redefined as maximizing the probability of the data given the parameters and maximizing the marginal probability of the parameters, where $p(D|\phi)$ corresponds to the data likelihood and $p(\phi)$ corresponds to a regularizer term:

$$= arg\max_{\phi} logp(D|\phi) + logp(\phi) \tag{2}$$

Equation (2) can be further broken down as follows, assuming that the data D consists of (input, label) pairs of (x_i, y_i):

$$= arg\max_{\phi} \sum_{i} logp(y_i|x_i, \phi) + logp(\phi) \tag{3}$$

However, if we deal with very large data D (as in most cases with complicated problems), our model will likely overfit. Even if we have a regularizer term here, it might not be enough to prevent that from happening.

The key problem that supervised meta-learning solves is **Is it feasible to get more data when dealing with supervised learning problem**?

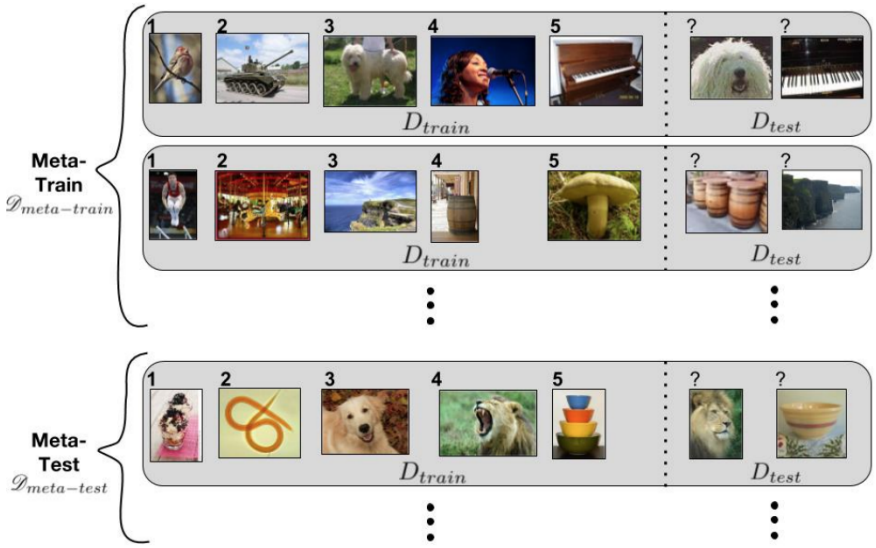

Figure 2: The Meta-Learning Setup described in [9]

Ravi and Larochelle [9] is the first paper that provides a standard formulation of the meta-learning setup, as seen in figure 2. They reframe equation (1) to equation (4) below, where $D_{meta-train}$ is the meta-

training data that allows our model to learn more efficiently. Here, $D_{meta-train}$ corresponds to a set of datasets for predefined tasks $D_1, D_2, \cdots, D_n$.

$$arg\max_{\phi} logp(\phi|D, D_{meta-train}) \tag{4}$$

Next, they design a set of **meta-parameters** $\theta = p(\theta|D_{meta-train})$, which includes the necessary information about $D_{meta-train}$ in order to solve the new tasks.

Mathematically speaking, with the introduction of this intermediary variable θ, the full likelihood of parameters for the original data given the meta-training data (in equation (4)) can be expressed as an integral over the meta-parameters θ:

$$logp(\phi|D, D_{meta-train}) = log \int_{\Theta} p(\phi|D, \theta)p(\phi|D_{meta-train})d\theta \tag{5}$$

Equation (5) can be approximated further with a point estimate for our parameters:

$$\approx logp(\phi|D, \theta^*) + logp(\theta^*|D_{meta-train}) \tag{6}$$

- $p(\phi|D, \theta^*)$ is the **adaptation** task that collects task-specific parameters ϕ for a new task - assuming that it has access to the data from that task D and meta-parameters θ.
- $p(\theta^*|D_{meta-train})$ is the **meta-training** task that collects meta-parameters θ - assuming that it has access to the meta-training data $D_{meta-train}$.

To sum it up, the meta-learning paradigm can be broken down into two phases:

- The adaptation phase: $\phi^* = arg\max_{\phi} logp(\phi|D, \theta^*)$ (first term in (6))
- The meta-training phase: $\theta^* = arg\max_{\theta} logp(\theta|D_{meta-train})$ (second term in (6))

2.2 Loss Optimization

Let's look at the optimization of meta-learning method. Initially, our meta-training data consists of pairs of training-test set for every task:

$$D_{meta-train} = \{(D_1^{train}, D_1^{test}), \cdots, (D_n^{train}, D_n^{test})\} \tag{7}$$

There are k feature-label pairs (x, y) in the training set D_i^{train} and l feature-label pairs (x, y) in the test set D_i^{test}:

$$D_i^{train} = \{(x_1^i, y_1^i), \cdots, (x_k^i, y_k^i)\}; D_i^{test} = \{(x_1^i, y_1^i), \cdots, (x_l^i, y_l^i)\} \tag{8}$$

During the adaptation phase, we infer a set of task-specific parameters ϕ^*, which is a function that takes as input the training set D^{train} and returns as output the task-specific parameters: $\phi^* = f_{\theta^*}(D^{train})$. Essentially, we want to learn a set of meta-parameters θ such that, the function $\phi_i = f_\theta(D_i^{train})$ is good enough for the test set D_i^{test}.

During the meta-learning phase, to get the meta-parameters θ^*, we want to maximize the probability of the task-specific parameters ϕ being effective at new data points in the test set D_i^{test}:

$$\theta^* = \max_\theta \sum_{i=1}^{n} logp(\phi_i | D_i^{test}) \tag{9}$$

2.3 Meta-Learning Paradigm

According to Chelsea Finn [10], there are two views of the meta-learning problem: a deterministic view and a probabilistic view.

The **deterministic** view is straightforward: we takes as input a training data set D^{train}, a test data point x_{test}, and the meta-parameters θ to produce the label corresponding to that test input y_{test}. The way we learn this function is via the $D_{meta-train}$ as discussed earlier.

$$y_{test} = f(D^{train}, x_{test}; \theta) \tag{10}$$

The **probabilistic** view incorporates Bayesian inference: we perform a maximum likelihood inference over the task-specific parameters ϕ_i - assuming that we have the the training dataset D_i^{train} and a set of meta-parameters θ:

$$\max_{\theta} \sum_{i} logp(\phi_i | D_i^{train}) \tag{11}$$

Regardless of the view, there two steps to design a meta-learning algorithm:

- Step 1 is to design the function $p(\phi_i | D_i^{train}, \theta)$ during the adaptation phase.
- Step 2 is to optimize θ with respect to $D_{meta-train}$ during the meta-training phase.

In this report, I will only pay attention to the deterministic view of meta-learning. In the remaining sections, I focus on the three different approaches to build up the meta-learning algorithm: (1) The black-box approach, (2) The optimization-based approach, and (3) The non-parametric approach. More specifically, I will go over their formulation, architectures used, and challenges associated with each approach.

3 Black-Box Meta-Learning

3.1 Formulation

The black-box meta-learning approach uses neural network architecture to generate the distribution $p(\phi_i | D_i^{train}, \theta)$.

- Our task-specific parameters are: $\phi_i = f_\theta(D_i^{train})$.
- A neural network with meta-parameters θ (denoted as f_θ) takes in the training data D_i^{train} as input and returns the task-specific parameters ϕ_i as output.
- Another neural network (denoted as $g(\phi_i)$) takes in the task-specific parameters ϕ_i as input and returns the predictions about test data points D_i^{test} as output.

During optimization, we maximize the log likelihood of the outputs from $g(\phi_i)$ for all the test data

points. This is applied across all the tasks in the meta-training set:

$$\max_{\theta} \sum_{T_i} \sum_{(x,y)\sim D_i^{test}} log g_{\phi_i}(y|x) \tag{12}$$

The log likelihood of $g(\phi_i)$ equation (12) is essentially the loss between a set of task-specific parameters ϕ_i and a test data point D_i^{test}:

$$\sum_{(x,y)\sim D_i^{test}} log g_{\phi_i}(y|x) = L(\phi_i, D_i^{test}) \tag{13}$$

Then in equation (12), we optimize the loss between the function $f_\theta(D_i^{train})$ and the evaluation on the test set D_i^{test}:

$$\max_{\theta} \sum_{T_i} L(f_\theta(D_i^{train}), D_i^{test}) \tag{14}$$

This is the black-box meta-learning algorithm in a nutshell:

- We sample a task T_i, as well as the training set D_i^{train} and test set D_i^{test} from the task dataset D_i.
- We compute the task-specific parameters ϕ_i given the training set D_i^{train}: $\phi_i \leftarrow f_\theta(D_i^{train})$.
- Then, we update the meta-parameters θ using the gradient of the objective with respect to the loss function between the computed task-specific parameters ϕ_i and D_i^{test}: $\nabla_\theta L(\phi_i, D_i^{test})$.
- This process is repeated iteratively with gradient descent optimizers.

3.2 Challenges

The main challenge with this black-box approach occurs when ϕ_i happens to be massive. If ϕ_i is a set of all the parameters in a very deep neural network, then **it is not scalable to output** ϕ_i.

Santoro et al. [11] and Mishra et al. [14] are two research papers that tackle this. Instead of having a neural network that outputs all of the parameters ϕ_i, they output a low-dimensional vector h_i, which is then used alongside meta-parameters θ to make predictions. The new task-specific parameters ϕ_i has the form: $\phi_i = \{h_i, \theta_g\}$, where θ_g represents all of the parameters other than h.

Overall, the general form of this black-box approach is as follows:

$$y^{ts} = f_\theta(D_i^{train}, x^{ts}) \tag{15}$$

Here, y^{ts} corresponds to the labels of test data, x^{ts} corresponds to the features of test data, and D_i^{train} corresponds to pairs of training data.

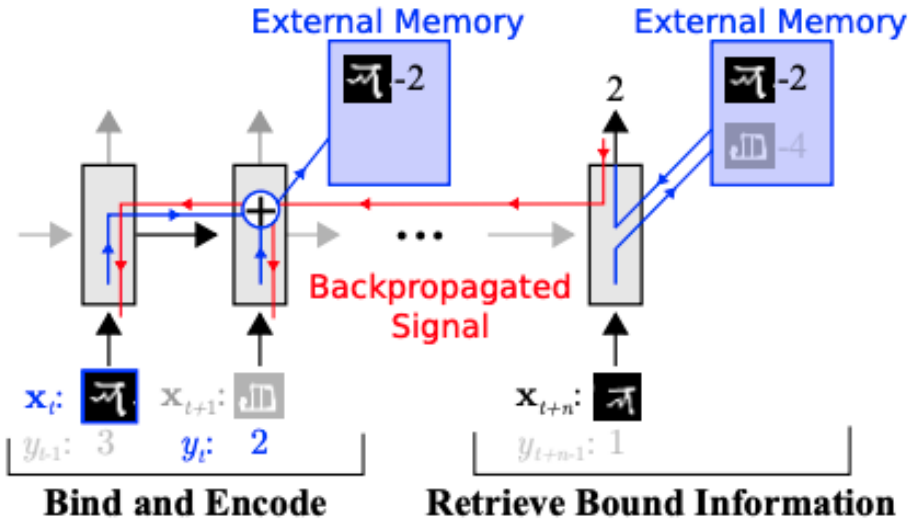

Figure 3: The memory-augmented neural network as described in [11].

3.3 Architectures

So what are the different model architectures to represent this function f?

- **Memory-Augmented Networks** by Santoro et al. [11] uses Long Short-Term Memory and Neural Turing Machine architectures to represent f. Both architectures have an external memory mechanism to store information from the training data point and then access that information during inference in a differentiable way, as seen in figure 3.
- **Conditional Neural Processes** by Garnelo et al. [12] represents f via 3 steps: (1) using a feed-forward neural network to compute the training data information, (2) aggregating that information, and (3) passing that information to another feed-forward network for inference.
- **Meta Networks** by Munkhdalai and Yu [13] uses other external memory mechanisms with slow and fast weights that are inspired by neuro-science to represent f. Specifically, the slow weights

are designed for meta-parameters θ and the fast weights are designed for task-specific parameters ϕ.

- **Neural Attentive Meta-Learner** by Mishra et al. [14] uses an attention mechanism to represent f. Such mechanism allow the network to pick out the most important information that it gathers, thus making the optimization process much more efficient, as seen in figure 4.

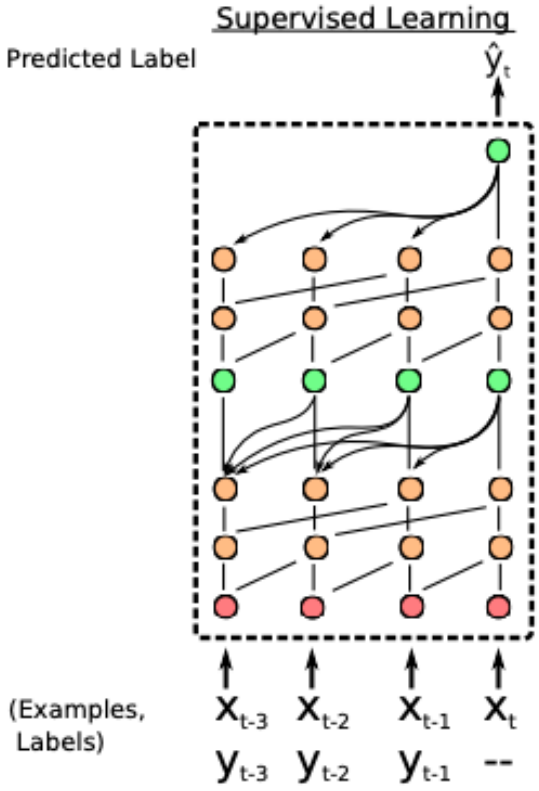

Figure 4: The simple neural attentive meta-learner as described in [14].

In conclusion, black-box meta-learning approach has a high learning capacity. Given that neural networks are universal function approximators, the black-box meta-learning algorithm can represent any function of our training data. However, as neural networks are fairly complex and the learning process usually happens from scratch, black-box approach usually requires a large amount of training data and a large number of tasks in order to perform well.

4 Optimization-Based Meta-Learning

Okay, so how else can we represent the distribution $p(\phi_i|D_i^{train}, \theta)$ in the adaptation phase of meta-learning? If we want to infer all the parameters of our network, we can treat this as an optimization procedure. The key idea behind optimization-based meta-learning is that we can optimize the process of getting the task-specific parameters ϕ_i so that we will get a good performance on the test set.

4.1 Formulation

Recall that the meta-learning problem can be broken down into two terms below, one that maximizes the likelihood of training data given the task-specific parameters and one that maximizes the likelihood of task-specific parameters given meta-parameters:

$$\max_{\phi_i} logp(D_i^{train}|\phi_i) + logp(\phi_i|\theta) \tag{16}$$

Here the meta-parameters θ are pre-trained during training time and fine-tuned during test time. The equation below is a typical optimization procedure via gradient descent, where α is the learning rate.

$$\phi \leftarrow \theta - \alpha \nabla_\theta L(\theta, D^{train}) \tag{17}$$

To get the pre-trained parameters, we can use standard benchmark datasets such as ImageNet for computer vision, Wikipedia Text Corpus for language processing, or any other large and diverse datasets that we have access to. As expected, this approach becomes less effective with *small amount of training data*.

Model-Agnostic Meta-Learning (MAML) from Finn et al. [15] is an algorithm that address this exact problem. Taking the optimization procedure in equation (17), it adjusts the loss so that only the best-performing task-specific parameters ϕ on test data points are considered. This happens for all the tasks:

$$\min_\theta \sum_{task_i} L(\theta - \alpha \nabla_\theta L(\theta, D_i^{train}), D_i^{test}) \tag{18}$$

The key idea is to learn θ for all the assigned tasks in order for θ can transfer effectively via the optimization procedure.

This is the optimization-based meta-learning algorithm in a nutshell:

- We sample a task T_i, as well as the training set D_i^{train} and test set D_i^{test} from the task dataset D_i.
- We compute the task-specific parameters ϕ_i given the training set D_i^{train} using the optimization procedure described above: $\phi_i \leftarrow \theta - \alpha \nabla_\theta L(\theta, D_i^{train})$.
- Then, we update the meta-parameters θ using the gradient of the objective with respect to the loss function between the computed task-specific parameters ϕ_i and D_i^{test}: $\nabla_\theta L(\phi_i, D_i^{test})$
- This process is repeated iteratively with gradient descent optimizers.

As provided in the previous section, the black-box meta-learning approach has the general form: $y^{test} = f_\theta(D_i^{train}, x^{test})$. The optimization-based MAML method described above has a similar form below, where $\phi_i = \theta - \alpha \nabla_\theta L(\phi, D_i^{train})$:

$$y^{test} = f_{MAML}(D_i^{train}, x^{test}) = f_{\phi_i}(x^{test}) \tag{19}$$

To prove the effectiveness of the MAML algorithm, in [16], Finn and Levine shows that the MAML algorithm can approximate any function of D_i^{train}, x^{test} for a very deep function f. This finding demonstrates that the optimization-based MAML algorithm is as expressive as any other black-box algorithms mentioned previously.

4.2 Architectures

In [17], Grant et al. provides another MAML formulation as a method for **probabilistic inference via hierarchical Bayes**. Let's say we have a graphical model as illustrated in figure 5, where J is the task, x_{j_n} is a data point in that task, ϕ_j are the task-specific parameters, and θ are the meta-parameters.

To do inference with respect to this graphical model, we want to maximize the likelihood of the data

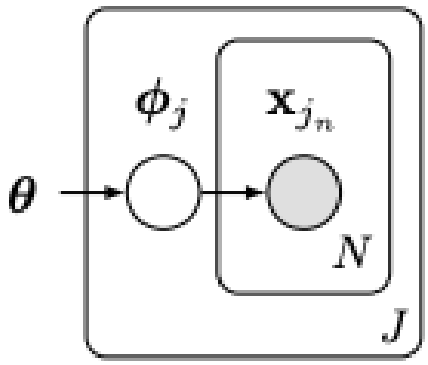

Figure 5: The probabilistic graphical model for which MAML provides an inference procedure as described in [17].

given the meta-parameters:

$$\max_{\theta} log \prod_{j} p(D_j|\theta) \tag{20}$$

The probability of the data given the meta-parameters can be expanded into the probability of the data given the task-specific parameters and the probability of the task-specific parameters given the meta-parameters. Thus, equation (20) can be rewritten as:

$$= log \prod_{j} \int p(D_j|\phi_j)p(\phi_j|\theta)d_{\phi_j} \tag{21}$$

This integral in equation (21) can be approximated with a Maximum a Posteriori estimate for ϕ_j:

$$\approx log \prod_{j} p(D_i|\hat{\phi}_j)p(\hat{\phi}_j|\theta) \tag{22}$$

In order to compute this Maximum a Posteriori estimate, Grant et al. [17] performs inference on Maximum a Posteriori under an *implicit Gaussian prior* - with mean that is determined by the initial parameters and variance that is determined by the number of gradient steps and the step size.

There have been other attempts to compute the Maximum a Posteriori estimate in equation (22):

- Rajeswaran et al. [18] proposes an **implicit MAML algorithm** that uses gradient descent with an

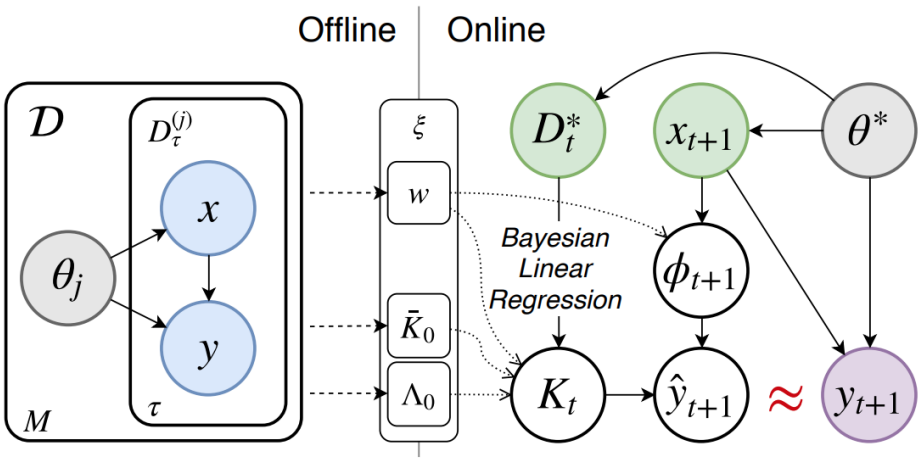

Figure 6: The ALPaCA algorithm which uses Bayesian linear regression as described in [19].

explicit Gaussian prior. More specifically, they regularize the inner optimization of the algorithm to be close to the meta-parameters θ: $\phi \leftarrow \min_{\phi'} L(\phi', D^{train}) + \frac{\lambda}{2}||\theta - \phi'||^2$. The mean and the variance of this explicit Gaussian prior is a function of λ regularizer.

- Harrison et al. [19] proposes the **ALPaCA algorithm** that uses an efficient Bayesian linear regression on top of the learned features from the inner optimization loop to represent the mean and variance of that regression as meta-parameters themselves (illustrated in figure 6). The inclusion of prior information here reduces computational complexity and adds more confidence to the final predictions.

- Bertinetto et al. [20] attempts to solve meta-learning with **differentiable closed-form solutions**. In particular, they apply a ridge regression as a base learner for the features in the inner optimization loop. The mean and variance predictions from the ridge regression are then used as meta-parameters in the outer optimization loop.

- Lee et al. [21] attempts to solve meta-learning with **differentiable convex optimization solutions**. The proposed method, called **MetaOptNet**, uses a support vector machine to learn the features from the inner optimization loop (as seen in figure 7).

4.3 Challenges

The MAML method requires very deep neural architecture in order to effectively get a good inner gradient update. Therefore, the first challenge lies in **choosing that architecture**. Kim et al. [22] proposes

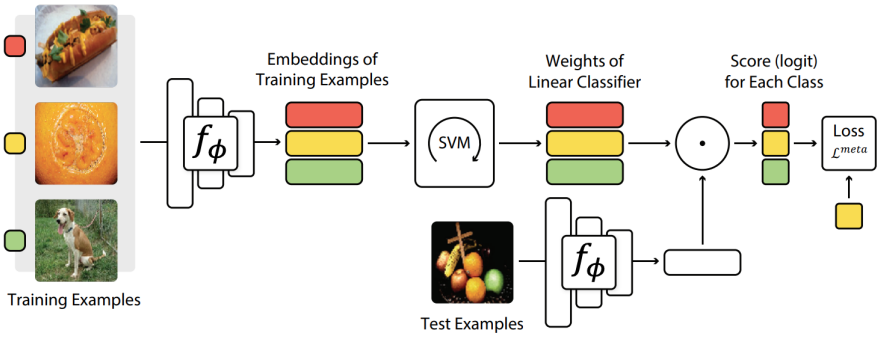

Figure 7: The MetaOptNet approach as described in [21].

Auto-Meta, which searches for the MAML architecture. They found that the highly non-standard architectures with deep and narrow layers tend to perform very well.

The second challenge that comes up lies in **the unreliability of the two-degree optimization paradigm**. There are many different optimization tricks that can be useful in this scenario:

- Li et al. [23] proposes **Meta-SGD** that learns the initialization parameters, the direction of the gradient updates, and the value of the inner learning rate in an end-to-end fashion. This method has proven to increase speed and accuracy of the meta-learner.
- Behl et al. [24] comes up with **Alpha-MAML**, which is an extension of the vanilla MAML. Alpha-MAML uses an online hyper-parameter adaptation scheme to automatically tune the learning rate, making the training process more robust.
- Zhou et al. [25] devises **Deep Meta-Learning**, which performs meta-learning in a concept space. As illustrated in figure 8, the concept generator generates the concept-level features from the inputs, while the concept discriminator discriminates the features generated from that first step. The final loss function includes both the loss from the discriminator and the loss from the meta-learner.
- Zintgraf et al. [26] designs **CAVIA**, which stands for fast context adaptation via meta-learning. To handle the overfitting challenge with vanilla MAML, CAVIA optimizes only a subset of the input parameters in the inner loop at test time (deemed *context parameters*), instead of the whole neural network. By separating the task-specific parameters and task-independent parameters, they show that training CAVIA is highly efficient.

- Antoniou et al. [27] ideates **MAML++**, which is a comprehensive guideline on reducing the hyper-parameter sensitivity, lowering the generalization error, and improving MAML stability. One interesting idea is that they disentangle both the learning rate and the batch-norm statistics per step of the inner loop.

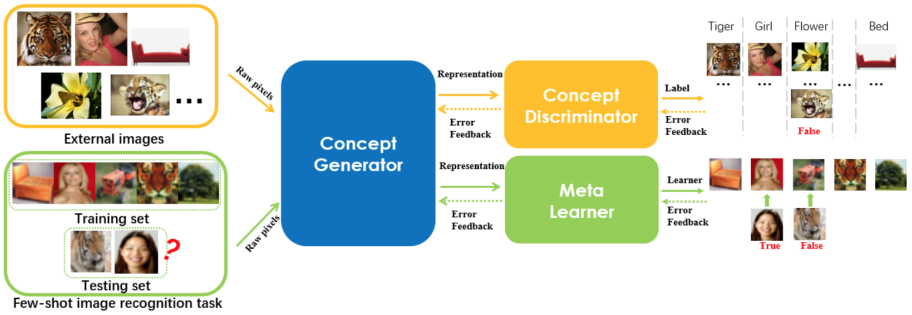

Figure 8: The Deep Meta Learning as described in [25].

The third challenge lies in **the computational expense associated with back-propagation**. The more inner gradient steps, the more challenging the optimization process is. There are two approaches to deal with this:

- Finn et al. [15] and Nichol et al. [28] truncate the back-propagation by approximating the $\frac{d_{\phi_i}}{d\theta}$ matrix as an identity function. This proves to work for simple few-shot learning problems.
- Rajeswaran et al. [18] uses a theorem to compute the meta-gradient $\frac{d_{\phi_i}}{d\theta}$ implicitly. The key benefit of this algorithm is that the outcome only depends on the inner optimization's solution, but not the number of inner gradient steps required.

In conclusion, optimization-based meta learning works by constructing a two-degree optimization procedure, where the inner optimization computes the task-specific parameters ϕ and the outer optimization computes the meta-parameters θ. The most representative method is the Model-Agnostic Meta-Learning algorithm, which has been studied and improved upon extensively since its conception.

The big benefit of MAML is that we can optimize the model's initialization scheme, in contrast to the black box approach where the initial optimization procedure is not optimized. Furthermore, MAML is highly consistent, which extrapolates well to learning problems where the data is out-of-distribution (compared to what the model has seen during meta-training). Unfortunately, because optimization-based meta learning requires second-order optimization, it is very computationally expensive.

5 Non-Parametric Meta Learning

So can we perform the learning procedure described above **without a second-order optimization**? This is where non-parametric methods fit in.

Non-parametric methods are very effective at learning with small amount of data (k-Nearest Neighbor, decision trees, support vector machines). In **non-parametric meta learning**, we compare the test data with the training data using some sort of similarity metric. If we find the training data that are most similar to the test data, we assign the labels of those training data as the label of the test data.

5.1 Formulation

This is the non-parametric meta-learning algorithm in a nutshell:

- We sample a task T_i, as well as the training set D_i^{train} and test set D_i^{test} from the task dataset D_i.
- We predict the test label $\hat{y}^{test}$ via the similarity between training data and test data (represented by f_θ): $\hat{y}^{test} = \sum_{x_k, y_k \in D^{train}} f_\theta(x^{test}, x_k) y_k$.
- Then we update the meta-parameters θ of this learned embedding function with respect to the loss function of how accurate our predictions are on the test set: $\nabla_\theta L(\hat{y}^{test}, y^{test})$
- This process is repeated iteratively with gradient descent optimizers.

Unlike the black-box and optimization-based approaches, we no longer have the task-specific parameters ϕ, which is not required for the comparison between training and test data.

5.2 Architectures

Now let's go over the different architectures used in non-parametric meta-learning methods.

Koch et al. [29] proposes a **Siamese network** which consists of two tasks: verification task and one-shot task. Taking in pairs of images during training time, the network verifies whether they are of the same class or different classes. At test time, the network performs one-shot learning: comparing each image x_{test} to the images in the training set D_j^{train} for a respective task, and predicting the label of x_{test} that corresponds to the label of the closest image. Figure 9 illustrates this strategy.

Vinyals et al. [30] proposes **Matching Networks**, which matches the actions happening during training

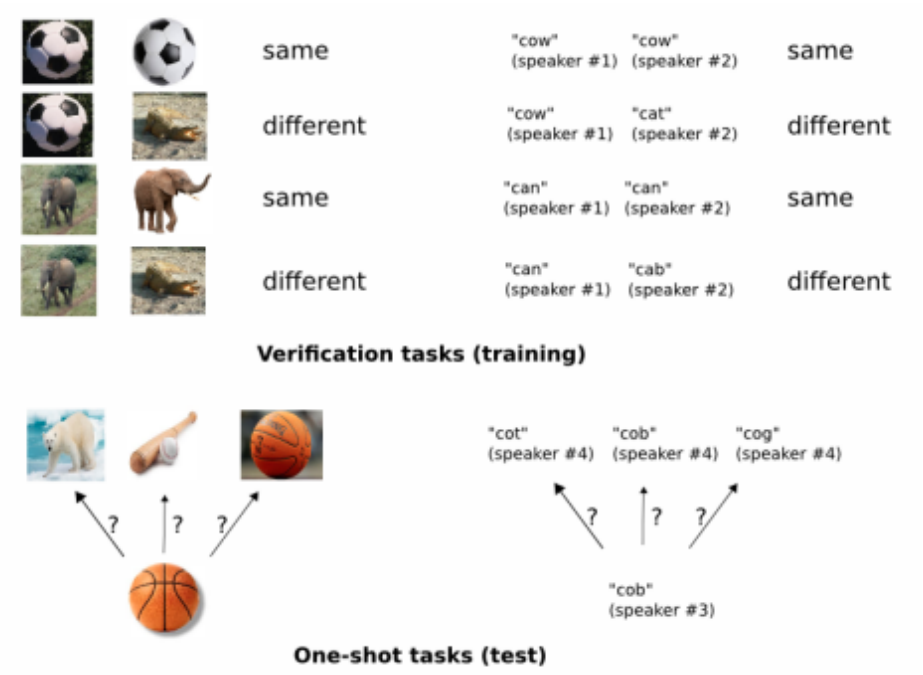

Figure 9: The Siamese Network strategy devised in [29].

time at test time. The network takes the training data and the test data and embeds them into their respective embedding spaces. Then, the network compares each pair of train-test embeddings to make the final label predictions:

$$\hat{y}^{test} = \sum_{x_k, y_k \in D^{train}} f_\theta(x^{test}, x_k) y_k \tag{23}$$

The Matching Network architecture used in Matching Networks includes a convolutional encoder network to embed the images and a bi-directional Long-Short Term Memory network to produce the embeddings of such images. As seen in figure 10, the examples in training match the examples in test.

Snell et al. [31] proposes **Prototypical Networks**, which create prototypical embeddings for all the classes in the given data. Then, the network compares those embeddings to make the final label predictions for the corresponding class.

Figure 11 provides a concrete illustrations of how Prototypical Networks look like in the few-shot sce-

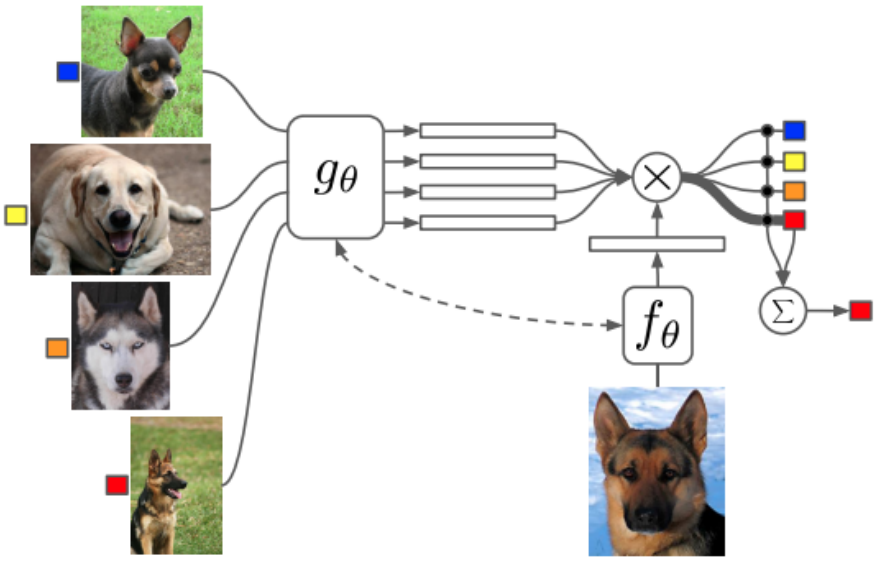

Figure 10: The Matching Networks architecture described in [30]

nario. c_1, c_2, and c_3 are the class prototypical embeddings, which are computed as:

$$c_k = \frac{1}{D_i^{train}} \sum_{(x,y) \in D_i^{train}} f_\theta(x) \tag{24}$$

Then, we compute the distances from x to each of the prototypical class embeddings: $D(f_\theta(x), c_k)$.

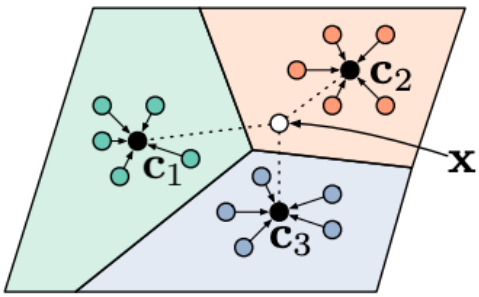

Figure 11: Prototpyical Networks as described in [31]

To get the final class prediction $p_\theta(y = k|x)$, we look at the probability of the negative distances after a softmax activation function, as seen below:

$$p_\theta(y = k|x) = \frac{softmax(-D(f_\theta(x), c_k))}{\sum_{k'} softmax(-D(f_\theta(x), c_{k'}))}) \tag{25}$$

5.3 Challenge

For non-parametric meta-learning, **how can we learn deeper interactions between our inputs?** Nearest neighbor probably will not work well when our data is high-dimensional. Here are three papers that attempt to accomplish this:

- Sung et al. [32] comes up with **RelationNet** (figure 12), which has two modules: the embedding module and the relation module. The embedding module embeds the training and test inputs to training and test embeddings. Then the relation module takes in the embeddings and learns a deep distance metric to compare those embeddings (function D in equation (25)).
- Allen et al. [33] proposes an **Infinite Mixture of Prototypes**. This is an extension of the Prototypical Networks, in the sense that it adaptively sets the model capacity based on the data complexity. By assigning each class its own cluster, this method allows the use of unsupervised clustering, which is helpful for many purposes.
- Garcia and Bruna [34] uses a **Graph Neural Network** in their meta-learning paradigm. By mapping the inputs into their graphical representation, they can easily learn the similarity between training and test data via the edge and node features.

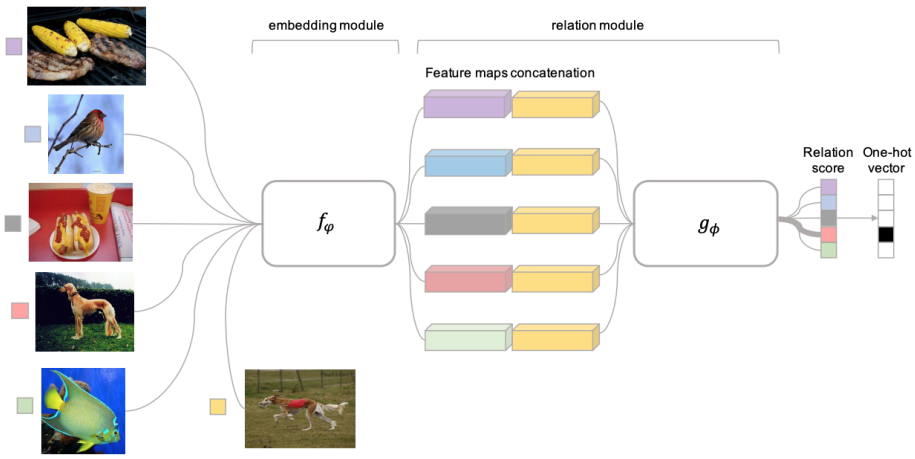

Figure 12: Relation Networks as described in [32]

6 Conclusion

In this report, I have discussed the motivation for meta-learning, the basic formulation and optimization objective for meta-learning, as well as the three approaches regarding the design of meta-learning algorithm. In particular:

- **Black-box meta-learning algorithms** have very strong learning capacity, in the sense that neural networks are universal function approximators. But if we impose certain structure into the function, there is no guarantee that black-box models will produce consistent results. Additionally, we can use black-box approaches with different types of problem settings such as reinforcement learning and self-supervised learning. However, because black-box models always learn from scratch, they are very data-hungry.
- **Optimization-based meta-learning algorithms** can be reduced down to gradient descent; thus, it's reasonable to expect consistent predictions. For a deep enough neural networks, optimization-based models also has very high capacity. Because the initialization is optimized internally, optimization-based models has a better head-start than black-box models. Furthermore, we can try out different architectures without any real difficulty, as evidenced by the Model-Agnostic Meta-Learning (MAML) learning paradigm. However, the second-order optimization procedure makes optimization-based approaches quite computationally expensive.
- **Non-parametric meta-learning algorithms** have good learning capacity for most choice of architectures as well as good learning consistency under the assumption that the learned embedding space is effective enough. Furthermore, non-parametric approaches do not involve any back-propagation, so they are computationally fast and easy to optimize. The downside is that they are hard to scale to large batches of data, because they are non-parametric.

There are a lot of exciting directions for the field of meta-learning, such as Bayesian Meta-Learning (the probabilistic view of meta-learning) and Meta Reinforcement Learning (the use of meta-learning in the reinforcement learning setting). I'd certainly expect to see more real-world applications in wide-ranging domains such as healthcare and manufacturing using meta-learning under the hood.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. *Attention Is All You Need*. Dec 2017. arXiv ePrint 1706.03762.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. Feb 2019. Technical Report, OpenAI.

[3] Samy Bengio , Yoshua Bengio , Jocelyn Cloutier , Jan Gecsei. *On the Optimization of a Synaptic Learning Rule*. 1992. Conference on Optimality in Biological and Artificial Networks.

[4] Rich Caruana. *Multitask Learning*. July 1997. Machine Learning 28, 41–75. https://doi.org/10.1023/A:1007379606734

[5] Sebastian Thrun. *Is learning the n-th thing any easier than learning the first?*. November 1995. NIPS'95: Proceedings of the 8th International Conference on Neural Information Processing Systems, pages 640-646.

[6] Roee Aharoni, Melvin Johnson, Orhan Firat. *Massively Multilingual Neural Machine Translation*. June 2019. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1, pages 3874-3884.

[7] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, Sergey Levine. *One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning*. Feb 2018. Robotics: Science and Systems 2018.

[8] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, Ed Chi. *Recommending What Video to Watch Next: A Multitask Ranking System*. September 2019. RecSys '19: Proceedings of the 13th ACM Conference on Recommender Systems, pages 43-51.

[9] Sachin Ravi, Hugo Larochelle. *Optimization As A Model For Few-Shot Learning*. March 2017. Published at the International Conference on Learning Conference (ICLR), 2017.

[10] Chelsea Finn. *Learning to learn with Gradients*. August 2018. PhD Thesis, University of California, Berkeley.

[11] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, Timothy Lillicrap. *One-shot Learning with Memory-Augmented Neural Networks*. May 2016. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016.

[12] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, S. M. Ali Eslami. *Conditional Neural Processes*. July 2018. In Proceedings of the 35th International Conference on Machine Learning, PMLR 80:1704-1713, 2018.

[13] Tsendsuren Munkhdalai, Hong Yu. *Meta Networks*. June 2017. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

[14] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, Pieter Abbeel. *A Simple Neural Attentive Meta-Learner*. February 2018. In the Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[15] Chelsea Finn, Pieter Abbeel, Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. August 2017. In the Proceedings of the 34th International Conference on Machine Learning (ICML), 2017.

[16] Chelsea Finn, Sergey Levine. *Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm*. February 2018. In the Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[17] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, Thomas Griffiths. *Recasting Gradient-Based Meta-Learning as Hierarchical Bayes*. January 2018. In the Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[18] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, Sergey Levine. *Meta-Learning with Implicit Gradients*. September 2019. In the Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019.

[19] James Harrison, Apoorva Sharma, Marco Pavone. *Meta-Learning Priors for Efficient Online Bayesian Regression*. October 2018. In the Workshop on the Algorithmic Foundations of Robotics (WAFR), 2018.

[20] Luca Bertinetto, João F. Henriques, Philip H.S. Torr, Andrea Vedaldi. *Meta-learning with differentiable closed-form solvers*. July 2019. In the International Conference on Learning Representations (ICLR), 2019.

[21] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, Stefano Soatto. *Meta-Learning with Differentiable Convex Optimization*. April 2019. In the Proceeedings of the conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[22] Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, Jiwon Kim. *Auto-Meta: Automated Gradient Based Meta Learner Search*. December 2018. In the Workshop on Meta-Learning at NeurIPS 2018.

[23] Zhenguo Li, Fengwei Zhou, Fei Chen, Hang Li. *Meta-SGD: Learning to Learn Quickly for Few-Shot Learning*. September 2017. arXiv ePrint 1707.09835v2.

[24] Harkirat Singh Behl, Atılım Güneş Baydin, Philip H.S. Torr. *Alpha MAML: Adaptive Model-Agnostic Meta-Learning*. May 2019. In the 6th ICML Workshop on Automated Machine Learning (2019).

[25] Fengwei Zhou, Bin Wu, Zhenguo Li. *Deep Meta-Learning: Learning to Learn in the Concept Space*. February 2018. arXiv ePrint 1802.03596.

[26] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, Shimon Whiteson. *Fast Context Adaptation via Meta-Learning*. June 2019. Published at the International Conference on Machine Learning (ICML), 2019.

[27] Antreas Antoniou, Harrison Edwards, Amos Storkey. *How to train your MAML*. March 2019. Published at the International Conference on Learning Conference (ICLR), 2019.

[28] Alex Nichol, Joshua Achiam, John Schulman. *On First-Order Meta-Learning Algorithms*. October 2018. arXiv ePrint 1803.02999.

[29] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov. *Siamese Neural Networks for One-shot Image Recognition*. 2015. In Proceedings of the 32nd International Conference on Machine Learning (ICML), Lille, France, 2015.

[30] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, Daan Wierstra. *Matching Networks for One Shot Learning*. June 2016. In Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS), 2016.

[31] Jake Snell, Kevin Swersky, Richard S. Zemel. *Prototypical Networks for Few-shot Learning*. June 2017. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS), 2017.

[32] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, Timothy M. Hospedales. *Learning to Compare: Relation Network for Few-Shot Learning*. March 2018. In the Proceeedings of the conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[33] Kelsey Allen, Evan Shelhamer, Hanul Shin, Joshua Tenenbaum. *Infinite Mixture Prototypes for Few-shot Learning*. 2019. In Proceedings of the 36th International Conference on Machine Learning, PMLR 97:232-241, 2019.

[34] Victor Garcia, Joan Bruna. *Few-Shot Learning with Graph Neural Networks*. February 2018. Published at the International Conference on Learning Conference (ICLR), 2018.