Meta Learning at a Glance

Seminar 2 on *Nôm OCR*

Problem statement: MNIST vs. Omniglot

MNIST:
few classes, many examples

Omniglot:
many classes, few examples

Transfer learning via fine-tuning approach

Key idea: Solve target task $\mathcal{T}_b$ after solving source task(s) $\mathcal{T}_a$ by *transferring* knowledge learned from $\mathcal{T}_a$.

parameters pre-trained on $\mathcal{D}_a$

$$\phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_b^{tr})$$

training data for new task $\mathcal{T}_b$

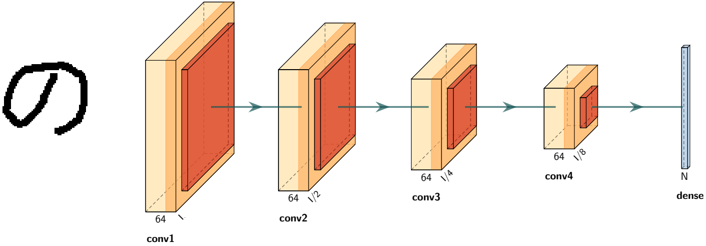
Common assumption: Cannot access data $\mathcal{D}_a$ during transfer.

Some problems/applications where transfer learning might make sense:

When $\mathcal{D}_a$ is very large and $\mathcal{D}_b$ is somehow smaller.

When you don't care about solving $\mathcal{T}_a$ & $\mathcal{T}_a$ simultaneously.

Transfer learning on Omniglot



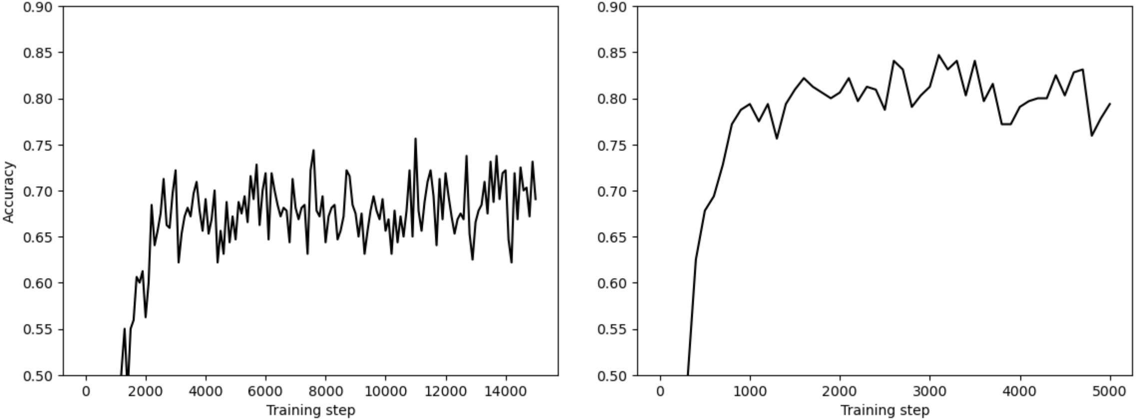
Model's backbone consists of four CNNs with ReLUs and Maxpooling layers

Pre-training split: contains $N = 1200$ characters/classes.

Fine-tuning split: contains another $N = 423$ characters/classes.

The goal: is to obtain a model that is initialized with the pre-trained parameters and can distinguish between new 423 classes after being fine-tuned.

How does transfer learning work on Omniglot?



Accuracy of pre-training vs. fine-tuning phases on validation sets

Meta Learning

Learning to Learn

From Transfer Learning to Meta-Learning

Transfer learning: Initialize model. Hope that it helps the target task.

Meta-learning: Can we explicitly *optimize* for transferability?

Given a set of training tasks, can we optimize for the ability to learn these tasks quickly?
so that we can learn new tasks quickly too.

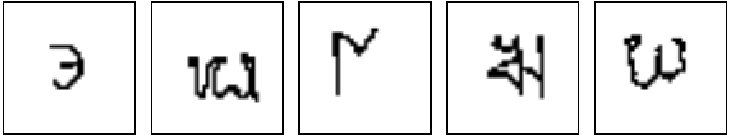
Scenario: what if dataset $\mathcal{D}_i^{tr}$ for task $\mathcal{T}_i$ is small?
(number of examples per class is small)

Learning a task: $\mathcal{D}_i^{tr} \xrightarrow{f_\theta} \theta$

Can we optimize this function f_θ?

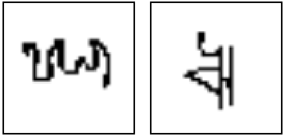
How does meta-learning work? An example

Given 1 example of 5 classes:



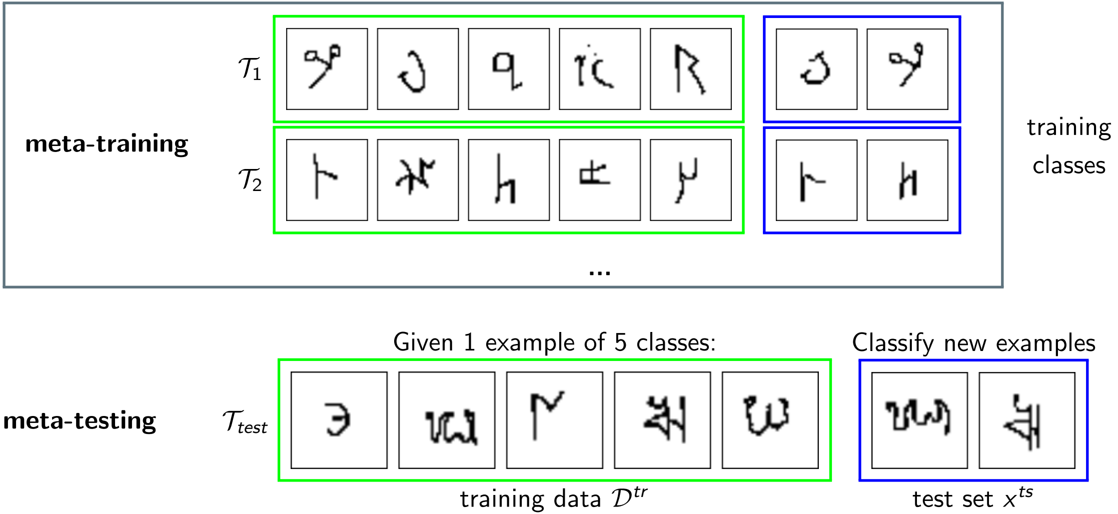
training data $\mathcal{D}^{tr}$

Classify new examples



test set x^{ts}

How does meta-learning work? An example



meta-training

$\mathcal{T}_1$

$\mathcal{T}_2$

...

training classes

meta-testing

$\mathcal{T}_{test}$

Given 1 example of 5 classes:

Classify new examples

training data $\mathcal{D}^{tr}$

test set x^{ts}

Can replace image classification with: regression, language generation, **any ML problem**.

Meta Learning problem

Transfer Learning with *Many Source Tasks*

Given data from $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n$, solve new task $\mathcal{J}_{test}$ more quickly/proficiently/stably.

Key assumption: meta-training tasks and meta-test task drawn i.i.d. from same task distribution

$$\mathcal{J}_1, \ldots, \mathcal{J}_n \sim p(\mathcal{J}), \ \mathcal{J}_{test} \sim p(\mathcal{J}).$$

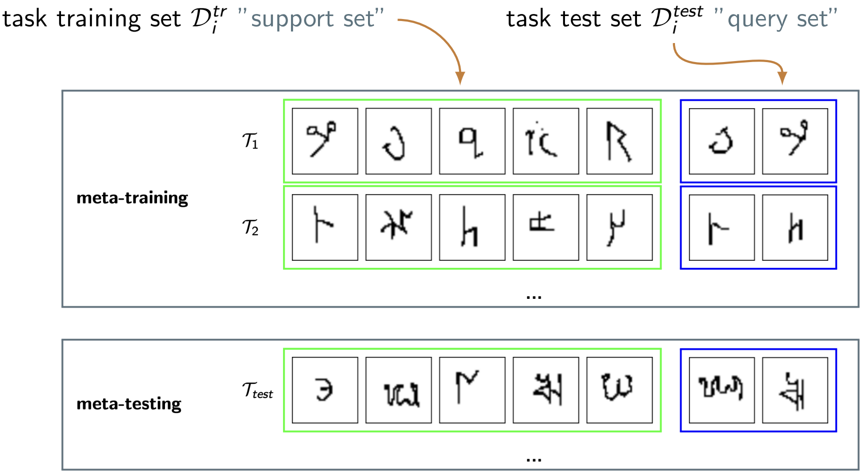
Tasks must share underlying structure

What do the tasks correspond to?

- Recognizing handwritten digits from different languages,
- Giving feedback to different students on different exams,
- Classifying flower species in different regions of the world,
- A robot performing different tasks.

How many tasks do we want for a training stage? The more the better.

Some terminologies



task training set $\mathcal{D}_i^{tr}$ "support set"

task test set $\mathcal{D}_i^{test}$ "query set"

meta-training

$\mathcal{T}_1$

$\mathcal{T}_2$

...

meta-testing

$\mathcal{T}_{test}$

...

k-shot learning: learning with k examples per class

N-way classification: choosing between N classes

Meta-Learning formulation

Conventional supervised learning:

inputs: $\mathbf{x}$

$\mathbf{y} = g_{\phi}(\mathbf{x})$

outputs: $\mathbf{y}$

Data: $\{(\mathbf{x}, \mathbf{y})_{\mathrm{i}}\}$

Meta supervised learning:

inputs: $\mathcal{D}^{tr}$ $\mathbf{x}^{ts}$

$\{(\mathbf{x}, \mathbf{y})_{1:\mathrm{K}}\}$

$\mathbf{y}^{ts} = \mathrm{f}_{\theta}(\mathcal{D}^{tr}, \mathbf{x}^{ts})$

outputs: $\mathbf{y}^{ts}$

Data: $\{\mathcal{D}_i\}$

$\mathcal{D}_i$: $\{(\mathbf{x}, \mathbf{y})_{\mathrm{i}}\}$

θ is called meta-parameters

Meta-Learning on Omniglot

Model: Using the *backbone* mentioned in the slide earlier.

Training and evaluating meta-learning algorithms on **5-way 1-shot** tasks with **15 query examples** per task.

Data:*

- Training set: contains 1100 characters/classes,
- Validation set: contains 100 characters/classes,

} exactly from the *pre-training split*

- Testing set: contains 423 characters/classes.

Only some minor changes in the training setups are introduced to produce a highly fair comparison.

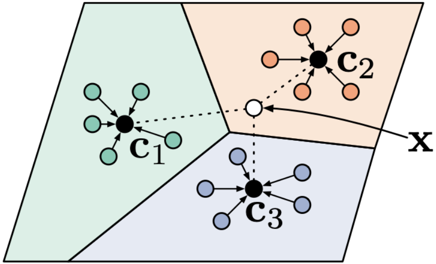
** The three sets are disjoint with each other, of course.*

Some Meta-Learning Algorithms

- Prototypical Networks
- Model-Agnostic Meta-Learning
- Proto-MAML

Prototypical **Net**works concept

Key idea: *Mapping* images to features such that images of same class are *close to each other* in feature space.



Snell, Jake, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning." Advances in neural information processing systems 30 (2017).

ProtoNet formulations

Using squared Euclidean distance to **measure distance** between $f_\theta(x)$ of an image x and each of the prototypes:

$$d(f_\theta(x), c_n) = \|f_\theta(x) - c_n\|^2$$

Applying the softmax operation to **classify the image** by obtaining the proper probabilities over classes:

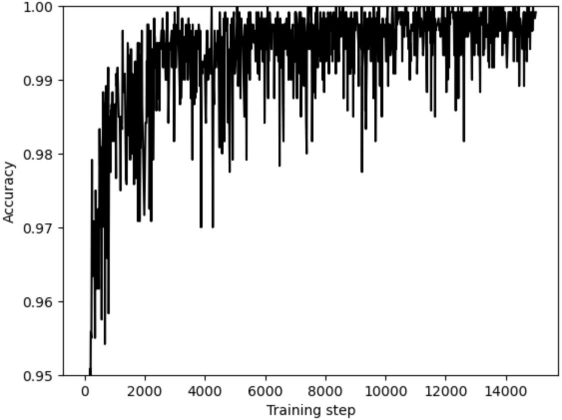
$$p(x, c_n) = p(y = n | x, c_n) = \frac{\exp(-d(f_\theta(x), c_n))}{\sum_{n'=1}^{N} \exp(-d(f_\theta(x), c_{n'}))}$$

ProtoNet algorithm

Algorithm ProtoNet for N-way K-shot Meta Supervised Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$: learning rate
Randomly initialize meta-parameters $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do
        Sample $K$ datapoints $\mathcal{D}_i^{tr}$ from $\mathcal{T}_i$
        Sample disjoint datapoints $\mathcal{D}_i^{query}$ from $\mathcal{T}_i$
        Compute prototypes $c_{n \in 1:N} \leftarrow \frac{1}{K} \sum_{(x,y)_j \in \mathcal{D}_i^{tr} : y_j = n} f_\theta(x_j)$
        Compute distribution over classes $\mathcal{P}_i \leftarrow p(x_j, c_{n \in 1:N})_{x_j \in \mathcal{D}_i^{query}}$
        Compute loss $\ell_{\mathcal{T}_i} \leftarrow \mathcal{L}(\theta, \mathcal{P}_i)$
    end for
    Update $\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{\mathcal{T}_i} \ell_{\mathcal{T}_i}$
end while
```

ProtoNet result on Omniglot



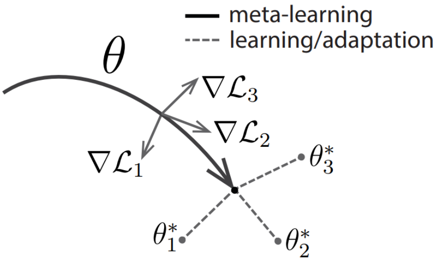
Accuracy on query set during validating

Some Meta-Learning Algorithms

- Prototypical Networks
- Model-Agnostic Meta-Learning
- Proto-MAML

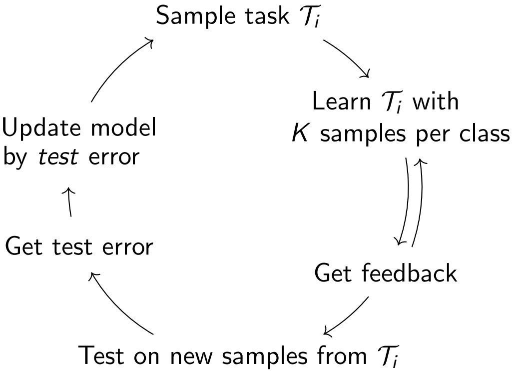
Model-**A**gnostic **M**eta-**L**earning concept

Key idea: To meta-learn **initial** θ that can be quickly adapted **via gradient descent** to a new task with a **small amount** of data.



Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." International conference on machine learning. PMLR, 2017.

A high-level view of MAML



Sample task $\mathcal{T}_i$

Learn $\mathcal{T}_i$ with K samples per class

Get feedback

Test on new samples from $\mathcal{T}_i$

Get test error

Update model by *test* error

Meta-training lifecycle

Objective: Learning initial parameters θ such that *small changes* in these parameters will produce *high performance* on any unseen task $\mathcal{J}_{test}$.

MAML algorithm

Algorithm MAML for N-way K-shot Meta Supervised Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha, \beta$: learning rates for inner and outer loop
Randomly initialize meta-parameters $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do                                              ▷ start of outter loop
        Sample $K$ datapoints $\mathcal{D}_i^{tr}$ from $\mathcal{T}_i$
        Sample disjoint datapoints $\mathcal{D}_i^{query}$ from $\mathcal{T}_i$
        Initialize task-specific parameters $\phi_i \leftarrow \theta$
        Optimize $\phi_i \leftarrow \phi_i - \alpha \nabla_{\phi_i} \mathcal{L}(\phi_i, \mathcal{D}_i^{tr})$          ▷ inner loop
        Compute loss $\ell_{\mathcal{T}_i} \leftarrow \mathcal{L}(\phi_i, \mathcal{D}_i^{query})$
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i} \ell_{\mathcal{T}_i}$                          ▷ end of outter loop
end while
```

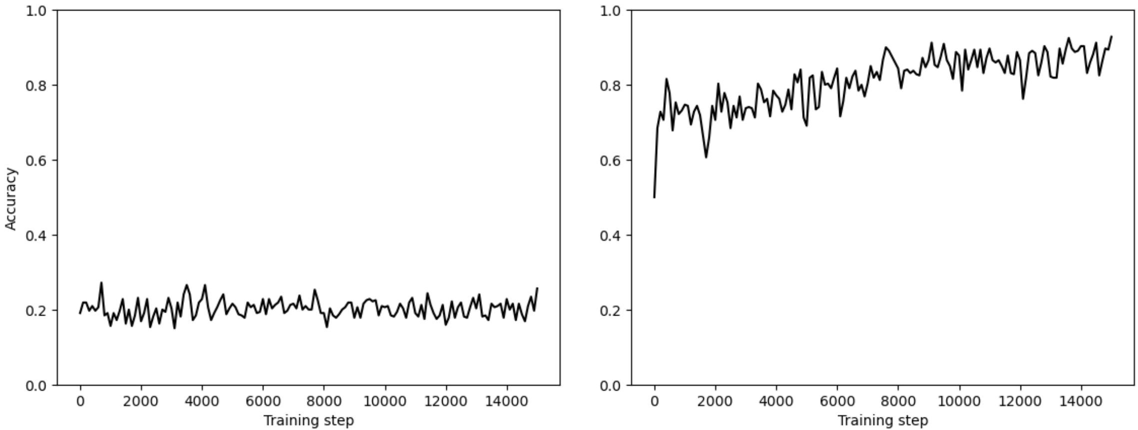
First-order MAML approximation

Second-order derivatives!!!

11: Update $\theta \leftarrow \theta - \beta \boxed{\nabla_\theta \sum_{\mathcal{T}_i} \ell_{\mathcal{T}_i}}$

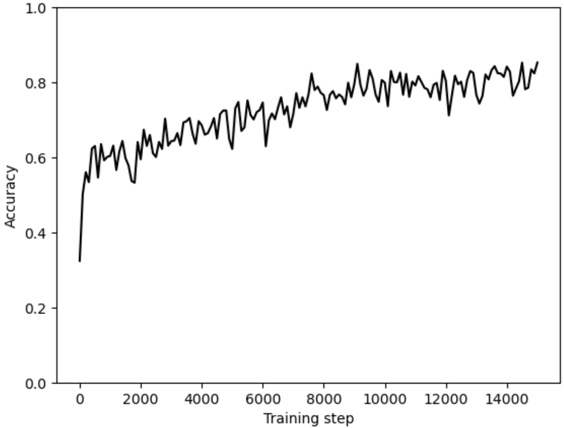
But, PyTorch will give you a hand

fo-MAML results on Omniglot



Pre- and Post-adaption accuracies on support set during validating

fo-MAML results on Omniglot



Accuracy on query set during validating

Some Meta-Learning Algorithms

- Prototypical Networks
- Model-Agnostic Meta-Learning
- Proto-MAML

Intergrating ProtoNet's outputing to MAML

Remember that ProtoNet use Euclidean distance $d(f_\theta(x), c_n)$ to classify image x?

The **output distance** with respect to a prototype c_n of ProtoNet can be unpacked as:

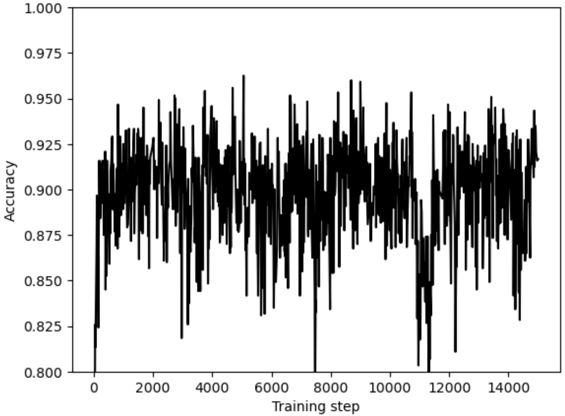
$$
\begin{aligned}
-d(f_\theta(x), c_n) = -\|f_\theta(x) - c_n\|^2 &= -f_\theta(x)^\intercal f_\theta(x) + 2c_n^\intercal f_\theta(x) - c_n^\intercal c_n \\
&= \underbrace{2c_n^\intercal}_{w_n = 2c_n} f_\theta(x) \underbrace{- c_n^\intercal c_n}_{b_n = -c_n^\intercal c_n} + \underbrace{\mathit{constant}}_{\text{just ignore}^{*}}
\end{aligned}
$$

A Dense output layer!!!

** Since it is a class-independent scalar, it leaves the output probabilities unchanged.*

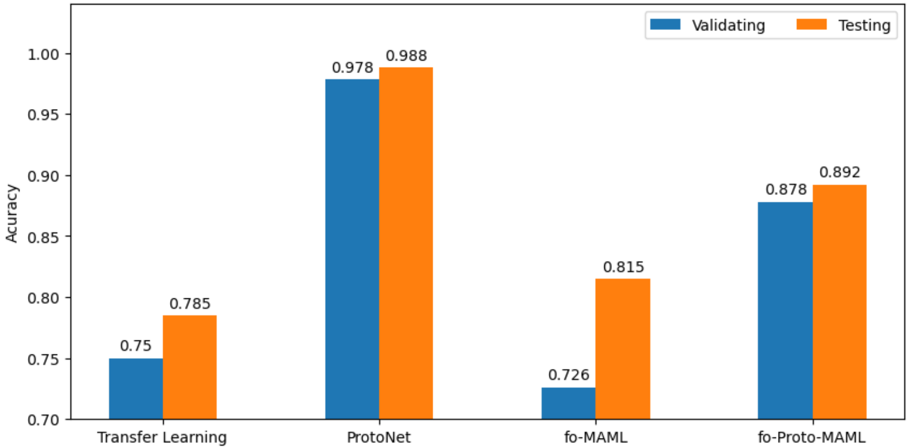
Triantafillou, Eleni, et al. "Meta-dataset: A dataset of datasets for learning to learn from few examples." arXiv preprint arXiv:1903.03096 (2019).

fo-Proto-MAML result on Omniglot



Accuracy on query set during validating

Experimental results



Comparison of accuracy on query set over approaches during validating/testing time

Acknowledgements

This work was mostly derived from (1) the CS330 coursework at Stanford University, and (2) the UvA Deep Learning tutorial 16.

`https://cs330.stanford.edu/`

`https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial16/Meta_Learning.html`

After credits

Codework repositories and their associated experiments:

- Three meta-learning algorithms on Omniglot: `https://github.com/nthehai01/meta-learning-methods`,
- Transfer learning on Omniglot: `https://github.com/nthehai01/non-episodic-approach`.