# Learning to Learn

Charlie Wu

May 2020

## 1 Introduction

Learning to learn can be described simply as *meta-learning*, or the process of being able to design and train successful optimization algorithms in a similar way that features are currently learned in vanilla machine learning models. In recent years of machine learning research, the learning process of creating well-refined models to solve specific problems with large quantities of training data has become very well refined and effective, in comparison to the previous manual processes.[4] While various researchers differ in exact *meta-learning* terminology, we will utilize definitions described by DeepMind researchers in their seminal paper published in 2016.[2] The process of using optimization algorithms created by humans, while very accurate when trained with discerning data on specific task(s), are not able to quickly train models that are generalized over wide sets of similar problems, or families of related tasks. This severely restricts adaptability when training many models involving many related tasks, and creates issues with training efficacy when new training data is limited or when training on novel examples.

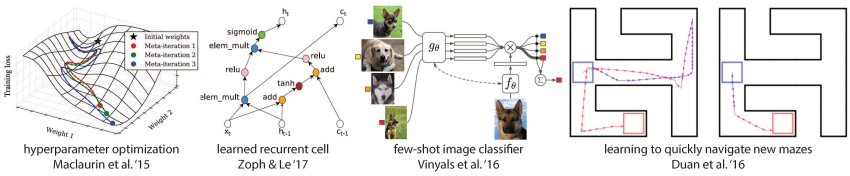


Figure 1: Various examples of meta-learning in action [3]

## 2 Background

The general goal of *meta-learning* systems is to obtain a learning algorithm that functions well when applied to training models for an entire class of problems, rather than training a singular model on one specific problem from the set of problems in a class. Such a learning algorithm should be able to train new, accurate models more quickly compared a learning algorithm defined by hand. The

creation of aforementioned learning algorithm is viewed as a learning problem in and of itself.

Here, we will focus on the *optimization-based* definition of *meta-learning* involving training these optimization algorithms used for training models. In this lecture, we will refer to the vanilla model being trained as the *base-learner* and the high-level model that is training to create an optimization algorithm for the *base-learner* as the *meta-learner*. The *meta-learner* will train to continually try to improve efficiency and minimize loss as various *base-learners* train on their differing tasks (over a family of tasks). Therefore, the *meta-learner* is learning at a higher, meta-level, learning how to best learn across a set of tasks.

We will also need to define the term of *a family of tasks*. In this instance, a family of tasks will refer to any set of tasks that can be trained to solve, with each task holding some fundamental relation to each other. In terms of the solutions of these tasks, these tasks can be visualized as interconnected by strategy or necessary "skillset" to be completed. For example, this could include a robot learning how to ride motorcycles after learning how to ride bicycles and jet-skis, new maze environments posed to a navigator that had learned to learn how to effectively navigate previous maze setups, or an experienced video game player quickly learning to succeed at a new game.

For right now, we will focus on simpler tasks, and restrict ourselves to supervised learning. This will allow us to gain a better understanding on how the meta-learner functions on fairly generic problems.

# 3 Process

The general definition of a meta-learner is as follows.[1] Assume an optimizer is to be trained for models that perform simple (supervised learning) classification of objects from sample subsets of sets of training data using standard neural networks. In other words, given the meta-training set $M$, $M_{\text{train}}$: $\{(X_{\text{i}}, c_{\text{i}})\}_{i=1}^{m}$. Each base training set within $M$, $S_{\text{train}}$: $(X_{\text{i}}, c_{\text{i}})$ can be viewed as its own sample dataset with features, inputs and labels. The goal of the meta-learner is to sequentially train the base-learner on each of the $S_{\text{train}}$. When the base-learner updates, it does so using the optimizer function trained by (and given by) the meta-learner. These updates are carried out with the meta-learner-trained optimizer-function in lieu of a generic optimization function. While the base-learner model is trained, losses for each training subset are calculated. One pass through training the base-learner with all of the sampled datasets can be referred to as a *meta-forward pass*. It is important to remember that in a minute view, the base-learner is trained in the same fashion a simple classification neural network would be trained, only with less extensive training data (as each of the many sampled datasets from $M$ will not be very extensive/thorough).

The meta-learner itself then performs its own gradient calculations from the *meta-loss*, a combination of the aforementioned losses from the base-learner during the meta-forward pass. The meta-learner then does a *meta-back pass* through each of the model training processes by the base-learner (through each

of the training subsets) in the meta-forward pass to compute *meta-gradients* for our meta-learner optimizer. These meta-losses and calculated meta-gradients are then fed into an optimizer (for the meta-learner model itself) to update the meta-learner weights. This optimizer, known as the *meta-optimizer*, **could** be another optimizer trained by an even higher-level meta-learner, but in common practice this would be an generic optimization function.

# 4 Pseudocode

Basic meta-learner to train a simple *meta-learning* optimization function:[5]

```
function train(forward_model, backward_model, optimizer,
    meta_optimizer, total_train_data, meta_epochs):
  # train a meta-learner
  # meta-train loop to train optimizer function
  for each meta_epoch:
    losses = 0
    # meta-train forward pass
    for inputs, labels in total_train_data:
      # do forward pass
      output = forward_model(inputs)
      # calculate loss
      loss = loss(output, labels)
      # add onto sum of losses
      losses += loss
      # backward pass; store gradients in forward model
      loss.backward()
      # use trained optimizer to update current model
      optimizer(forward_model,
                backward_model)
    # take meta loss, simply, as the sum of all models' losses
    meta_loss = losses
    # meta-train backward pass; store gradients in forward model
    meta_loss.backward()
    # manually update optimizer using stored
    #    gradients and meta-optimizer function
    meta_optimizer.step()
```

# 5 Conclusion

The application of learning to learn (more quickly and efficiently!) stretches far and wide, beyond just the scope of research, into real-world implementations. Experienced humans innately have the ability to learn to learn, and adapt previously learnt learning techniques to quickly learn to solve new tasks posed to them. At a high level, the ability to learn to learn could eventually prove to

be a monumental step in advancing the scope and ability of machine learning systems.

# References


[1] Ricardo Vilalta & co. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 2002.

[2] Google DeepMind. Learning to learn by gradient descent by gradient descent. *arXiv*, 2016.

[3] Chelsea Finn. Various recent meta-learning approaches. 2017.

[4] Sebastian Thrun. Learning to learn: Introduction. 1996.

[5] Thomas Wolf. From zero to research—an introduction to meta-learning. 2018.