Introduction to meta-learning

Brooks Paige
ML Seminar

Term 2, 2021

What is meta-learning?

Most machine learning models are

- . . . trained to solve a single, specific task

What is meta-learning?

Most machine learning models are

- . . . trained to solve a single, specific task
- . . . from a single, fixed dataset

What is meta-learning?

Most machine learning models are

- . . . trained to solve a single, specific task
- . . . from a single, fixed dataset
- . . . using a specific learning algorithm.

What is meta-learning?

Meta-learning asks how we can **learn to learn**:

- If we have already solved n problems, shouldn't solving the $(n+1)^{\text{th}}$ be somehow easier?

What is meta-learning?

Meta-learning asks how we can **learn to learn**:

- If we have already solved n problems, shouldn't solving the $(n+1)^{\text{th}}$ be somehow easier?
- How can our learning algorithms gain "experience" over the course of many different but related learning problems?

What is meta-learning?

Meta-learning asks how we can **learn to learn**:

- If we have already solved n problems, shouldn't solving the $(n+1)^{\text{th}}$ be somehow easier?
- How can our learning algorithms gain "experience" over the course of many different but related learning problems?
- We no longer need to hand-design learning rules, or representations. Why do we hand-design learning algorithms?

End-to-end learning in computer vision

Replacing hand-engineered features with end-to-end learning:

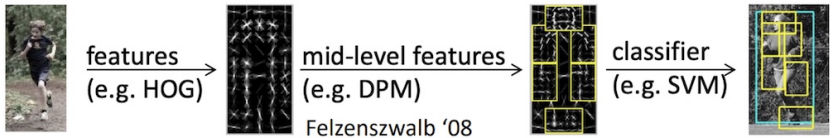
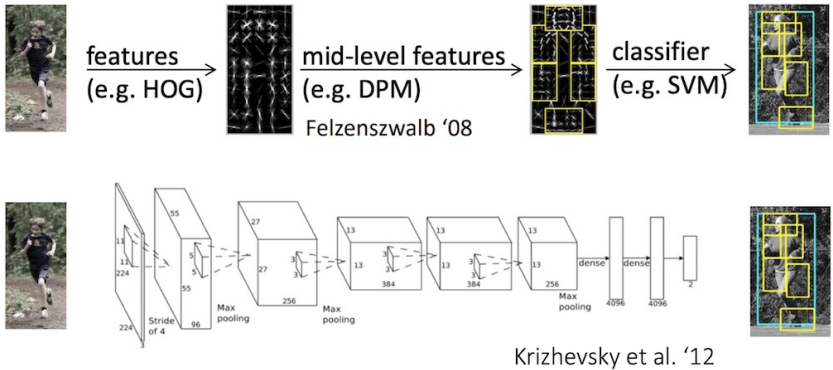


Figure: Chelsea Finn (cs330)

End-to-end learning in computer vision

Replacing hand-engineered features with end-to-end learning:



There's still a lot of hard-coded inductive bias in that architecture — and the learning algorithm is pre-defined

Figure: Chelsea Finn (cs330)

Challenges

- Deep learning (and machine learning generally) requires huge amounts of labelled training data
- This is acceptable for solving isolated problems, but has a large real-world expense

Challenges

- Deep learning (and machine learning generally) requires huge amounts of labelled training data
- This is acceptable for solving isolated problems, but has a large real-world expense
- Are giant training sets really the path to solving "AI"?
- Never going to be appropriate in low-data regimes: personalized medicine, personalized recommendations, translating rare languages, ...

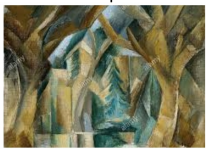
Challenges

- Deep learning (and machine learning generally) requires huge amounts of labelled training data
- This is acceptable for solving isolated problems, but has a large real-world expense
- Are giant training sets really the path to solving "AI"?
- Never going to be appropriate in low-data regimes: personalized medicine, personalized recommendations, translating rare languages, ...
- Humans don't have this problem — we learn quickly!

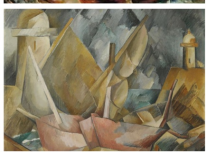
Learning when we don't have much data

training data

Braque



Cezanne



test datapoint

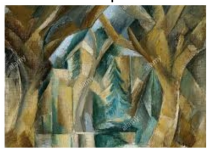
By Braque or Cezanne?

Figure: Chelsea Finn (cs330)

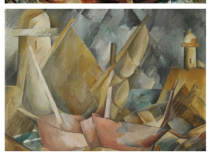
Learning when we don't have much data

training data

Braque



Cezanne



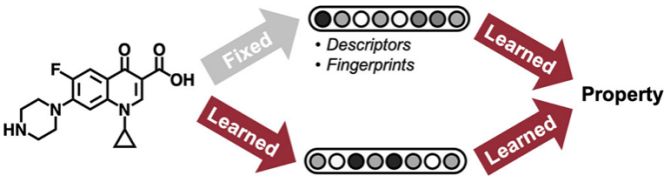
test datapoint

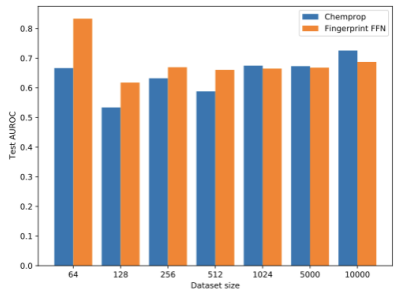
By Braque or Cezanne?

Why are you able to do this? **Prior experience**.

Figure: Chelsea Finn (cs330)

Learning when data is expensive to collect





Molecular property prediction: on small datasets, hand-crafted features can still outperform deep learning.

[MSc project last year: addressing this with meta-learning!]

Yang, Swanson, et al. (2020); Pappu & Paige (2020)

Difficulties in definitions

The term "meta-learning" is used in many different communities, to mean many different things, and with varying terminology.

In general, for machine learning problems, **more data $\Rightarrow$ better predictions**.

Difficulties in definitions

The term "meta-learning" is used in many different communities, to mean many different things, and with varying terminology.

In general, for machine learning problems, **more data $\Rightarrow$ better predictions**.

The goal in meta-learning: **more tasks $\Rightarrow$ better learning algorithms**.

Learning to learn: general framework

1. In "conventional" machine learning, a **model** improves its performance (e.g. its predictions) as it acquires more instances data

Learning to learn: general framework

1. In "conventional" machine learning, a **model** improves its performance (e.g. its predictions) as it acquires more instances data
   - This machine learning model is called our **base learner**, which is trained to solve a specific task (e.g. a single classification problem) using an *inner* or *base* learning algorithm.

Learning to learn: general framework

1. In "conventional" machine learning, a **model** improves its performance (e.g. its predictions) as it acquires more instances data
   - This machine learning model is called our **base learner**, which is trained to solve a specific task (e.g. a single classification problem) using an *inner* or *base* learning algorithm.
2. In meta-learning, a **learning algorithm** improves its performance over the course of multiple "learning episodes"

Learning to learn: general framework

1. In "conventional" machine learning, a **model** improves its performance (e.g. its predictions) as it acquires more instances data
   - This machine learning model is called our **base learner**, which is trained to solve a specific task (e.g. a single classification problem) using an *inner* or *base* learning algorithm.
2. In meta-learning, a **learning algorithm** improves its performance over the course of multiple "learning episodes"
   - This is called our **meta-learning** or *outer learning* algorithm. It is responsible for training the inner learning algorithm!

Learning to learn: general framework

1. In "conventional" machine learning, a **model** improves its performance (e.g. its predictions) as it acquires more instances data
   - This machine learning model is called our **base learner**, which is trained to solve a specific task (e.g. a single classification problem) using an *inner* or *base* learning algorithm.
2. In meta-learning, a **learning algorithm** improves its performance over the course of multiple "learning episodes"
   - This is called our **meta-learning** or *outer learning* algorithm. It is responsible for training the inner learning algorithm!
   - A **learning episode** consists of a base algorithm, a model, and some measure of performance (e.g. generalization error, or convergence speed). Multiple learning episodes taken together provide the training data for the outer learning algorithm.

Terminology

In our base learner, for each particular task or episode,

- we train on our **training set**,
- and evaluate on a **test set**.

Terminology

In our base learner, for each particular task or episode,

- we train on our **training set**,
- and evaluate on a **test set**.

For meta-learning an algorithm A or its parameters ω,

- we train on a **meta-training set**, composed of many different tasks,
- and evaluate on a **meta-test set** of unseen tasks or data.

Terminology

In our base learner, for each particular task or episode,

- we train on our **training set**, *[support set]*
- and evaluate on a **test set**. *[query set]*

For meta-learning an algorithm A or its parameters ω,

- we train on a **meta-training set**, composed of many different tasks, *[training set]*
- and evaluate on a **meta-test set** of unseen tasks or data. *[test set]*

[alternative terminology]

Learning and meta-learning

Anatomy of a typical ML learning scenario:

- Dataset $\mathcal{D}$, e.g. $\mathcal{D} = \{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}$ for supervised learning
- Model specification to learn, e.g. parameters $\boldsymbol{\theta}$ of a predictor $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$
- Loss $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}, \omega)$ which we minimize, e.g. error between true labels y and predicted labels $\hat{y}$.

Learning and meta-learning

Anatomy of a typical ML learning scenario:

- Dataset $\mathcal{D}$, e.g. $\mathcal{D} = \{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}$ for supervised learning
- Model specification to learn, e.g. parameters $\boldsymbol{\theta}$ of a predictor $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$
- Loss $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}, \omega)$ which we minimize, e.g. error between true labels y and predicted labels $\hat{y}$.

This definition of the loss makes explicit a dependence on ω, which are all the parameters of the learning process, including

- the learning algorithm A,

Learning and meta-learning

Anatomy of a typical ML learning scenario:

- Dataset $\mathcal{D}$, e.g. $\mathcal{D} = \{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}$ for supervised learning
- Model specification to learn, e.g. parameters $\boldsymbol{\theta}$ of a predictor $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$
- Loss $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}, \omega)$ which we minimize, e.g. error between true labels y and predicted labels $\hat{y}$.

This definition of the loss makes explicit a dependence on ω, which are all the parameters of the learning process, including

- the learning algorithm A,
- the function class or architecture of the model f,

Learning and meta-learning

Anatomy of a typical ML learning scenario:

- Dataset $\mathcal{D}$, e.g. $\mathcal{D} = \{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}$ for supervised learning
- Model specification to learn, e.g. parameters $\boldsymbol{\theta}$ of a predictor $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$
- Loss $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}, \omega)$ which we minimize, e.g. error between true labels y and predicted labels $\hat{y}$.

This definition of the loss makes explicit a dependence on ω, which are all the parameters of the learning process, including

- the learning algorithm A,
- the function class or architecture of the model f,
- the explicit form of the loss function,

Learning and meta-learning

Anatomy of a typical ML learning scenario:

- Dataset $\mathcal{D}$, e.g. $\mathcal{D} = \{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}$ for supervised learning
- Model specification to learn, e.g. parameters $\boldsymbol{\theta}$ of a predictor $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$
- Loss $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}, \omega)$ which we minimize, e.g. error between true labels y and predicted labels $\hat{y}$.

This definition of the loss makes explicit a dependence on ω, which are all the parameters of the learning process, including

- the learning algorithm A,
- the function class or architecture of the model f,
- the explicit form of the loss function,
- ...

Cost per episode

- A **task** $\mathcal{T}$ is defined as a dataset $\mathcal{D}$ and a loss function $\mathcal{L}$, i.e. $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$.

Cost per episode

- A **task** $\mathcal{T}$ is defined as a dataset $\mathcal{D}$ and a loss function $\mathcal{L}$, i.e. $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$.
- An **episode** is defined by minimizing the loss $\mathcal{L}$ on a single dataset $\mathcal{D} = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$. For example, assuming a probabilistic model with parameters $\boldsymbol{\theta}$, and a classification problem, this loss is

$$\mathcal{L}(\mathcal{D}; \omega) = \frac{1}{|\mathcal{D}_{test}|} \sum_{\{\mathbf{x}_i, y_i\} \in \mathcal{D}_{test}} -\log p(y_i | \mathbf{x}_i, \boldsymbol{\theta}^\star)$$

Cost per episode

- A **task** $\mathcal{T}$ is defined as a dataset $\mathcal{D}$ and a loss function $\mathcal{L}$, i.e. $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$.
- An **episode** is defined by minimizing the loss $\mathcal{L}$ on a single dataset $\mathcal{D} = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$. For example, assuming a probabilistic model with parameters $\boldsymbol{\theta}$, and a classification problem, this loss is

$$\mathcal{L}(\mathcal{D}; \omega) = \frac{1}{|\mathcal{D}_{test}|} \sum_{\{\mathbf{x}_i, y_i\} \in \mathcal{D}_{test}} -\log p(y_i | \mathbf{x}_i, \boldsymbol{\theta}^\star)$$

- The value of $\boldsymbol{\theta}^\star$ for a particular episode is found by running the base learning algorithm A, with parameters ω; i.e.

$$\boldsymbol{\theta}^\star = A(\mathcal{L}, \mathcal{D}_{train}; \omega).$$

Distributions of tasks

One way to formalize meta-learning is to learn an algorithm that performs well across a distribution of tasks, which we denote $p(\mathcal{T})$.

Distributions of tasks

One way to formalize meta-learning is to learn an algorithm that performs well across a distribution of tasks, which we denote $p(\mathcal{T})$.

- We can then write the overall meta-learning objective as

$$\min_{\omega} \mathcal{L}^{meta}(\omega) = \mathbb{E}_{\{\mathcal{L},\mathcal{D}\}\sim p(\mathcal{T})} \left[\mathcal{L}(\mathcal{D};\omega)\right].$$

Distributions of tasks

One way to formalize meta-learning is to learn an algorithm that performs well across a distribution of tasks, which we denote $p(\mathcal{T})$.

- We can then write the overall meta-learning objective as

$$\min_{\omega} \mathcal{L}^{meta}(\omega) = \mathbb{E}_{\{\mathcal{L},\mathcal{D}\}\sim p(\mathcal{T})} \left[\mathcal{L}(\mathcal{D};\omega)\right].$$

- If we have M initial example tasks $\mathcal{T}^1, \ldots, \mathcal{T}^M \sim p(\mathcal{T})$, then our target optimal $\omega^\star$ are

$$\omega^\star = \arg\min_{\omega} \sum_{m=1}^{M} \mathcal{L}^m(\mathcal{D}^m;\omega).$$

Distributions of tasks

One way to formalize meta-learning is to learn an algorithm that performs well across a distribution of tasks, which we denote $p(\mathcal{T})$.

- We can then write the overall meta-learning objective as

$$\min_{\omega} \mathcal{L}^{meta}(\omega) = \mathbb{E}_{\{\mathcal{L},\mathcal{D}\}\sim p(\mathcal{T})} \left[\mathcal{L}(\mathcal{D};\omega)\right].$$

- If we have M initial example tasks $\mathcal{T}^1, \ldots, \mathcal{T}^M \sim p(\mathcal{T})$, then our target optimal $\omega^\star$ are

$$\omega^\star = \arg\min_{\omega} \sum_{m=1}^{M} \mathcal{L}^m(\mathcal{D}^m;\omega).$$

- In the meta-test phase, we can then make predictions on new training sets $\mathcal{D}'_{train}$ by using $\omega^\star$, with $\boldsymbol{\theta}^\star = A(\mathcal{L}', \mathcal{D}'_{train}; \omega^\star)$.

Distributions of tasks

One way to formalize meta-learning is to learn an algorithm that performs well across a distribution of tasks, which we denote $p(\mathcal{T})$.

- We can then write the overall meta-learning objective as

$$\min_{\omega} \mathcal{L}^{meta}(\omega) = \mathbb{E}_{\{\mathcal{L},\mathcal{D}\}\sim p(\mathcal{T})} \left[\mathcal{L}(\mathcal{D};\omega)\right].$$

- If we have M initial example tasks $\mathcal{T}^1, \ldots, \mathcal{T}^M \sim p(\mathcal{T})$, then our target optimal $\omega^\star$ are

$$\omega^\star = \arg\min_{\omega} \sum_{m=1}^{M} \mathcal{L}^m(\mathcal{D}^m;\omega).$$

- In the meta-test phase, we can then make predictions on new training sets $\mathcal{D}'_{train}$ by using $\omega^\star$, with $\boldsymbol{\theta}^\star = A(\mathcal{L}', \mathcal{D}'_{train}; \omega^\star)$.

In practice, this often corresponds to solving a two-level optimization problem.

What are different tasks?

- Image classification, but on **previously unseen classes**
- Image classification, but in **new lighting conditions** (or **weather conditions**, etc)
- Same objective, but on new data: e.g. customized to a **different person**
- Same conceptual goal, but **different loss** or objective function
- Machine translation, but on **new languages** or with **new vocabulary**
- ...

What are different tasks?

- Image classification, but on **previously unseen classes**
- Image classification, but in **new lighting conditions** (or **weather conditions**, etc)
- Same objective, but on new data: e.g. customized to a **different person**
- Same conceptual goal, but **different loss** or objective function
- Machine translation, but on **new languages** or with **new vocabulary**
- ...

Different tasks need to share at least some structure. But actually, many things we might want to do using machine learning have a lot of shared structure. (Tasks aren't "random"!)

Some meta-learning-adjacent fields

A lot of different subareas in machine learning are similar to meta-learning.

Some meta-learning-adjacent fields

A lot of different subareas in machine learning are similar to meta-learning.

- **Multi-task** learning also considers a dataset of many different tasks at training time. However, it aims to learn a set of parameters $\boldsymbol{\theta}$ which are appropriate for **all** tasks (instead of learning meta-parameters ω). It's also generally assumed all tasks are simultaneously accessible.

Some meta-learning-adjacent fields

A lot of different subareas in machine learning are similar to meta-learning.

- **Multi-task** learning also considers a dataset of many different tasks at training time. However, it aims to learn a set of parameters $\boldsymbol{\theta}$ which are appropriate for **all** tasks (instead of learning meta-parameters ω). It's also generally assumed all tasks are simultaneously accessible.

- **Transfer learning** aims to take knowledge from previous tasks, and use it to accelerate future tasks. This typically involves fitting $\boldsymbol{\theta}$ to previous task(s), and then using those parameters as initialization on a new tasks. The main difference from meta-learning is that there is **no meta-objective** — the previous tasks are not trained in a way which is aware that $\boldsymbol{\theta}$ will be used on new data later.

Some meta-learning-adjacent fields

A lot of different subareas in machine learning are similar to meta-learning.

- **AutoML** aims to automate large parts of the machine learning pipeline — from algorithm selection, to dataset transforms, to neural network architectures. It's generally a broader topic than meta-learning, but meta-learning can be useful.

  For example, searching for neural architectures is slow and expensive, so it's desirable to find architectures which are generally helpful across a wide variety of problems, and to leverage knowledge that existing architectures (and algorithms) are useful on particular datasets.

Three things to look for when reading meta-learning papers

Hospedales et al. (2020) set out the following useful taxonomy for meta-learning:

https://arxiv.org/abs/2004.05439

Three things to look for when reading meta-learning papers

Hospedales et al. (2020) set out the following useful taxonomy for meta-learning:

- **Meta-representation**: **What** is being meta-learned? This could be (for example) an initial value of model parameters, but could be anything useful for learning subsequent tasks.

https://arxiv.org/abs/2004.05439

Three things to look for when reading meta-learning papers

Hospedales et al. (2020) set out the following useful taxonomy for meta-learning:

- **Meta-representation**: **What** is being meta-learned? This could be (for example) an initial value of model parameters, but could be anything useful for learning subsequent tasks.
- **Meta-optimizer**: **How** do we do the outer-level optimization of ω? This might be gradient descent through the inner optimization of $\boldsymbol{\theta}$, or could be something else entirely (RL, random search, ...).

https://arxiv.org/abs/2004.05439

Three things to look for when reading meta-learning papers

Hospedales et al. (2020) set out the following useful taxonomy for meta-learning:

- **Meta-representation**: **What** is being meta-learned? This could be (for example) an initial value of model parameters, but could be anything useful for learning subsequent tasks.
- **Meta-optimizer**: **How** do we do the outer-level optimization of ω? This might be gradient descent through the inner optimization of $\boldsymbol{\theta}$, or could be something else entirely (RL, random search, ...).
- **Meta-objective**: **Why** are we doing any of this? The meta-objective corresponds to the choice of task distribution and meta-loss.

https://arxiv.org/abs/2004.05439

Amortization

One other aspect to keep an eye out for is the amount of **amortization**.

- Idea of amortizing runtime costs: pay a large cost up front, in exchange for fast computation later
- Popular e.g. for running Bayesian inference repeatedly using the same model, on different datasets
- Different meta-learning methods have different degrees of amortization (e.g. learning an initialization less so than generating parameters directly)

Case study: few-shot learning

Learning from a single example

Which of these is like the other?

Lake et al. (2015)

Learning from a single example

Which of these is like the other?

Lake et al. (2015)

Few-shot learning as meta-learning

Suppose we want to know how to classify an image from very few examples (maybe only one!)

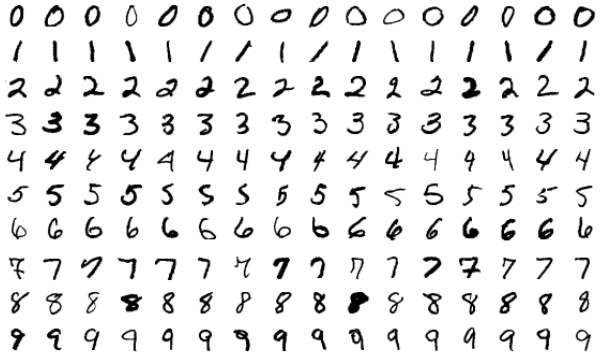
- We want to learn an algorithm A that outputs good parameters $\boldsymbol{\theta}$ for a model M given a very small training dataset $\mathcal{D}$
- If we collect many such few-shot learning tasks, we can learn the algorithm A directly from data
- This will require a suitable **meta-objective**.

Most machine learning datasets

MNIST dataset

- 10 classes
- 60,000 examples

$\approx$ 6,000 images each



Most real-world datasets?

Omniglot dataset

- 1,623 classes
- . . . across 50 alphabets

. . . with only 20 images each

Lake et al. (2015)

Most real-world datasets?

Omniglot dataset

- 1,623 classes
- . . . across 50 alphabets

. . . with only 20 images each

[“transposed” MNIST]

Lake et al. (2015)

Example task

Here's an example task.

- Suppose we are shown 5 images, one each from five different classes:

Example task

Here's an example task.

- Suppose we are shown 5 images, one each from five different classes:

- Then, we are asked to predict the correct class of two brand-new test images:

Example task

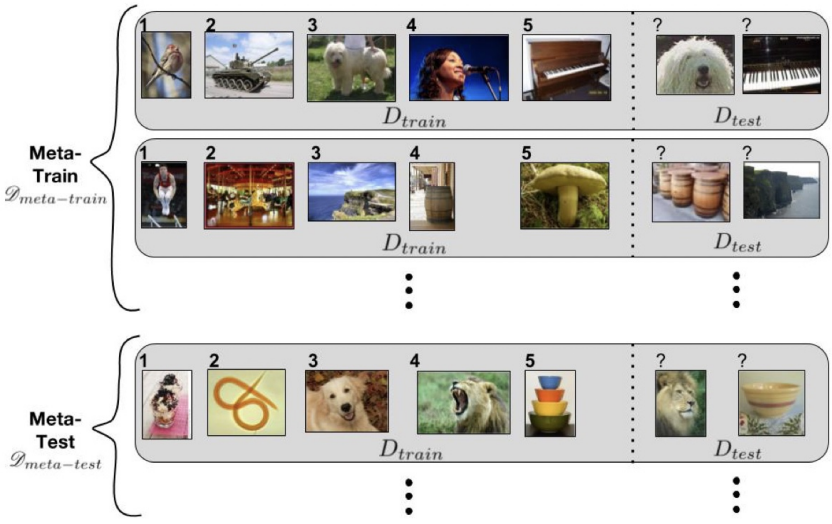
Here's an example task.

- Suppose we are shown 5 images, one each from five different classes:

- Then, we are asked to predict the correct class of two brand-new test images:

These test images are from classes we've never seen before these five new images.

Mini-ImageNet



Vinyals et al. (2016), Ravi & Larochelle (2017)

Choice of meta-representation

One approach is to start from an existing machine learning algorithm, add extra parameters, and learn those with meta-learning. We will look at three of these for classification:

- kNN or kernel classifier $\rightarrow$ Matching networks, Vinyals et al. (2016)
- Gaussian classifier $\rightarrow$ Prototypical networks, Snell et al. (2017)
- Gradient descent $\rightarrow$ MAML, Finn et al. (2017)

Slide adapted from Hugo Larochelle

Choice of meta-representation

One approach is to start from an existing machine learning algorithm, add extra parameters, and learn those with meta-learning. We will look at three of these for classification:

- kNN or kernel classifier $\rightarrow$ Matching networks, Vinyals et al. (2016)
- Gaussian classifier $\rightarrow$ Prototypical networks, Snell et al. (2017)
- Gradient descent $\rightarrow$ MAML, Finn et al. (2017)

It's also possible to take a completely black-box approach, where the entire algorithm is learned by a neural network:

- Memory-augmented neural networks, Santoro et al. (2016)
- SNAIL, Mishra et al. (2018)

Slide adapted from Hugo Larochelle

Matching networks

Classifier for new $\mathbf{x}'$:

$$\hat{y} = \sum_{i=1}^{M} a(\mathbf{x}', \mathbf{x}_i) y_i$$

where $\{\mathbf{x}_i, y_i\}$ are the train set $\mathcal{D}_{train}$ and $\mathbf{x}' \in \mathcal{D}_{test}$. We want to learn $a(\dots)$.

Vinyals et al. (2016)

Matching networks

Classifier for new $\mathbf{x}'$:

$$\hat{y} = \sum_{i=1}^{M} a(\mathbf{x}', \mathbf{x}_i) y_i$$

where $\{\mathbf{x}_i, y_i\}$ are the train set $\mathcal{D}_{train}$ and $\mathbf{x}' \in \mathcal{D}_{test}$. We want to learn $a(\dots)$.

Advantages:

- Non-parametric method
- Can recover nearest neighbors and kernel density estimators
- Can also function as an associative memory that "points" to nearest training examples

Vinyals et al. (2016)

Matching networks

Classifier for new $\mathbf{x}'$:

$$\hat{y} = \sum_{i=1}^{M} a(\mathbf{x}', \mathbf{x}_i) y_i$$

where $\{\mathbf{x}_i, y_i\}$ are the train set $\mathcal{D}_{train}$ and $\mathbf{x}' \in \mathcal{D}_{test}$. We want to learn $a(\dots)$.

Idea: parameterize this in terms of an attention mechanism

$$a(\mathbf{x}', \mathbf{x}_i) = \frac{\exp\{c(f(\mathbf{x}'), g(\mathbf{x}_i))\}}{\sum_{j=1}^{M} \exp\{c(f(\mathbf{x}'), g(\mathbf{x}_j))\}},$$

where f, g are learnable functions, and c is the cosine distance

$$c(\mathbf{f}, \mathbf{g}) = 1 - \frac{\mathbf{f}^\top \mathbf{g}}{\|\mathbf{f}\| \|\mathbf{g}\|}.$$

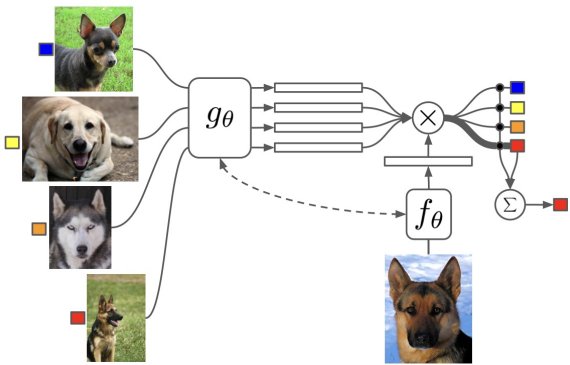
Vinyals et al. (2016)

Matching networks

One trick: instead of just $f_{\boldsymbol{\theta}}(\mathbf{x}')$ and $g_{\boldsymbol{\theta}}(\mathbf{x}_i)$, also provide access to an embedding of the entire training set, e.g. learning

- $f_{\boldsymbol{\theta}}(\mathbf{x}') = f_{\boldsymbol{\theta}}(\mathbf{x}', \mathcal{D}_{train})$
- $g_{\boldsymbol{\theta}}(\mathbf{x}_i) = g_{\boldsymbol{\theta}}(\mathbf{x}_i, \mathcal{D}_{train})$

where both are parameterized by sequence models (e.g. LSTMs).



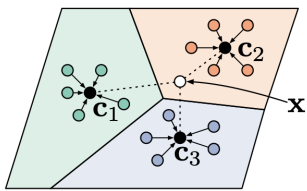
Vinyals et al. (2016)

Prototypical networks

Learn an embedding $f_{\boldsymbol{\theta}}(\mathbf{x})$, and compute class centroids $\mathbf{c}_k$ as

$$\mathbf{c}_k = \frac{1}{|\mathcal{S}_k|} \sum_{\{\mathbf{x}_i, y_i\} \in \mathcal{S}_k} f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

where $\mathcal{S}_k \subset \mathcal{D}_{train}$ is the set of training points with class label $y_i = k$.



Classifier for new $\mathbf{x}'$:

$$p(y = k|\mathbf{x}') = \frac{\exp\{-d(f_{\boldsymbol{\theta}}(\mathbf{x}'), \mathbf{c}_k)\}}{\sum_{j=1}^{K} \exp\{-d(f_{\boldsymbol{\theta}}(\mathbf{x}'), \mathbf{c}_j))\}}.$$

where d is a distance function.

Snell et al. (2017)

Prototypical networks

Since class predictions only depend on the means of the embeddings $\mathbf{c}_k$, this is much simpler than matching networks — there is no need for a complex embedding of the training sets.

- With a Euclidean distance $d(\cdot, \cdot) = \|f_{\boldsymbol{\theta}}(\mathbf{x}') - \mathbf{c}_k\|^2$, this is interpretable as learning a linear model in the embedding space $f_{\boldsymbol{\theta}}(\mathbf{x})$:

$$\begin{aligned} -\|f_{\boldsymbol{\theta}}(\mathbf{x}') - \mathbf{c}_k\|^2 &= -f_{\boldsymbol{\theta}}(\mathbf{x}')^\top f_{\boldsymbol{\theta}}(\mathbf{x}') + 2\mathbf{c}_k^\top f_{\boldsymbol{\theta}}(\mathbf{x}') - \mathbf{c}_k^\top \mathbf{c}_k \\ &= \underbrace{(2\mathbf{c}_k)}_{\mathbf{w}_k}^\top f_{\boldsymbol{\theta}}(\mathbf{x}') - \underbrace{\mathbf{c}_k^\top \mathbf{c}_k}_{b_k} + \text{const} \end{aligned}$$

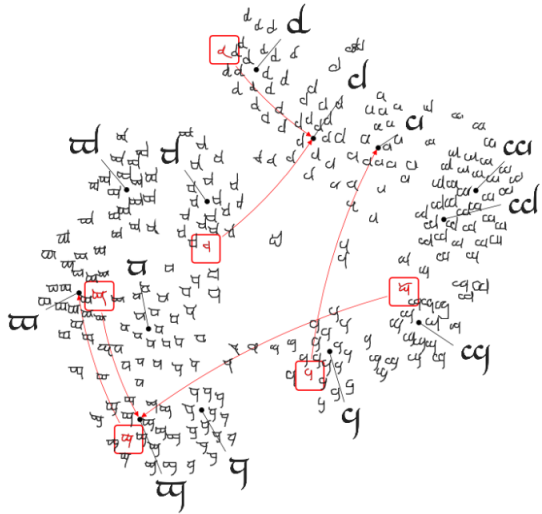
- The learned network $f_{\boldsymbol{\theta}}$ produces, in a feedforward manner, the embedding space as well as the weights of the corresponding linear classifier.

Snell et al. (2017)

Prototypical networks

Example embedding for an Omniglot meta-test task

- T-SNE visualization
- Black highlights are class centers
- Red highlights are misclassification errors



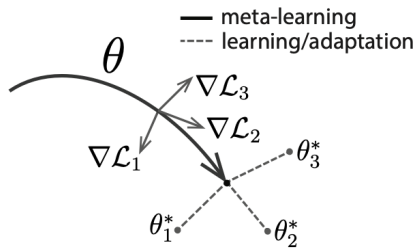
Snell et al. (2017)

(This one is simple enough to implement that I can actually do a live demo)

Model-agnostic meta learning (MAML)

A downside of the previous method was that it is not obvious how to apply outside of classification settings.

- MAML: meta-representation is a parameter vector $\boldsymbol{\theta}$ for some (any) deep learning model
- Learn a general **initialization**, from which we can quickly fine-tune to new classes
- Unlike "typical" transfer learning, where the initial $\boldsymbol{\theta}$ subject to fine-tuning come from some unrelated network, here there is an explicit meta-learning objective



Finn et al. (2017)

Model-agnostic meta learning (MAML)

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Finn et al. (2017)

Black-box approaches

Black-box approach

Train a neural architecture to directly output predictions at the "next" task.

1. Train a network which directly outputs parameters of a model, e.g. for task $\mathcal{T}_i$,

$$\omega_i = f_\theta(\mathcal{D}^i_{train})$$

2. Make predictions on test data with

$$y = g_{\omega_i}(\mathbf{x}_{i,test})$$

Clear training objective — this is now just supervised learning. But, complex models are required, and extending to large datasets is hard.

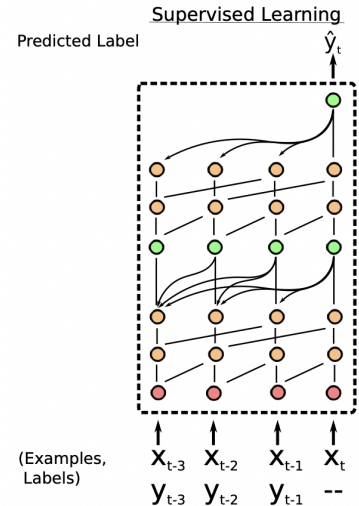


Figure: Mishra et al. (2018)

What else is there?

Other things you can meta-learn

- **Neural network architectures**: This is a big one — network architectures (e.g. convnets, resnets, LSTMs, GNNs) have strong inductive biases. Can we learn architectures (or building blocks) which generalize well to many tasks?

Other things you can meta-learn

- **Neural network architectures**: This is a big one — network architectures (e.g. convnets, resnets, LSTMs, GNNs) have strong inductive biases. Can we learn architectures (or building blocks) which generalize well to many tasks?
- **Optimizers**: Crudely, an optimizer is a function that takes a current value $\boldsymbol{\theta}^{(t)}$ and a gradient $\mathbf{g}$, which returns a new value $\boldsymbol{\theta}^{(t+1)}$. Can we learn this function?

Other things you can meta-learn

- **Neural network architectures**: This is a big one — network architectures (e.g. convnets, resnets, LSTMs, GNNs) have strong inductive biases. Can we learn architectures (or building blocks) which generalize well to many tasks?
- **Optimizers**: Crudely, an optimizer is a function that takes a current value $\boldsymbol{\theta}^{(t)}$ and a gradient $\mathbf{g}$, which returns a new value $\boldsymbol{\theta}^{(t+1)}$. Can we learn this function?
- **Losses or rewards**: what if you wanted an alternative objective function for your inner learning tasks — for example, a loss function which was robust to label noise? Can it generalize to other datasets?

Other things you can meta-learn

- **Neural network architectures**: This is a big one — network architectures (e.g. convnets, resnets, LSTMs, GNNs) have strong inductive biases. Can we learn architectures (or building blocks) which generalize well to many tasks?
- **Optimizers**: Crudely, an optimizer is a function that takes a current value $\boldsymbol{\theta}^{(t)}$ and a gradient $\mathbf{g}$, which returns a new value $\boldsymbol{\theta}^{(t+1)}$. Can we learn this function?
- **Losses or rewards**: what if you wanted an alternative objective function for your inner learning tasks — for example, a loss function which was robust to label noise? Can it generalize to other datasets?
- **Data augmentation**: if you have many different potential data transformations, which are likely to be label-preserving, which ones should you include when training? Can we learn a data augmentation policy?

Other things you can meta-learn

- **Neural network architectures**: This is a big one — network architectures (e.g. convnets, resnets, LSTMs, GNNs) have strong inductive biases. Can we learn architectures (or building blocks) which generalize well to many tasks?
- **Optimizers**: Crudely, an optimizer is a function that takes a current value $\boldsymbol{\theta}^{(t)}$ and a gradient $\mathbf{g}$, which returns a new value $\boldsymbol{\theta}^{(t+1)}$. Can we learn this function?
- **Losses or rewards**: what if you wanted an alternative objective function for your inner learning tasks — for example, a loss function which was robust to label noise? Can it generalize to other datasets?
- **Data augmentation**: if you have many different potential data transformations, which are likely to be label-preserving, which ones should you include when training? Can we learn a data augmentation policy?
- ...

. . . and other applications

- Domain adaptation,
- continual learning,
- defense against adversarial attacks, . . .

Many more ideas (with citations) in Hospedales et al. (2020).

Thank you!