Pattern Recognition (Pattern Classification)

Meta Learning (Learning to Learn)

Second Edition

- A baby is testing the interactive cactus toy
- How did she/he learn to test the toy using her/his voice?
- Can we learn her/his learning schema?

Contents

1. Introduction
2. Meta-Learning Problem Statement
3. Model-Agnostic Meta-Learning
4. MetaOptNet
5. Error decomposition in Few-Shot Learning

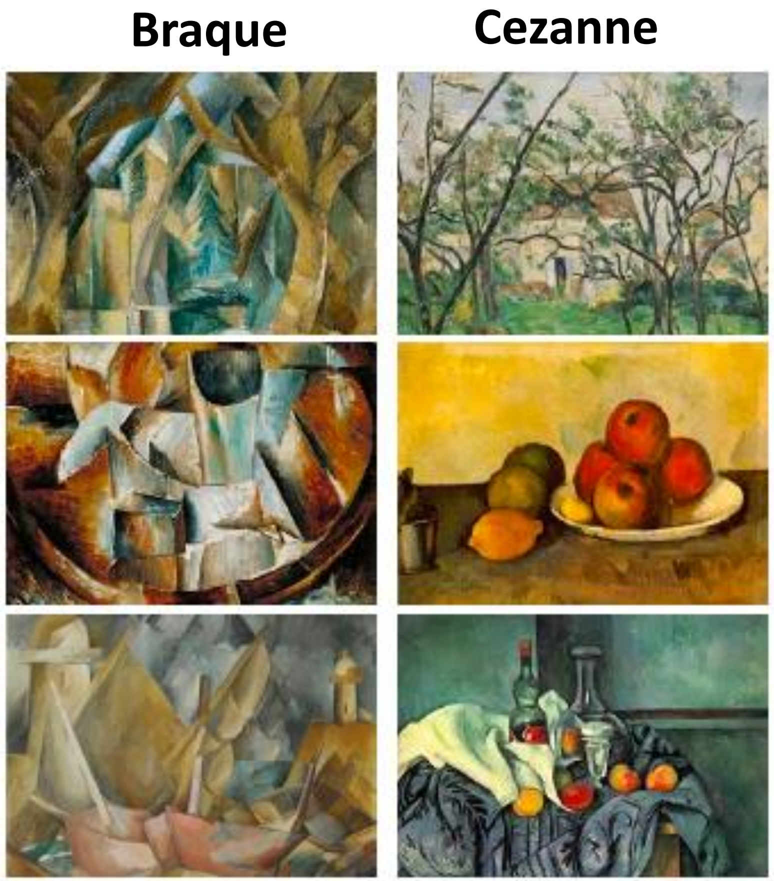
1- Introduction

Introduction

- Problem with deep networks is that we need to have a large training set to train our model and it will fail abruptly when we have very few data points.
- Let's say we trained a deep learning model to perform task A. Now, when we have a new task, B, that is closely related to A, we can't use the same model. We need to train the model from scratch for task B.
- So, for each task, we need to train the model from scratch although they might be related.

What if: Available Data set is small and Need to quickly learn a new task

Training Data:



By Braque or Cezanne?

Test Data:

How did you accomplish this?

By leveraging prior experience!

This is where elements of multi-task learning can come into play

Is deep learning really the true AI?

- We generalize our learning to multiple **concepts** and learn from there.
- But current learning algorithms master only one task.
- Here is where meta learning comes in.
- Meta learning produces a versatile AI model that can **learn** to perform various **tasks** without having to train them from scratch.
- We train a meta learning model on various related tasks with few data points, so for a new related task, it can make use of the learning obtained from the previous tasks and we don't have to train them from scratch. Many researchers and scientists believe that meta learning can get us closer to achieving **Artificial General Intelligence** (AGI).
    - AGI is the ability of an intelligent agent to understand or learn any intellectual task that a human being can.

Single Task Learning

- A single task supervised machine learning problem considers a set of training data (x_n, y_n) sampled from a single task $\mathcal{T}$, where the goal is to learn a hypothesis $x \longmapsto \hat{y}$.
- We rely on three different data sets:
  - Training set D^{trian}
  - Validation set $D^{validation}$
  - Test set D^{test}

- Learning algorithm optimizes parameters θ on the training set D^{train} and do model selection using $D^{validation}$, and evaluate its generalization error on the test set D^{test}

Multi-Task Learning (Meta-Learning)

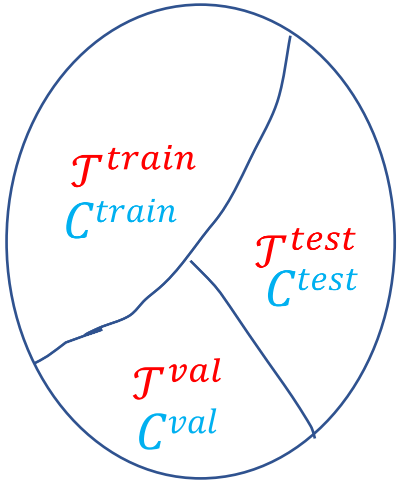
- In meta-learning, we rely on 3 different **tasks sets**:
- Training task set: $\mathcal{T}^{train} = T = \{T_i\}_{i=1}^{I}$, containing examples of C^{train} classes
- Validation task set: $\mathcal{T}^{val} = V = \{V_v\}_{v=1}^{V}$, containing examples of C^{val} classes
- Testing task set: $\mathcal{T}^{test} = S = \{S_j\}_{j=1}^{J}$, containing examples of C^{test} classes
- In order to measure the **model's generalization** to unseen classes, C^{train}, C^{val} and C^{test} are chosen to be mutually disjoint

- Each T_i or V_v or S_j has been called an episode

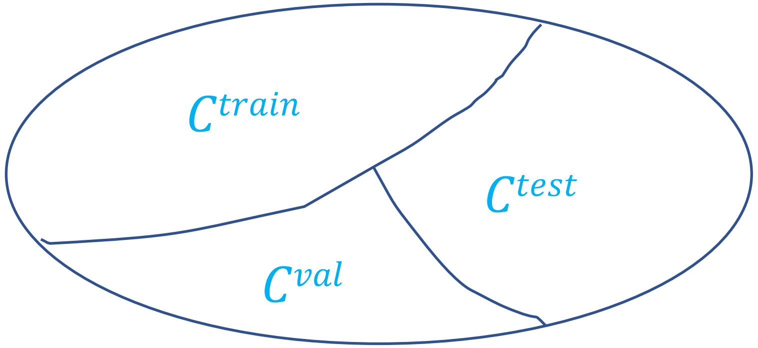
Task-Set, Class-Set

- Suppose the task-set $\mathcal{T} = \{T_1 \quad T_2 \quad \dots \quad T_m\}$ is available
- Suppose all classes involve in the task-set $\mathcal{T}$ is given by $C = \{C_1 \quad C_2 \quad \dots \quad C_L\}$

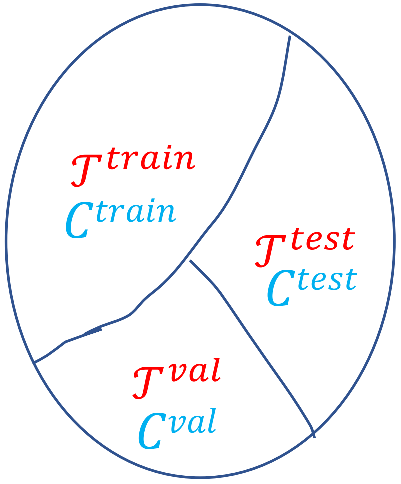
$\mathcal{T} = \{T_1 \quad T_2 \quad \dots \quad T_m\}$

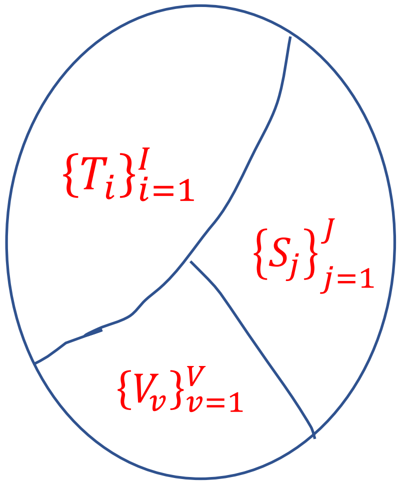


$C = \{C_1 \quad C_2 \quad \dots \quad C_L\}$



Meta Learning Data Sets of T, V and S





$$T_i \longrightarrow \mathcal{D}_i = \{D_i^{train}, D_i^{test}\}$$

$$S_j \longrightarrow \mathcal{D}_j = \{D_j^{train}, D_j^{test}\}$$

$$V_v \longrightarrow \mathcal{D}_v = \{D_v^{train}, D_v^{test}\}$$

Meta Learning Data Sets of T, V and S

1. **Meta-training tasks set** $T = \{T_i\}_{i=1}^{I} = \{\mathcal{D}_i\}_{i=1}^{I} = \mathcal{D}^{meta-training}$,

   $\mathcal{D}_i = \{D_i^{train}, D_i^{test}\}$ that is needed for "**meta-training**"

1. **Meta-validation tasks set** $V = \{V_v\}_{v=1}^{V} = \{\mathcal{D}_v\}_{v=1}^{V} = \mathcal{D}^{meta-validation}$,

   $\mathcal{D}_v = \{D_v^{train}, D_v^{test}\}$ for model selection (hyperparameter setting),

1. **Meta-testing tasks set** $S = \{S_j\}_{j=1}^{J} = \{\mathcal{D}_j\}_{j=1}^{J} = \mathcal{D}^{meta-testing}$,

   $\mathcal{D}_j = \{D_j^{train}, D_j^{test}\}$ for the evaluation of generalization performance of the trained meta learner.

Constructing $\mathcal{T} = \{T_i\}_{I=1}^{I} = \{(\overset{\overset{support}{\uparrow}}{D_i^{train}}, \overset{\overset{query}{\uparrow}}{D_i^{test}})\}_{i=1}^{I}$ in few-Shot classification

- Suppose the overall set of classes for meta-training is C^{train}

A task (episode) $T_i = \left(D_i^{train}, D_i^{test}\right)$ is sampled as follows:

- For task i, **classes set $\boldsymbol{C_i}$** containing N classes are first sampled (with replacement) from C^{train}
- Next, training set $D_i^{train} = \{(x_n, y_n)|n = 1, \dots, N \times K, y_n \in C_i\}$ consisting of K examples per N classes is sampled. Test set $D_i^{test} = \{(x_n, y_n)|n = 1, \dots, Q \times K, y_n \in C_i\}$ consisting of K examples per Q classes is sampled
- We need to select the train and test samples of each episodes without replacement, i.e., $D_i^{train} \cap D_i^{test} = \emptyset$, to optimize the generalization error

- So, we select $\mathcal{T} = \left\{\left(D_i^{train}, D_i^{test}\right)\right\}_{i=1}^{I}$ consisting of I tasks (episodes), each tasks consists of K examples of each Nclasses for D_i^{train}, and of each Q classes for D_i^{test}

Constructing $S = \{\mathcal{D}_j\}_{I=1}^{J}$ and $V = \{\mathcal{D}_i\}_{v=1}^{V}$

- The overall set of classes for meta-validation is C^{val}
- In the same manner, meta-validation set is constructed on the fly from C^{val}

- The overall set of classes for meta-testing is C^{test}
- In the same manner, meta-test set is constructed on the fly from C^{test}

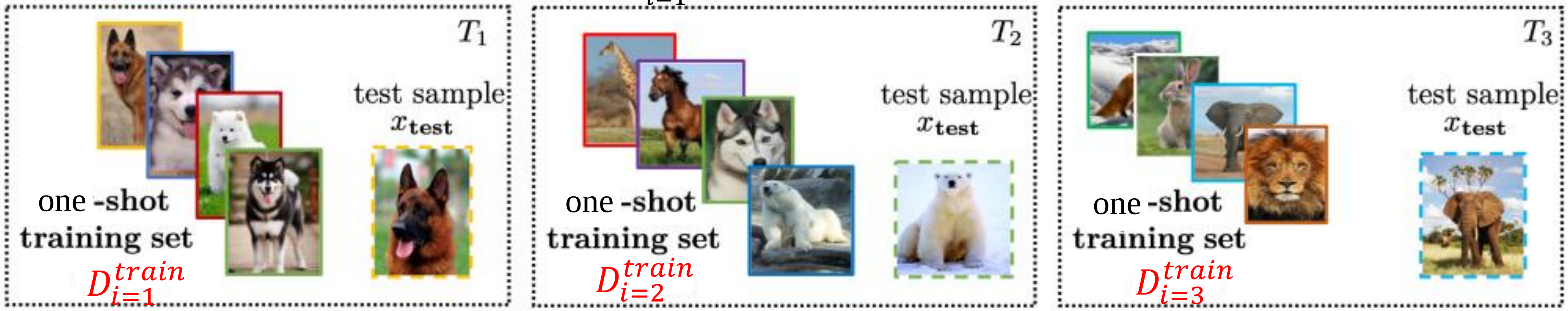
- A set of tasks are said to be mutually-exclusive if a single model cannot solve them all at once

Meta learning and few-shot

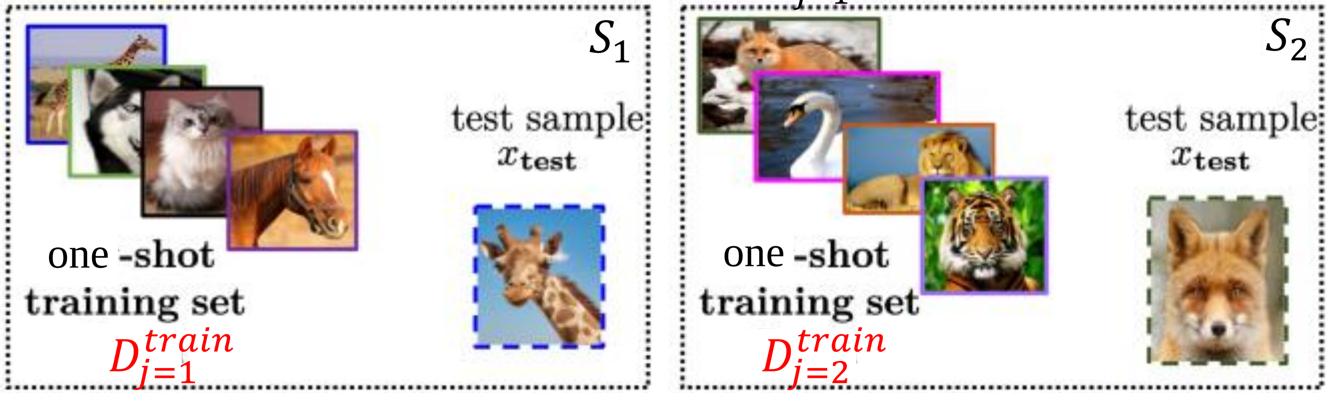
- Learning from fewer data points is called few-shot learning or K-shot learning where K denotes the number of examples in each of the classes in the dataset.
- Let's say we are performing the image classification of dogs and cats.
  - If we have exactly one dog and one cat image then it is called one-shot learning, that is, we are learning from just one example per class.
  - If we have, say 10 images of a dog and 10 images of a cat, then that is called 10- shot learning.
  - So K in K-shot learning implies a number of examples we have per class.
  - There is also zero-shot learning where we don't have any example for some classes. In this case, we will not have examples, but we will have meta information about each of the classes and we will learn from the meta information.
  - Since we have two classes in our dataset, that is, dog and cat, we can call it two-way K-shot learning; so N-way means the number of classes we have in our dataset.

T:meta training tasks - S:meta-test tasks

meta-training tasks set $T = \{(D_i^{train}, D_i^{test})\}_{i=1}^{I}$ N= 4-class, Q=1-class, K=1-shot meta-learning tasks set



meta-testing tasks set $S = \{(D_j^{train}, D_j^{test})\}_{j=1}^{J}$ 4-class, 1-shot meta-testing tasks set



Critical Assumption

- Different tasks need to share some structure.
- If this doesn’t hold, you are better off using single-task learning.
- The good news: There are many tasks with shared structure.

- Even if the tasks are seemingly unrelated:
  - The laws of physics underlay real data
  - People are all organisms with intentions
  - The rules of English underlay English language data
  - Languages all develop for similar purposes

Contents

1. Introduction
2. Meta-Learning Problem Statement
3. Model-Agnostic Meta-Learning
4. MetaOptNet
5. Error decomposition in Few-Shot Learning

2- Meta-Learning Problem Statement

Meta-Learning Problem Statement

- Recalling the Supervised Learning (classification):

$$\boldsymbol{y_j} = h_{\varphi}\left(D^{train}, \boldsymbol{x}_j\right), D^{train} = \{(\boldsymbol{x}, \boldsymbol{y})_i\}, D^{test} = \left\{(\boldsymbol{x}, \boldsymbol{y})_j\right\}$$

- Meta Supervised Learning (classification):

$$\boldsymbol{y}^{D_j^{test}} = \boldsymbol{f_\theta}\left(\boldsymbol{T}, D_j^{test}\right)$$

$$T = \{T_i\}_{I=1}^{I} = \left\{\left(D_i^{train}, D_i^{test}\right)\right\}_{i=1}^{I}$$

$$S = \left\{S_j\right\}_{j=1}^{J} = \mathcal{D}^{meta-testing}, S_j = \{D_j^{train}, D_j^{test}\}$$

- $\boldsymbol{\theta}$ is learnable meta-parameters

Types of meta learning

- Meta learning can be categorized in several ways.

- We will categorize meta learning into the following three categories:
  1. Learning the metric space (Metric based)
  2. Learning the optimizer (Optimization based)
  3. Learning the initializations (Model based)

1. Learning the metric space

- We will learn the appropriate metric space.
- Let's say we want to learn the similarity between two images.
- In the metric-based setting, we use a simple neural network that extracts the features from two images and **finds the similarity** by computing the distance between features of these two images.
- This approach is widely used in a few-shot learning setting where we don't have many data points.
- Example of metric-based learning algorithms are Siamese networks, prototypical networks, and relation networks.

2. Learning the optimizer

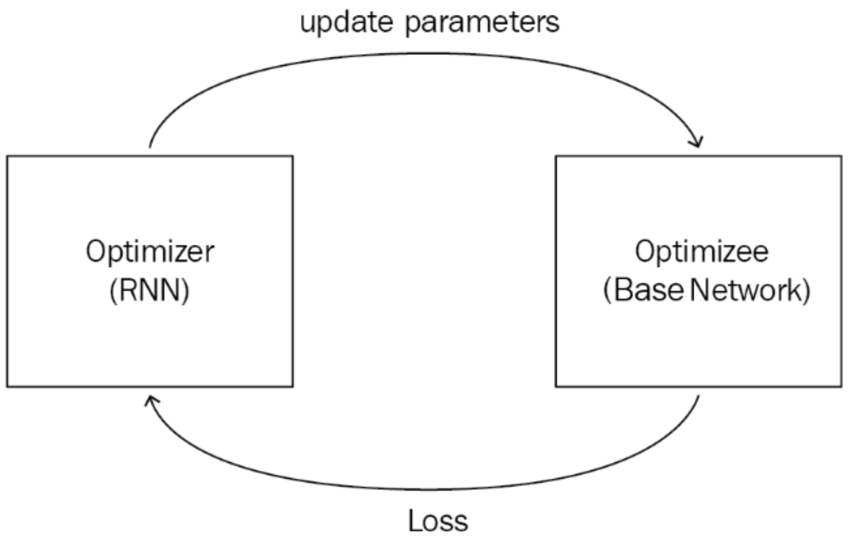
- How do we generally optimize our deep learner? We optimize by training on a large dataset and minimize the loss using gradient descent.
- But in the few-shot learning setting, gradient descent fails as we will have a smaller dataset.
- So, in this case, we will learn the optimizer itself.
- We will have two networks:
  - a base network that actually tries to learn the concept and
  - a meta network that optimizes the base network

2. Learning the optimizer: learning to learn gradient descent

- It is one of the simplest meta learning algorithms.
- We know that, in meta learning, our goal is to learn the learning process.
- In general, how do we train our deep networks? We train our network by computing loss and minimizing the loss through gradient descent.
- Instead of using gradient descent **we learn this optimization process automatically.**

2. Learning the optimizer using Recurrent Neural Network (RNN)

- We call our RNN, an optimizer and our base network an optimizee.
- Let's say we have a model $\boldsymbol{f}$ parameterized by some parameters $\boldsymbol{\theta}$.
- We need to find the optimal $\boldsymbol{\theta}$, so that we can minimize the loss.
- We use the RNN for finding the optimal $\boldsymbol{\theta}$.
- So the RNN (optimizer) finds the optimal parameter and sends it to the optimizee; the optimizee uses this parameter, computes the loss, and sends the loss to the RNN.
- Then, RNN optimizes itself through gradient descent and updates the model parameters $\boldsymbol{\theta}$.



3. Learning $\boldsymbol{\theta}$ as the initializations (Model Based)

- In this method, we try to **learn optimal initial parameter values**.

Suppose we are building a deep network to classify images:

- First, we initialize random weights, calculate loss, and minimize the loss through a gradient descent.
  - So, we will find the optimal weights through gradient descent and minimize the loss.
- Instead of initializing the weights randomly, we initialize the weights with optimal values or close to optimal values, then we can attain the convergence faster and we can learn very quickly.
- We can find these optimal initial parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ with algorithms such as MAML, Reptile, and Meta-SGD.

Finding the best initial parameters means that the algorithms are optimization based

Using the initializations $\boldsymbol{\theta_{init} = \theta^*}$

- Meta learner learns the best parameters $\boldsymbol{\theta = \theta^*}$
- We train any new task (unseen task during meta-training) initializing its parameters on $\boldsymbol{\theta_{init} = \theta^*}$, then we expect **fast convergence** to a **high accuracy** even the **training set is not large**.

Contents

1. Introduction
2. Meta-Learning Problem Statement
3. Model-Agnostic Meta-Learning
4. MetaOptNet
5. Error decomposition in Few-Shot Learning

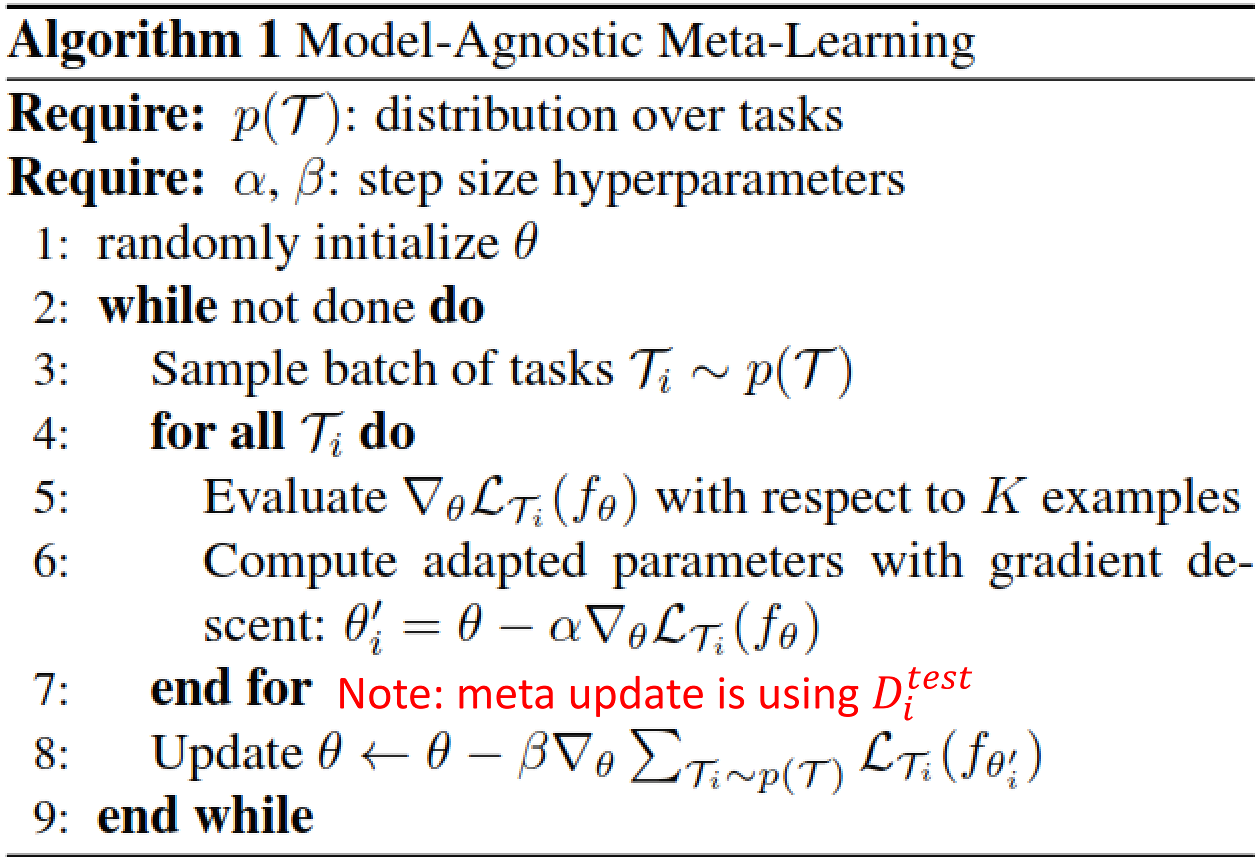
3- Model-Agnostic Meta-Learning

Chelsea Finn, Pieter Abbeel, Sergey Levine. **Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks**, International conference on Machine Learning, Sydney, Australia, PMLR 70, 2017

Cited by 7814

MAML Algorithm - Meta Learning

$$\mathcal{T}_i = T = \{T_i\}_{i=1}^{Bach\ Size} = \left\{\left(D_i^{train}, D_i^{test}\right)\right\}_{i=1}^{I} = \mathcal{D}^{meta-training}$$



Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

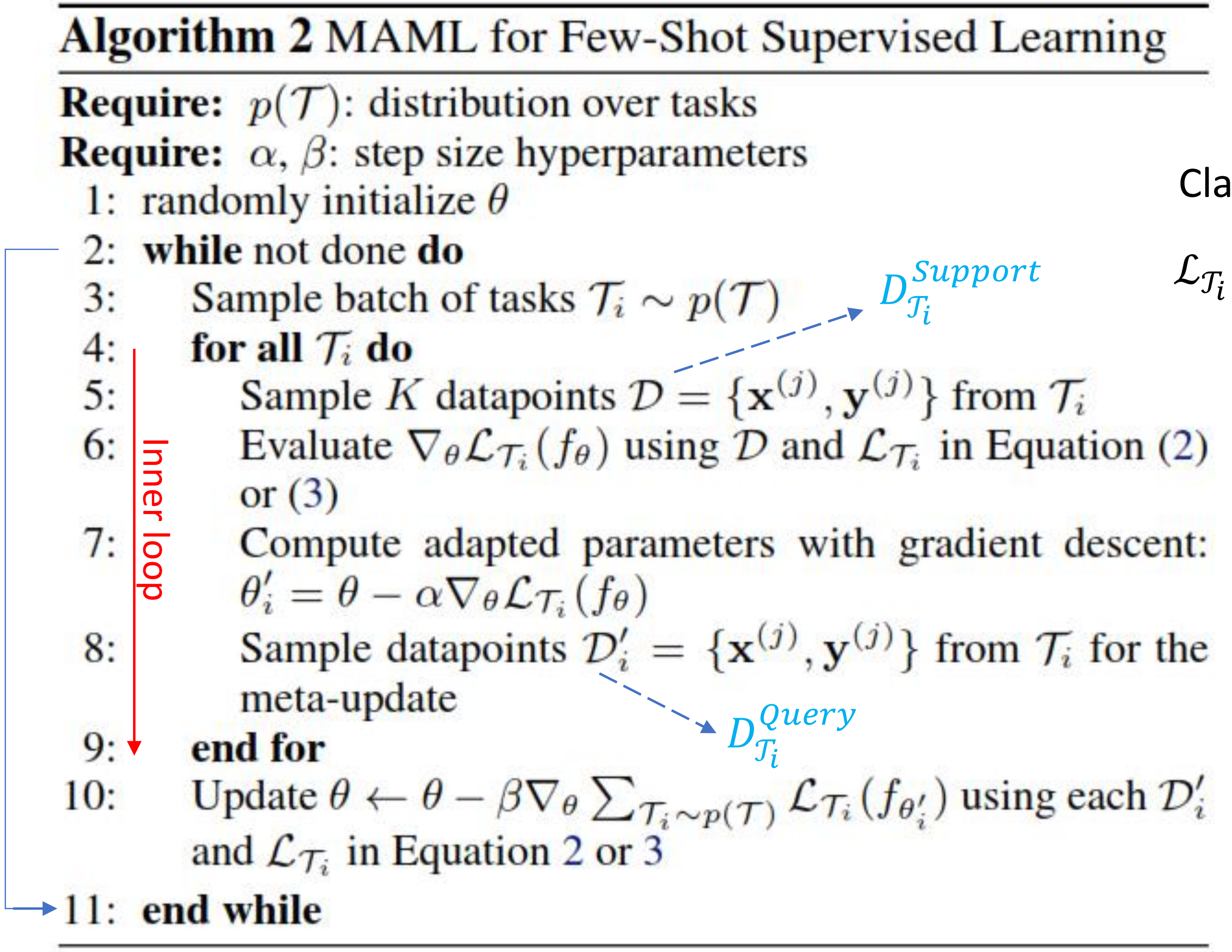
Require: α, β: step size hyperparameters

```
randomly initialize $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do
        Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples
        Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while
```

Note: meta update is using D_i^{test}

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

MAML Algorithm - Few Shot Learning



Algorithm 2 MAML for Few-Shot Supervised Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
randomly initialize $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do
        Sample $K$ datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
        Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
        Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
        Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each $\mathcal{D}'_i$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
end while
```

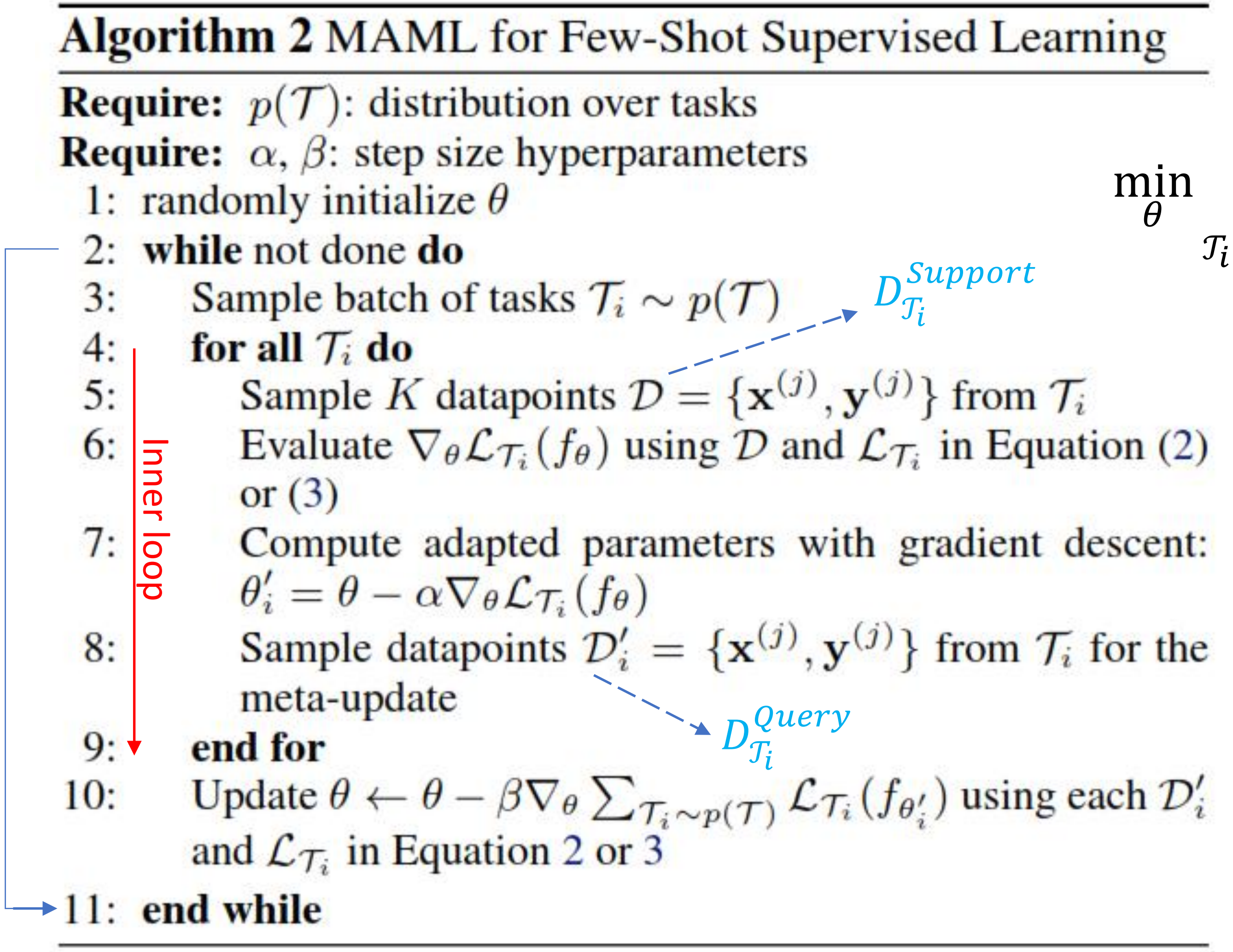
Classification loss function: (3)

$$\mathcal{L}_{\mathcal{T}_i}(f_\theta) = \sum_{x,y\sim\mathcal{T}_i} yLogf_\theta(x) + (1-y)Log(1-f_\theta(x))$$

Regression loss function:

$$\mathcal{L}_{\mathcal{T}_i}(f_\theta) = \sum_{x,y\sim\mathcal{T}_i} \|f_\theta(x) - y\|_2^2 \qquad (2)$$

MAML Algorithm - Few Shot Learning



Algorithm 2 MAML for Few-Shot Supervised Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
randomly initialize $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do
        Sample $K$ datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
        Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2)
        or (3)
        Compute adapted parameters with gradient descent:
        $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
        Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the
        meta-update
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each $\mathcal{D}'_i$
    and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
end while
```

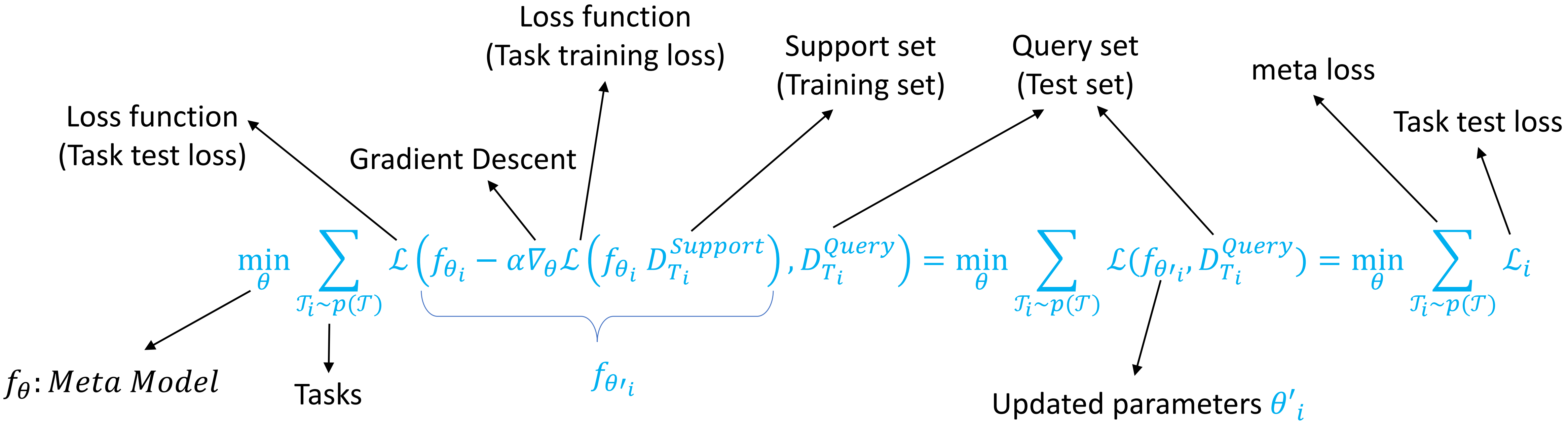
$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\Big(\underbrace{f_{\theta_i} - \alpha \nabla_\theta \mathcal{L}\big(f_{\theta i}, \overbrace{D_{T_i}^{Support}}^{D_i^{train}}\big)}_{f_{\theta'_i}}, \overbrace{D_{T_i}^{Query}}^{D_i^{test}}\Big)$$

$$\equiv$$

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(f_{\theta'_i}, D_{T_i}^{Query})$$

MAML Optimization function

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(\theta - \alpha \nabla_{\theta} \mathcal{L}\left(\theta, D_{T_i}^{Support}\right), D_{T_i}^{Query}\right) = \min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(\theta', D_{T_i}^{Query}\right) = \min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_i$$



$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(\underbrace{f_{\theta_i} - \alpha \nabla_{\theta} \mathcal{L}\left(f_{\theta_i}\ D_{T_i}^{Support}\right)}_{f_{\theta'_i}}, D_{T_i}^{Query}\right) = \min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(f_{\theta'_i}, D_{T_i}^{Query}) = \min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_i$$

Loss function (Task training loss)

Support set (Training set)

Query set (Test set)

meta loss

Loss function (Task test loss)

Gradient Descent

Task test loss

$f_{\theta}: Meta\ Model$

Tasks

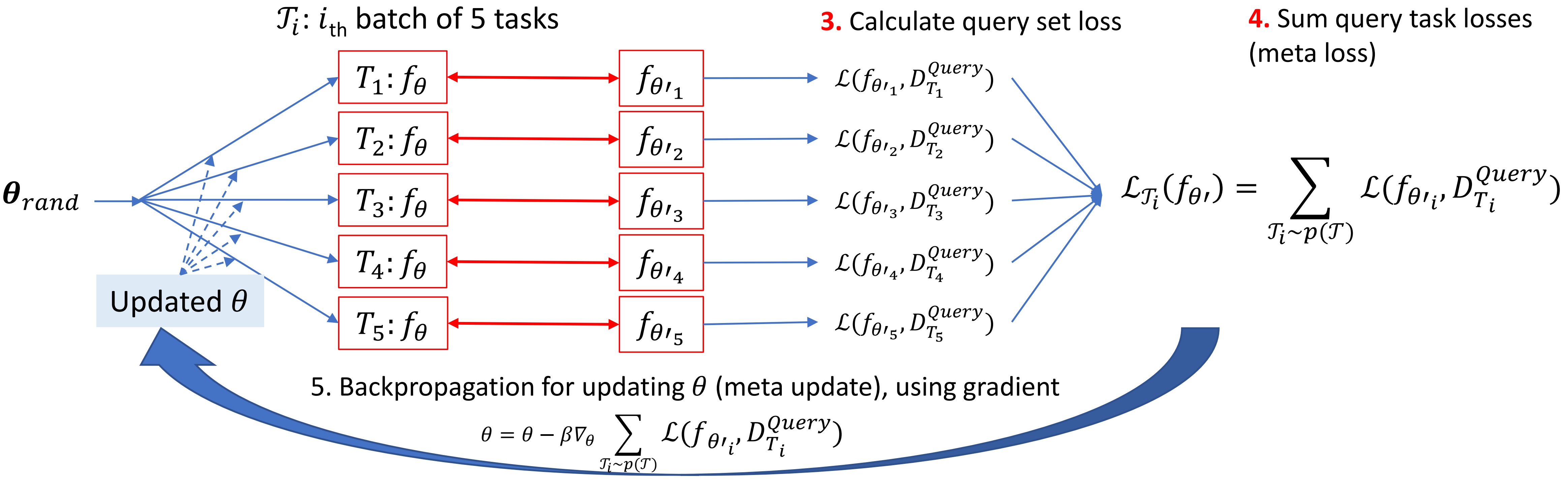
Updated parameters θ'_i

MAML Training

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(f_{\theta_i} - \alpha\nabla_\theta \mathcal{L}\left(f_{\theta i}, D_{T_i}^{Support}\right), D_{T_i}^{Query}\right) = \min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(f_{\theta'_i}, D_{T_i}^{Query})$$

1. A task learner f parametrized by $\boldsymbol{\theta}$

2. Each task individually is trained by its own Support set. θ' is leaner parameters after training is done (Inner loop: several times to converge θ', $\theta'_i = \theta - \alpha\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$)



$\mathcal{T}_i$: i_{th} batch of 5 tasks

3. Calculate query set loss

4. Sum query task losses (meta loss)

$\boldsymbol{\theta}_{rand}$

Updated θ

$T_1: f_\theta$ — $f_{\theta'_1}$ — $\mathcal{L}(f_{\theta'_1}, D_{T_1}^{Query})$

$T_2: f_\theta$ — $f_{\theta'_2}$ — $\mathcal{L}(f_{\theta'_2}, D_{T_2}^{Query})$

$T_3: f_\theta$ — $f_{\theta'_3}$ — $\mathcal{L}(f_{\theta'_3}, D_{T_3}^{Query})$

$T_4: f_\theta$ — $f_{\theta'_4}$ — $\mathcal{L}(f_{\theta'_4}, D_{T_4}^{Query})$

$T_5: f_\theta$ — $f_{\theta'_5}$ — $\mathcal{L}(f_{\theta'_5}, D_{T_5}^{Query})$

$$\mathcal{L}_{\mathcal{T}_i}(f_{\theta'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(f_{\theta'_i}, D_{T_i}^{Query})$$

5. Backpropagation for updating θ (meta update), using gradient

$$\theta = \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(f_{\theta'_i}, D_{T_i}^{Query})$$

After $\boldsymbol{\theta}$ is updated, another batch of tasks is selected and step 2 starts. After several epochs final $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ is emerged.
Final meta loss shows how well multiclass training is down for all batches of tasks together

MAML Training.

- Initialization is done by maintaining an explicit set of model parameters $\boldsymbol{\theta}$
- Adaptation procedure, or “inner loop”, takes $\boldsymbol{\theta}$ as input and returns θ'_i adapted specifically for **task instance $\boldsymbol{T_i}$** , by iteratively using gradient descent, and termination, which is handled simply by choosing a fixed number of optimization steps in the “inner loop”.
- MAML updates $\boldsymbol{\theta}$ by differentiating through the “inner loop” in order to minimize errors of **task instance-specific adapted models** $f_{\theta'_i}$ on the corresponding query set (test set), which contributes to the meta loss.

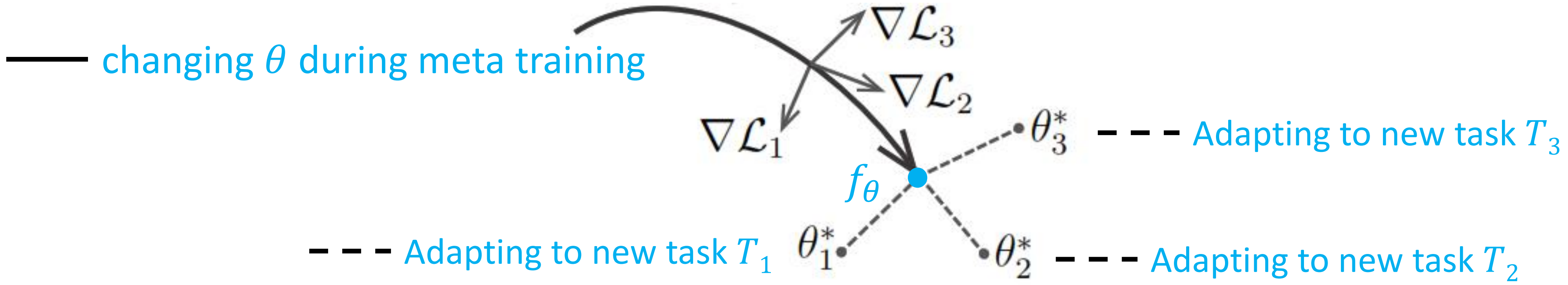
New task training (adaptation)

- Finally, $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ can be used as initial parameters for learning any new task
- Using $\boldsymbol{\theta}_{init} = \boldsymbol{\theta}^*$ can quickly learn a new task using only:
    - a few training examples
    - a few training iterations

Learning Model f_θ

- **We make no assumption on the form of the learner $\boldsymbol{f_\theta}$**, other than to assume that it is parametrized by some parameter vector $\boldsymbol{\theta}$, and that the loss function is smooth enough in $\boldsymbol{\theta}$ that we can use gradient-based learning techniques.

- We find model parameters $\boldsymbol{\theta}$ that are sensitive to changes in the task, such that small changes in the parameters will produce large improvements on the loss function of any task drawn from $p(T)$.



changing θ during meta training

Adapting to new task T_3

Adapting to new task T_1

Adapting to new task T_2

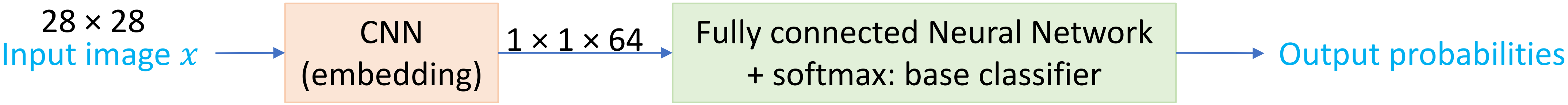
Experimental setups

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

- $\boldsymbol{f_{\theta}}$: A simple CNN as the embedding function is used, followed by a fully connected layer, then, followed by a softmax non-linearity is used to define the Baseline Classifier.

- CNN architecture:
  - CNN consisting of a stack of modules, each of which is a 3×3 convolution with 64 filters followed by batch normalization, a Relu non-linearity and 2×2 max-pooling.
  - We resized all the images to 28×28 so that, when we stack 4 modules, the resulting feature map is $1 \times 1 \times 64$, resulting in our embedding function.

f_{θ}


28 × 28

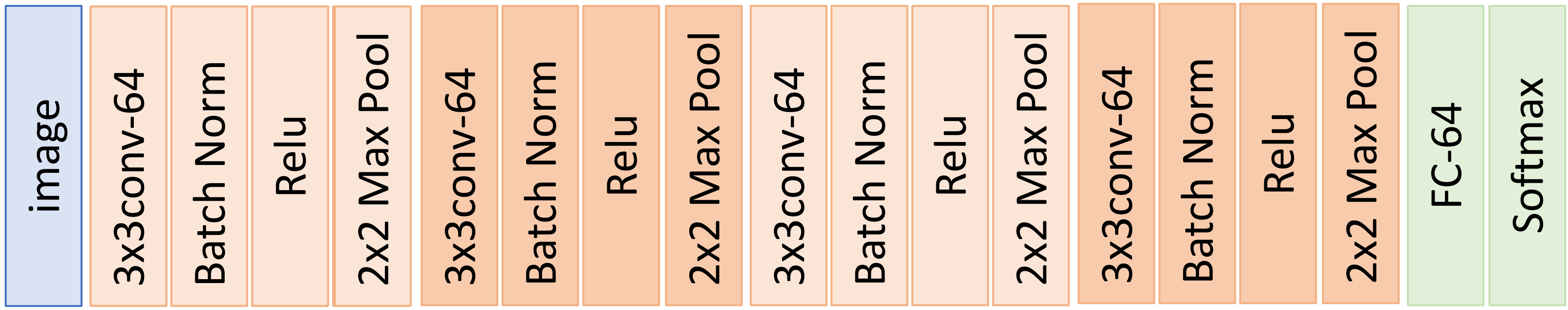
Input image x

CNN (embedding)

1 × 1 × 64

Fully connected Neural Network + softmax: base classifier

Output probabilities



image

3x3conv-64

Batch Norm

Relu

2x2 Max Pool

3x3conv-64

Batch Norm

Relu

2x2 Max Pool

3x3conv-64

Batch Norm

Relu

2x2 Max Pool

3x3conv-64

Batch Norm

Relu

2x2 Max Pool

FC-64

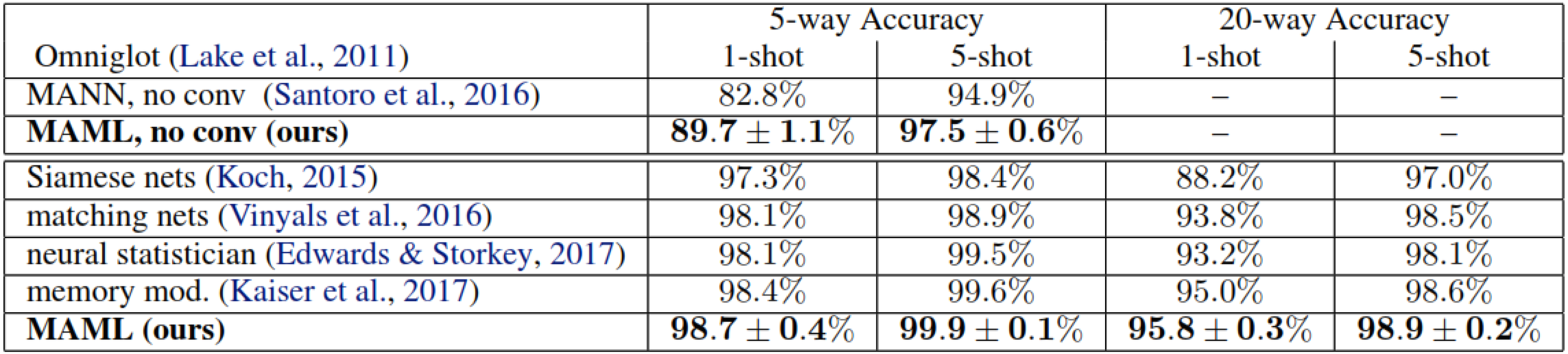
Softmax

Omniglot dataset

- Omniglot dataset consists of 20 examples of 1623 characters from 50 different alphabets. Each example was written by a different person.

Source : https://paperswithcode.com/dataset/omniglot-1

Table 1- Few-shot classification on held-out Omniglot characters



Omniglot (Lake et al., 2011)	5-way Accuracy 1-shot	5-way Accuracy 5-shot	20-way Accuracy 1-shot	20-way Accuracy 5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	**89.7 ± 1.1%**	**97.5 ± 0.6%**	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	**98.7 ± 0.4%**	**99.9 ± 0.1%**	**95.8 ± 0.3%**	**98.9 ± 0.2%**

The ∓ shows 95% confidence intervals over tasks.

Chelsea Finn, Pieter Abbeel, Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, International conference on Machine Learning, Sydney, Australia, PMLR 70, 2017

Discussion on Table 1

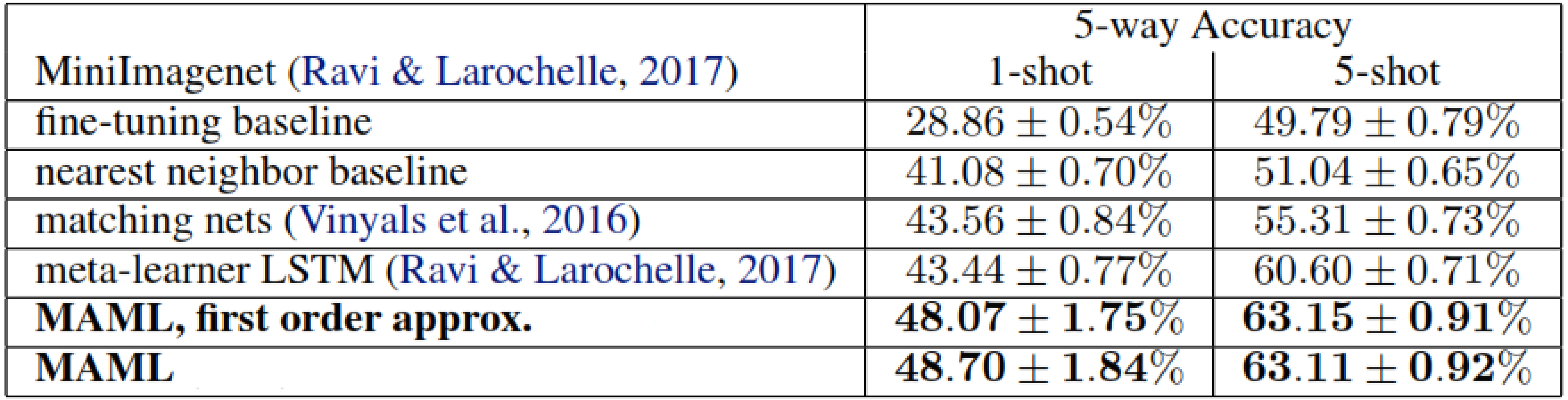
- MAML achieves results that are comparable to or outperform state-of-the-art convolutional and recurrent models.
- MAML uses fewer overall parameters compared to matching networks and the meta-learner LSTM, since the algorithm **does not introduce any additional parameters beyond the weights of the classifier itself.**
- Siamese nets, matching nets, and the memory module approaches are all specific to classification, and are not directly applicable to regression or RL scenarios.

MiniImagenet dataset

- Mini-Imagenet is proposed by **Matching Networks for One Shot Learning.** This dataset consists of 50000 training images and 10000 testing images, evenly distributed across 100 classes.

Table 2

- Few-shot classification on the MiniImagenet test set.
- MiniImagenet dataset involves 64 training classes, 12 validation classes, and 24 test classes.

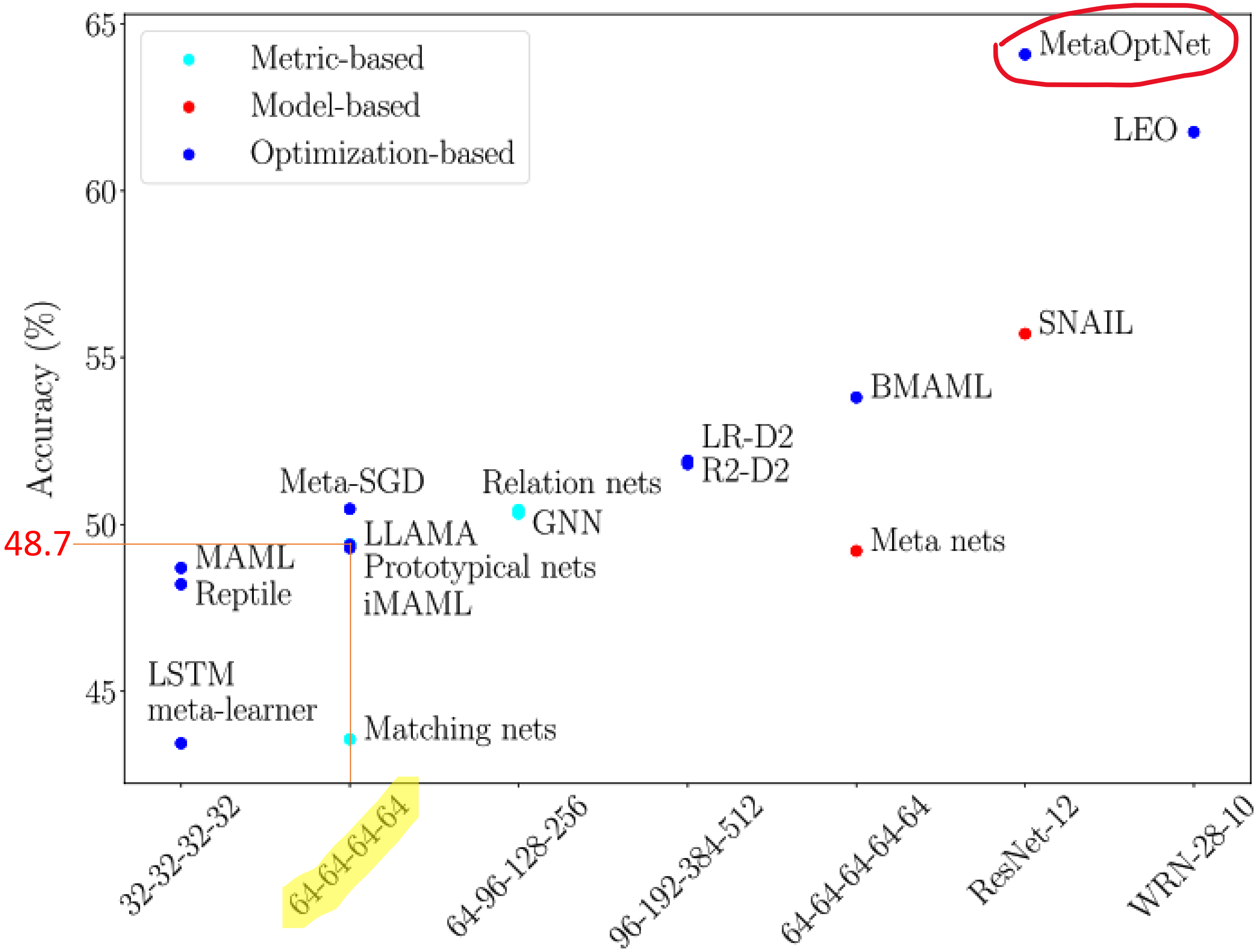


	5-way Accuracy	
MiniImagenet (Ravi & Larochelle, 2017)	1-shot	5-shot
fine-tuning baseline	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
nearest neighbor baseline	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
matching nets (Vinyals et al., 2016)	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
meta-learner LSTM (Ravi & Larochelle, 2017)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML, first order approx.	$\mathbf{48.07 \pm 1.75\%}$	$\mathbf{63.15 \pm 0.91\%}$
MAML	$\mathbf{48.70 \pm 1.84\%}$	$\mathbf{63.11 \pm 0.92\%}$

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Progress in Meta Learning

Fig. 1 The accuracy scores of the covered techniques on 1-shot miniImageNet classification. The used feature embedding CNN is displayed on the x-axis. As one can see, there is a strong relationship between the network complexity and the classification performance



From: Huisman2021_Article_ASurveyOfDeepMeta-learning

Contents

1. Introduction
2. Meta-Learning Problem Statement
3. Model-Agnostic Meta-Learning
4. MetaOptNet
5. Error decomposition in Few-Shot Learning

4- MetaOptNet

MetaOptNet

- Lee K, Maji S, Ravichandran A, Soatto S (2019) Meta-learning with diferentiable convex optimization. In Proceedings of the IEEE conference on computer vision and pattern recognition, IEEE, pp10657–10665

- MetaOptNet is related to learning the initializations (Model based) using backpropagation for optimization procedures.

SVM in inner loop

- MetaOptNet advocates the use of SVM which can be formulated as convex learning problems.
- The loss objective is a **quadratic program** (QP) which can be efficiently solved to obtain its **global optima** using **gradient-based** techniques.
- Solution to convex problems can be characterized by their Karush-KuhnTucker (KKT) conditions which allow to backpropagate through SVM using the **implicit function theorem**.
- We can use the formulation of Amos and Kolter [1] which provides efficient GPU routines for computing solutions to QPs and their gradients.

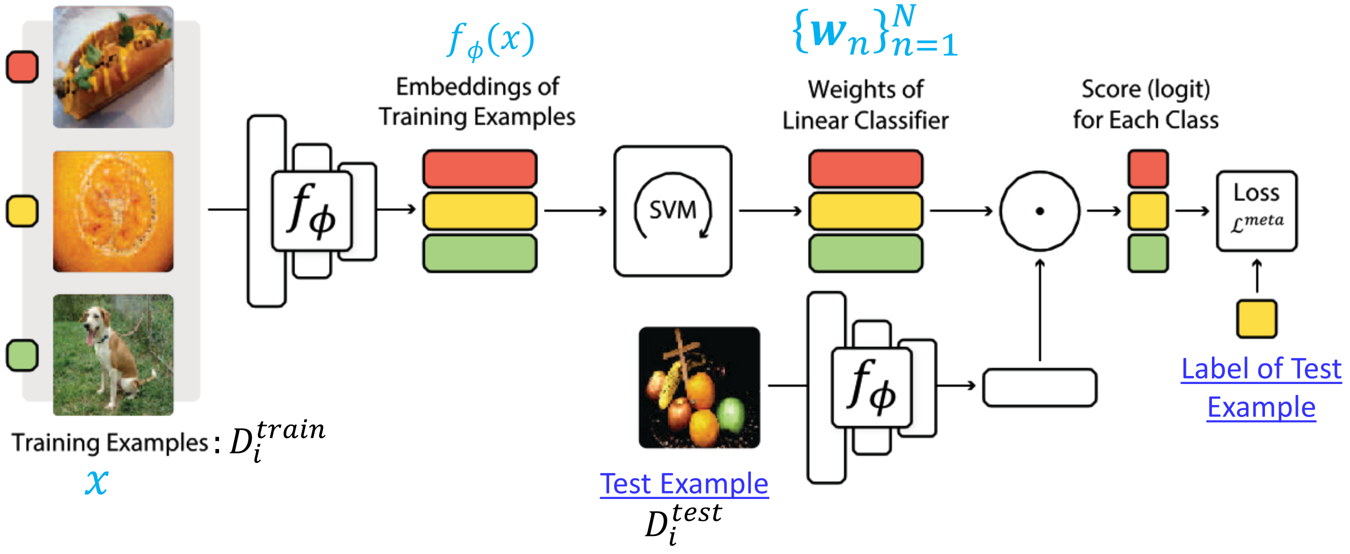
1. Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In ICML, 2017.

Figure 1. Illustration of the training and testing for a single task

- Schematic illustration of MetaOptNet on an 1-shot 3-way classification task. The meta-training objective is to learn the parameters ϕ of a feature embedding model f_{ϕ} that generalizes well across tasks when used with regularized linear classifiers (e.g., SVMs). A task is a tuple of a few-shot training (support) set and a test (query) set.

- SVM is used to learn a classifier given a set of labeled training examples and the generalization error is computed on a test set of examples from the same task.

- In order to make the system end-to-end trainable, we require that the solution of the SVM solver should be differentiable with respect to its input, so the gradients of the SVM objective can be used during backpropagation.



$f_{\phi}(x)$
Embeddings of Training Examples
$\{w_n\}_{n=1}^{N}$
Weights of Linear Classifier
Score (logit) for Each Class
f_{ϕ}
SVM
Loss $\mathcal{L}^{meta}$
Training Examples : D_i^{train}
x
Test Example
D_i^{test}
Label of Test Example

$f_{\phi}(x)$ is embedded feature vector of example x

Meta-learning approaches for few-shot learning

- Meta-learning approaches for few-shot learning aim to minimize the generalization error across a distribution of tasks sampled from a task distribution.
- This can be thought of as learning over a collection of tasks:
  - $T = \left\{\left(D_i^{train}, D_i^{test}\right)\right\}_{i=1}^{I}$, often referred to as a meta training set.
    - Tuple $\left(D_i^{train}, D_i^{test}\right)$ describes a training and a test dataset, or a **task**.

Problem formulation

- Given I tasks data set $T = \left\{\left(D_i^{train}, D_i^{test}\right)\right\}_{i=1}^{I}$, each task consisting of training set $D_i^{support} = D_i^{train} = \{x_t, y_t\}_{t=1}^{T_i}$, and test set $D_i^{query} = D_i^{test} = \{x_t, y_t\}_{t=1}^{Q_i}$
- Goal of the base learner $\mathcal{A}$ is to learn from $D^{train} = \left\{\left(D_i^{train}\right)\right\}_{i=1}^{I}$ and estimate parameters $\boldsymbol{\theta}$ of the predictor $y = f(x;\ \boldsymbol{\theta})$ so that it generalizes well to the unseen test set $D^{test} = \left\{\left(D_i^{test}\right)\right\}_{i=1}^{I}$.
- It is often assumed that the training and test set are sampled from the same task distribution $p(\mathcal{T})$, and the domain is mapped to a feature space using an embedding model f_ϕ parameterized by ϕ.
- The objective is to learn an embedding modelf_ϕ that minimizes generalization (or test) error across tasks given a base learner $\mathcal{A}$.

Training the base learner

- For optimization based learners, the parameters are obtained by minimizing the empirical loss over training data ,D^{train} ,along with a regularization that encourages simpler models. This can be written as:

$$\theta = \mathcal{A}\left(D^{train}; \phi\right) = arg\,\min_{\theta} \mathcal{L}^{base}\left(D^{train}; \theta, \phi\right) + \lambda \mathcal{R}(\theta) \quad (1)$$

- Where $\mathcal{L}^{base}$ is a loss function, such as the negative loglikelihood of labels, and $\mathcal{R}(\theta)$ is a regularization term. Regularization plays an important role in generalization when training data is limited
- ϕ is embedding learnable parameters and θ is classifier learnable parameters

Meta-Training and Meta Testing

- Formally, the learning objective for task i is:

$$\min_{\phi} \mathbb{E}_T\left[\mathcal{L}^{meta}\left(D_i^{test}; \theta, \phi\right)\right], where\ \theta = \mathcal{A}\left(D_i^{train}; \phi\right) \qquad (2)$$

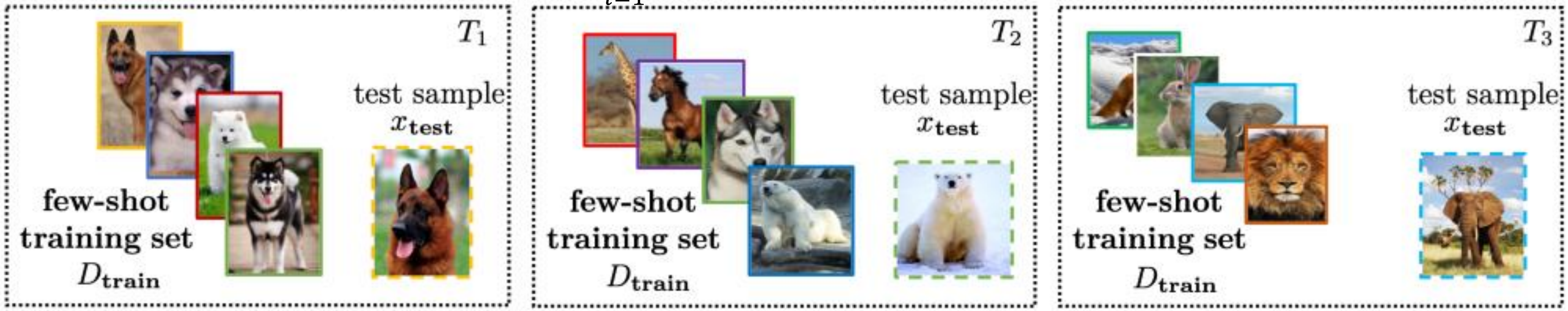
- Once an embedding model f_ϕ is learned for all I tasks, its generalization is estimated on a set of held-out tasks (often referred to as a meta-test set) $S = \left\{\left(D_j^{train}, D_j^{test}\right)\right\}_{j=1}^{J}$ computed as:

$$\mathbb{E}_S\left[\mathcal{L}^{meta}\left(D_j^{test}; \theta, \phi\right), where\ \theta = \mathcal{A}\left(D_j^{train}; \phi\right)\right] \qquad (3)$$

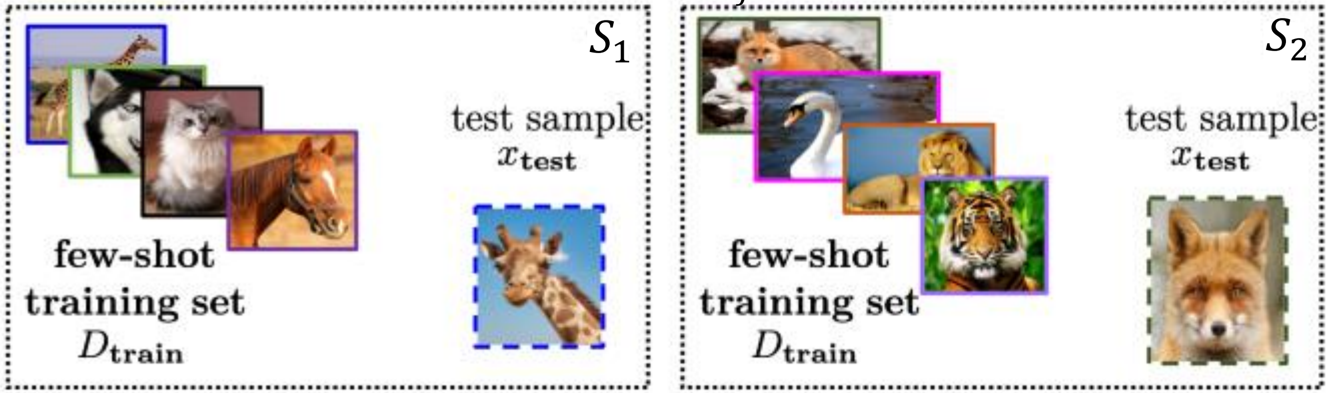
- Expectation in Equation 2 and 3 are meta-training and meta-testing respectively.

T:meta training set - S:meta-test set

meta-training tasks $T = \{(D_i^{train}, D_i^{test})\}_{i=1}^{I}$



meta-testing tasks $S = \{(D_j^{train}, D_j^{test})\}_{j=1}^{J}$



Episodic sampling of tasks

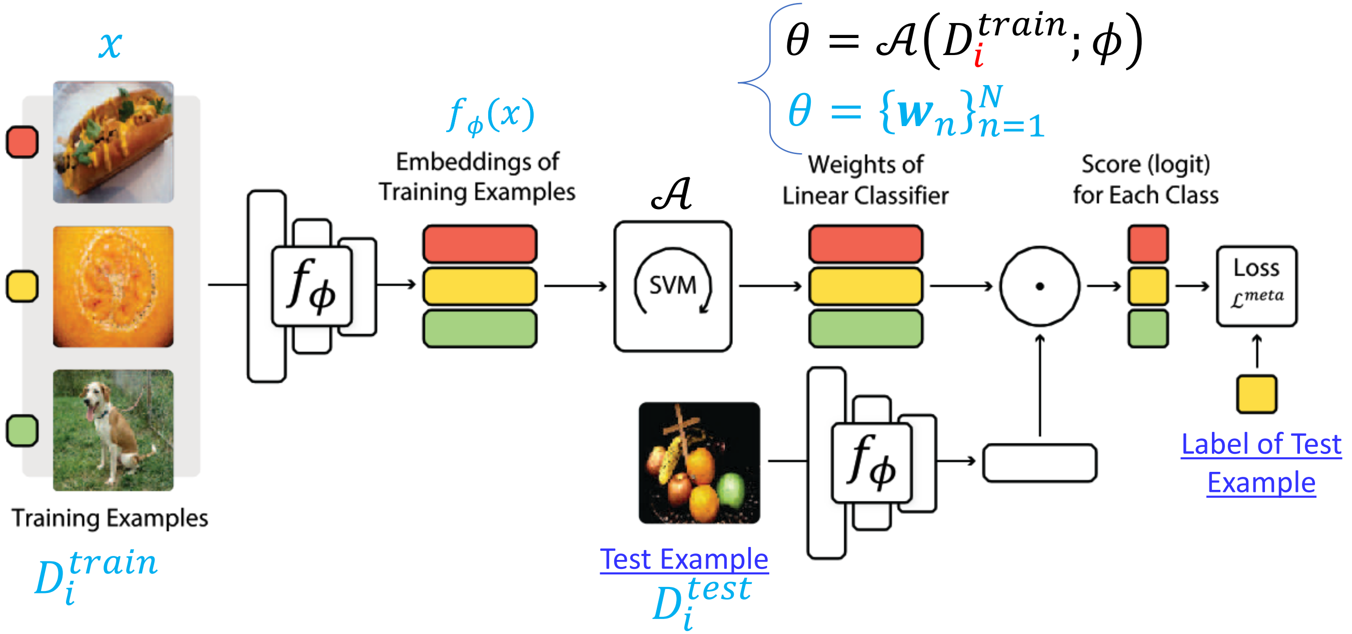
- Standard few-shot learning benchmarks such as miniImageNet evaluate models in N-way, K-shot classification tasks.
- Here N denotes the number of classes, and K denotes the number of training examples per class.
    - Few shot learning techniques are evaluated for **small values of K, typically K ∈ {1, 5}.**
- In practice, these datasets do not explicitly contain tuples $\left(D_i^{train}, D_i^{test}\right)$, but **each task for meta-learning (commonly described as an episode)** is constructed “on the fly” during the meta training stage,

Figure 1. Illustration of the training and testing for a single task

- Schematic illustration of MetaOptNet on **one classification task T_i** : 1-shot 3-way
- Using T ,meta-training objective is to learn the parameters ϕ of a feature embedding model f_ϕ that generalizes well across tasks when multi class SVM is used.

$$T_i = \left(D_i^{train}, D_i^{test}\right)$$

$$\begin{cases} \theta = \mathcal{A}\left(D_i^{train}; \phi\right) \\ \theta = \{\boldsymbol{w}_n\}_{n=1}^{N} \end{cases}$$



$f_\phi(x)$ is embedded feature vector of example x

Convex base learners

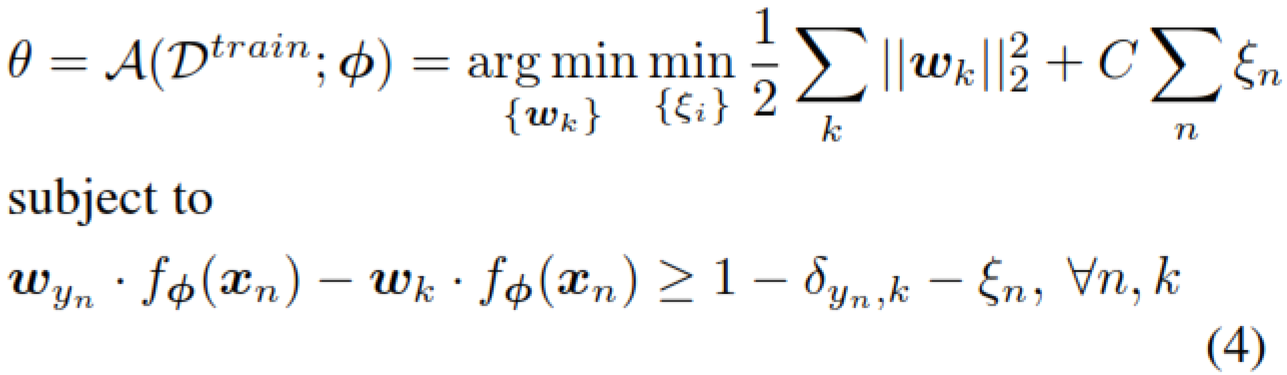
- Choice of the base learner $\mathcal{A}$ has a significant impact on Equation 2

$$\min_{\phi} \mathbb{E}_T\left[\mathcal{L}^{meta}\left(D_i^{test};\theta,\phi\right)\right], where\ \theta = \mathcal{A}\left(D_i^{train};\phi\right) \qquad (2)$$

- The base learner that computes $\theta = \mathcal{A}\left(D_i^{train};\phi\right)$ has to be efficient since the expectation has to be computed over a distribution of tasks.
- Moreover, to estimate parameters ϕ of the embedding model the gradients of the task test error $\mathcal{L}^{meta}\left(D_i^{test};\theta,\phi\right)$ with respect to ϕ have to be efficiently computed. This has motivated simple base learners such as nearest class mean for which the parameters of the base learner θ are easy to compute and the objective is differentiable.

SVM as the base learner

- We consider base learners based on multi-class linear classifiers (e.g., SVMs with hinge loss and ℓ_2 regularization) where the base learner's objective is convex.
- For example, a N-class linear SVM can be written as $\theta = \{\boldsymbol{w}_n\}_{n=1}^{N}$
- The Crammer and Singer [1] formulation of the multi-class SVM is:



$$\theta = \mathcal{A}(\mathcal{D}^{train};\phi) = \underset{\{\boldsymbol{w}_k\}}{\arg\min}\ \min_{\{\xi_i\}} \frac{1}{2}\sum_k ||\boldsymbol{w}_k||_2^2 + C\sum_n \xi_n$$

subject to

$$\boldsymbol{w}_{y_n}\cdot f_\phi(\boldsymbol{x}_n) - \boldsymbol{w}_k\cdot f_\phi(\boldsymbol{x}_n) \geq 1 - \delta_{y_n,k} - \xi_n,\ \forall n,k \tag{4}$$

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res., 2:265–292, Mar. 2002.

Meta learning objective

- To measure the performance of the model we evaluate the negative log-likelihood of the test data sampled from the same task. Hence, we can re-express the meta-learning objective of Equation 2 as:

$$
\begin{aligned}
&\mathcal{L}^{meta}(\mathcal{D}^{test}; \theta, \phi, \gamma) = \\
&\sum_{(\boldsymbol{x},y)\in\mathcal{D}^{test}} \left[-\gamma \underline{\boldsymbol{w}_y \cdot f_\phi(\boldsymbol{x})} + \log \sum_k \exp(\gamma \underline{\boldsymbol{w}_k \cdot f_\phi(\boldsymbol{x})})\right] \qquad (12)
\end{aligned}
$$

prediction score

γ: a learnable scale parameter

- Where γ is a learnable scale parameter.
- Prior work in few-shot learning suggest that adjusting the prediction score by a learnable scale parameter γ provides better performance under nearest class mean.
- We empirically find that inserting γ is beneficial for the meta-learning with SVM base learner as well.
- While other choices of test loss, such as hinge loss, are possible, loglikelihood worked the best in our experiments.

Implementation details of MetaOptNet-1

- ResNet-12 network is used for learning embedding function f_ϕ.
- Let Rk denote a residual block that consists of three

  {3×3 convolution with k filters, batch normalization, Leaky ReLU(0.1)}
- Let MP denote a 2×2 max pooling.
- DropBlock regularization is used, it is a form of structured Dropout.
- Let $DB(k, b)$ denote a DropBlock layer with $keep\ rate = k$ and $block\ size = b.$
- The network architecture for ImageNet derivatives is:

$$R64 - MP - DB(0.9,1) - R160 - MP - DB(0.9,1) - R320 - MP - DB(0.9,5) - R640 - MP - DB(0.9,5)$$

- The network architecture used for CIFAR derivatives is:

$$R64 - MP - DB(0.9,1) - R160 - MP - DB(0.9,1) - R320 - MP - DB(0.9,2) - R640 - MP - DB(0.9,2)$$

Implementation details of MetaOptNet -2

- As an **optimizer**, SGD is used with Nesterov momentum of 0.9 and weight decay of 0.0005.
- Each mini-batch consists of **8 episodes**. The model was meta-trained for 60 epochs, with each epoch consisting of 1000 episodes.
- The learning rate was initially set to 0.1, and then changed to 0.006, 0.0012, and 0.00024 at epochs 20, 40 and 50.

Base-learner setup

- For linear SVM training, the quadratic programming (QP) solver OptNet [1] is used.
- Regularization parameter C of SVM was set to 0.1.
- **Early stopping**: Although we can run the optimizer until convergence, in practice we found that running the QP solver for a fixed number of iterations (**just three**) works well in practice.
- Early stopping acts an additional regularizer and even leads to a slightly better performance.

[1] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017

Experiments on ImageNet derivatives-1

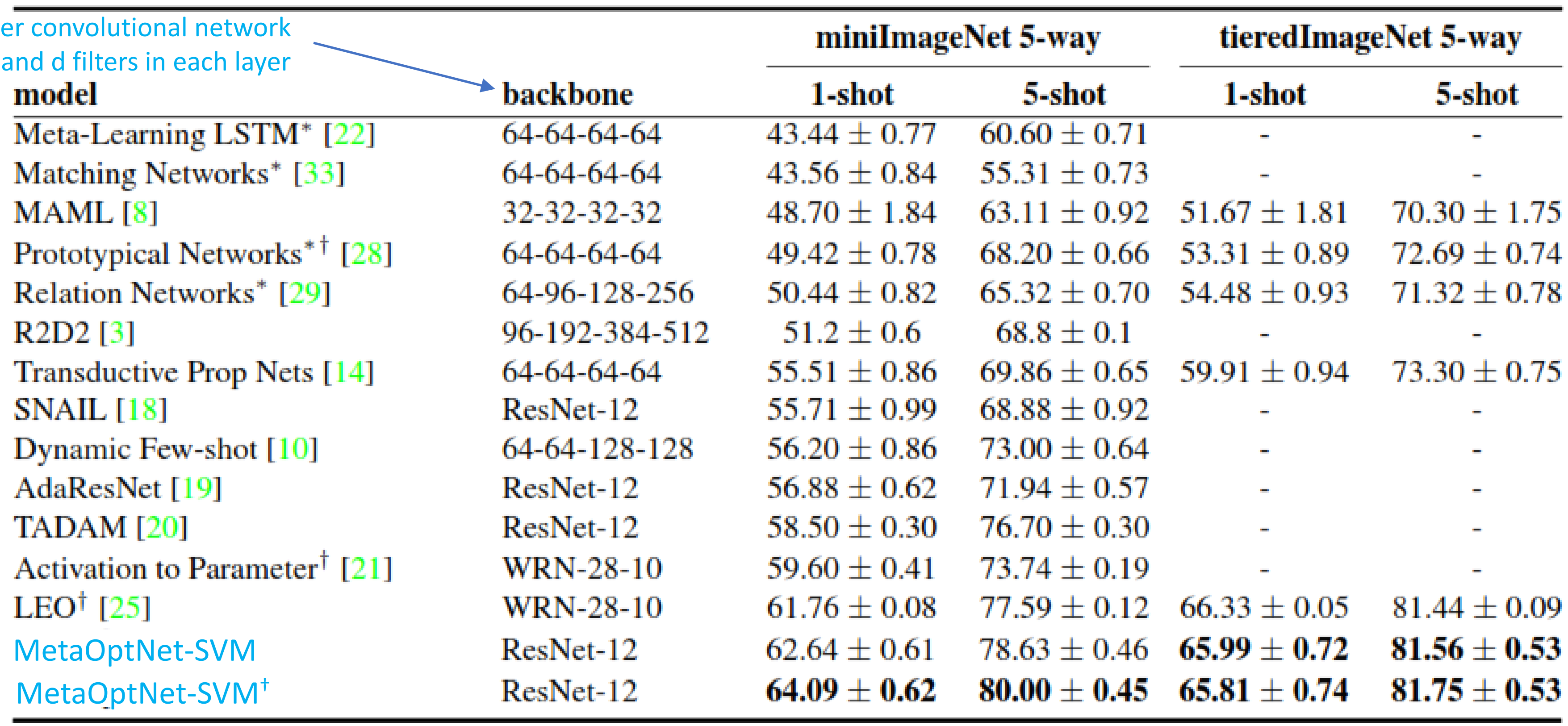
- The **miniImageNet** dataset is a standard benchmark for few-shot image classification benchmark, consisting of 100 randomly chosen classes from ILSVRC-2012.
- These classes are randomly split into 64, 16 and 20 classes for **meta-training**, **meta-validation**, and **meta-testing** respectively.
- Each class contains 600 images of size 84×84.

Experiments on ImageNet derivatives-2

- The **tieredImageNet** benchmark is a larger subset of ILSVRC-2012, composed of 608 classes grouped into 34 high-level categories.
- These are divided into 20 categories for meta-training, 6 categories for meta-validation, and 8 categories for meta-testing.
- This corresponds to 351, 97 and 160 classes for meta-training, meta-validation, and meta-testing respectively.
- All images are of size 84 × 84.

Table 1. Comparison to prior work on miniImageNet and tieredImageNet

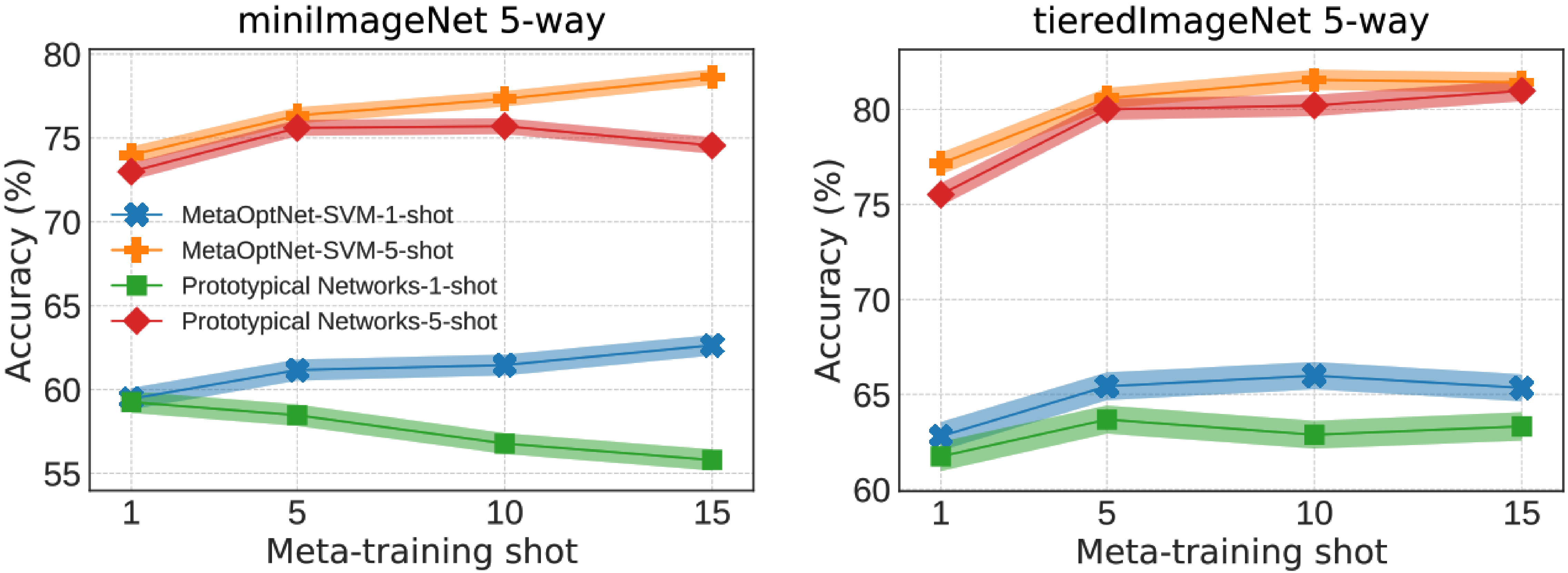
a-b-c-d denotes a 4-layer convolutional network with a, b, c, and d filters in each layer



model	backbone	miniImageNet 5-way		tieredImageNet 5-way	
		1-shot	5-shot	1-shot	5-shot
Meta-Learning LSTM* [22]	64-64-64-64	43.44 ± 0.77	60.60 ± 0.71	-	-
Matching Networks* [33]	64-64-64-64	43.56 ± 0.84	55.31 ± 0.73	-	-
MAML [8]	32-32-32-32	48.70 ± 1.84	63.11 ± 0.92	51.67 ± 1.81	70.30 ± 1.75
Prototypical Networks*† [28]	64-64-64-64	49.42 ± 0.78	68.20 ± 0.66	53.31 ± 0.89	72.69 ± 0.74
Relation Networks* [29]	64-96-128-256	50.44 ± 0.82	65.32 ± 0.70	54.48 ± 0.93	71.32 ± 0.78
R2D2 [3]	96-192-384-512	51.2 ± 0.6	68.8 ± 0.1	-	-
Transductive Prop Nets [14]	64-64-64-64	55.51 ± 0.86	69.86 ± 0.65	59.91 ± 0.94	73.30 ± 0.75
SNAIL [18]	ResNet-12	55.71 ± 0.99	68.88 ± 0.92	-	-
Dynamic Few-shot [10]	64-64-128-128	56.20 ± 0.86	73.00 ± 0.64	-	-
AdaResNet [19]	ResNet-12	56.88 ± 0.62	71.94 ± 0.57	-	-
TADAM [20]	ResNet-12	58.50 ± 0.30	76.70 ± 0.30	-	-
Activation to Parameter† [21]	WRN-28-10	59.60 ± 0.41	73.74 ± 0.19	-	-
LEO† [25]	WRN-28-10	61.76 ± 0.08	77.59 ± 0.12	66.33 ± 0.05	81.44 ± 0.09
MetaOptNet-SVM	ResNet-12	62.64 ± 0.61	78.63 ± 0.46	**65.99 ± 0.72**	**81.56 ± 0.53**
MetaOptNet-SVM†	ResNet-12	**64.09 ± 0.62**	**80.00 ± 0.45**	**65.81 ± 0.74**	**81.75 ± 0.53**

Average classification accuracies (%) with 95% confidence intervals on miniImageNet and tieredImageNet meta-test splits.
†Used the union of meta-training set and meta-validation set to meta-train the meta-learner.

Figure 2: Test accuracies, varying meta-training shot



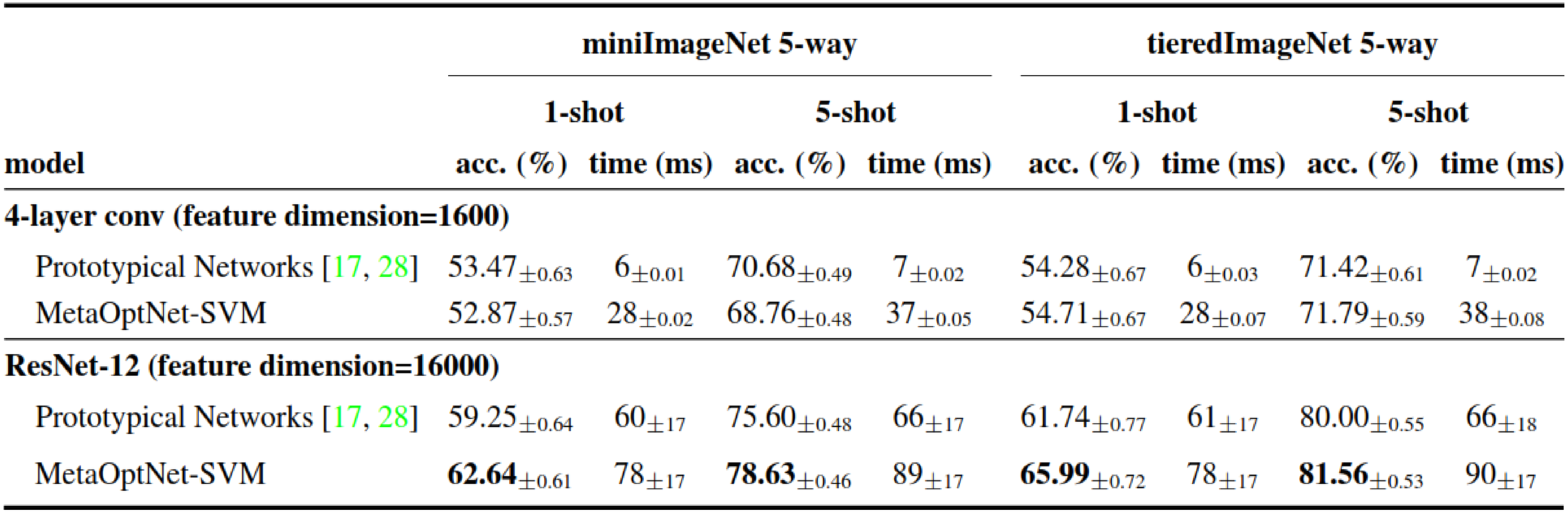
Meta-training shot vs Meta-testing shot

- For prototypical networks, we match the **meta-training shot** to **meta-testing shot** following the usual practice.
- For SVM, we observe that keeping **meta-training shot** higher than **meta-testing shot** leads to better test accuracies as shown in Figure 2.
- Hence, during meta-training, we set **training shot** to 15 for miniImageNet with ResNet-12; 5 for miniImageNet with 4-layer CNN (in Table 3); 10 for tieredImageNet; 5 for CIFAR-FS; and **15** for **FC100**

Comparisons between base learners

- When we use a standard 4-layer convolutional network where the feature dimension is low (1600), we do not observe a substantial benefit of adopting discriminative classifiers for few-shot learning.
- When the embedding dimensional is much higher (16000), SVMs yield better few-shot accuracy than other base learners.
- Thus, regularized linear classifiers provide robustness when high dimensional features are available. See next slide
- For ResNet-12 (16000 features), compared to **nearest class mean classifier (ProtoNet)**, the additional overhead around 30-50% for the SVM base learner.
- Performance on both 1-shot and 5-shot tests, generally increases with increasing meta-training shot.
- This makes the approach more practical as we can **meta-train the embedding once with a high shot for all meta-testing shots.**

Table 3: Comparisons between base learners (Accuracy, Training time)



	miniImageNet 5-way				tieredImageNet 5-way			
	1-shot		5-shot		1-shot		5-shot	
model	**acc. (%)**	**time (ms)**	**acc. (%)**	**time (ms)**	**acc. (%)**	**time (ms)**	**acc. (%)**	**time (ms)**
4-layer conv (feature dimension=1600)								
Prototypical Networks [17, 28]	53.47$_{\pm 0.63}$	6$_{\pm 0.01}$	70.68$_{\pm 0.49}$	7$_{\pm 0.02}$	54.28$_{\pm 0.67}$	6$_{\pm 0.03}$	71.42$_{\pm 0.61}$	7$_{\pm 0.02}$
MetaOptNet-SVM	52.87$_{\pm 0.57}$	28$_{\pm 0.02}$	68.76$_{\pm 0.48}$	37$_{\pm 0.05}$	54.71$_{\pm 0.67}$	28$_{\pm 0.07}$	71.79$_{\pm 0.59}$	38$_{\pm 0.08}$
ResNet-12 (feature dimension=16000)								
Prototypical Networks [17, 28]	59.25$_{\pm 0.64}$	60$_{\pm 17}$	75.60$_{\pm 0.48}$	66$_{\pm 17}$	61.74$_{\pm 0.77}$	61$_{\pm 17}$	80.00$_{\pm 0.55}$	66$_{\pm 18}$
MetaOptNet-SVM	**62.64**$_{\pm 0.61}$	78$_{\pm 17}$	**78.63**$_{\pm 0.46}$	89$_{\pm 17}$	**65.99**$_{\pm 0.72}$	78$_{\pm 17}$	**81.56**$_{\pm 0.53}$	90$_{\pm 17}$

Overfitting in Meta-learning

- Meta-learning algorithms aim to learn two components:
  - a model that embeds the examples of all training tasks
  - a base learner that is trained when given examples from a new task
- This additional level of learning can be powerful, but it also creates another potential source of overfitting, since we can now **overfit in either the model or the base learner**

Meta overfitting in MetaOptNet

- At the end of meta-training MetaOptNet-SVM with ResNet- 12 achieves nearly 100% **test accuracy on all the meta training datasets** except the tieredImageNet.
- To alleviate overfitting, we use the **union of the meta-training and meta-validation sets** to meta-train the embedding (augmenting meta-training set). And also, we terminate the meta-training after 21 epochs for mini- ImageNet, 52 epochs for tieredImageNet.

Meta Overfiting

- Meta-learning has two levels of optimization: an inner-loop optimization and an outer-loop optimization
- In meta learning, there are two types of overfitting that can happen: 1) memorization overfitting and 2) learner overfitting
- Memorization overfitting occurs when the base learner and D_i^{train} does not impact the model's prediction of D_i^{test}
- Learner overfitting occurs when the model and base learner leverage both D_i^{train} and D_i^{test}, but fails to generalize to the meta-test set $S_j = \{D_j^{train}, D_j^{test}\}$.

Concluding MetaOpNet

- MetaOptNet is a meta-learning approach with convex base learners for few-shot learning.
- Linear classifiers offer better generalization than nearest neighbor classifiers at a modest increase in computational costs (as seen in Table 3).
- Experiments suggest that regularized linear models allow significantly higher embedding dimensions with reduced overfitting.
- Future work can be the kernel SVMs. This would allow the ability to incrementally increase model capacity as more training data becomes available for a task.

Contents

1. Introduction
2. Meta-Learning Problem Statement
3. Model-Agnostic Meta-Learning
4. MetaOptNet
5. Error decomposition in Few-Shot Learning

5- Error decomposition in Few-Shot Learning

Error decomposition in Few-Shot Learning

- In any machine learning problem, usually there are prediction errors and one cannot obtain perfect predictions. In this section, we illustrate the core issue of FSL based on error decomposition in supervised machine learning.
- This analysis applies to FSL
  - supervised learning including classification,
  - regression,
  - also provide insights for understanding FSL reinforcement learning

Empirical Risk Minimization

- Given a hypothesis h we want to minimize its expected risk R, which is the loss measured with respect to $p(x, y)$. Specifically,

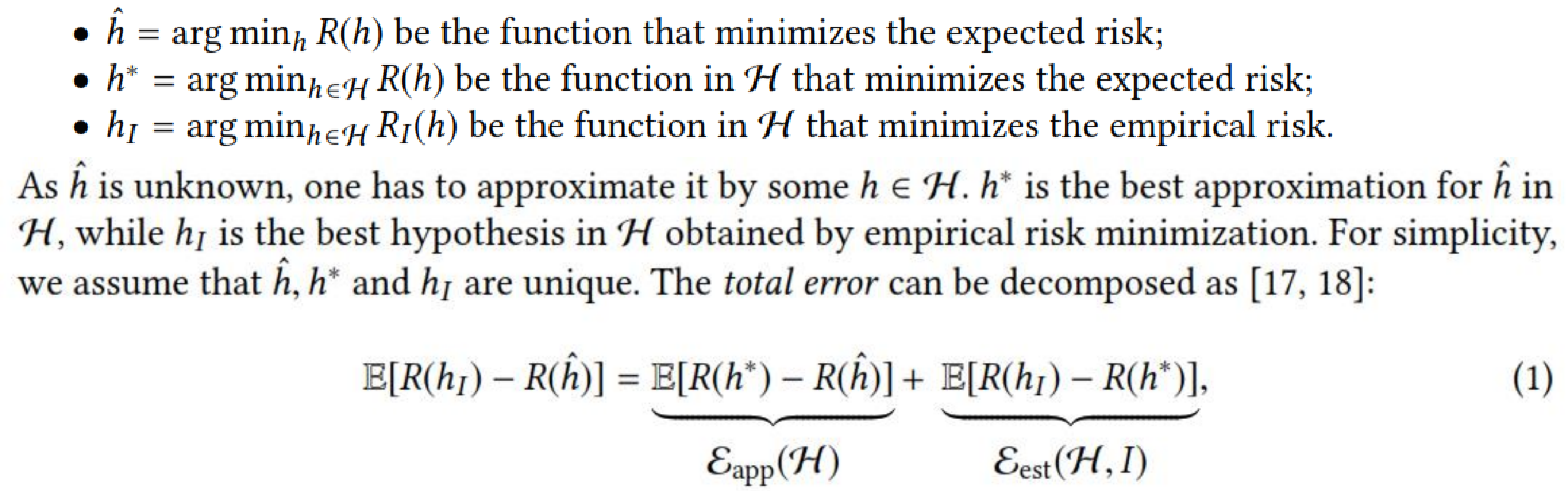
$$R(h) = \int \ell(h(x), y)\, dp(x, y) = \mathbb{E}[\ell(h(x), y)].$$

- As $p(x, y)$ is unknown, the empirical risk (which is the average of sample losses over the training set D_{train} of I samples)

$$R_I(h) = \frac{1}{I} \sum_{i=1}^{I} \ell(h(x_i), y_i),$$

Total Error Decomposition

- It is usually used as a proxy for $R(h)$, leading to empirical risk minimization (**with possibly some regularizes**). For illustration, let



- $\hat{h} = \arg\min_h R(h)$ be the function that minimizes the expected risk;
- $h^* = \arg\min_{h \in \mathcal{H}} R(h)$ be the function in $\mathcal{H}$ that minimizes the expected risk;
- $h_I = \arg\min_{h \in \mathcal{H}} R_I(h)$ be the function in $\mathcal{H}$ that minimizes the empirical risk.

As $\hat{h}$ is unknown, one has to approximate it by some $h \in \mathcal{H}$. h^* is the best approximation for $\hat{h}$ in $\mathcal{H}$, while h_I is the best hypothesis in $\mathcal{H}$ obtained by empirical risk minimization. For simplicity, we assume that $\hat{h}$, h^* and h_I are unique. The *total error* can be decomposed as [17, 18]:

$$\mathbb{E}[R(h_I) - R(\hat{h})] = \underbrace{\mathbb{E}[R(h^*) - R(\hat{h})]}_{\mathcal{E}_{\text{app}}(\mathcal{H})} + \underbrace{\mathbb{E}[R(h_I) - R(h^*)]}_{\mathcal{E}_{\text{est}}(\mathcal{H}, I)}, \tag{1}$$

- where the expectation is with respect to the random choice of D_{train}.

Learning to reduce the total error

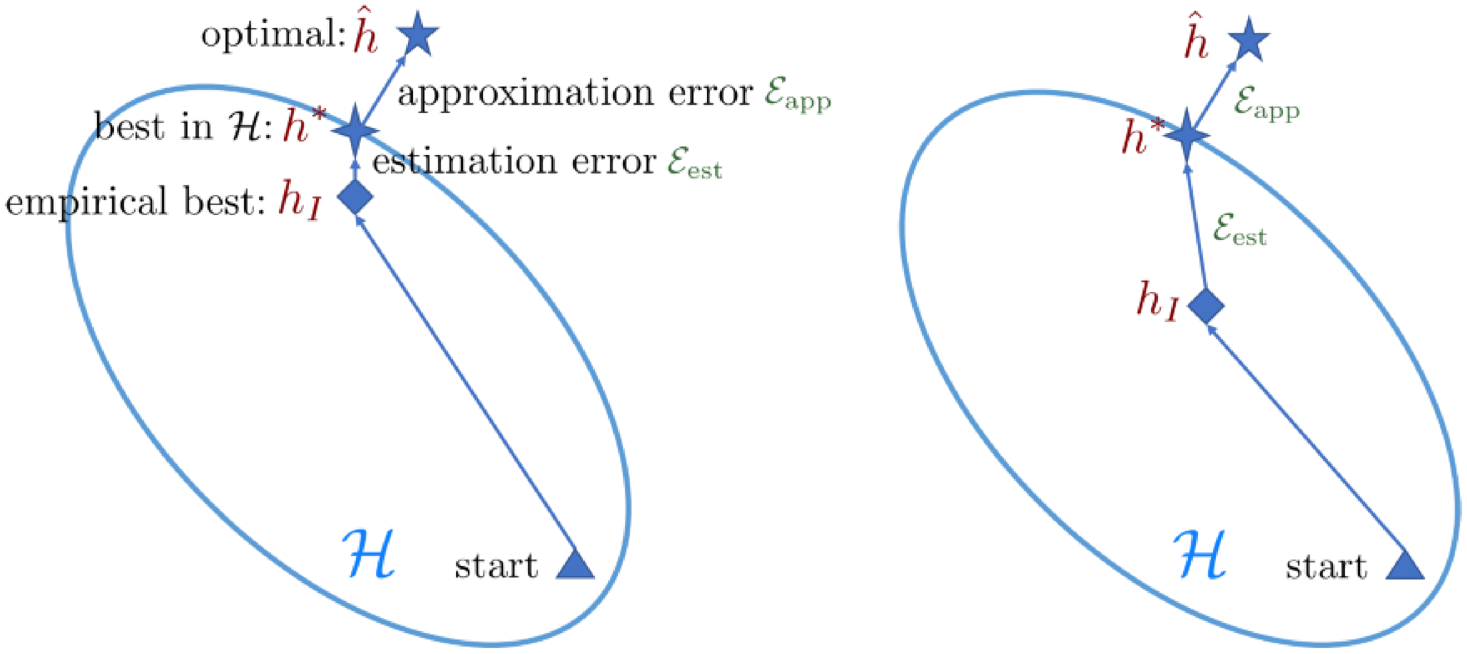
- The approximation error $\mathcal{E}_{app}(\mathcal{H})$ measures how close the functions in $\mathcal{H}$ can approximate the optimal hypothesis $\hat{h}$, and the estimation error $\mathcal{E}_{est}(\mathcal{H}, I)$ measures the effect of minimizing the empirical risk $R_I(h)$ instead of the expected risk $R(h)$ within $\mathcal{H}$.
- As shown, the total error is affected by $\mathcal{H}$ (hypothesis space) and I (number of examples in D_{train}). In other words, learning to reduce the total error can be attempted from the perspectives of:
- (i) data, which provides D_{train}
- (ii) model, which determines $\mathcal{H}$
- (iii) algorithm, which searches for the optimal $h_I \in \mathcal{H}$ that fits D_{train}

Unreliable Empirical Risk Minimizer.

- In general,$\mathcal{E}_{est}(\mathcal{H}, I)$ can be reduced by having a larger number of examples. Thus, when there is sufficient training data with supervised information (i.e., I is large), the empirical risk minimizer h_I can provide a good approximation $R(h_I)$ to the best possible $R(h^*)$ for h's in $\mathcal{H}$.
- However, in FSL, the number of available examples I is small. The empirical risk $R_I(h)$ may then be far from being a good approximation of the expected risk $R(h)$, and the resultant empirical risk minimizer h_I overfits.
- Indeed, this is the core issue of FSL supervised learning, i.e., the empirical risk minimizer h_I is no longer reliable. Therefore, FSL is much harder.

Fig. 1. Comparison of learning with sufficient and few training samples.

- A comparison of learning with sufficient and few training samples is shown in Figure 1.


optimal: $\hat{h}$
approximation error $\mathcal{E}_{\text{app}}$
best in $\mathcal{H}$: h^*
estimation error $\mathcal{E}_{\text{est}}$
empirical best: h_I
$\mathcal{H}$
start
$\hat{h}$
$\mathcal{E}_{\text{app}}$
h^*
$\mathcal{E}_{\text{est}}$
h_I
$\mathcal{H}$
start

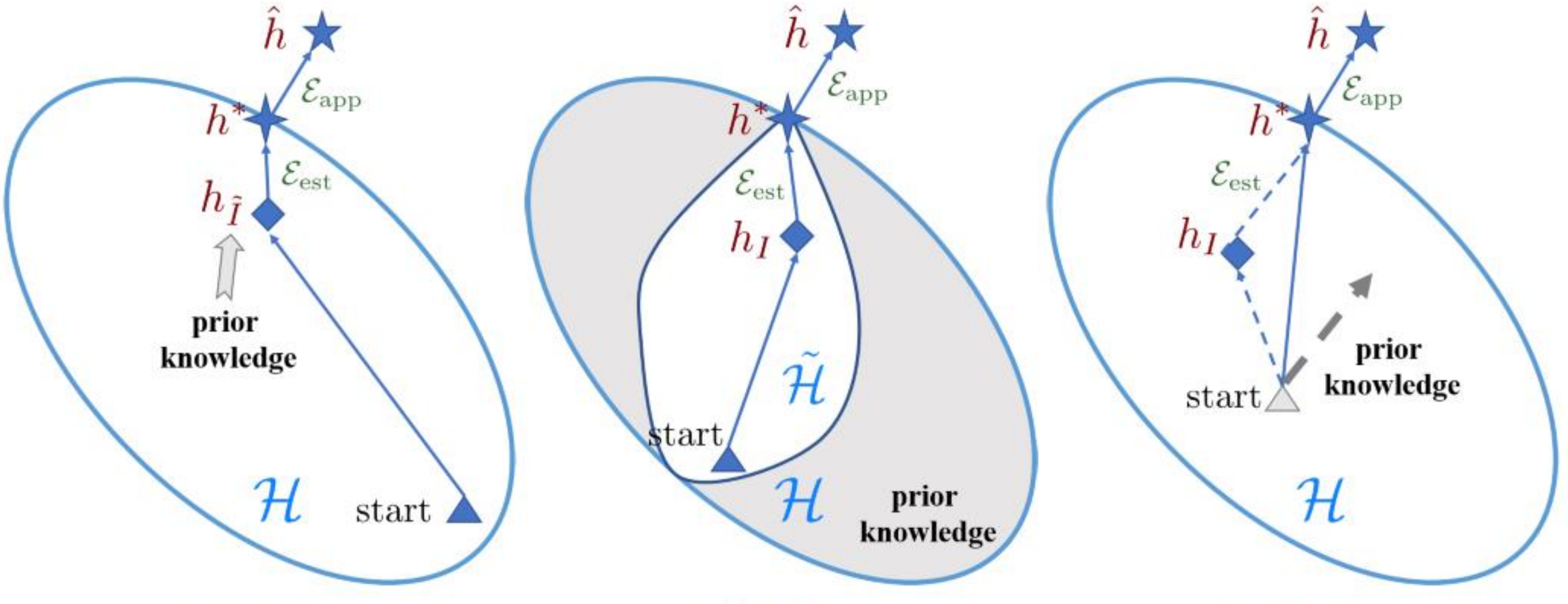
(a) Large I.

(b) Small I.

Controlling Estimation error in FLS

- To alleviate the problem of having an unreliable empirical risk minimizer h_I in FSL supervised learning, prior knowledge must be used.
- Based on which aspect is enhanced using prior knowledge, existing FSL works can be categorized into the following perspectives (Figure 2).

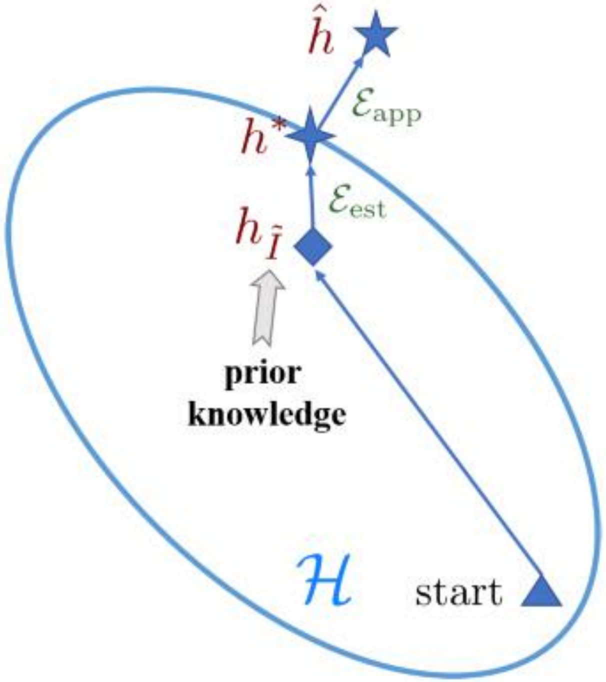
Fig. 2. Different perspectives on how FSL methods solve the few-shot problem.



(a) Data. (b) Model. (c) Algorithm.

(a) Data

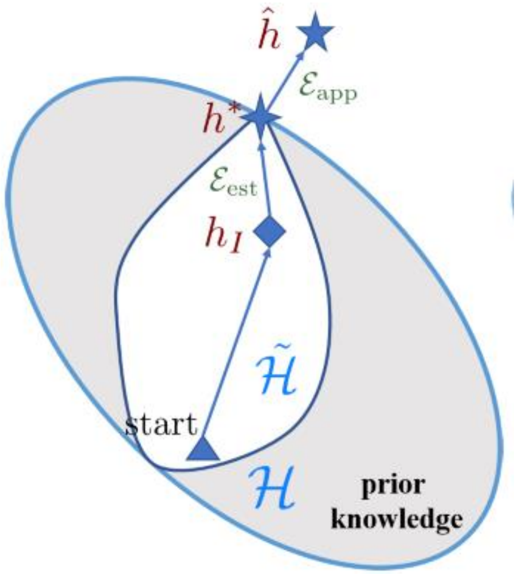
- These methods use prior knowledge to **augment** D_{train}, and increase the number of samples from I to $\hat{I}$, where $\hat{I} \gg I$.
- Standard machine learning models and algorithms can then be used on the **augmented data**, and a more accurate empirical risk minimizer $h_{\hat{I}}$ can be obtained (Figure 2(a)).



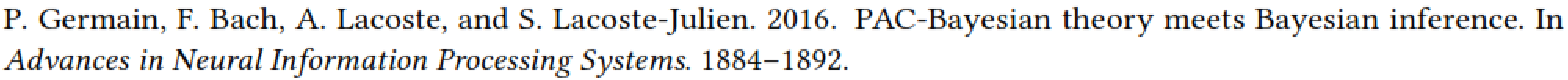
(a) Data.

(b) Model

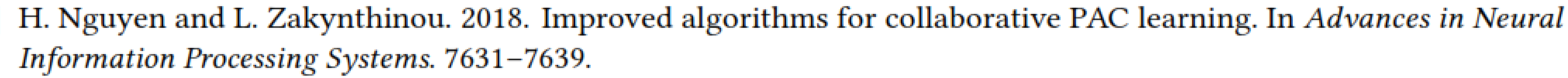
- These methods use prior knowledge to constrain the complexity of $\mathcal{H}$, which results in a much smaller hypothesis space $\widehat{\mathcal{H}}$
- As shown in Figure 2(b), the gray area is not considered for optimization as they are known to be unlikely to contain the optimal h^* according to prior knowledge. For this smaller $\mathcal{H}$, D_{train} is sufficient to learn a reliable h_I



(b) Model.



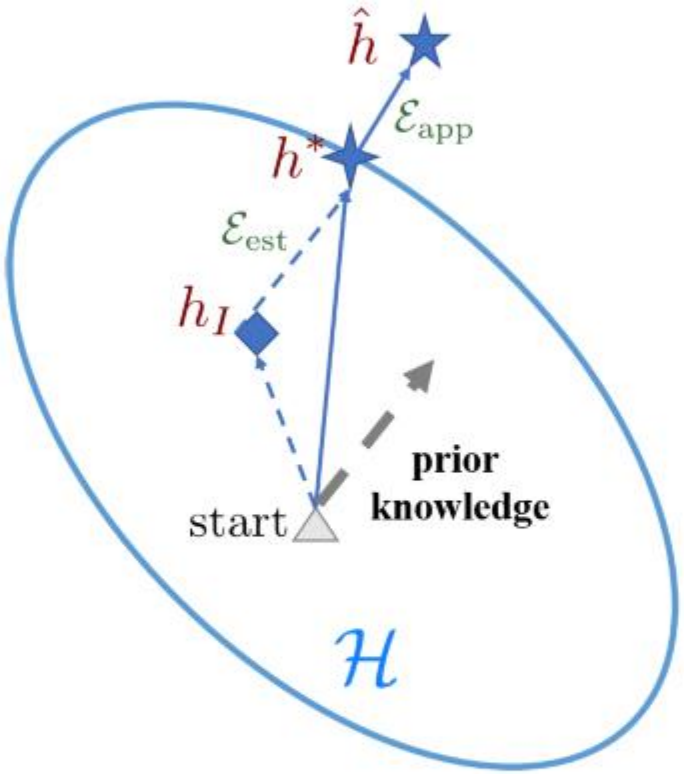
P. Germain, F. Bach, A. Lacoste, and S. Lacoste-Julien. 2016. PAC-Bayesian theory meets Bayesian inference. In *Advances in Neural Information Processing Systems*. 1884–1892.



H. Nguyen and L. Zakynthinou. 2018. Improved algorithms for collaborative PAC learning. In *Advances in Neural Information Processing Systems*. 7631–7639.

(c) Algorithm

- These methods use prior knowledge to search for the θ which parameterizes the best hypothesis h^* in $\mathcal{H}$
- Prior knowledge alters the search strategy by providing a good initialization △ , or guiding the search steps (the gray dotted line). For the latter, the resultant search steps are affected by both prior knowledge and empirical risk minimizer.



(c) Algorithm.