Universität St.Gallen

Meta-Learning | The MAML method

(1st Coding Lab – DL2025)

Why Meta-Learning?

- Humans can generalize from very few samples

Only one image is enough
for a kid to be able to
classify deers in the future!

- Is this the case for deep neural networks?
- Meta-learning: Methods that generalize from very few samples (e.g., 1 sample per class)
  - Model based (e.g., Hyper Networks)
  - Optimization based (e.g., MAML)
  - Metric based - aka Metric learning (e.g., Matching Nets)

Image classification meta-learning tasks

n-way, k-shot

Number of samples per class

Number of classes

Meta- training / testing phases

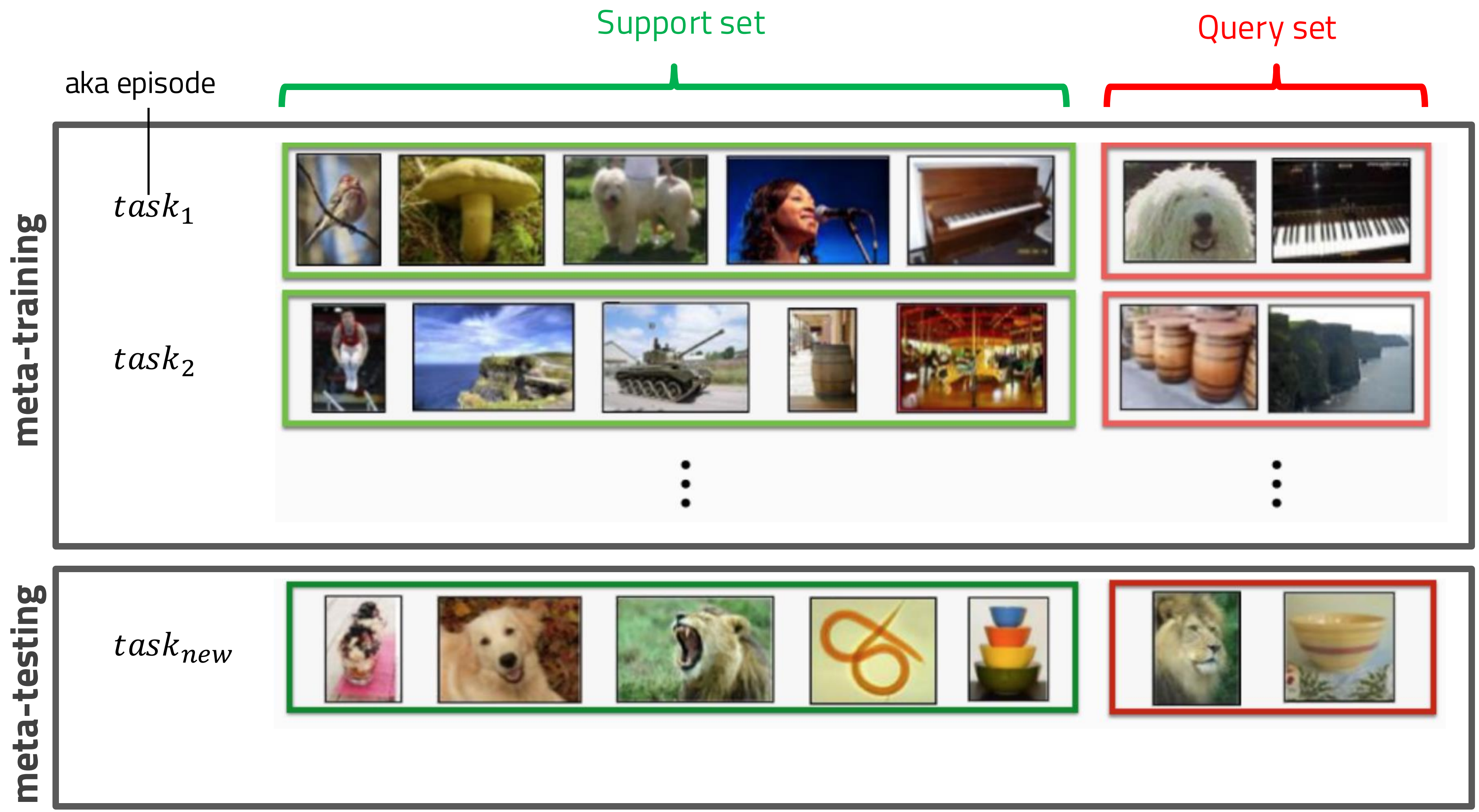
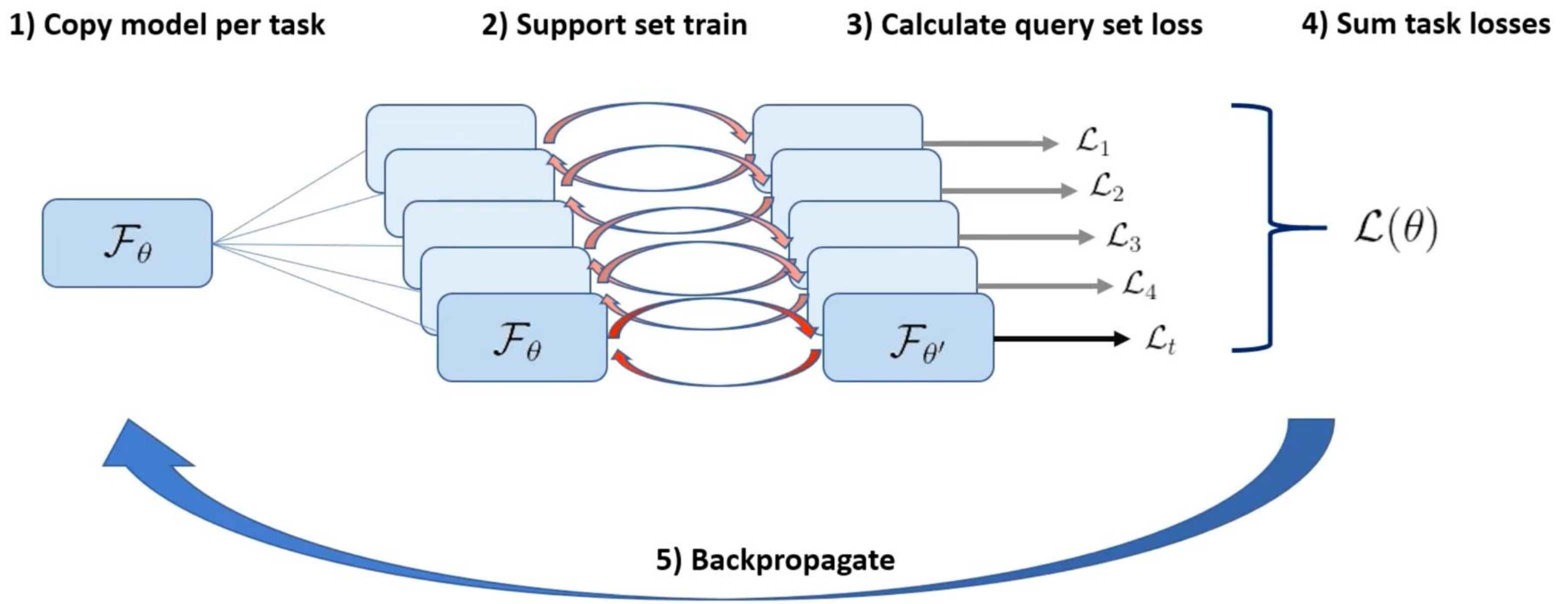
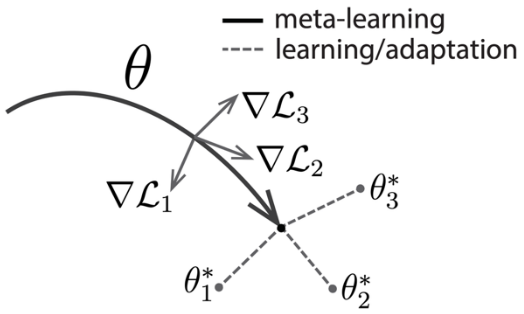


Image from: Deep Multi-Task and Meta Learning course (Fall 2020) – Stanford

MAML-Introduction

- MAML: **M**odel **A**gnostic **M**eta **L**earning
- An optimization based meta-learning method.





MAML-algorithm

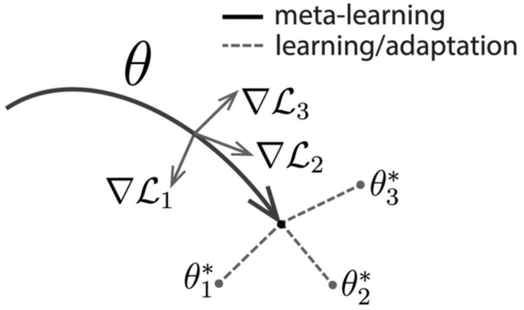
Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        Evaluate ∇_θ L_{T_i}(f_θ) with respect to K examples
        Compute adapted parameters with gradient descent: θ'_i = θ − α∇_θ L_{T_i}(f_θ)
    end for
    Update θ ← θ − β∇_θ Σ_{T_i~p(T)} L_{T_i}(f_{θ'_i})
end while
```

Inner task specific model updates

Outer meta-model updates



This gradient depends on the task specific model parameters:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

Which in turn depend on the per task gradient computation.

Computation of the Hessian vector products (2^{nd} derivative)

Coding Lab Contents

PyTorch Background:

1. Tensor Operations & Back Propagation
2. Stateful vs. Stateless Models

Meta-Learning Specific:

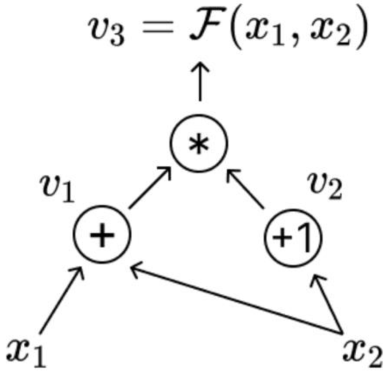
1. Task Family: Sine waves task distribution.
2. Meta-training with MAML.
3. Model Adaptation: Evaluating MAML against supervised pre-training.

Universität St.Gallen

Tensor Operations & Back Propagation

- What is a computational graph?

$$\mathcal{F}(x_1, x_2) = (x_1 + x_2)(x_2 + 1)$$



Forward Pass:

$$v_1 = x_1 + x_2$$
$$v_2 = x_2 + 1$$
$$v_3 = v_1 + v_2$$

Backward Pass:

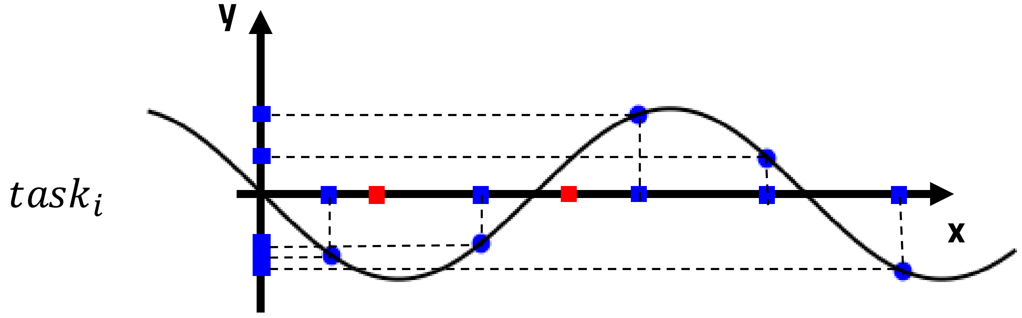
$$\frac{\partial \mathcal{F}}{\partial v_3} = \frac{\partial v_3}{\partial v_3} = 1$$
$$\frac{\partial \mathcal{F}}{\partial v_1} = \frac{\partial \mathcal{F}}{\partial v_3}\frac{\partial v_3}{\partial v_1} = v_2$$
$$\frac{\partial \mathcal{F}}{\partial v_2} = \frac{\partial \mathcal{F}}{\partial v_3}\frac{\partial v_3}{\partial v_2} = v_1$$
$$\frac{\partial \mathcal{F}}{\partial x_1} = \frac{\partial \mathcal{F}}{\partial v_1}\frac{\partial v_1}{\partial x_1} = v_2$$
$$\frac{\partial \mathcal{F}}{\partial x_2} = \frac{\partial \mathcal{F}}{\partial v_1}\frac{\partial v_1}{\partial x_2} + \frac{\partial \mathcal{F}}{\partial v_2}\frac{\partial v_2}{\partial x_2} = v_1 + v_2$$

Meta-learning regression setting

- Regression task
  - The model has to learn the mapping:
    F: X->Y
  - The Mean Squared Error (MSE) loss is used for supervision.

- Tasks distribution: y = amplitude * sin(phase + x)
  - A family of sines defined by: (amplitude_min, amplitude_max, phase_min, phase_max, x_min, x_max)

- Single task
  - Support set: The x and y (target) coordinates (in blue).
  - Query set: Only the x coordinates (in red).



5-shot setting

Supplementary material

- MAML paper [here]
- Blog on pytorch's *create_graph* option for 2nd order derivatives [here]
- Video tutorials:
  - Meta-learning (the general setting)
    - CS 182: Lecture 21: Part 1: Meta-Learning (UC Berkeley) [here]
  - MAML (intuitive description) [here]