EE5239 Project Proposal

Carter Blum and Ashley Law

Fall 2020

1 Problem Formulation

Deep learning using stochastic gradient descent is a very common method for solving both regression and classification tasks. In both settings, a task τ is defined by a set of input data points $\{x_i\}$ and corresponding regression targets $\{y_i\}$. In machine learning settings, a task is split into training and test sets, which we'll denote as $\tau^{(r)}$ and $\tau^{(s)}$ respectively. The goal is to learn some θ that parameterizes a function f such that

$$\theta = \arg\min_{\theta} \sum_i L_\tau \Big(y_i, f(x_i; \theta) \Big) \tag{1}$$

With low-bias models such as neural networks, this can require large amounts of data to achieve competitive performance. This is not always feasible in real-world settings, where data can be sparse and quick acquisition of new skills is highly prioritized. Meta-learning seeks to alleviate these issues by learning a joint model for a number of different but related tasks. Intuitively, this can be thought of as learning reasonable biases for tasks in this domain. Somewhat more formally, meta-learning can be written as trying to minimize the following loss (although the term is fairly vague, so there are many formulations).

$$\theta = \arg\min_{\theta} \sum_i E_{x_i, y_i \sim \tau_i^{(s)}} \Big[L_{\tau_i} \Big(y_i, g_i(x_i) \Big) \Big] \tag{2}$$

where

$$g_i = f(\tau_i^{(r)}, \theta)$$

is usually assumed to be some model that is fine-tuned for τ_i using the respective training data. Perhaps the most famous approach in this domain is Model-Agnostic Meta-Learning[1] (MAML), which tries to learn good starting parameters θ that allow for solid results after a limited number of steps. In the formulation above, f would correspond to training (and returning) a model for a number of steps on task $\tau_i^{(r)}$, starting from parameters θ.

This problem is closely related to continuous learning, in which a model is presented with data that is not i.i.d. This distributional shift often takes the form of different tasks, trained one after another, and the primary difficulty is finding a single model that is able to handle new tasks well while not experiencing catastrophic forgetting of old tasks.

2 Proposed Approach

As remarked in equation (1), all deep learning approaches attempt to minimize model parameters θ. The primary contribution of this work is to propose decomposing the θ into the product of a weight matrix θ_W (shared between all tasks) and a (nearly) binary mask θ_M.

More specifically, we propose taking the θ above and reformulating it as

$$\theta = \theta_W \odot \sigma(\theta_M) \tag{3}$$

where $\odot$ is the Hadamard product (element-wise multiplication) and σ is the element-wise sigmoid function, $\sigma(s) = \frac{e^s}{1+e^s}$. We note that this formulation does not lose any expressivity, as setting $\theta_W^{(ij)} = \frac{\theta^{(ij)}}{\sigma(\theta_M^{(ij)})}$ trivially results in recovering any desired θ.

The intuition here is that θ_W learns an embedding from the input space to some latent space of features that are useful for multiple tasks. However, not all features are useful for all tasks, so it may be desirable to ignore some features for some tasks. This is achieved via the binary mask, $\sigma(\theta_M)$. For most values of $\theta_M^{(ij)}$, $\sigma(\theta_M)^{(ij)}$ will be near either 0 or 1. This results in effectively allowing θ_M to turn different elements of θ_W 'on' or 'off' for each task.

One can compare this idea to the lottery ticket hypothesis [2], which showed that, even in randomly initialized networks, there usually exist subnetworks that are able to perform reasonably well on any given task. Any subnetwork of a larger network is equivalent to a selectively masked version of the larger network [6]. By this sequence of reasoning, even if θ_W is poorly optimized for any given task, we can expect that there exists some mask defined by θ_M for which the network defined by equation (3) performs well on that task.

2.1 Details

Given a set of tasks T, the proposed paradigm is as follows:

1. Randomly initialize θ_W
2. Randomly select $\tau_i \in T$ with uniform distribution

3. If τ_i has not been selected before, randomly initialize a new corresponding $\theta_M^{(i)}$
4. Train the network given by $f(\theta = \theta_W \odot \sigma(\theta_M))$ with steepest gradient descent for N steps
5. Save the updates to θ_W and $\theta_M^{(i)}$
6. Repeat from (2)

Crucially, due to the nature of the back-propagation algorithm, one update of θ_W and $\theta_M^{(i)}$ has the same time complexity as ordinary backpropagation of θ. This is in contrast with other methods such as MAML [1], Meta-gradient Reinforcement Learning [5], MetaGenRL [3] and others, which often require computing the Hessian of the weights.

During test time, if the testing task has been seen before, the corresponding mask weights $\theta_M^{(i)}$ can be looked up from training time. If the task has not been seen before, it's possible to use a linear combination of weights from existing strategies, as is done in [4] for continual learning.

One notable difficulty is that the gradient $\nabla_s \sigma(s) = (1 - \sigma(s))\sigma(s)$, gets very small as $\sigma(s)$ gets close to either 0 or 1. Ideally, we would like $\sigma(s)$ to take on values that are close enough to 0 & 1 to turn values on and off, but not so small as to make the gradient near zero, since that would result in very slow convergence. Luckily, this issue is largely rectified by adding an L_2 regularizer to the weights of θ_W, which encourages the weights to take on small values. Because the gradient of σ is greatest at $s = 0$, this encourages the weights to stay roughly in the zone where the gradient flows nicely. One possible source of difficulty is that this may cause the values of $\sigma(s)$ to stay around 0.5, which minimizes the regularizer, so it is important to make sure that the weight of the regularizer is tuned correctly.

3 Experiments

3.1 Proof of Concept: Linear Regression

This experiment checks the situation where a linear model of the form $y = (w_W \odot w_M)^T x + (b_W \odot b_M)$ is correct and attempts to see if our method can find it. Namely, we test linear regression on 5 variables for each task. For each task, the weight of the k^{th} variable is either w_k or 0, as is the bias, where w_k are shared between all tasks. In this scenario, our proposed model is perfectly capture the dynamics of the system, so it works as a simple debugging example and proof-of-concept.

3.2 Simple Generalization: Linear Regression with Sinusoidal Output

In this experiment, the data takes the form $y = \text{sine}\Big((w_W \odot w_M)^T x + (b_W \odot b_M)\Big)$. As in the above experiment, w and b vary from task to task according to the previously described rules. This should require a 2+ layer network, where the optimal values for the first layer are the same as the optimal values from the proof of concept, and the optimal values for the later layers are shared between all tasks. This test should demonstrate whether or not the method is able to identify when all weights of a layer should be shared between all functions.

3.3 Sinusoidal Regression without Linear Assumption

In this experiment, each task consists of learning linear regression on a sinusoidal function given by $\text{sine}(w^T x + b_1) + b_2$. Unlike the previous task, there is no relation between the values of w, b_1 and b_2 from one task to another. This experiment is intended to demonstrate whether or not the method is able to learn dynamic-valued weights.

3.4 Sinusoidal Regression with Occasional Cosine Replacement

This experiment is another proof of concept. It is the same as the previous experiment (3.3), except that sine is occasionally replaced with cos, which is equivalent to sometimes adding a bias of $\frac{\pi}{2}$. If the model is successful, we should see the bias term take on a value of $\frac{\pi}{2}$ in the cos cases, with the corresponding mask being nearly 0 or 1 for each task.

3.5 Logistic Regression

In this experiment, logistic regression is attempted, with the same assumptions as the first experiment (3.1). This is to test whether the method is compatible with classification as well as regression, which it theoretically should be.

3.6 Image Classification - MNIST

In this experiment, a convolutional neural network (CNN) is used to classify black & white images of digits, as used in class. For this particular example, there is no meta-learning component; it is simply to examine whether our method performs similarly to a baseline CNN.

3.7 Image Classification - Colored MNIST

This experiment is the same as MNIST, except that there are now a number of tasks that each take the black background in the MNIST digits and replace it

with a background of a random color. The intent of this experiment is to test whether the algorithm is able to scale to non-trivial domains.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[3] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[4] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *arXiv preprint arXiv:2006.14769*, 2020.

[5] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, pages 2396–2407, 2018.

[6] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3597–3607, 2019.