Meta Learning: Learning to Learn

By: Chandresh Kumar Maurya,
CSE deptt, IIT Indore

Some Slides adapted from Chelsea Finn Talk on Meta learning Tutorial

Large, diverse data
(+ large models)

Russakovsky et al. ‘14

GPT-2

Radford et al. ‘19

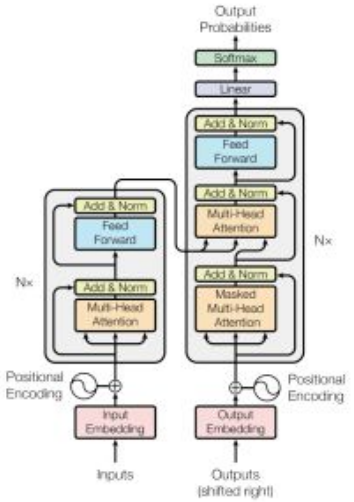


Figure 1: The Transformer - model architecture.

Vaswani et al. ‘18

Under the paradigm of supervised learning.

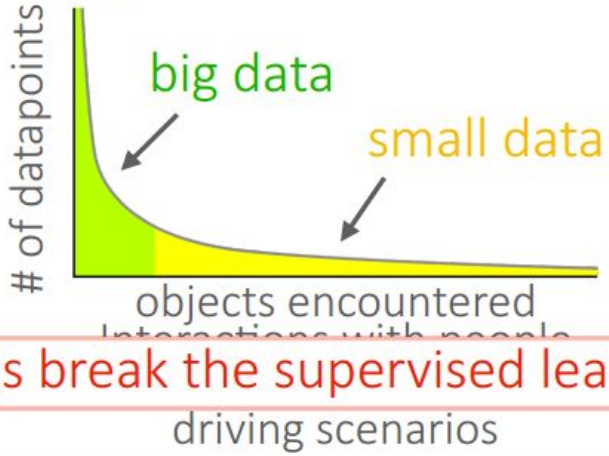
What if you don't have a large dataset?

medical imaging robotics personalized education,
translation for rare languages recommendations

What if you want a general-purpose AI system in the real world?

Need to continuously adapt and learn on the job.
Learning each thing from scratch won't cut it.

What if your data has a long tail?



These settings break the supervised learning paradigm.

Qs: slido.com/meta

Two ways to view meta-learning

Mechanistic view

- Deep neural network model that can read in an entire dataset and make predictions for new datapoints
- Training this network uses a meta-dataset, which itself consists of many datasets, each for a different task
- This view makes it easier to implement meta-learning algorithms

Probabilistic view

- Extract prior information from a set of (meta-training) tasks that allows efficient learning of new tasks
- Learning a new task uses this prior and (small) training set to infer most likely posterior parameters
- This view makes it easier to understand meta-learning algorithms

Problems with Current AI models

- They are data hungry
- Everytime you want to deploy model M trained on task A to for solving task B which is closely related to task A, you suffer with:
  a. Low performance
  b. Have to retrain from scratch
- Is this really a true AI model?

Problems with Current AI models

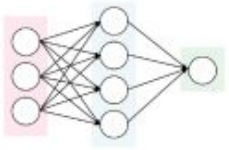
- They are data hungry
- Everytime you want to deploy model M trained on task A to for solving task B which is closely related to task A, you suffer with:
  a. Low performance
  b. Have to retrain from scratch
- Is this really a true AI model?
- No.
- How do as a human learn?
- We generalize our learning from one task to multiple tasks.
- Current learning algorithm master only one task.
- This is where meta learning comes in.

What is Meta Learning

- Meta learning produces a versatile AI model that can learn to perform various tasks without having to train them from scratch.
- We train our 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)

Problem definitions

supervised learning:



$$\arg\max_{\phi} \log p(\phi|\mathcal{D})$$

model parameters

training data

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

input (e.g., image)

label

$$= \arg\max_{\phi} \log p(\mathcal{D}|\phi) + \log p(\phi)$$

data likelihood

regularizer (e.g., weight decay)

$$= \arg\max_{\phi} \sum_i \log p(y_i|x_i, \phi) + \log p(\phi)$$

What is wrong with this?

- The most powerful models typically require large amounts of labeled data
- Labeled data for some tasks may be very limited

Problem definitions

supervised learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}) \qquad \mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

can we incorporate *additional* data?

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

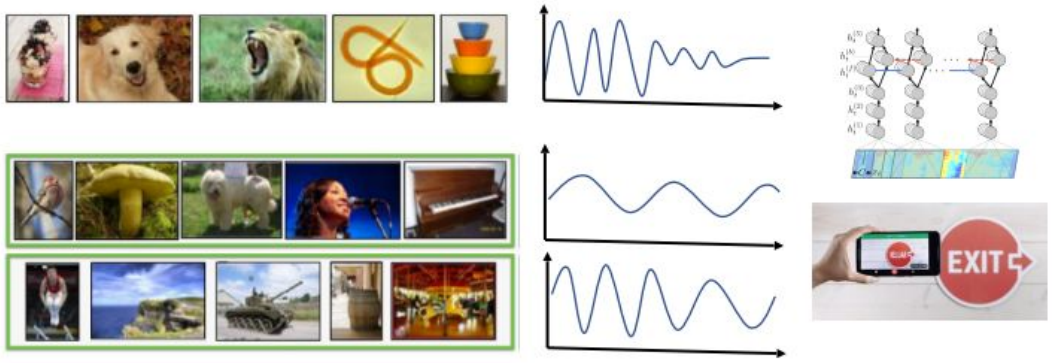
$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \qquad \mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$\mathcal{D}_{\text{meta-train}}$

$\mathcal{D}$

$\mathcal{D}_1$

$\mathcal{D}_2$



Qs: slido.com/meta

Image adapted from Ravi & Larochelle

The meta-learning problem

meta-learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

what if we don't want to keep $\mathcal{D}_{\text{meta-train}}$ around forever?

learn *meta-parameters* θ: $p(\theta|\mathcal{D}_{\text{meta-train}})$

whatever we need to know about $\mathcal{D}_{\text{meta-train}}$ to solve new tasks

this is the meta-learning problem

assume $\phi \perp\!\!\!\perp \mathcal{D}_{\text{meta-train}}|\theta$

$$\log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) = \log \int_{\Theta} p(\phi|\mathcal{D}, \theta) p(\theta|\mathcal{D}_{\text{meta-train}}) d\theta$$

$$\approx \log p(\phi|\mathcal{D}, \theta^\star) + \log p(\theta^\star|\mathcal{D}_{\text{meta-train}})$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \approx \arg\max_{\phi} \log p(\phi|\mathcal{D}, \theta^\star)$$

$$\theta^\star = \arg\max_{\theta} \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

Qs: slido.com/meta

A Quick Example

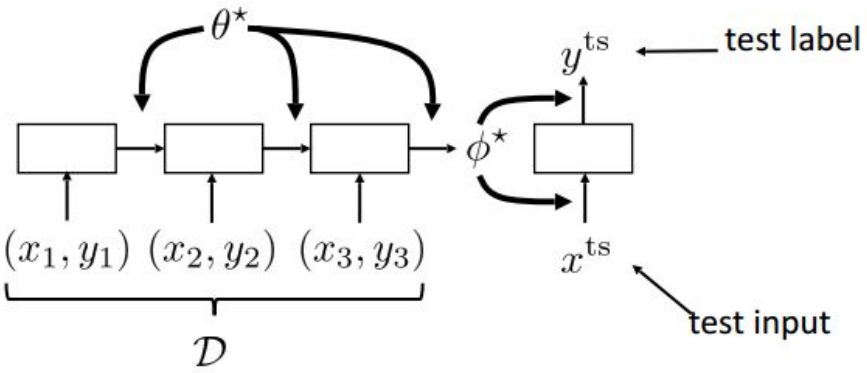
$$\text{meta-learning: } \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

$$\text{adaptation: } \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$



$\mathcal{D}$

$\mathcal{D}_{\text{meta-train}}$ $\mathcal{D}_1$

$\mathcal{D}_2$

How do we train this thing?

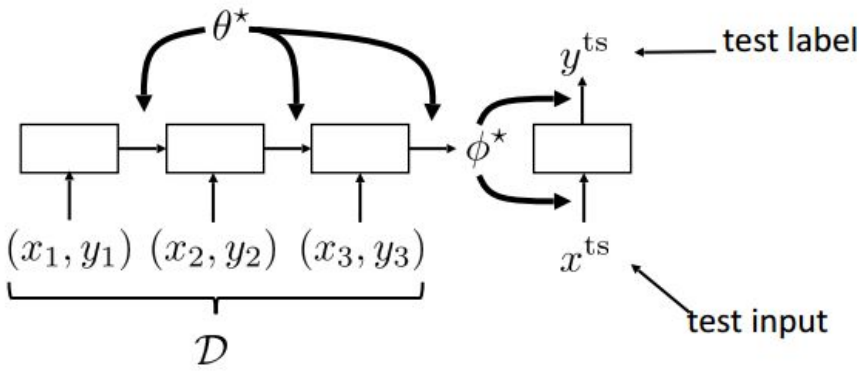
meta-learning: $\theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$

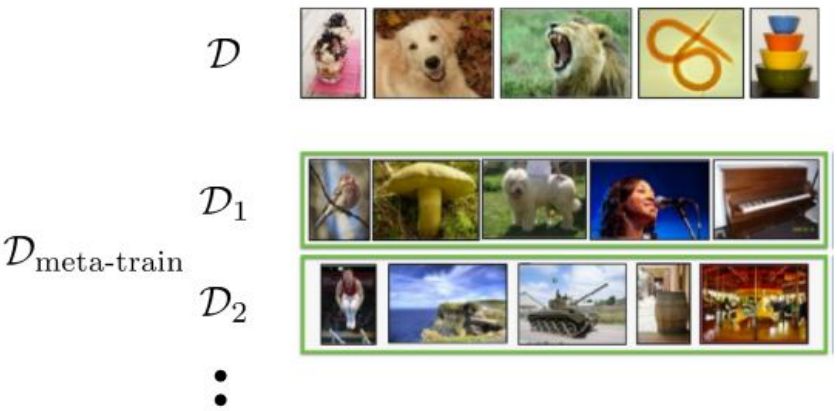
adaptation: $\phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)$

$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$

$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$

$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$





Key idea:

"our training procedure is based on a simple machine learning principle: test and train conditions must match"

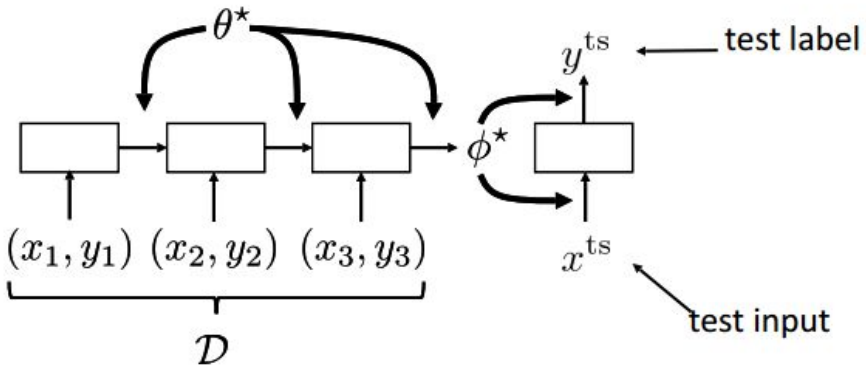
Vinyals et al., Matching Networks for One-Shot Learning

How do we train this thing?

$$\text{meta-learning: } \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

$$\text{adaptation: } \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)$$

(meta) test-time

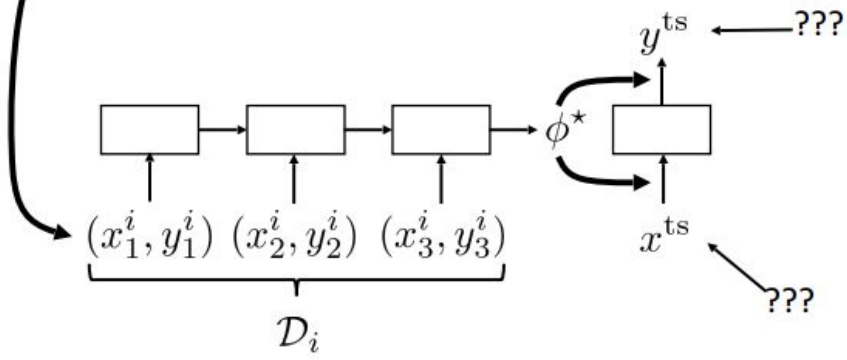


$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

(meta) training-time



Key idea:

"our training procedure is based on a simple machine learning principle: test and train conditions must match"

Vinyals et al., Matching Networks for One-Shot Learning

Reserve a test set for each task!

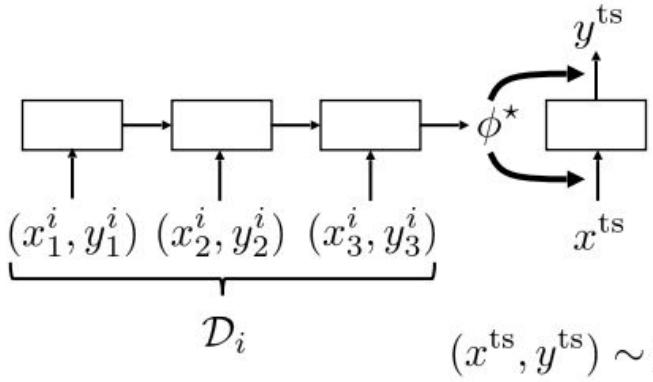
$\mathcal{D}_{\text{meta-train}}$ $\mathcal{D}_1$ $\mathcal{D}_2$ $\vdots$

(meta) training-time

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$$



Key idea:

"our training procedure is based on a simple machine learning principle: test and train conditions must match"

Vinyals et al., Matching Networks for One-Shot Learning

The complete meta-learning optimization

$$\text{meta-learning: } \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

$$\text{adaptation: } \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}^{\text{tr}}, \theta^\star)$$

$$\Updownarrow$$

$$\phi^\star = f_{\theta^\star}(\mathcal{D}^{\text{tr}})$$

learn θ such that $\phi = f_\theta(\mathcal{D}_i^{\text{tr}})$ is good for $\mathcal{D}_i^{\text{ts}}$

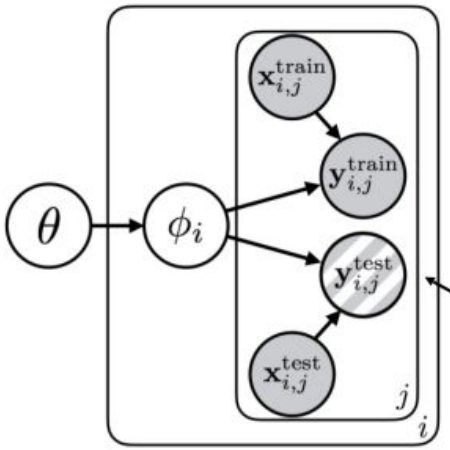
$$\theta^\star = \max_\theta \sum_{i=1}^{n} \log p(\phi_i|\mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$$



unobserved at meta-test time

Some meta-learning terminology

learn θ such that $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$ is good for $\mathcal{D}_i^{\text{ts}}$

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} \log p(\phi_i | \mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{T}_i \begin{cases} \mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\} \\ \mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\} \end{cases}$$

shot
(i.e., k-shot, 5-shot)

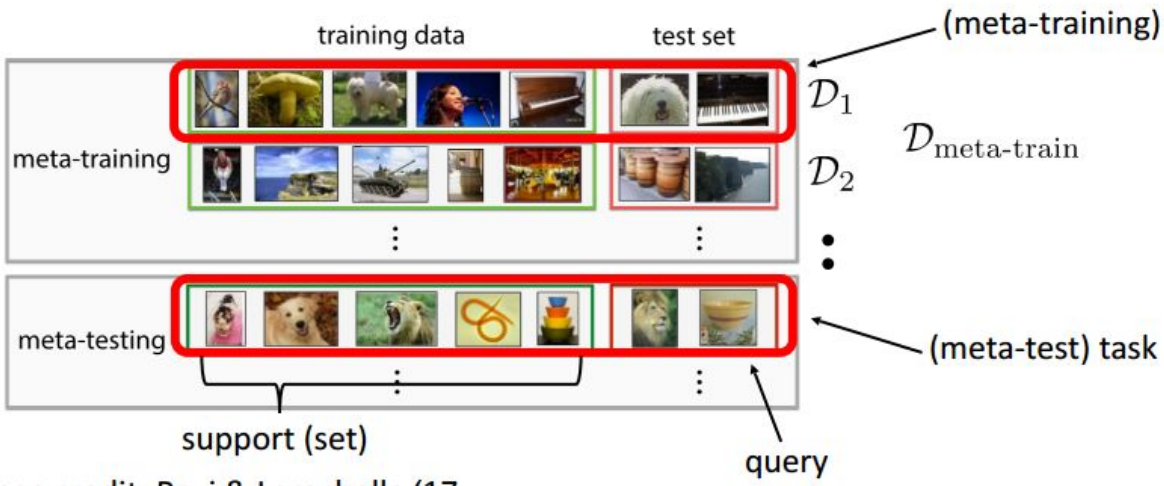


image credit: Ravi & Larochelle '17

Closely related problem settings

meta-learning:

$$\theta^\star = \max_\theta \sum_{i=1}^{n} \log p(\phi_i | \mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$$

multi-task learning: learn model with parameters $\theta^\star$ that solves multiple tasks $\theta^\star = \arg\max_\theta \sum_{i=1}^{n} \log p(\theta | \mathcal{D}_i)$

can be seen as special case where $\phi_i = \theta$ (i.e., $f_\theta(\mathcal{D}_i) = \theta$)

hyperparameter optimization & auto-ML: can be cast as meta-learning

hyperparameter optimization: θ = hyperparameters, ϕ = network weights

architecture search: θ = architecture, ϕ = network weights

very active area of research! but outside the scope of this tutorial

Qs: slido.com/meta

Closely Related Problem Settings

Transfer Learning

- **Multi-task learning** in which we learn multiple tasks T1,T2,..,Tn together
- **Fine-tuning** in which we take a pretrained model and tune to work in target domain. E.g. Imagenet fine tuning on medical data
- **Domain adaptation** similar to fine tuning except that only domain changes, not the labels

Meta learning

A form of lifelong learning in which we learn a model on multiple related tasks and test on a different but related task where we have very less data. Recently, meta-learning tends focus on finding "model agnostic" solutions where as multi-task learning remains deeply tied to model architecture.

Some More Terminology

- **K-shot learning** : use k examples to learn the task
- Special cases:
  - Zero-shot learning: k=0
  - One-shot learning: k=1
  - Few-shot learning: aka k-shot learning
- How can we learn when there are no data points at all?
- In this case, we will not have data points, but we will have meta information about each of the classes and we will learn from the meta information.

Types of meta learning

- Learning the metric space
- Learning the initializations
- Learning the optimizer

Learning the metric space

- 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.

Learning the Initializations

- learn optimal initial parameter values.
- Let's say we are a building a neural network to classify images.
- First, we initialize random weights, calculate loss, and minimize the loss through a gradient descent.
- Instead of initializing the weights randomly, if can 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.
- Algorithms such as MAML, Reptile, and Meta-SGD.

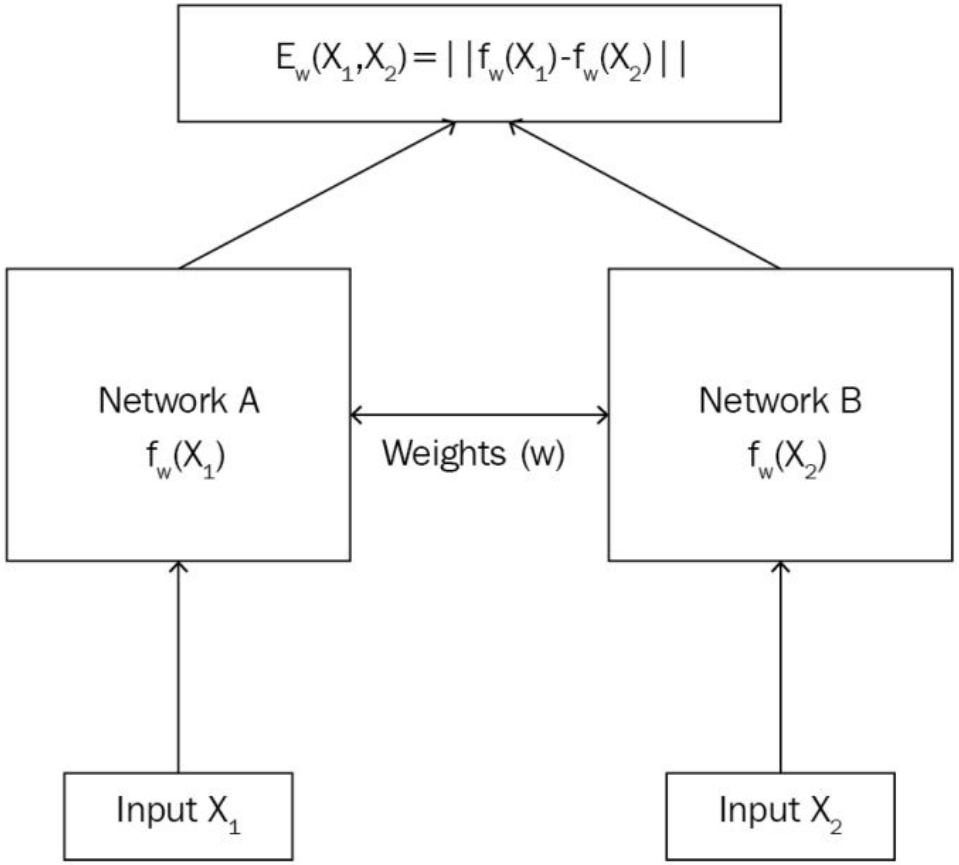
Learning the optimizer

- Try to learn the optimizer.
- How do we generally optimize our neural network?
- We optimize our neural network 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.
- Have two networks: a base network that actually tries to learn and a meta network that optimizes the base network.

Learning the Metric Space: Siamese Network: One-shot learning of faces

- Siamese network is type of NN where two symmetrical networks sharing the same weights and architecture are joined together at the end using some energy function.
- The objective of our siamese network is to learn whether two input values are similar or dissimilar.
- Let's say we have two images, X1 and X2, and we want to learn whether the two images are similar or dissimilar

Siamese Network: few-shot learning of faces



See 02 face and audio recognition folder for code.

Learning the Initializers: Modal Agnostic Meta Learning (MAML)

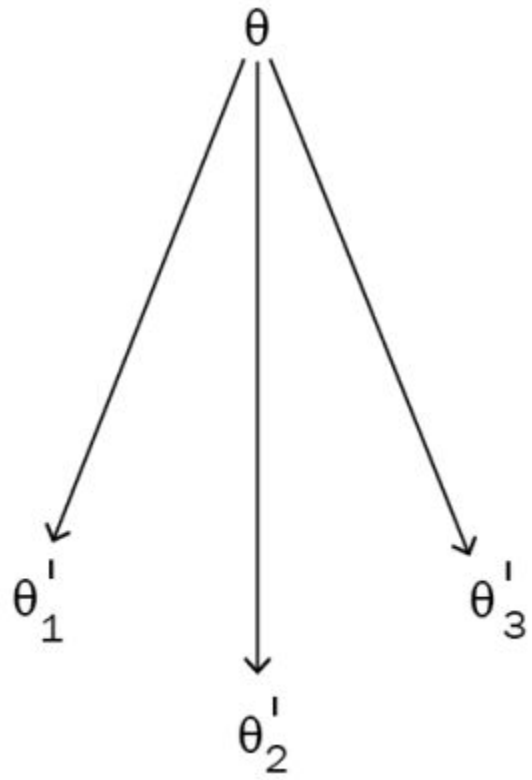
- MAML is one of the recently introduced and most popularly used meta learning algorithms.
- Major breakthrough in meta learning research.
- Learning to learn is the key focus of meta learning
- In meta learning, we learn from various related tasks containing only a small number of data points and the meta learner produces a quick learner that can generalize well on a new related task even with a lesser number of training samples.

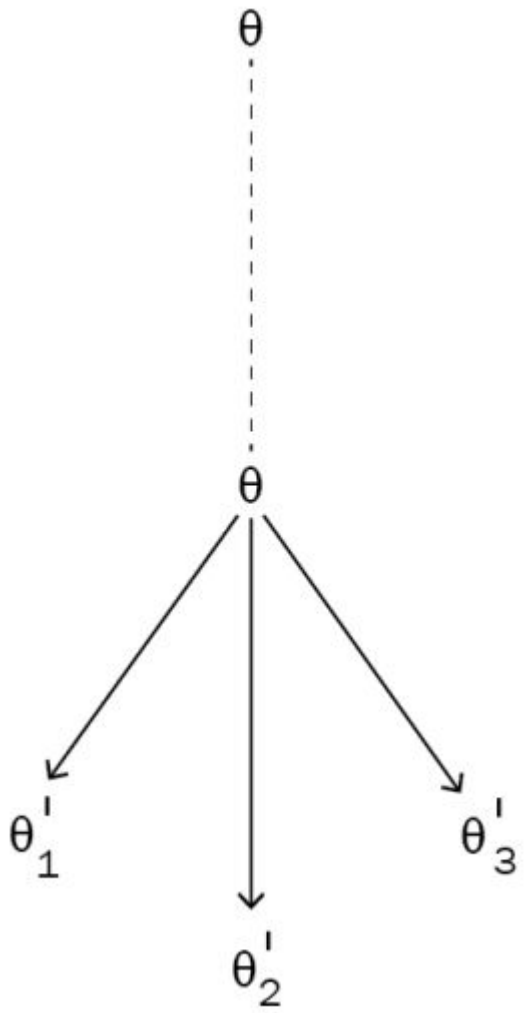
MAML

- The basic idea of MAML is to find a better initial parameter so that, with good initial parameters, the model can learn quickly on new tasks with fewer gradient steps.
- In NN, we learn by optimizing the loss using gradient descent.
- Gradient descent gives us the optimal weights.
- In MAML, we try to find these optimal weights by learning from the distribution of similar tasks.
- So, for a new task, we don't have to start with randomly initialized weights—instead, we can start with optimal weights, which will take fewer gradient steps to reach convergence and it doesn't require more data points for training.

Intuition

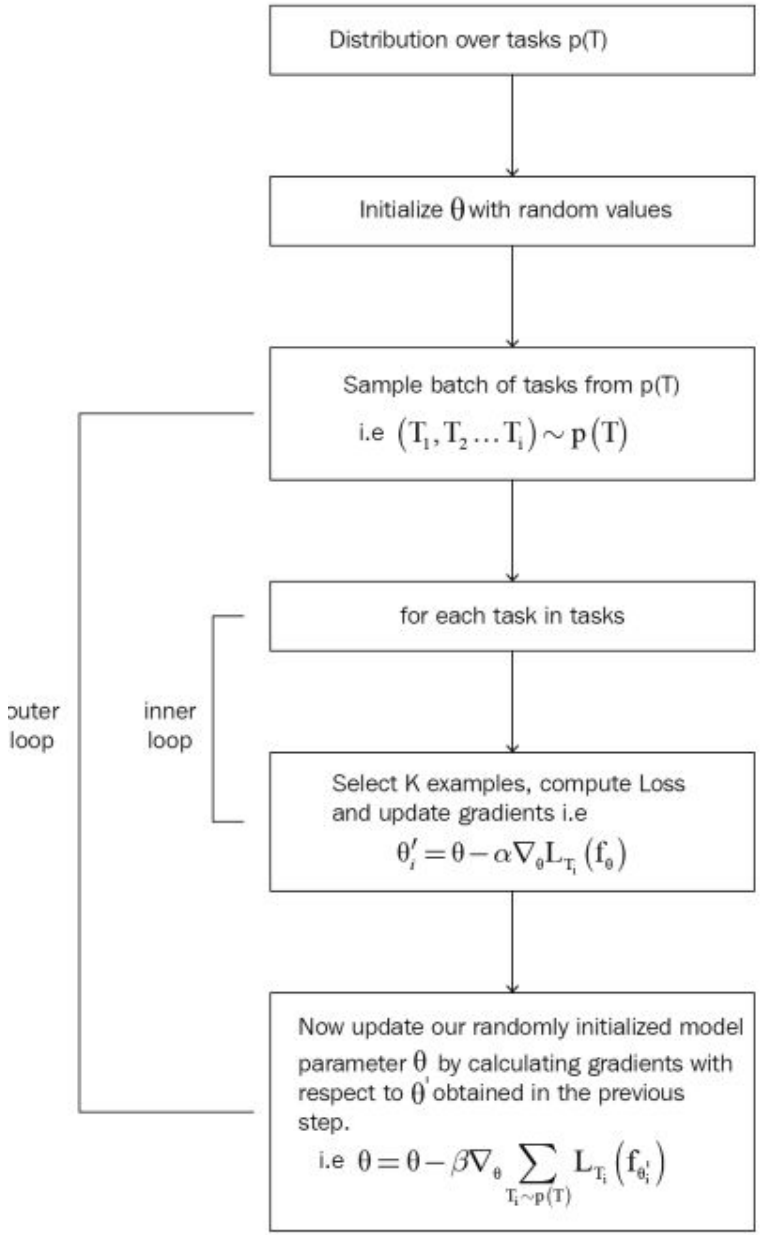
- We have three related tasks: T1, T2, and T3
- First, we randomly initialize our model parameter, θ. We train our network on task T1.
- Then, we try to minimize the loss L by gradient descent. We minimize the loss by finding the optimal parameter θ1*.
- Similarly, for tasks T2 and T3, we will start off with a randomly initialized model parameter, θ, and minimize the loss by finding the right set of parameters by gradient descent.
- Let's say θ2*, θ3* are the optimal parameters for the tasks, T2 and T3, respectively.


θ
θ'1
θ'2
θ'3


θ
θ
θ'1
θ'2
θ'3

- So, for a new related task, say, T4, we don't have to start with a randomly initialized parameter, θ. Instead, we can start with the optimal θ value so that it will take fewer gradient steps to attain convergence.
- MAML is **model agnostic**, meaning that we can apply MAML to any models that are trainable with gradient descent. But how exactly does MAML work?
- How do we shift the model parameters to an optimal position?

MAML in Detail



See the demo of MAML in python notebook

Ideas on Improving MAML?

Task-Agnostic Meta Learning (TAML)

- In meta learning, we sample a batch of tasks and from each batch sample a data points and train the model on it.
- Do you see any problem in this?

Task-Agnostic Meta Learning (TAML)

- In meta learning, we sample a batch of tasks and from each batch sample a data points and train the model on it.
- Do you see any problem in this?
- Model could be biased towards task sampled more often than others.
- Need a model which generalizes over unseen tasks which vary widely with the training tasks
- Need our model task-agnostic.

TAML

- Entropy maximization/reduction
- Inequality minimization

Entropy Maximization/Minimization

- Goal: Learn task-agnostic models
- We know that entropy is a measure of randomness. So, we maximize entropy by allowing the model to make a random guess over the predicted labels with equal probability.
- By making random guesses over the predicted label, we can prevent task bias.

Entropy Calculation

- How do we compute the entropy?
- Let's denote entropy by H. The entropy for Ti computed by sampling from xi over its output probabilities from PTi(xi), over predicted labels:

$$H_{T_i}(f_\theta) = -\mathbb{E}_{x_i \sim P_{T_i}(x)} \sum_{n=1}^{N} \hat{y}_i, nlog(\hat{y}_i, n)$$

Contd..

- Maximize the entropy before updating the model parameter.
- Next, we minimize the entropy after updating the model parameter.
- So, what do we mean by minimizing the entropy?
- Minimizing the entropy implies that we don't add any randomness over the predicted labels and we allow the model to predict the label with high confidence.
- So, our goal is to maximize the entropy reduction for each of the tasks and it can be represented as follows:

$$H_{T_i}(f_\theta) - H_{T_i}(f_{\theta_i})$$

Contd..

- Incorporate our entropy term with the meta objective and try to find the optimal parameter , so our meta objective becomes the following:

$$\theta = \theta - \beta \nabla_\theta \{ \mathbb{E}_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'}) + \lambda [-H_{T_i}(f_\theta) + H_{T_i}(f_{\theta_i})] \}$$

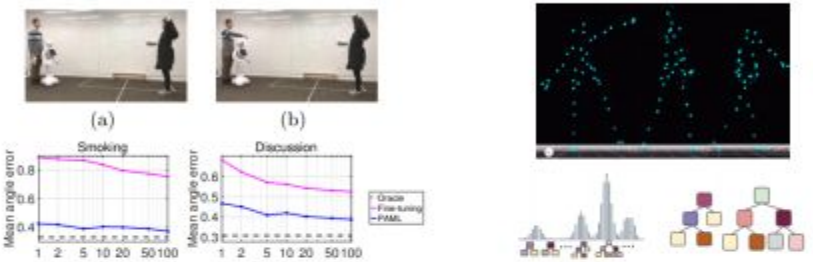
Applications of Meta learning

Applications in computer vision

few-shot image recognition

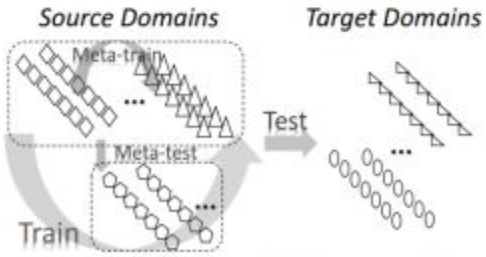
see, e.g.: Vinyals et al. **Matching Networks for One Shot Learning**, and many many others

human motion and pose prediction



see, e.g.: Gui et al. **Few-Shot Human Motion Prediction via Meta-Learning.**
Alet et al. **Modular Meta-Learning.**

domain adaptation



see, e.g.: Li, Yang, Song, Hospedales. **Learning to Generalize: Meta-Learning for Domain Adaptation.**

few-shot segmentation

see, e.g.: Shaban, Bansal, Liu, Essa, Boots. **One-Shot Learning for Semantic Segmentation.**
Rakelly, Shelhamer, Darrell, Efros, Levine. **Few-Shot Segmentation Propagation with Guided Networks.**
Dong, Xing. **Few-Shot Semantic Segmentation with Prototype Learning.**

Qs:

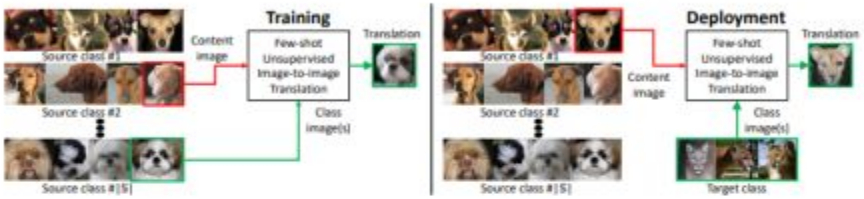
Applications of Meta learning

Applications in image & video generation

few-shot image generation

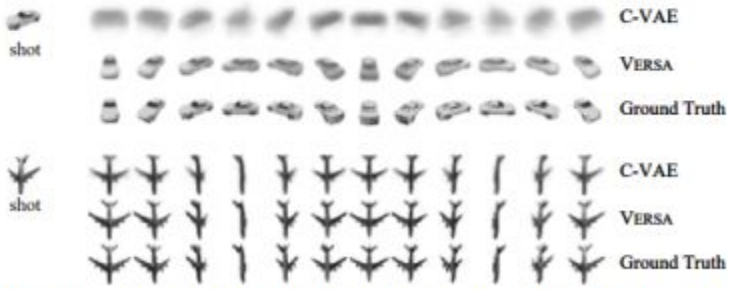
see, e.g.: Reed, Chen, Paine, van den Oord, Eslami, Rezende, Vinyals, de Freitas. **Few-Shot Autoregressive Density Estimation**. and many many others.

few-shot image-to-image translation



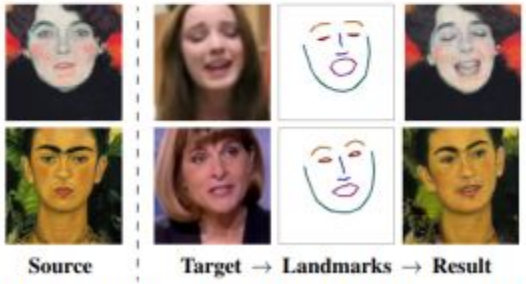
see, e.g.: Liu, Huang, Mallya, Karras, Aila, Lehtinen, Kautz. **Few-Shot Unsupervised Image-to-Image Translation.**

generation of novel viewpoints



see, e.g.: Gordon, Bronskill, Bauer, Nowozin, Turner. **VERSA: Versatile and Efficient Few-Shot Learning.**

generating talking heads from images



see, e.g.: Zakharov, Shysheya, Burkov, Lempitsky. **Few-Shot Adversarial Learning of Realistic Neural Talking Head Models**

Qs: s[illegible]