CS 194-129 Deep Neural Networks
Spring 2018 Finn Lecture 23

1 Lecture

Motivating question: *do neural networks need a large dataset?*

After all, humans can learn just from a few examples. This is probably because humans don't start from scratch; they are continually accumulating experience and acquiring common sense.

How do we move away from training neural networks from scratch? How can we develop systems that accumulate prior experience (so as to not require massive amounts of data for every individual task)?

Here's an idea: we can have our systems learn the *structure* underlying previous data, tasks, or experiences. Then we can use that structure to quickly learn new tasks.

One way to do this: *meta-learning* (learning to learn).

Meta-Learning

Some types of meta-learning:

- **AutoML:** automating the learning process
  - Hyperparameter optimization, architecture search, ...
- **One-shot/few-shot learning:** learning how to learn from small amounts of data
- **Meta-RL:** learning an RL algorithm (meta-learning in the context of RL)
- **Meta-X learning:** we can use meta-learning on top of anything (imitation, unsupervised, ...)

This lecture will focus on *few-shot learning.*

Applications of Learning to Learn

- **Few-shot image classification:** given only a few examples of each class, classify new images
- **Fast reinforcement learning:** given a small amount of experience, learn to solve a task

Problem Formulation

In *supervised learning*, we map inputs x to outputs y via the parameterized model $y = f(x; \theta)$. We learn this through a dataset of input/output pairs $\{(x, y)_i\}$.

In *meta-supervised learning*, our inputs consist of a training dataset $\mathcal{D}_{\text{train}} = \{(x, y)_{1:K}\}$ (K input/output pairs for the K-shot learning case). We're also given a new unlabeled data point x_{test} that we want to be able to make predictions y_{test} about. Thus our goal is to learn the mapping $y_{\text{test}} = f(\mathcal{D}_{\text{train}}, x_{\text{test}}; \theta)$. We learn this through a dataset of datasets $\{\mathcal{D}_i\}$, where each dataset $\mathcal{D}_i : \{(x, y)_j\}$ consists of (x, y) pairs.

(Each data point is a dataset; we're *learning how to learn* from these datasets.) This view is useful because it reduces the problem to the design and optimization of f.

Solution Class: Metric Learning Approach

If performing image classification, we can take the image x_{test} we want to classify and pair it with each of the images in our dataset $\mathcal{D}_{\text{train}}$. Then we can pick the class that is the most similar.

Note: if we view meta-learning as mapping from a training dataset $\mathcal{D}_{\text{train}}$ and test input x_{test} to the test output y_{test}, the mapping that we're learning (f) corresponds to nearest neighbors.

Metric-wise, an l_2 distance in pixel space won't cut it. We should learn how to compare (using the data that we have on all of the image classes); we should learn a feature space in which we can perform comparisons and also learn a distance metric with which to perform those comparisons.

- One approach: we can train a Siamese network to predict whether two images belong to the same class. Then, at test time, we can compare the image x_{test} to all of the images in our training dataset $\mathcal{D}_{\text{train}}$ and output the class of the image that's closest according to the Siamese network.
- Another approach: make the train and test conditions match, so that we can more effectively make comparisons. This can be done through a "matching network," trained for the classification problem that's going to be seen during test time.

Solution Class: Direct Black Box Approach

$$\mathcal{D}_{\text{train}}, x_{\text{test}} \rightarrow y_{\text{test}}$$

This is the meta-supervised learning problem. Why not train a model to directly approximate this mapping? For example, a recurrent neural network f (since $\mathcal{D}_{\text{train}}$ might have a variable number of data points) which takes in $\mathcal{D}_{\text{train}}$ (i.e. (x_i, y_i) data) along with x_{test}, and predicts y_{test}.

$$y_{\text{test}} = f(\mathcal{D}_{\text{train}}, x_{\text{test}}; \theta)$$

We can also learn an optimizer, i.e.

$$y_{\text{test}} = f(x_{\text{test}}; g(\mathcal{D}_{\text{train}}; \theta))$$

where g is a neural network that ingests the training dataset and outputs the weights for f.

Solution Class: Gradient-Based Approach

In fine-tuning, we assume we have some pre-trained parameters θ and some training data for a new task, and we run gradient descent starting from our pre-trained parameters:

$$\theta \rightarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{train}}(\theta)$$

This works well, but it will only go so far. Recall that the training dataset is only a few data points. Can we find a set of pre-trained parameters such that when we fine-tune with a small amount of data, we can effectively generalize from that data? (Essentially, we should account for the fact that we want to run fine-tuning in the meta-training process.)

Yep. We can optimize for a set of parameters θ, such that when we run a few steps of gradient descent on the training data points, we do well on test data points. We'll do this optimization across a wide range of different tasks (so that when we see a dataset for a new task, we can generalize effectively with only a small amount of data).

$$\min_\theta \sum_{\text{task } i} \mathcal{L}^i_{\text{test}}(\theta - \alpha \nabla_\theta \mathcal{L}^i_{\text{train}}(\theta))$$

The key idea is to train over many tasks, in order to learn a parameter vector θ that transfers well via fine-tuning with only a small amount of data. (Note: two objectives, $\mathcal{L}_{\text{test}}$ and $\mathcal{L}_{\text{train}}$.)

This algorithm is called MAML (model-agnostic meta-learning); it's agnostic to the function and objective that we use, as long as both are amenable to gradient-based optimization.

MAML can be viewed as running gradient descent at test time w.r.t. the training dataset, then making predictions with those new parameters:

$$y_{\text{test}} = f(x_{\text{test}}; \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{D}_{\text{train}}))$$

If it's not good enough, we can collect more data and continue taking gradient steps with that data.

For sufficiently deep f, the MAML function can approximate any function of $\mathcal{D}_{\text{train}}$ and x_{test}.

Takeaways

- **Meta-learning** can be seen as learning the function $\mathcal{D}_{\text{train}}, x_{\text{test}} \to y_{\text{test}}$.
- **Metric learning** (learning to compare test-time data points with training data points) works well for few-shot classification.
- A **direct black box approach** (learning an RNN that maps from $\mathcal{D}_{\text{train}}, x_{\text{test}}$ to y_{test}) is very general and powerful, but data inefficient and poor at extrapolation [to data outside of the distribution it was trained on].
- **Gradient-based approaches** (e.g. MAML) are general and good at extrapolation, but somewhat difficult to optimize (we have a gradient w.r.t. θ in the objective and are trying to optimize θ, so there are second derivatives in the optimization procedure).

Humans and deep networks can learn from a small amount of data, but only with a large amount of previous experience. So we do need a lot of data, but if we can amortize the experience across tasks, we can learn more efficiently for a new task.

Humans still require a lot of data, but not for each and every task (they reuse experience).
Data is *amortized* across tasks.

Open Questions / Problems

- Where do the tasks come from? (How do we define them? Can ML algorithms create their own tasks?)
- How does learning performance change, *theoretically*, across similar but extrapolated tasks?
- Can we create meta-RL algorithms that are consistent and universal?
- We need benchmarks, datasets, and environments for meta-learning (more than Omniglot).
- Can we use meta-learning for *continual learning*?