Meta Learning

Learning by examples : supervised / unsupervised
Learning by experience : reinforcement
Learning by learning : meta learning

⇓

tasks of similar nature ⟶ similar properties

↓

come across enough diverse tasks of certain type ⟶ develop model that masters that type of task

↓

typical: train smaller ML models on specific tasks. feed output to meta-learning model ⟶ meta-learning happens through multiple training episodes where the model optimises the learning algo

↓

(hopefully) better generalisation to this task especially when less data is given

Episodic Learning

For each episode :

1. Sample datapoints from D
   - support set → train, val
   - query set → train, val
2. Train on support set : } meta train
   (a) train baseline ML model on train
   (b) train meta learning using val
3. Test on query set } meta-test

Eventually model learns how to learn.

FORMAL DEFINITION

In general :

$$\boxed{\min_{\omega} E_{T\sim p(T)}\, \mathcal{L}(D;\omega)}$$

$p(T)$: distribution of all tasks

Meta-training:

$$D_{source} = \left\{\left(D_{source}^{train}, D_{source}^{val}\right)^{(i)}\right\}_{i=1}^{S}$$

S source tasks

Meta-testing:

$$D_{target} = \left\{\left(D_{target}^{train}, D_{target}^{val}\right)^{(j)}\right\}_{j=1}^{G}$$

G target tasks

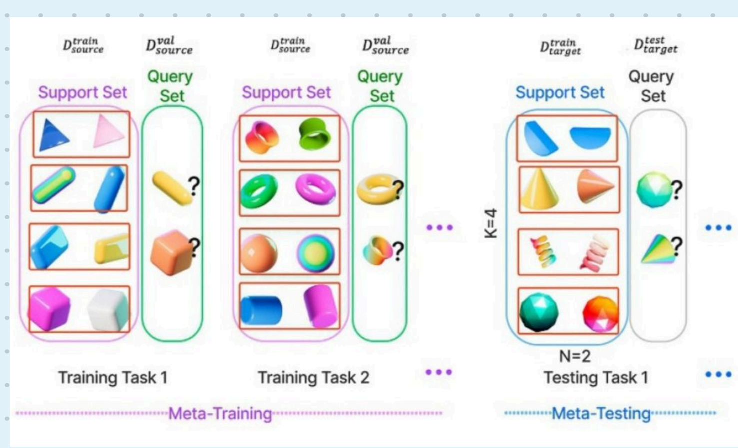
support sets   query sets

"k-way n-shot learning"
k-way → k classes; n-shot → n examples per class } for support set

"Bilevel optimisation"
inner loop → baseline ML model, train θ (model parameters)
outer loop → meta-learning model, train ω

$$\theta^{*(i)}(\omega) = \arg\min_{\theta} \mathcal{L}^{task}\left(\theta, \omega, D_{source}^{train\,(i)}\right)$$

$$\omega^{*} = \arg\min_{\omega} \sum_{i=1}^{S} \mathcal{L}^{meta}\left(\theta^{*(i)}(\omega), \omega, D_{source}^{val\,(i)}\right)$$



4-way 2-shot learning

Few-shot learning

meta-learning:
→ repeatedly see tasks in training with same structure (k-way n-shot), different classes
→ test set has unseen classes
→ has to learn how to discriminate data classes

regular few-shot:
→ model already trained on broad dataset
→ uses that knowledge to learn new categories
→ does not learn how to learn

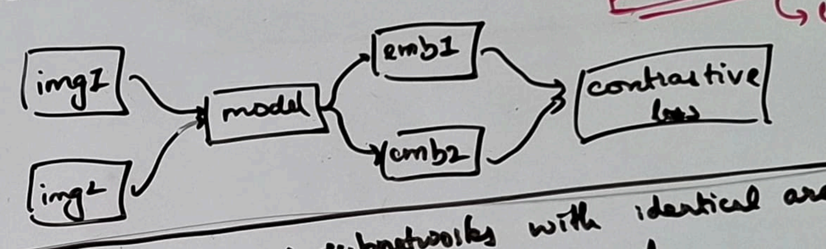
TYPES OF META-LEARNING

metric: map to metric space; same classes nearby and vice versa
model-based: constrain model arch. Eg: replay arch
optimisation based: same arch as normal, learn a new optimiser.

METRIC

1. Siamese networks

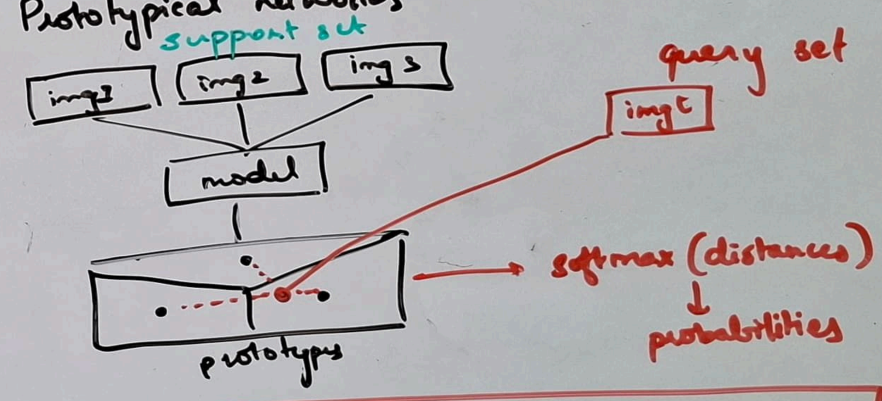
embeds all seen classes into diff. regions using pairwise comparisons ↳ loss



model = 2 or more subnetworks with identical arch. parameter updates mirrored

put img1 → get emb1 → put img2 → get emb2 → loss → backprop → optimise

2. Prototypical networks



prototype = mean of points under each class

k-way classification problem: compare query to set of other classes, loss based on classification

Optimisation: Model Agnostic Meta Learning

model just needs to be optimisable w/ gradient

finds best initial parameters θ to reduce training → finds best policy for RL

goal : $\min_{\theta} \left[E_{T\sim p(T)}\left(\mathcal{L}_T\left(U_T(\theta)\right)\right)\right]$

where U_T : optimiser that maps θ to ϕ, result of fine tuning θ on T

Steps

1. Pick some tasks T_i from $p(T)$
2. For each T_i, calc. $U_{T_i}(\theta)$: minimise $\mathcal{L}_{T_i,train}(\theta)$ for a few steps
3. Update θ by gradient descent to minimise $\mathcal{L}_{T_i,test}(\phi_i)$ on test

2. Easy $\phi_i = U_{T_i}(\theta)$: grad descent once

$$\boxed{\phi_i = \theta_i - \alpha \nabla_{\theta} \mathcal{L}_{T_i,train}(\theta)}$$

3a. calculate gradient wrt θ on the test using $U_T(\theta)$ loss

$$\boxed{\nabla_{\theta} \mathcal{L}(\theta) = \sum_i \nabla_{\theta} \mathcal{L}_{T_i,test}(\phi_i)}$$

3b. update θ

$$\boxed{\theta = \theta - \beta \nabla_{\theta} \mathcal{L}(\theta)}$$

NOTE: Even with only one gradient step for ϕ_i, we are calculating 2nd order derivative ⟶ computationally intensive

IMPROVEMENTS

1. CAML: Fast Context Adaptation
   - Split model parameters:
     θ → shared for all task; outer loop update
     ϕ_i → context for each task; inner loop update
   - Init $\phi_0 = 0$ for each task

$$\boxed{\phi_i = \phi_0 - \alpha \nabla_{\theta} \mathcal{L}_{T_i,train}(\phi_0, \theta)}$$

2. ADML : Adversarial
   - Train with clean & adversarial samples
   - Both used in inner and outer loops
   - Both contribute equally

$$\phi_{clean_i} = \theta - \alpha_1 \nabla_{\theta} \mathcal{L}_{clean_i,T_i}(\theta)$$
$$\phi_{adv_i} = \theta - \alpha_2 \nabla_{\theta} \mathcal{L}_{adv_i,T_i}(\theta)$$

$$\theta = \theta - \beta_1 \nabla_{\theta} \sum_{i=1}^{T} \mathcal{L}_{clean_i,T_i}(\phi_i)$$
$$\theta = \theta - \beta_2 \nabla_{\theta} \sum_{i=1}^{T} \mathcal{L}_{adv_i,T_i}(\phi_i)$$

3. Meta SGD

$$\phi = \theta - \alpha \nabla_{\theta} \mathcal{L}_{train_i,T_i}(\theta)$$

α → adaptation term

- Init α also as a random vector with size = size of θ
- Optimising θ lets us also learn update direction

4. Reptile
   - For each task, run SGD for some n iters.
   - Effectively calculates 2nd order derivative
   - Computationally efficient
   - Update model parameter in direction common to all individual tasks

$$\boxed{\theta = \theta + \varepsilon(\theta' - \theta)}$$

Simplification : FOMAML
just do first order in outer loop also
accuracy tradeoff; usually worth

Improvements
1. Meta-overfitting : memorises function to process meta-training data
2. Computationally intensive
3. Task heterogeneity
4. Lack of training task resources