Meta Learning (Part 1)

Hung-yi Lee

Introduction

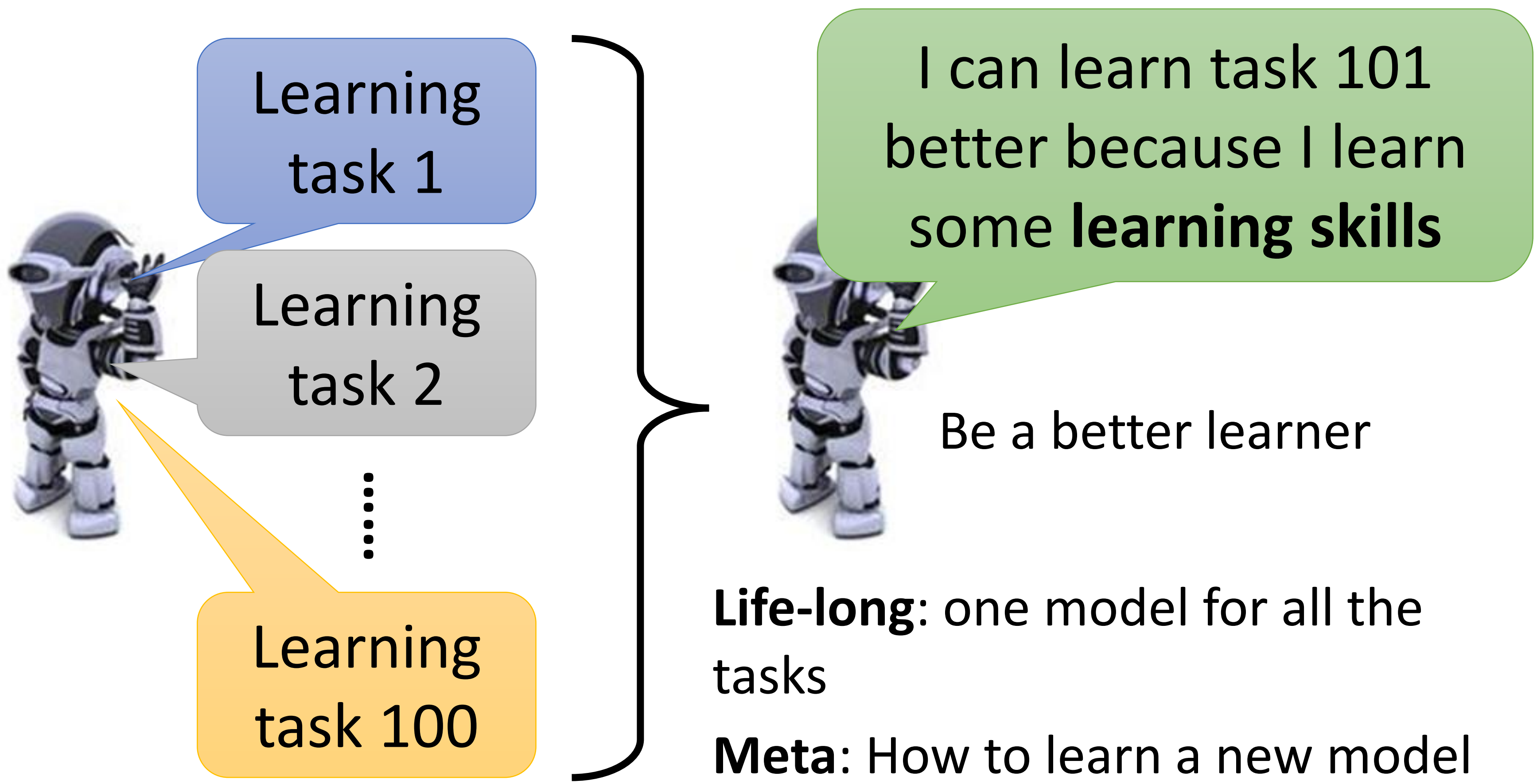
Task 1: speech recognition

Task 2: image recognition

⋮

Task 100: text classification

- Meta learning = Learn to learn



Life-long: one model for all the tasks

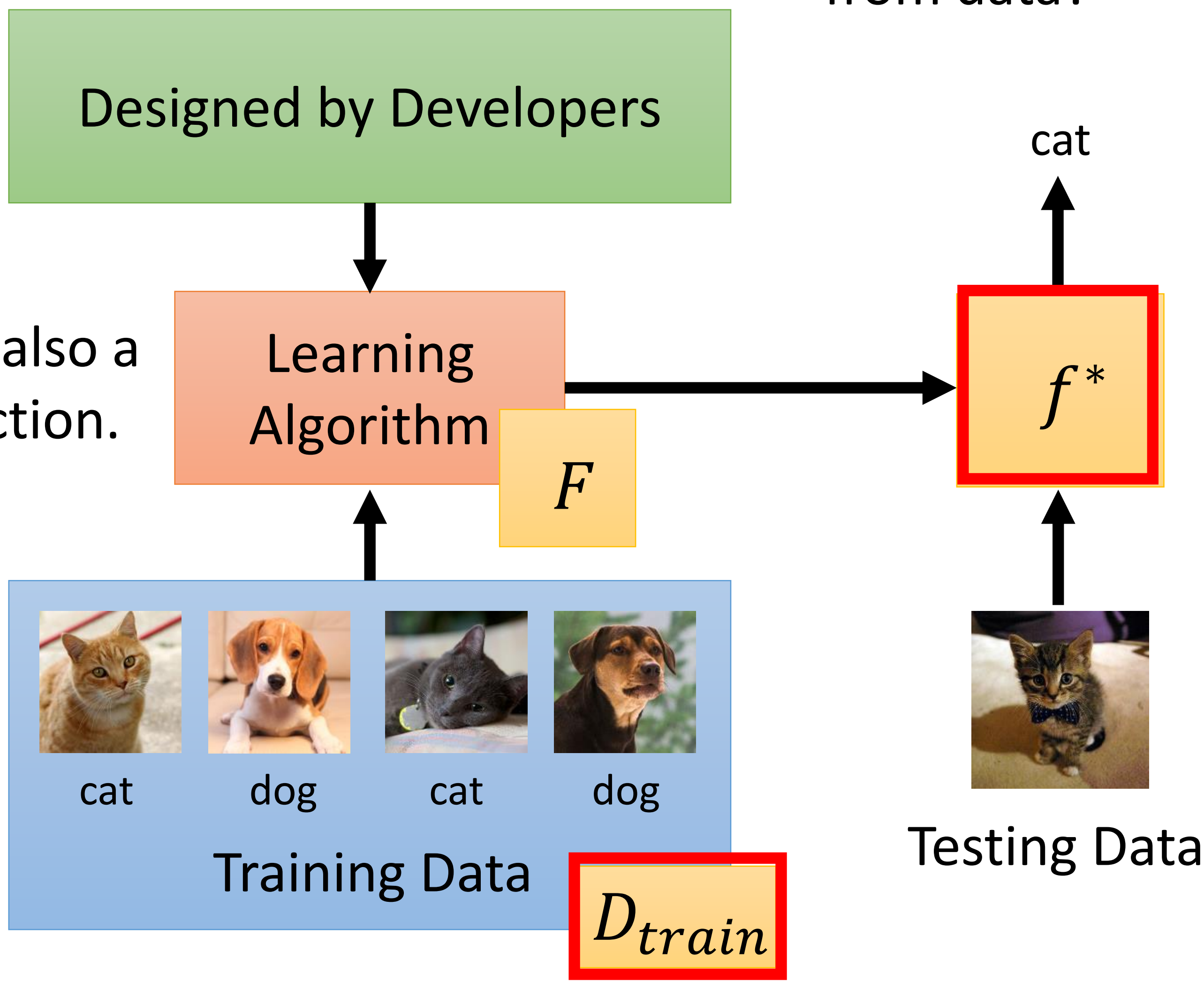
Meta: How to learn a new model

Meta Learning

$$f^* = F(D_{train})$$

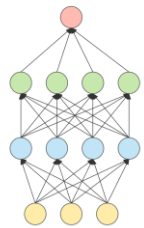
Can machine find F from data?

It is also a function.



Meta Learning

Machine Learning ≈ 根據資料找一個函數 f 的能力



$f($  $) =$ “Cat”

Meta Learning

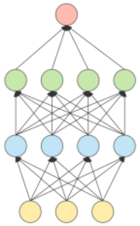
≈ 根據資料找一個找一個函數 f 的函數 F 的能力

$F($

cat dog cat dog

Training Data

$) = f^*$



~~Machine~~ Learning is Simple

Meta

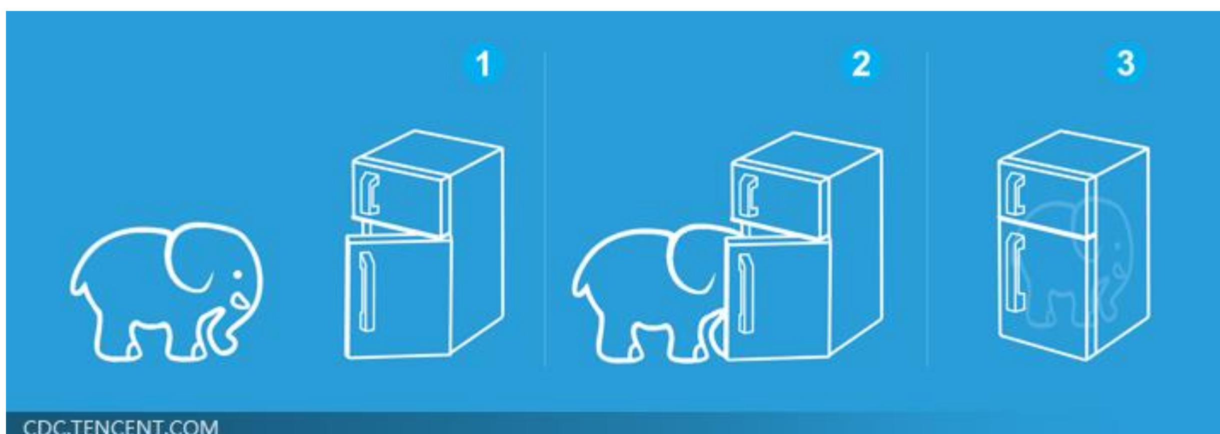
Step 1: define a set of ~~function~~

Step 2: goodness of ~~function~~

Step 3: pick the best ~~function~~

Function f ➡ Learning algorithm F

就好像把大象放進冰箱 ......

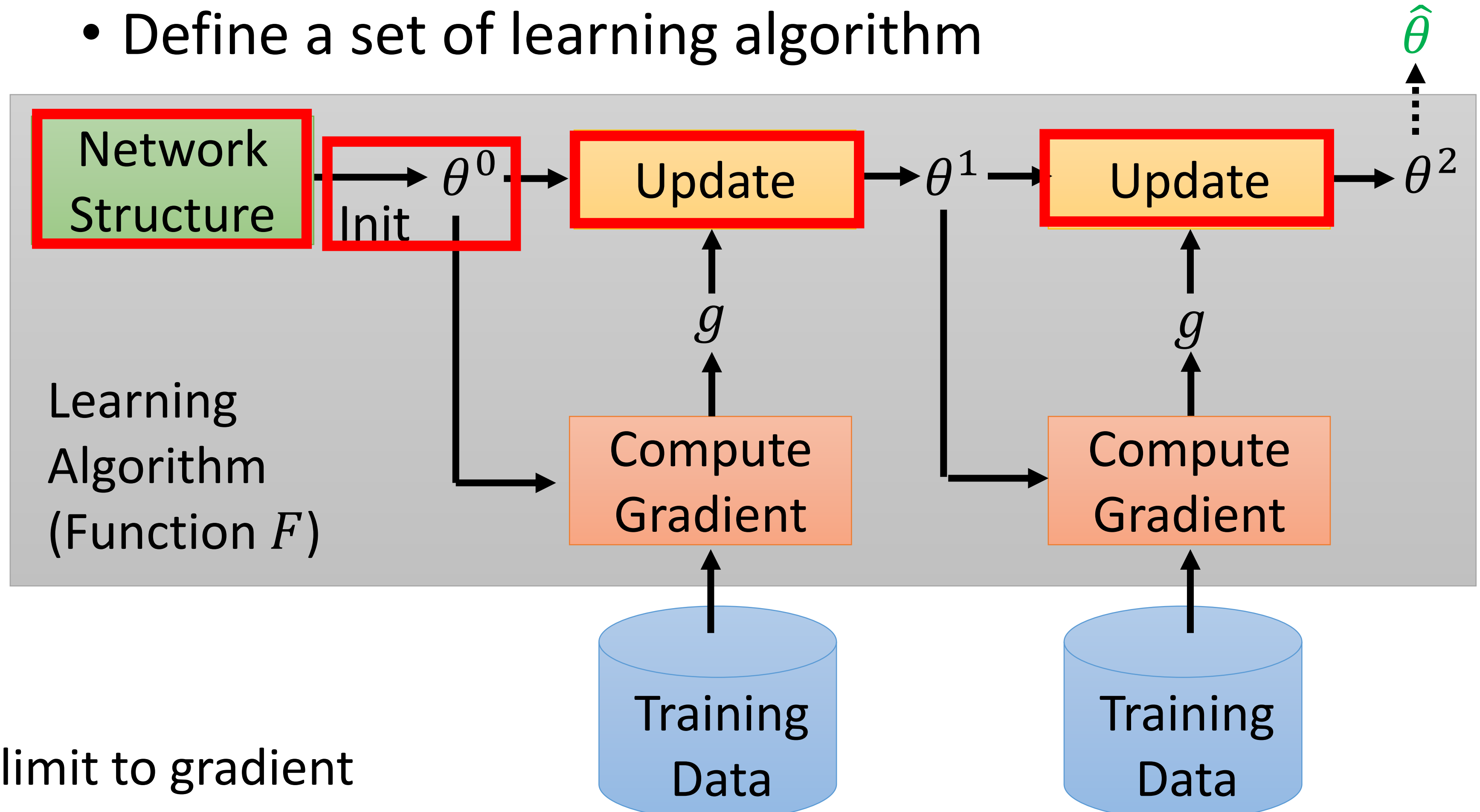


Meta Learning

Different decisions in the red boxes lead to different algorithms.

What happens in the red boxes is decided by humans until now.

- Define a set of learning algorithm



(limit to gradient descent based approach)

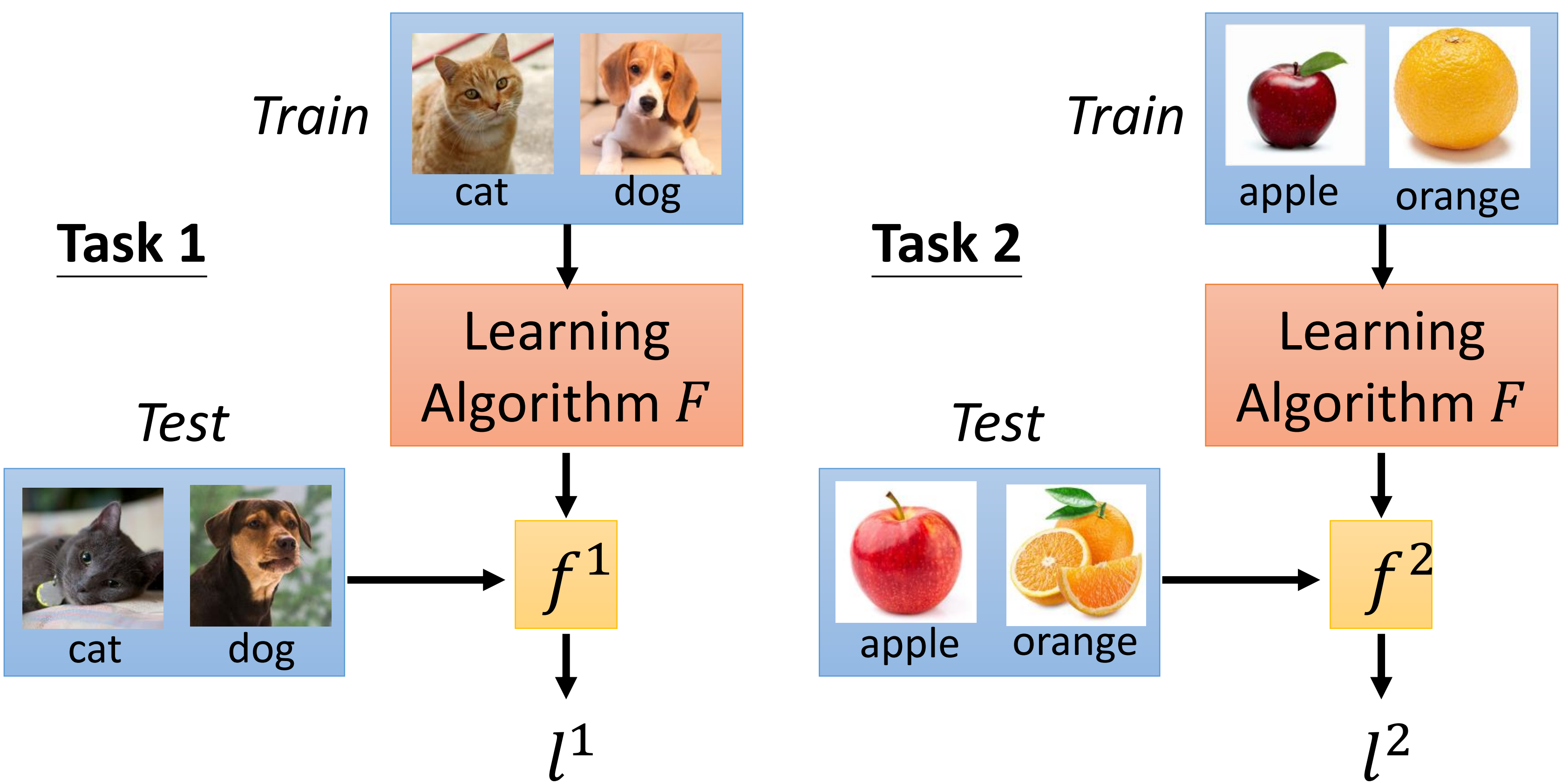
Meta Learning

$$L(F) = \sum_{n=1}^{N} l^n$$

N tasks

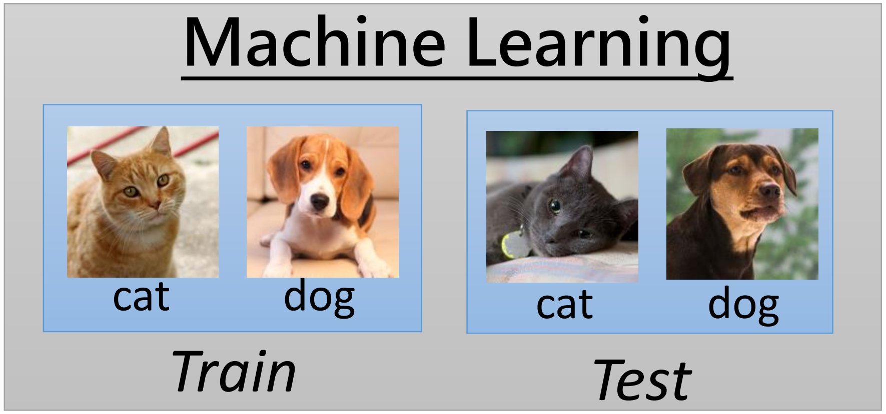
Testing loss for task n after training

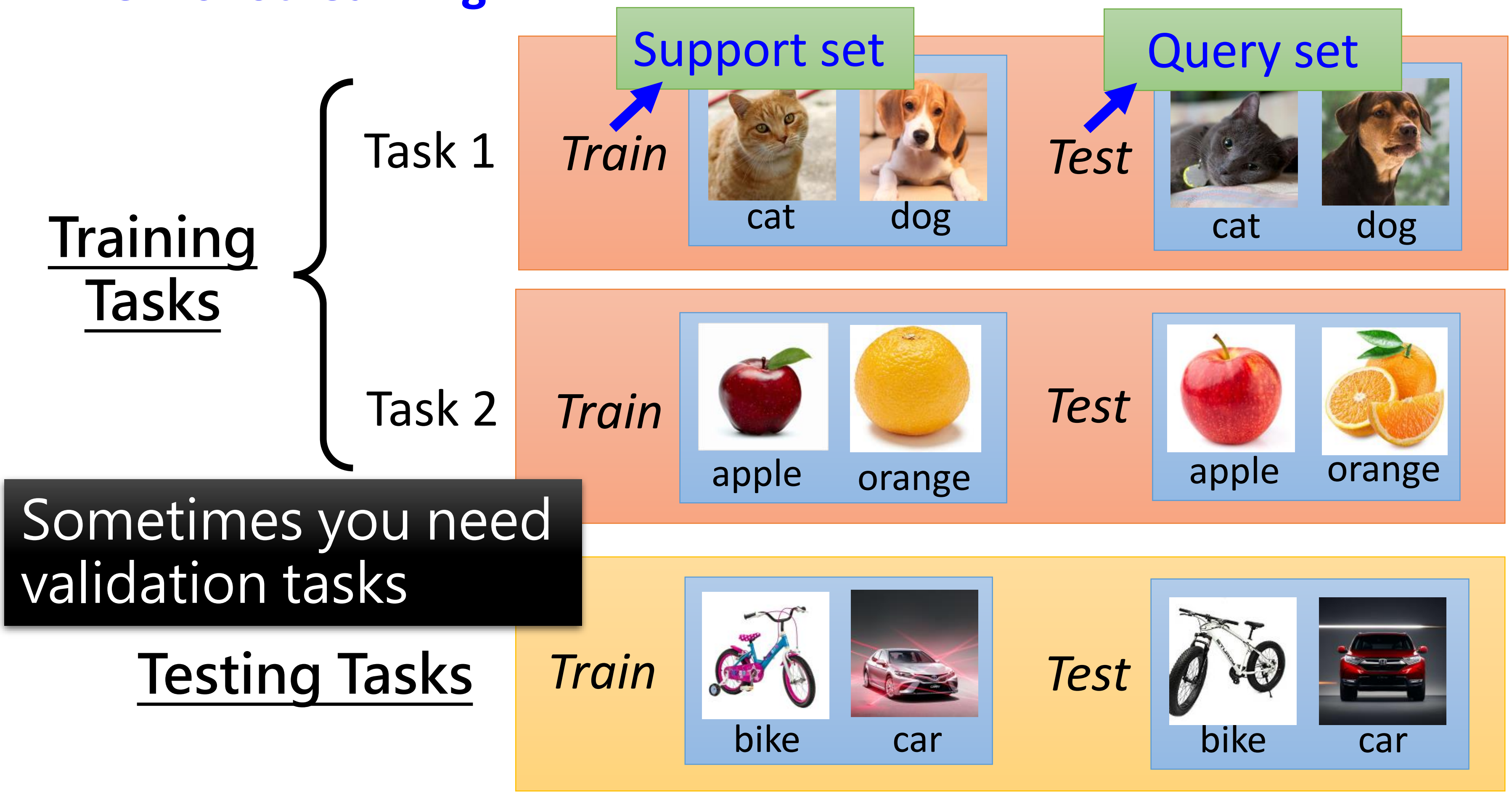
- Defining the goodness of a function F



Meta Learning

Widely considered in **few-shot learning**





Meta Learning

- Defining the goodness of a function F

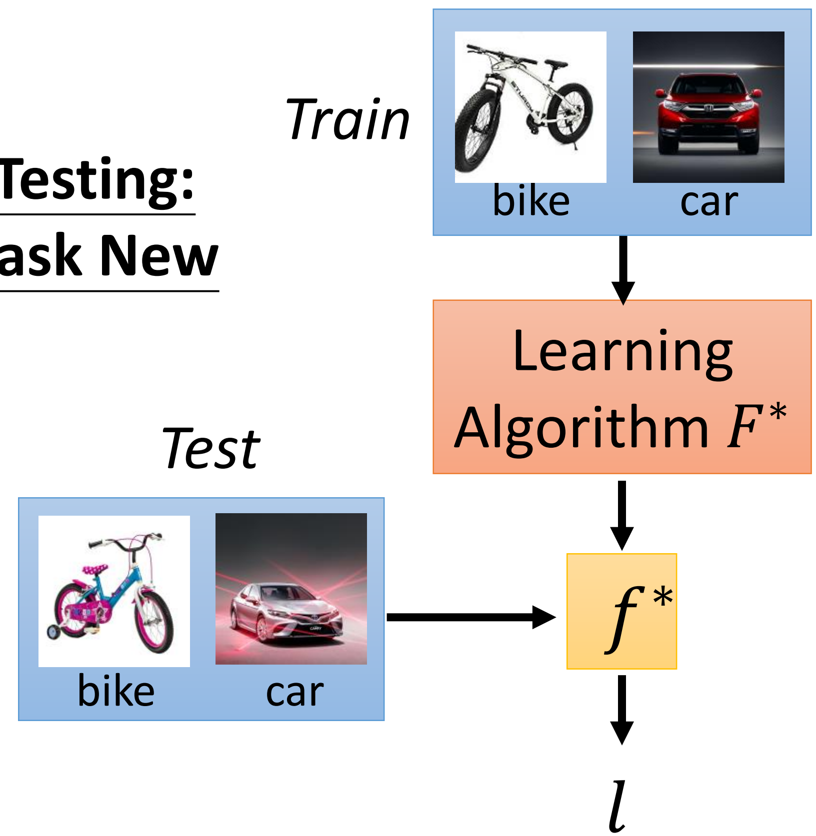
$$L(F) = \sum_{n=1}^{N} l^n$$

- Find the best function F^*

$$F^* = arg \min_{F} L(F)$$

<u>**Testing:**</u>

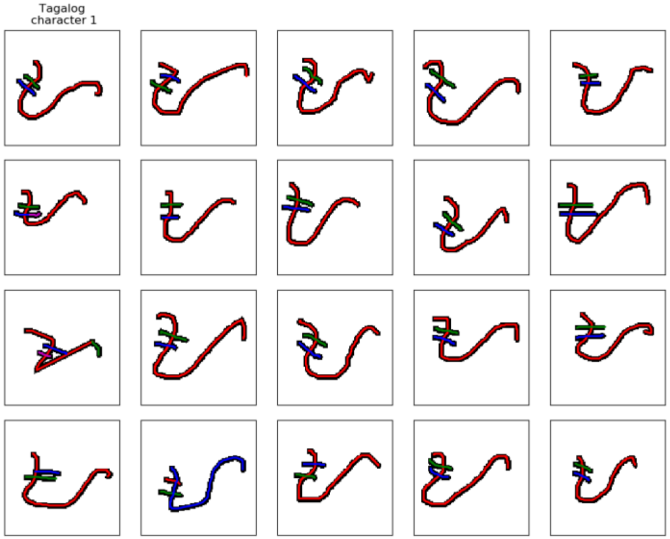
<u>**Task New**</u>



Omniglot

https://github.com/brendenlake/omniglot

- 1623 characters
- Each has 20 examples



Demo of Reptile:
https://openai.com/blog/reptile/

Omniglot – Few-shot Classification

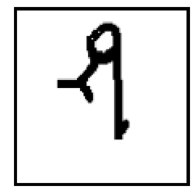
- N-ways K-shot classification: In each training and test tasks, there are N classes, each has K examples.

20 ways

1 shot

Each character represents a class

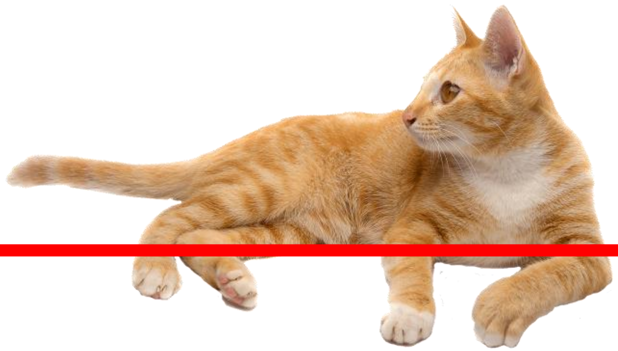
Training set (Support set)



Testing set (Query set)

- Split your characters into training and testing characters
  - Sample N training characters, sample K examples from each sampled characters → one training task
  - Sample N testing characters, sample K examples from each sampled characters → one testing task

Techniques Today



- **MAML**
  - Chelsea Finn, Pieter Abbeel, and Sergey Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”, ICML, 2017
- **Reptile**
  - Alex Nichol, Joshua Achiam, John Schulman, On First-Order Meta-Learning Algorithms, arXiv, 2018

MAML

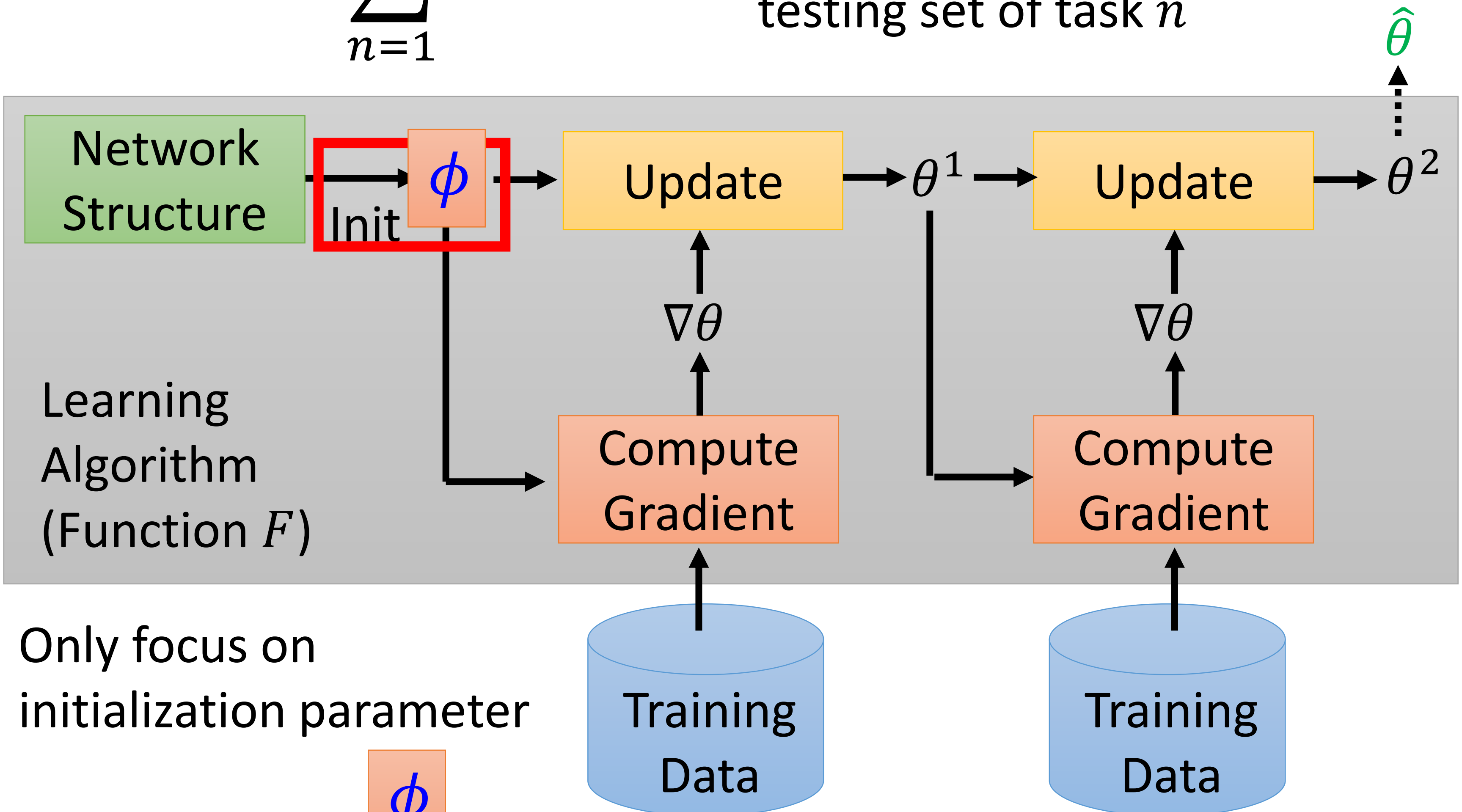
$\hat{\theta}^n$: model learned from task n

$\hat{\theta}^n$ depends on ϕ

Loss Function:

$$L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)$$

$l^n(\hat{\theta}^n)$: loss of task n on the testing set of task n



Only focus on initialization parameter

ϕ

MAML

$\hat{\theta}^n$: model learned from task n

$\hat{\theta}^n$ depends on ϕ

Loss Function:

$$L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)$$

$l^n(\hat{\theta}^n)$: loss of task n on the testing set of task n

How to minimize $L(\phi)$? Gradient Descent

$$\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)$$

Model Pre-training

Widely used in transfer learning

Loss Function:

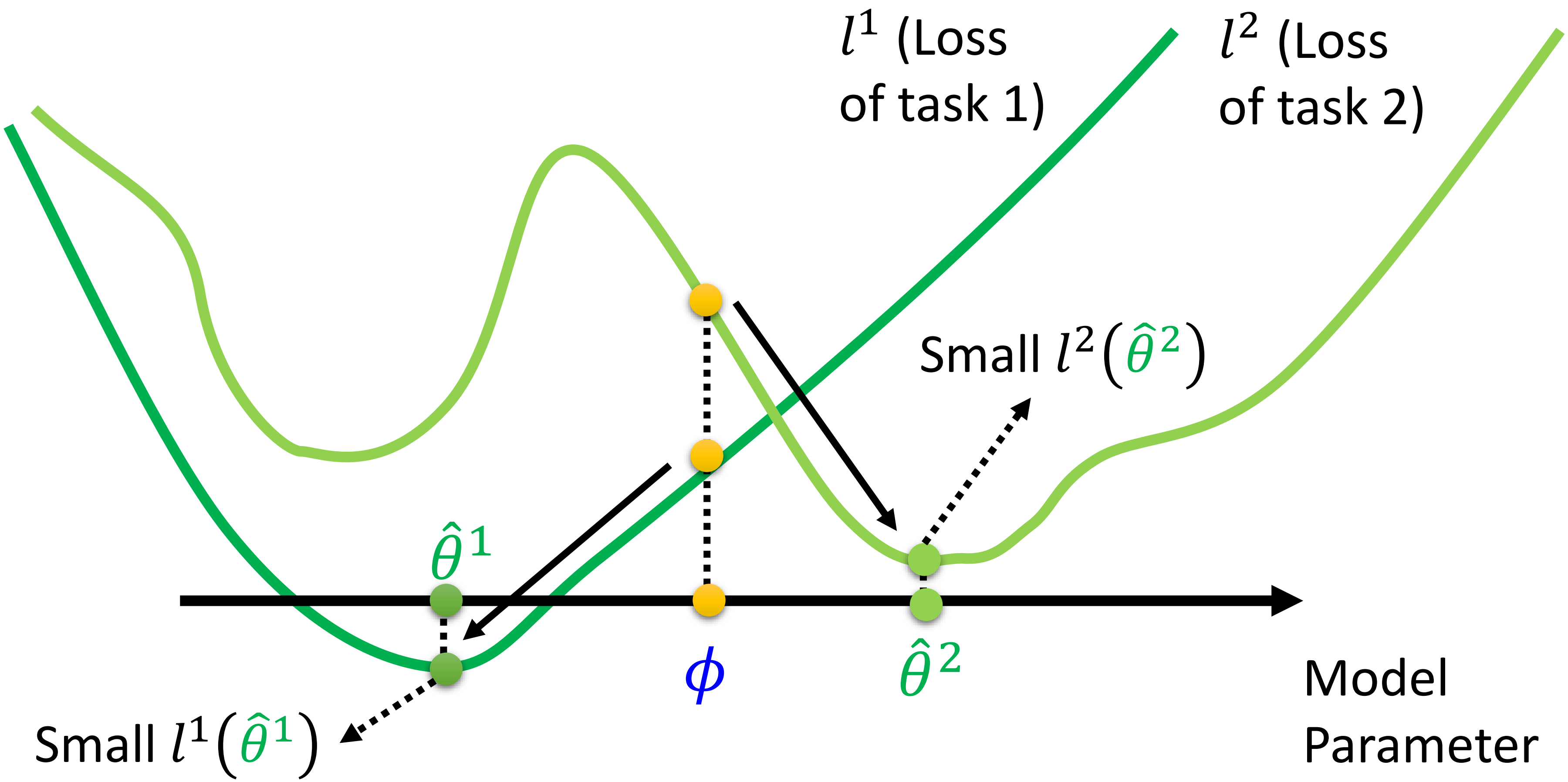
$$L(\phi) = \sum_{n=1}^{N} l^n(\phi)$$

MAML

$$L(\phi)=\sum_{n=1}^{N} l^n\left(\hat{\theta}^n\right)$$

我們不在意 ϕ 在 training task 上表現如何

我們在意用 ϕ 訓練出來的 $\hat{\theta}^n$ 表現如何

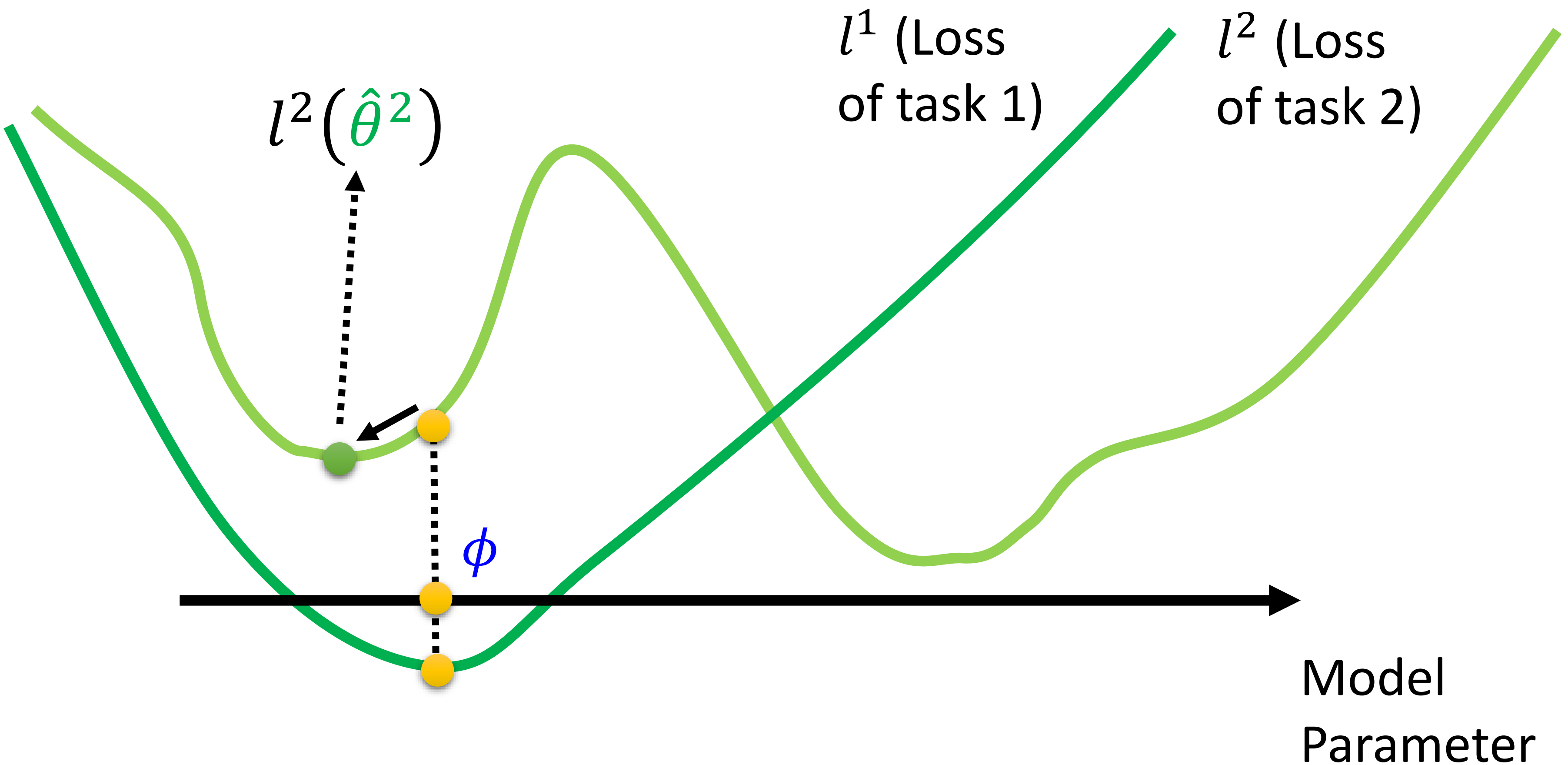


Model Pre-training

$$L(\phi) = \sum_{n=1}^{N} l^n(\phi)$$

找尋在所有 task 都最好的 ϕ

並不保證拿 ϕ 去訓練以後會得到好的 $\hat{\theta}^n$



MAML

$\hat{\theta}^n$: model learned from task n

$\hat{\theta}^n$ depends on ϕ

Loss Function:

$$L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)$$

$l^n(\hat{\theta}^n)$: loss of task n on the testing set of task n

How to minimize $L(\phi)$? Gradient Descent

$$\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)$$

Find ϕ achieving good performance **after training**	潛力

Model Pre-training

Widely used in transfer learning

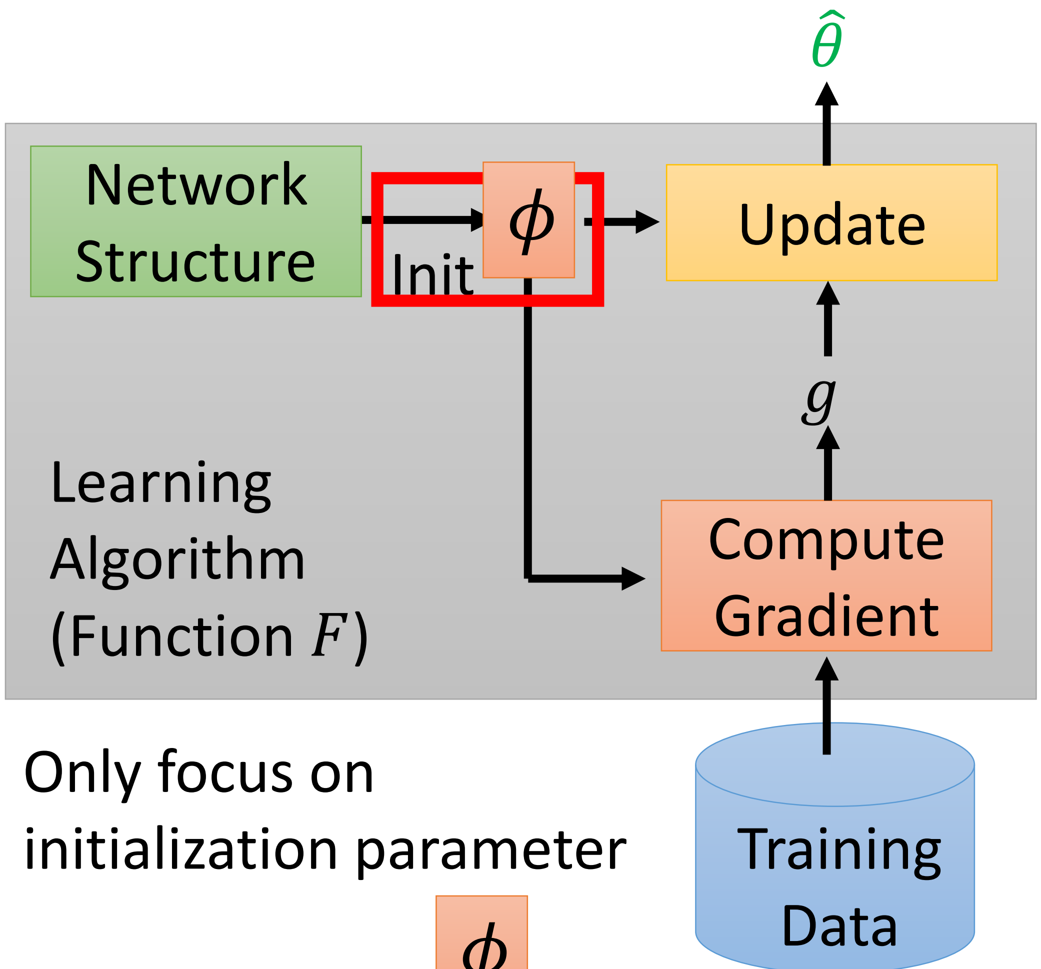
Loss Function:

$$L(\phi) = \sum_{n=1}^{N} l^n(\phi)$$

Find ϕ achieving good performance	現在表現如何

MAML

- Fast … Fast … Fast …
- Good to truly train a model with one step. ☺
- When using the algorithm, still update many times.
- Few-shot learning has limited data.



Only focus on initialization parameter ϕ

$$L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)$$

$$\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)$$

Considering one-step training:

$$\hat{\theta} = \phi - \varepsilon \nabla_{\phi} l(\phi)$$

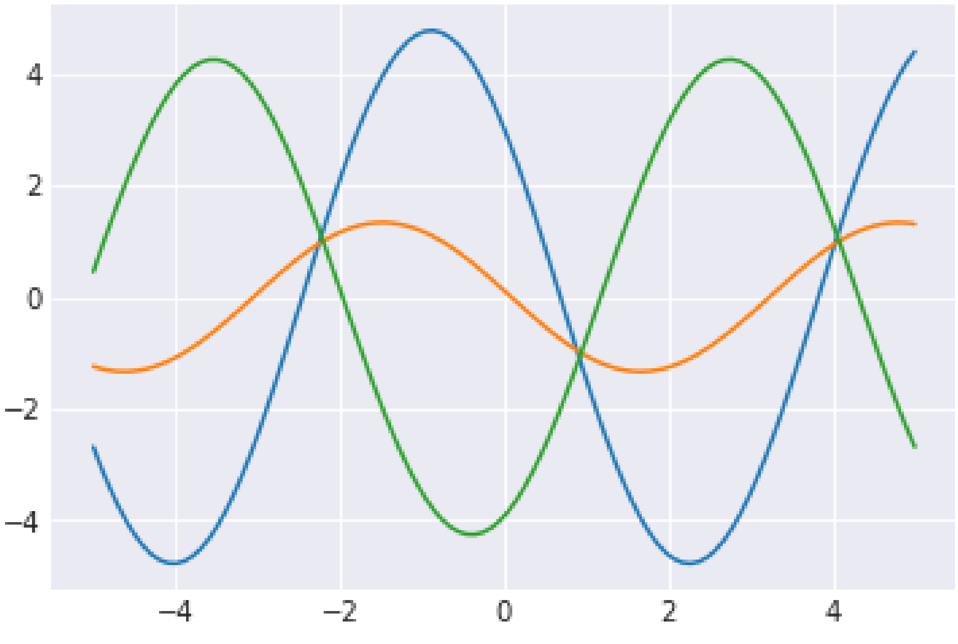
Toy Example

Source of images
https://towardsdatascience.com/paper-repro-deep-metalearning-using-maml-and-reptile-fd1df1cc81b0

Each task:

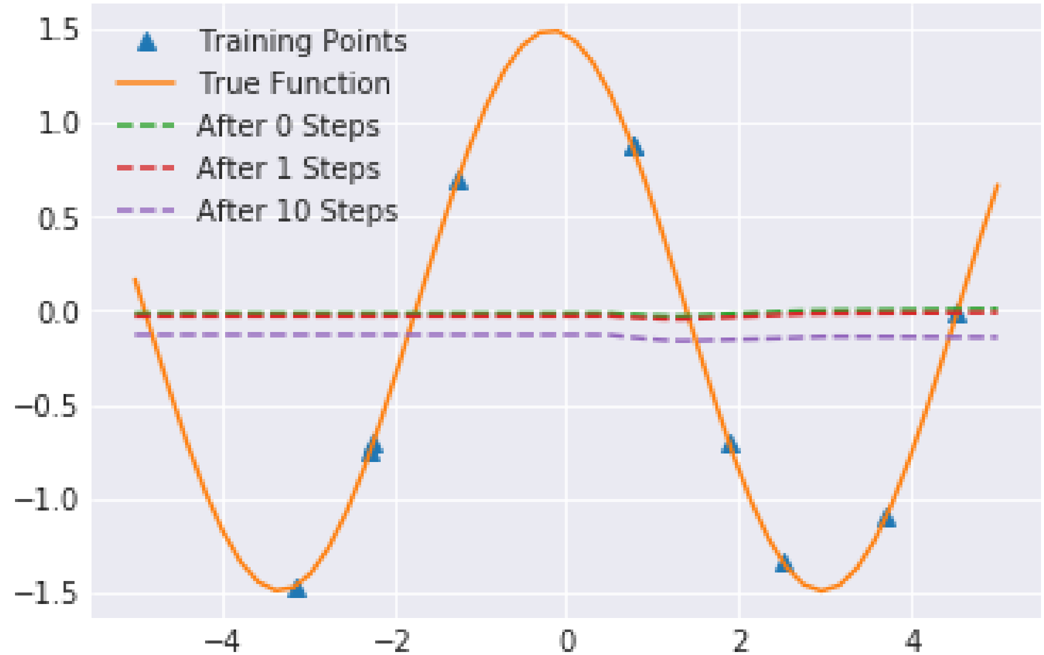
- Given a target sine function $y = a\ sin(x + b)$
- Sample K points from the target function
- Use the samples to estimate the target function

Sample a and b to form a task



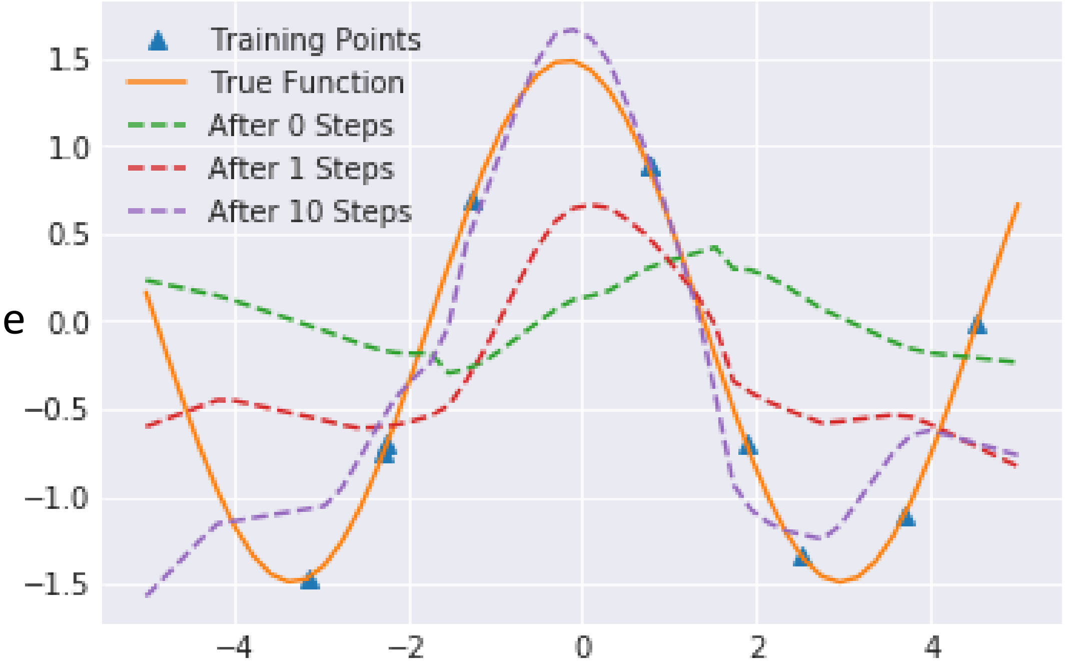
Toy Example

Model Pre-training



Source of images
https://towardsdatascience.com/paper-repro-deep-metalearning-using-maml-and-reptile-fd1df1cc81b0

MAML



Omniglot & Mini-ImageNet

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)	$\mathbf{89.7 \pm 1.1\%}$	$\mathbf{97.5 \pm 0.6\%}$	–	–
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)	$\mathbf{98.7 \pm 0.4\%}$	$\mathbf{99.9 \pm 0.1\%}$	$\mathbf{95.8 \pm 0.3\%}$	$\mathbf{98.9 \pm 0.2\%}$

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy 1-shot	5-way Accuracy 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. (ours)	$\mathbf{48.07 \pm 1.75\%}$	$\mathbf{63.15 \pm 0.91\%}$
MAML (ours)	$\mathbf{48.70 \pm 1.84\%}$	$\mathbf{63.11 \pm 0.92\%}$

https://arxiv.org/abs/1703.03400

Warning of Math

$$\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)$$

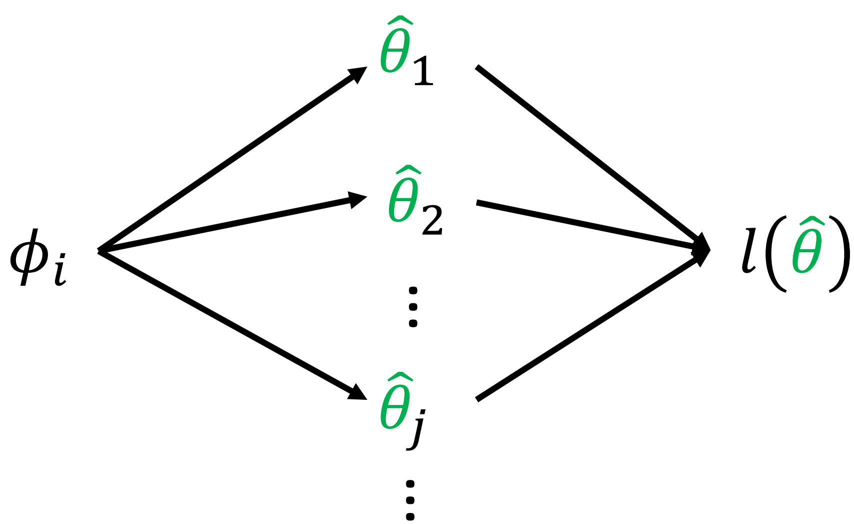
$$L(\phi) = \sum_{n=1}^{N} l^{n}\left(\hat{\theta}^{n}\right)$$

$$\hat{\theta} = \phi - \varepsilon \nabla_{\phi} l(\phi)$$

$$\nabla_{\phi} L(\phi) = \nabla_{\phi} \sum_{n=1}^{N} l^{n}\left(\hat{\theta}^{n}\right) = \sum_{n=1}^{N} \underline{\nabla_{\phi} l^{n}\left(\hat{\theta}^{n}\right)}$$

$$\frac{\partial l(\hat{\theta})}{\partial \phi_i} = \sum_{j} \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \frac{\partial \hat{\theta}_j}{\partial \phi_i}$$

$$\nabla_{\phi} l(\hat{\theta}) = \begin{bmatrix} \partial l(\hat{\theta}) / \partial \phi_1 \\ \partial l(\hat{\theta}) / \partial \phi_2 \\ \vdots \\ \boxed{\partial l(\hat{\theta}) / \partial \phi_i} \\ \vdots \end{bmatrix}$$



$$\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)$$

$$L(\phi) = \sum_{n=1}^{N} l^n\left(\hat{\theta}^n\right)$$

$$\boxed{\hat{\theta} = \phi - \varepsilon \nabla_{\phi} l(\phi)}$$

$$\nabla_{\phi} L(\phi) = \nabla_{\phi} \sum_{n=1}^{N} l^n\left(\hat{\theta}^n\right) = \sum_{n=1}^{N} \underline{\nabla_{\phi} l^n\left(\hat{\theta}^n\right)}$$

$$\frac{\partial l(\hat{\theta})}{\partial \phi_i} = \sum_j \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \boxed{\frac{\partial \hat{\theta}_j}{\partial \phi_i}} \approx \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_i}$$

$$\hat{\theta}_j = \phi_j - \varepsilon \frac{\partial l(\phi)}{\partial \phi_j}$$

$i \neq j$:

$$\frac{\partial \hat{\theta}_j}{\partial \phi_i} = -\varepsilon \cancel{\frac{\partial l(\phi)}{\partial \phi_i \partial \phi_j}} \approx 0$$

$i = j$:

$$\frac{\partial \hat{\theta}_j}{\partial \phi_i} = 1 - \varepsilon \cancel{\frac{\partial l(\phi)}{\partial \phi_i \partial \phi_j}} \approx 1$$

$$\nabla_{\phi} l(\hat{\theta}) = \begin{bmatrix} \partial l(\hat{\theta})/\partial \phi_1 \\ \partial l(\hat{\theta})/\partial \phi_2 \\ \vdots \\ \boxed{\partial l(\hat{\theta})/\partial \phi_i} \\ \vdots \end{bmatrix}$$

$$\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)$$

$$L(\phi) = \sum_{n=1}^{N} l^n\left(\hat{\theta}^n\right)$$

$$\hat{\theta} = \phi - \varepsilon \nabla_{\phi} l(\phi)$$

$$\nabla_{\phi} L(\phi) = \nabla_{\phi} \sum_{n=1}^{N} l^n\left(\hat{\theta}^n\right) = \sum_{n=1}^{N} \nabla_{\phi} l^n\left(\hat{\theta}^n\right)$$

$$\hat{\theta}^n$$

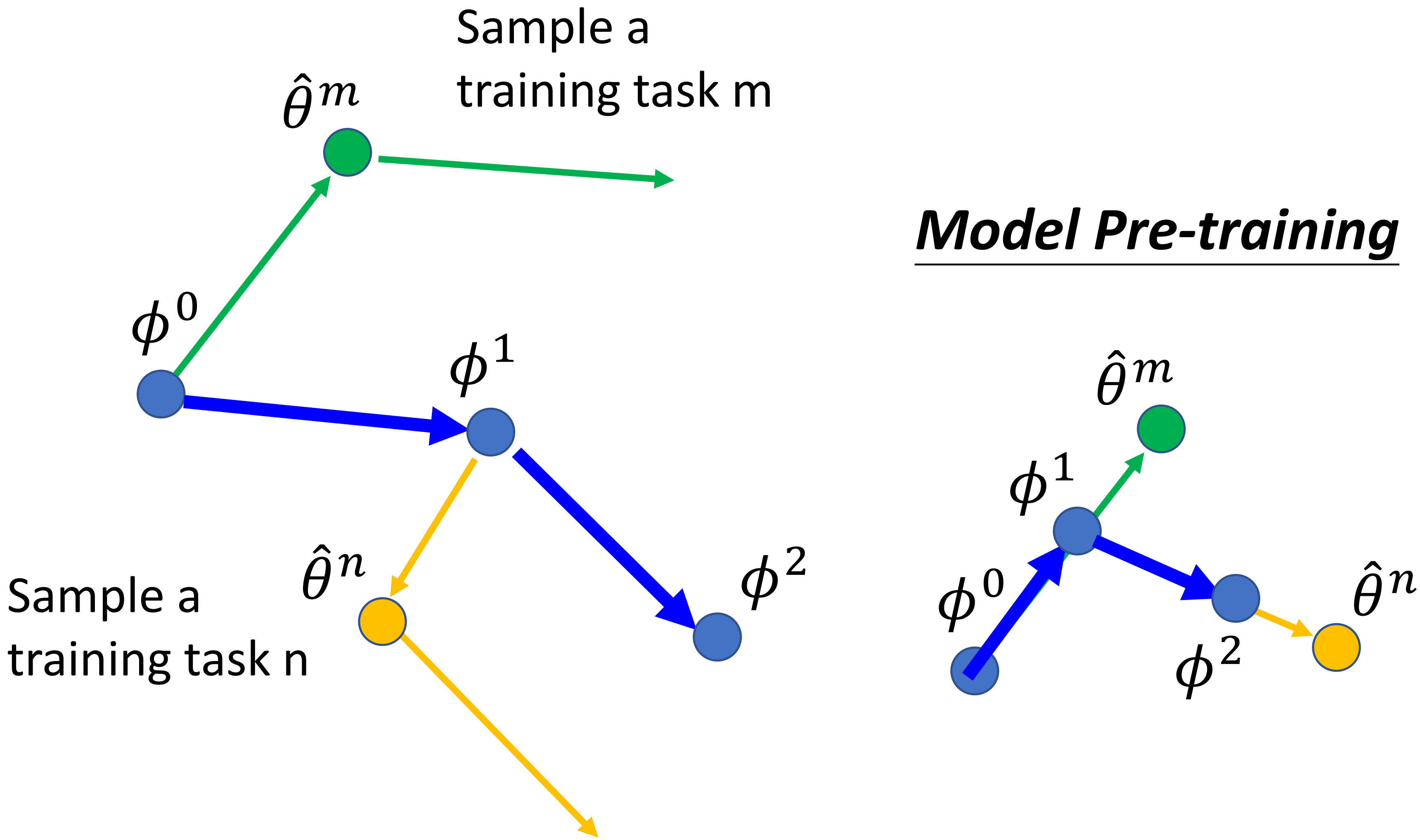
$$\frac{\partial l(\hat{\theta})}{\partial \phi_i} = \sum_j \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \frac{\partial \hat{\theta}_j}{\partial \phi_i} \approx \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_i}$$

products, which is supported by standard deep learning libraries such as TensorFlow (Abadi et al., 2016). In our experiments, we also include a comparison to dropping this backward pass and using a first-order approximation, which we discuss in Section 5.2.

$$\nabla_{\phi} l(\hat{\theta}) = \begin{bmatrix} \partial l(\hat{\theta})/\partial \phi_1 \\ \partial l(\hat{\theta})/\partial \phi_2 \\ \vdots \\ \partial l(\hat{\theta})/\partial \phi_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \partial l(\hat{\theta})/\partial \hat{\theta}_1 \\ \partial l(\hat{\theta})/\partial \hat{\theta}_2 \\ \vdots \\ \partial l(\hat{\theta})/\partial \hat{\theta}_i \\ \vdots \end{bmatrix} = \nabla_{\hat{\theta}} l(\hat{\theta})$$

End of Warning

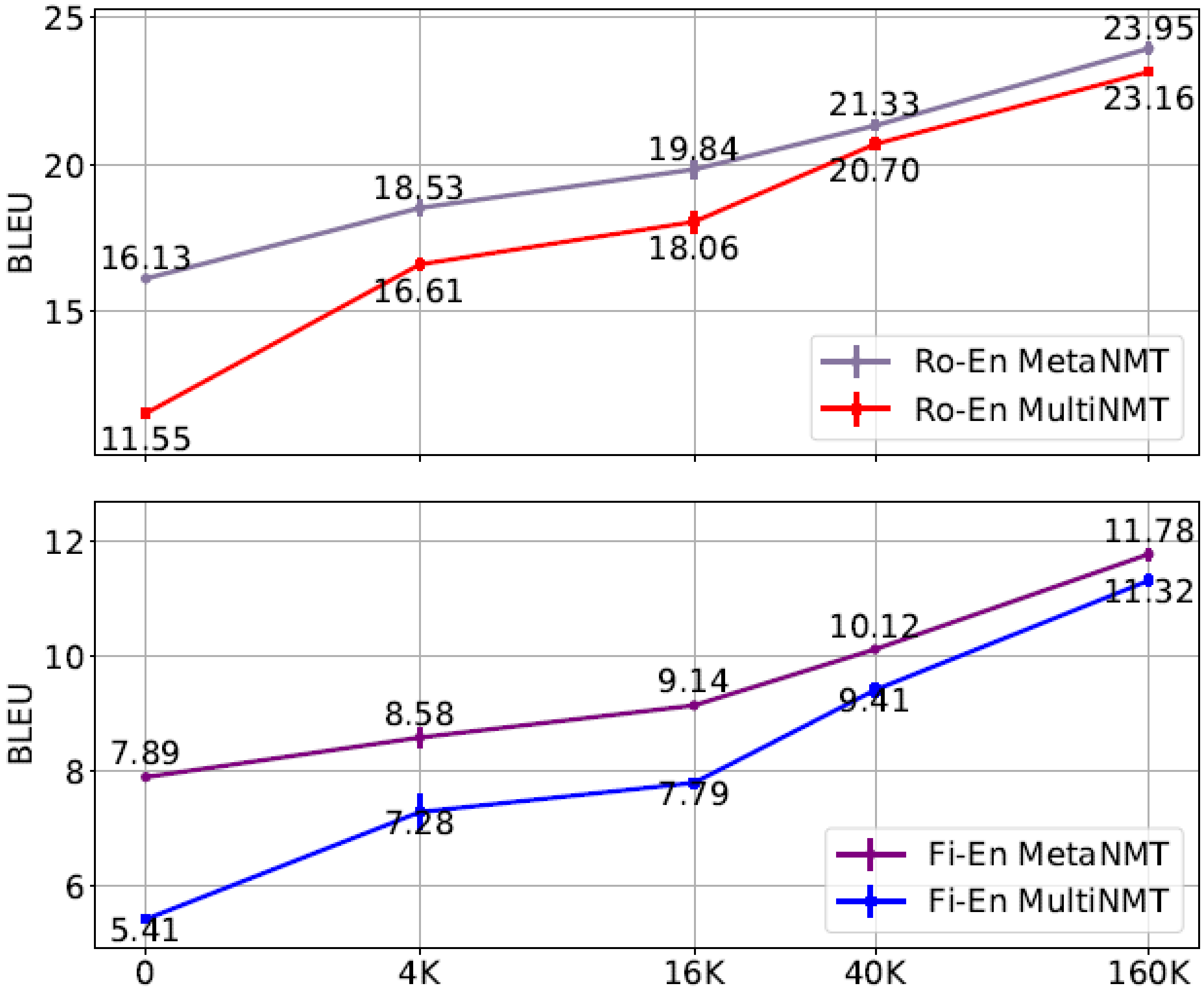
MAML – Real Implementation



Translation

18 training tasks: 18 different languages translating to English

2 validation tasks: 2 different languages translating to English



Ro = Romanian

Fi = Finnish

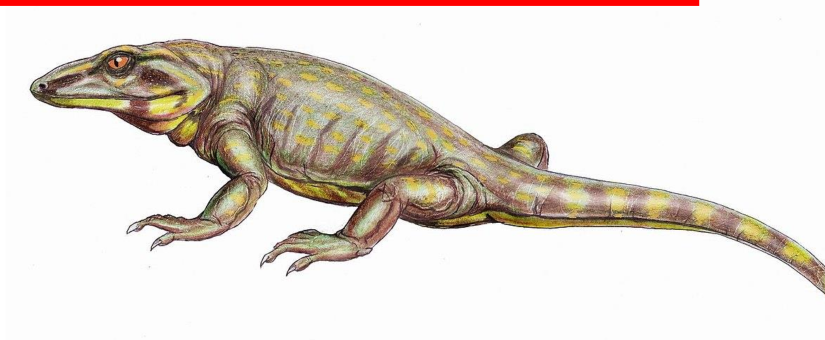
https://arxiv.org/abs/1808.08437

Techniques Today

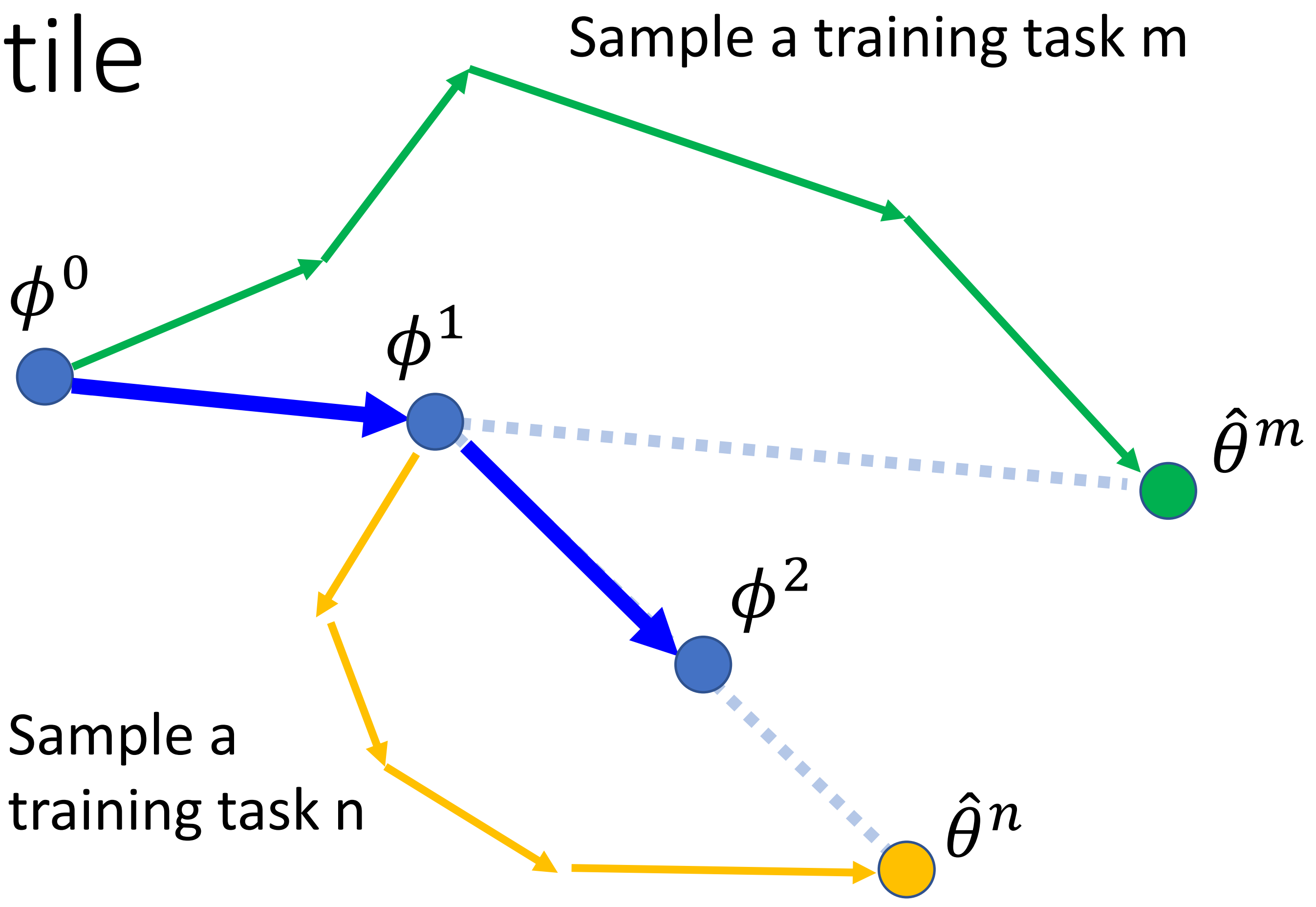
- **MAML**
  - Chelsea Finn, Pieter Abbeel, and Sergey Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”, ICML, 2017

- **Reptile**
  - Alex Nichol, Joshua Achiam, John Schulman, On First-Order Meta-Learning Algorithms, arXiv, 2018

https://openai.com/blog/reptile/



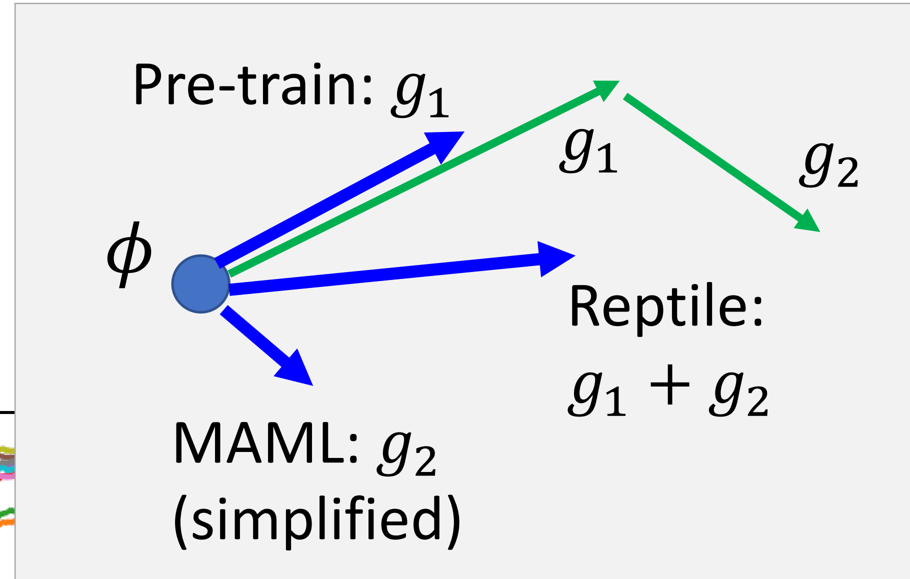
Reptile



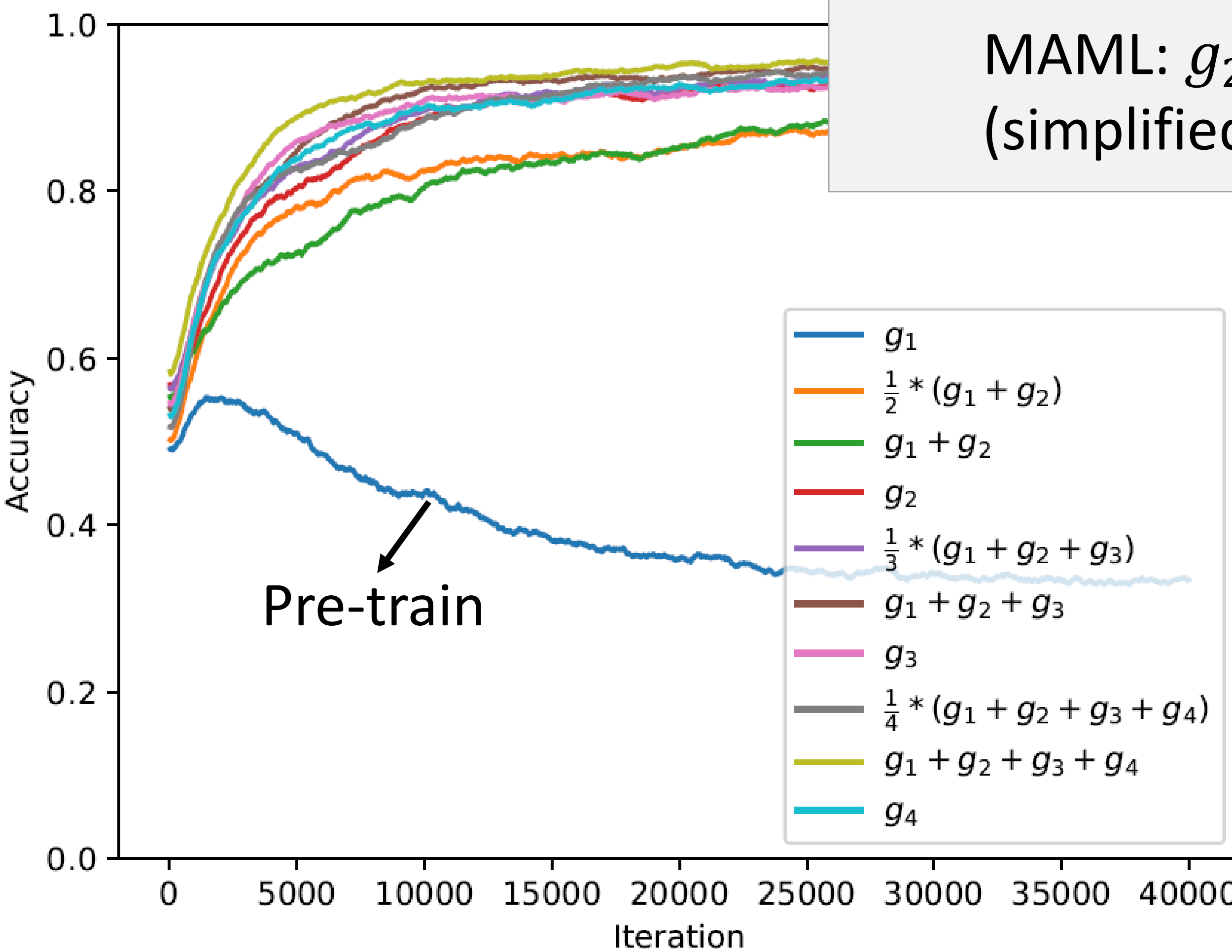
You might be thinking "isn't this the same as training on the expected loss $\mathbb{E}_\tau[L_\tau]$?" and then checking if the date is April 1st. Indeed, if the partial minimization consists of a single gradient step, then this algorithm corresponds to minimizing the expected loss:

(this sentence is removed in the updated version)

Reptile

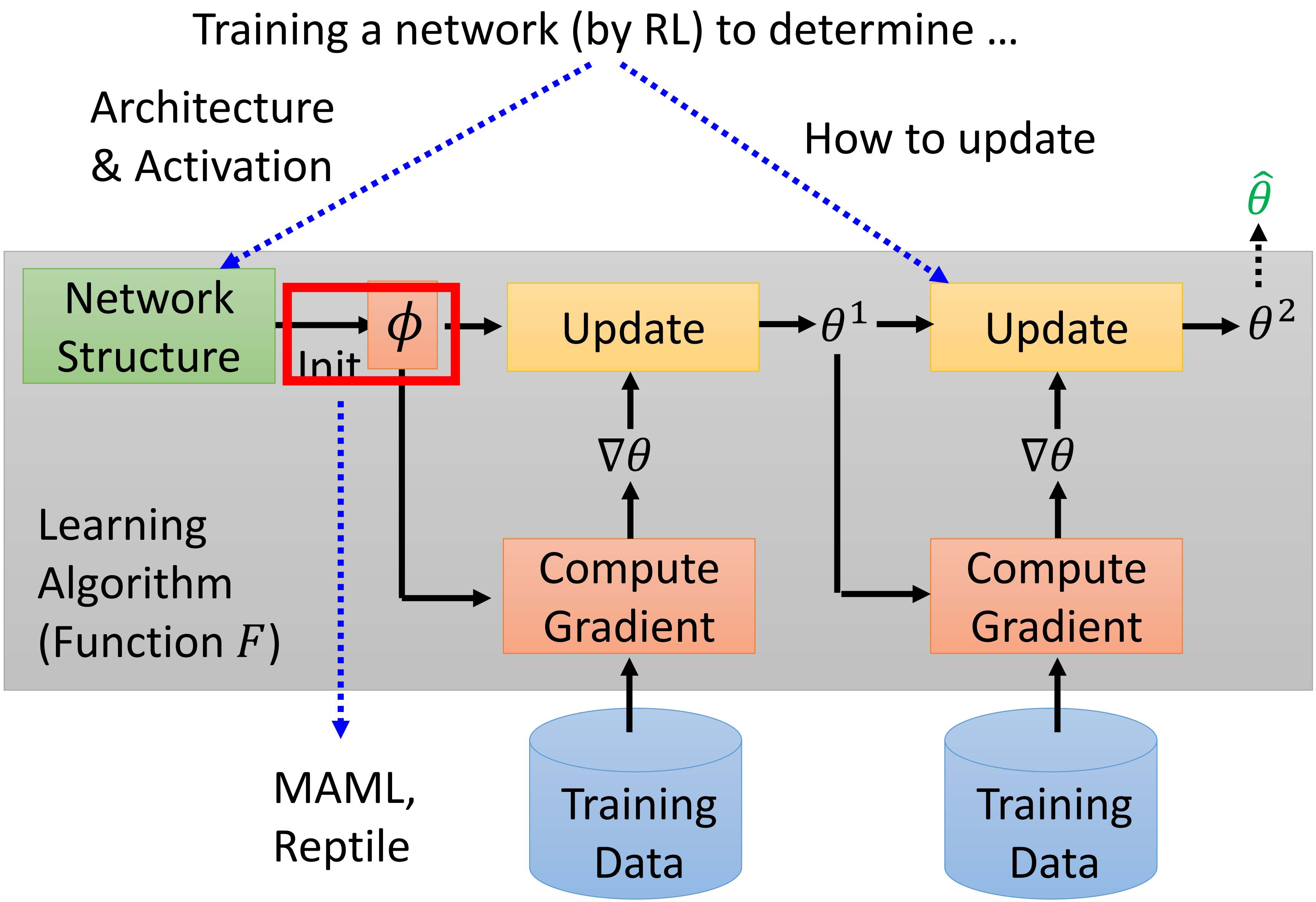


Pre-train: g_1
g_1
g_2
ϕ
Reptile:
$g_1 + g_2$
MAML: g_2
(simplified)



Accuracy
Iteration
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000
35000
40000
g_1
$\frac{1}{2} * (g_1 + g_2)$
$g_1 + g_2$
g_2
$\frac{1}{3} * (g_1 + g_2 + g_3)$
$g_1 + g_2 + g_3$
g_3
$\frac{1}{4} * (g_1 + g_2 + g_3 + g_4)$
$g_1 + g_2 + g_3 + g_4$
g_4
Pre-train

More … Video: https://www.youtube.com/watch?v=c10nxBcSH14



Turtles all the way down ……?

- We learn the initialization parameter ϕ by gradient descent
- What is the initialization parameter ϕ^0 for initialization parameter ϕ?

Learn

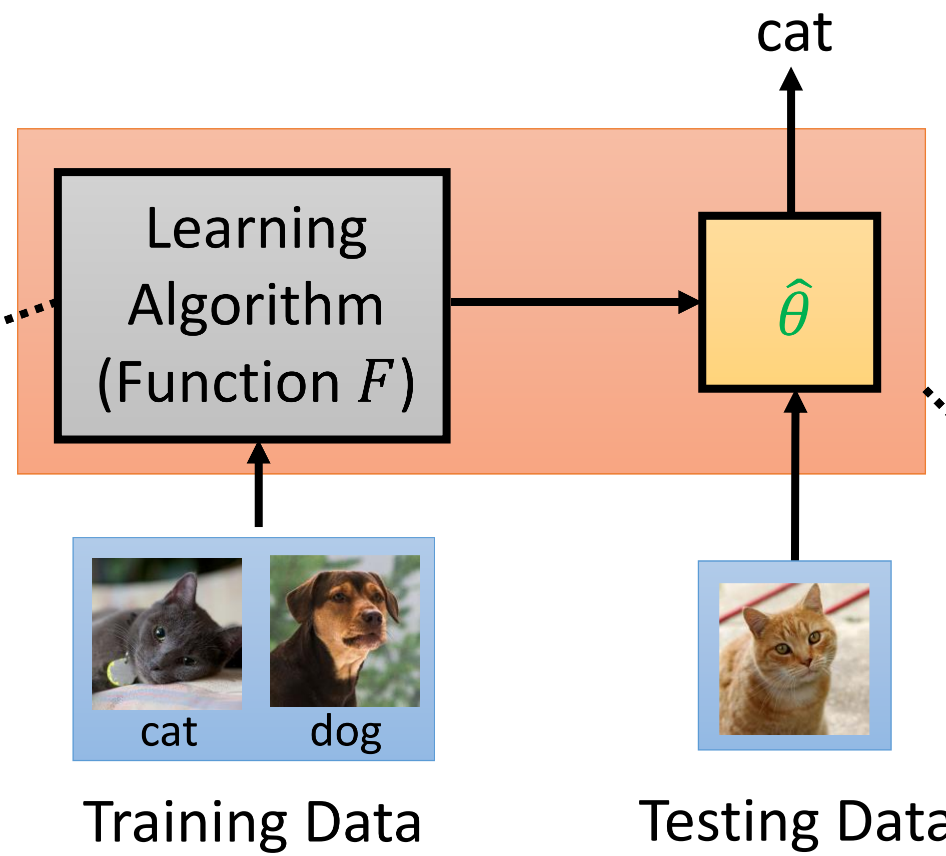
Learn to learn

Learn to learn to learn

Crazy Idea?

下回分解 ☺

- How about learning algorithm beyond gradient descent?



Just a network

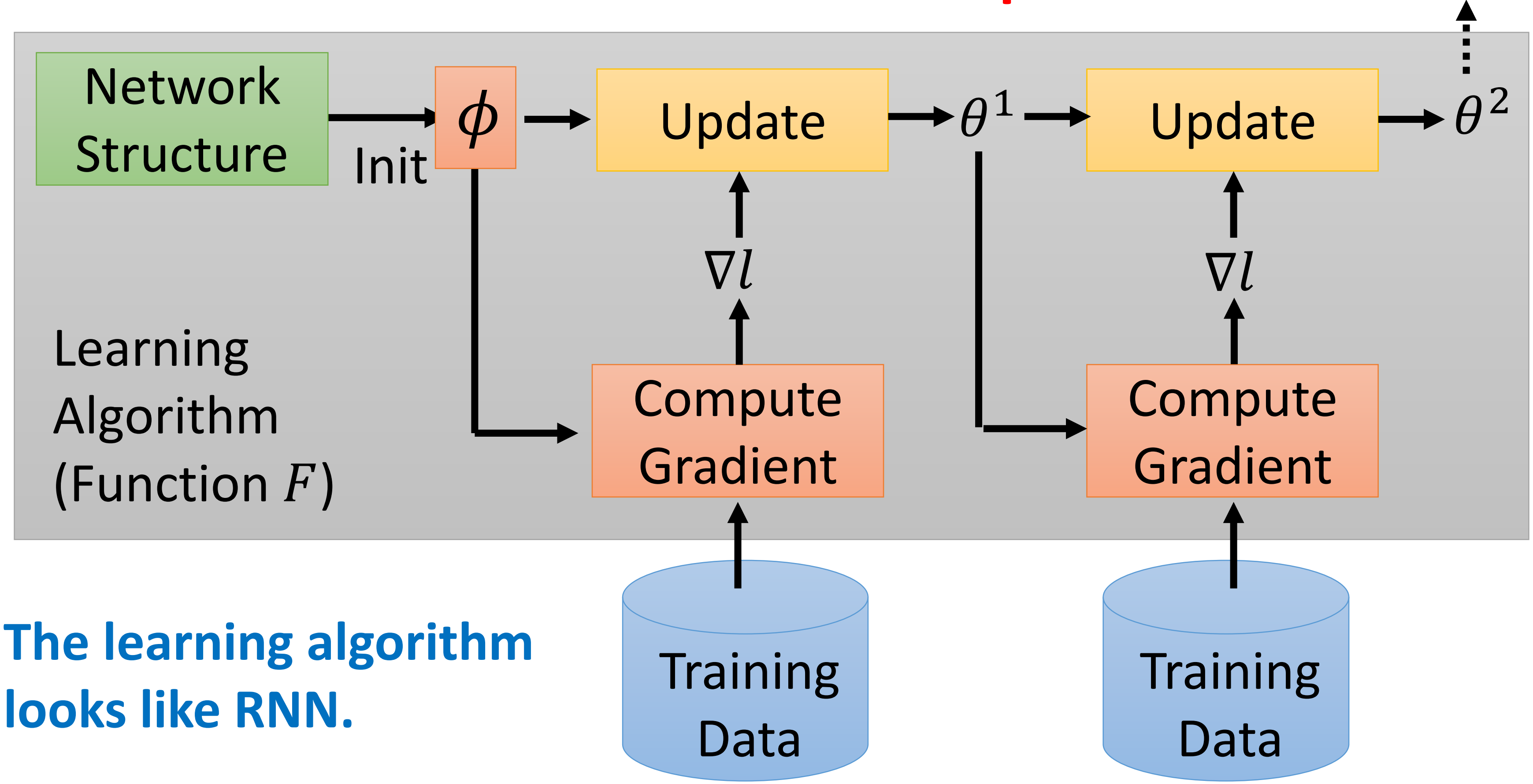
Learn an even bigger function

Design network architecture
= Design training algorithm?

Meta Learning (Part 2):
Gradient Descent as LSTM

Hung-yi Lee

Can we learn more than initialization parameters?



The learning algorithm looks like RNN.

OPTIMIZATION AS A MODEL FOR FEW-SHOT LEARNING

Sachin Ravi* and Hugo Larochelle
Twitter, Cambridge, USA
{sachinr,hugo}@twitter.com

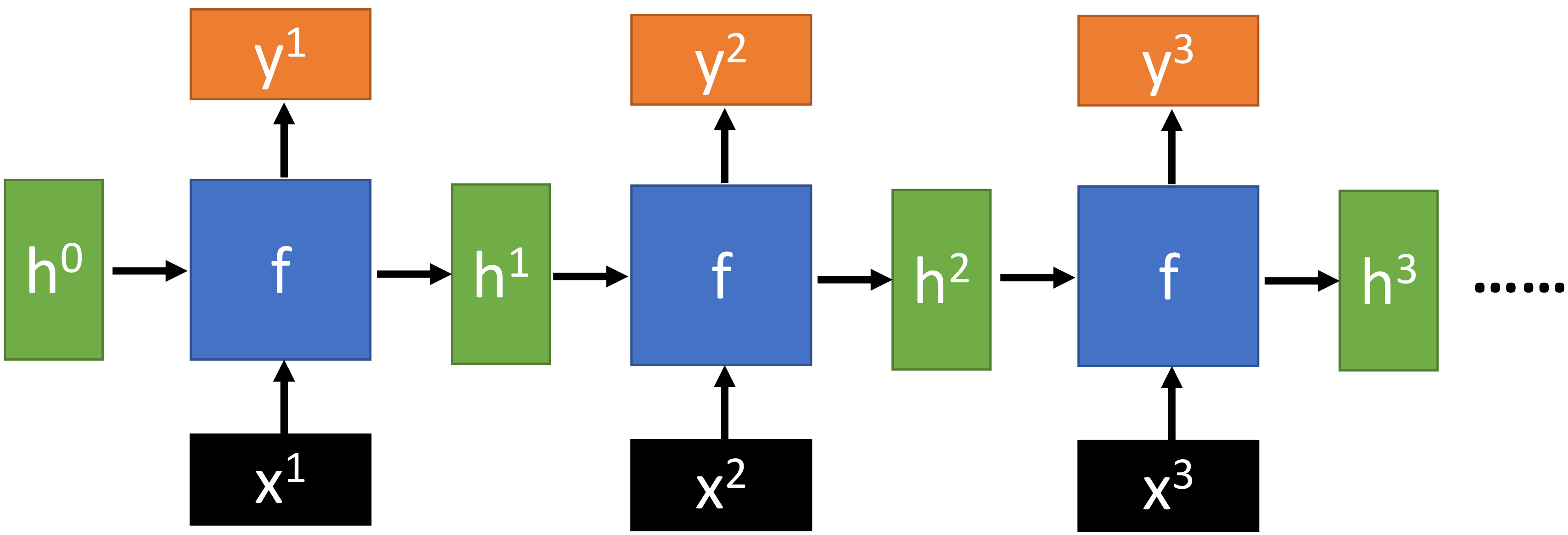
Learning to learn by gradient descent by gradient descent

Marcin Andrychowicz[1], Misha Denil[1], Sergio Gómez Colmenarejo[1], Matthew W. Hoffman[1], David Pfau[1], Tom Schaul[1], Brendan Shillingford[1,2], Nando de Freitas[1,2,3]
[1]Google DeepMind [2]University of Oxford [3]Canadian Institute for Advanced Research

Recurrent Neural Network

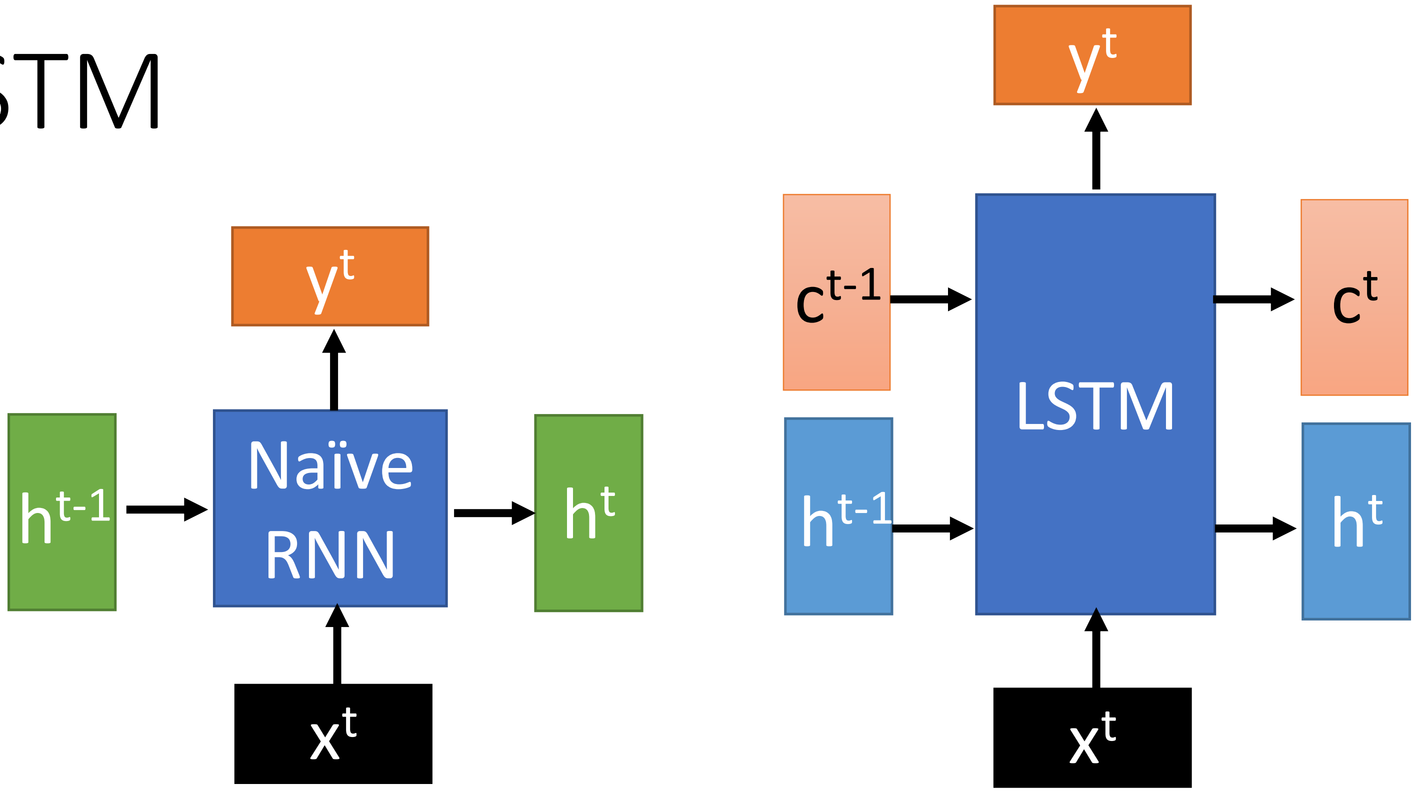
- Given function f: $h', y = f(h, x)$

h and h' are vectors with the same dimension



No matter how long the input/output sequence is, we only need one function f

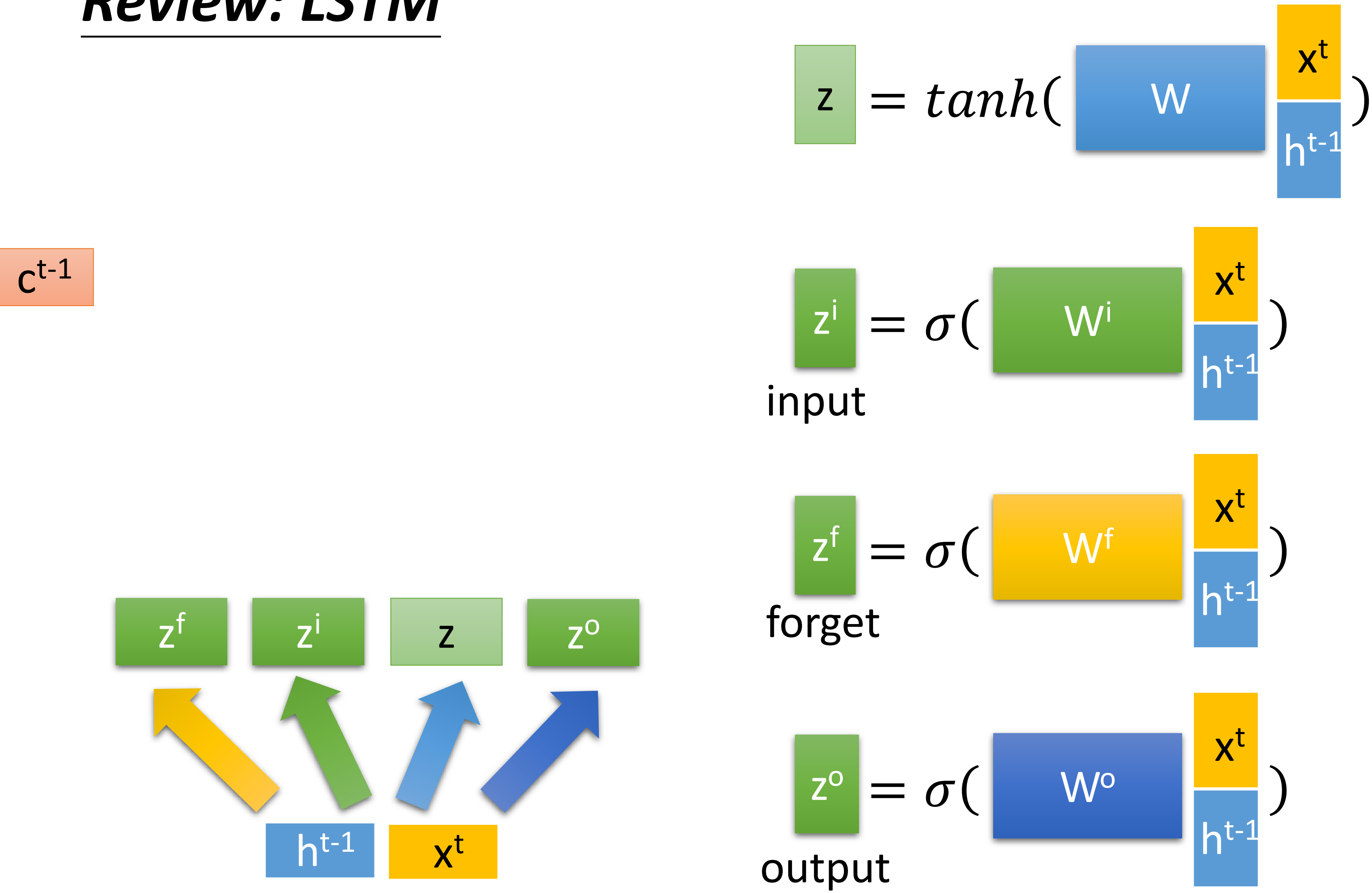
LSTM



c change slowly ➡ c^t is c^{t-1} added by something

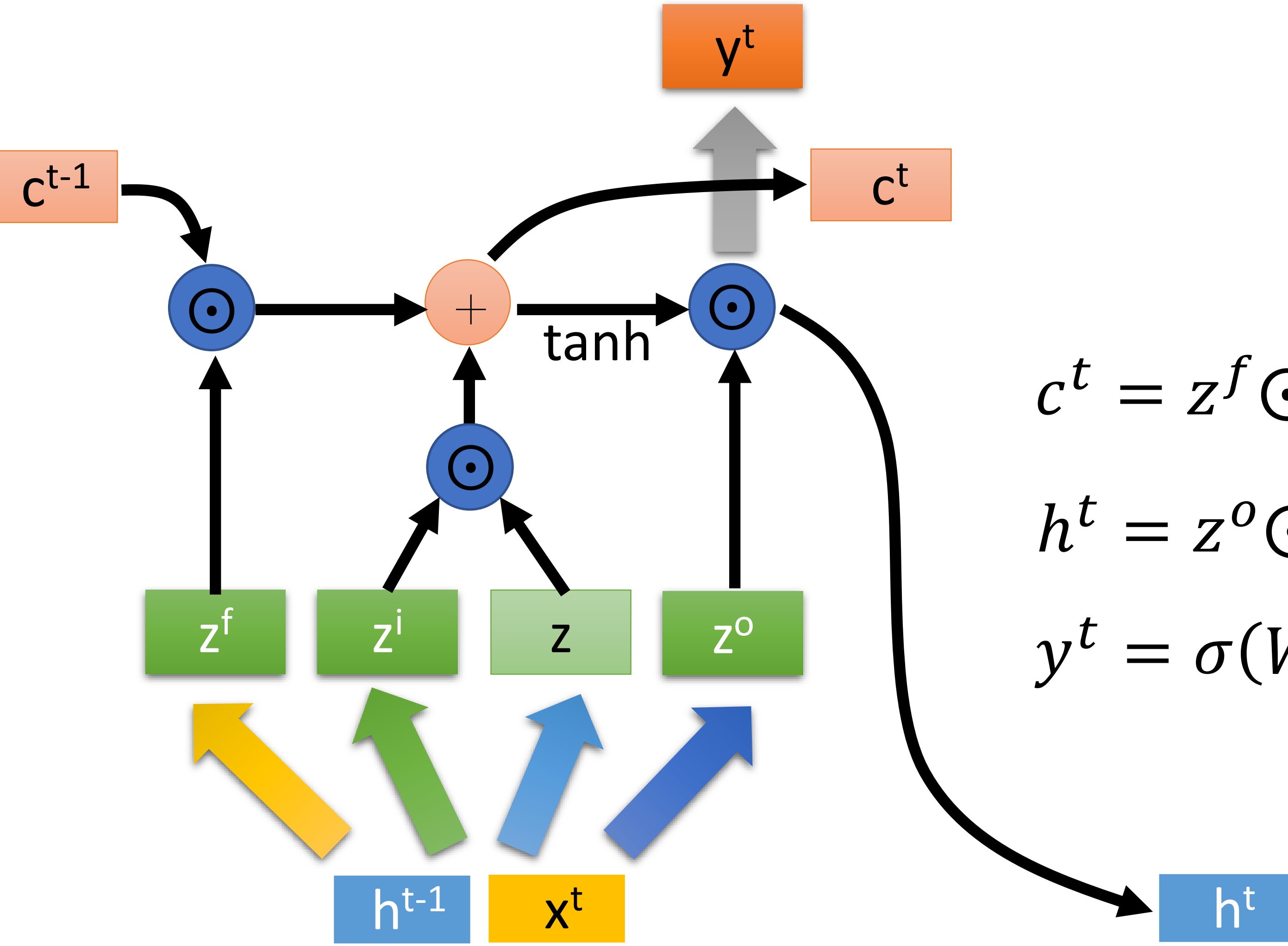
h change faster ➡ h^t and h^{t-1} can be very different

Review: LSTM



$z = tanh(W \; x^t \; h^{t-1})$
$z^i = \sigma(W^i \; x^t \; h^{t-1})$
input
$z^f = \sigma(W^f \; x^t \; h^{t-1})$
forget
$z^o = \sigma(W^o \; x^t \; h^{t-1})$
output
c^{t-1}
z^f z^i z z^o
h^{t-1} x^t

Review: LSTM

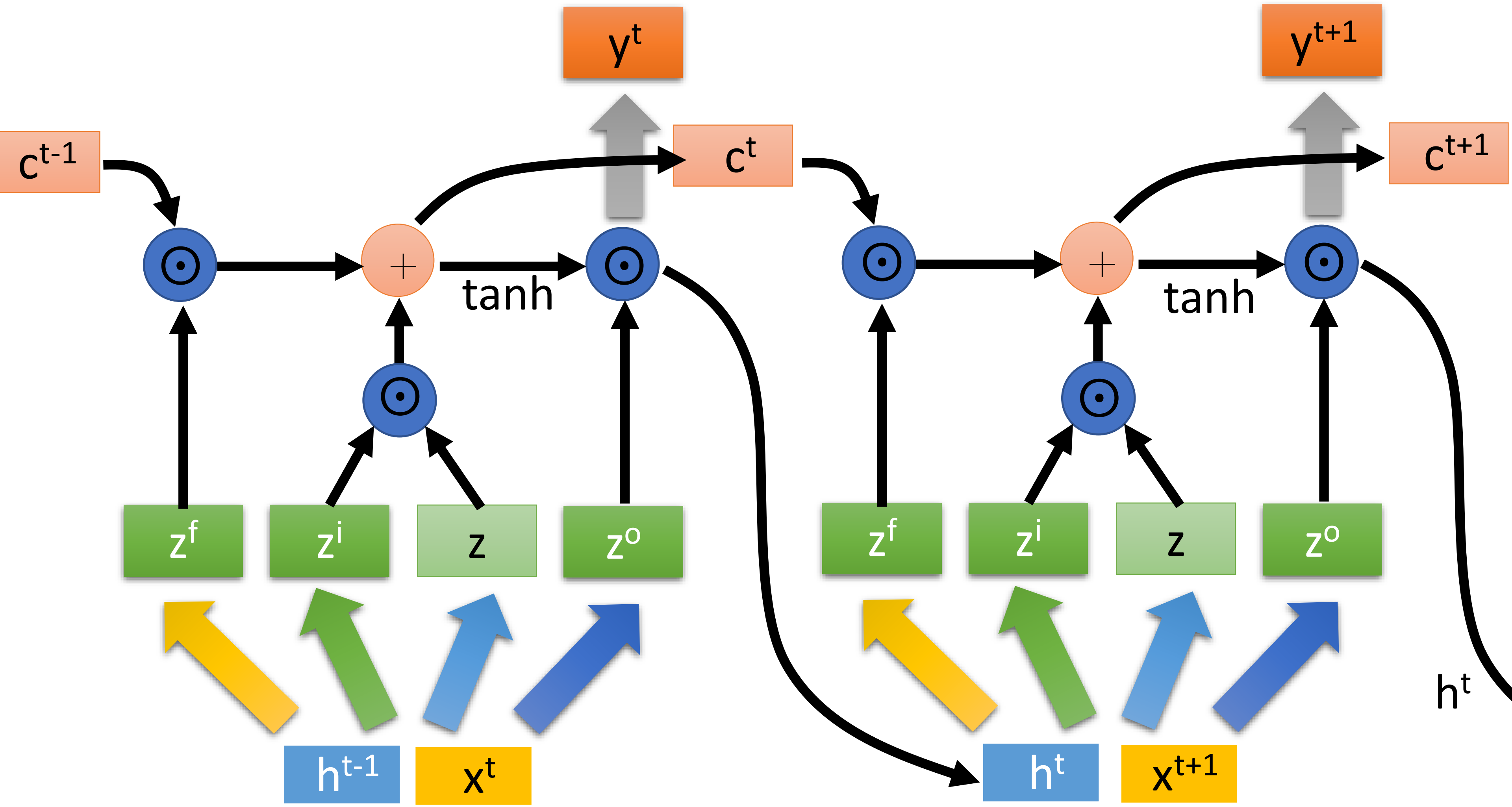


$$c^t = z^f \odot c^{t-1} + z^i \odot z$$

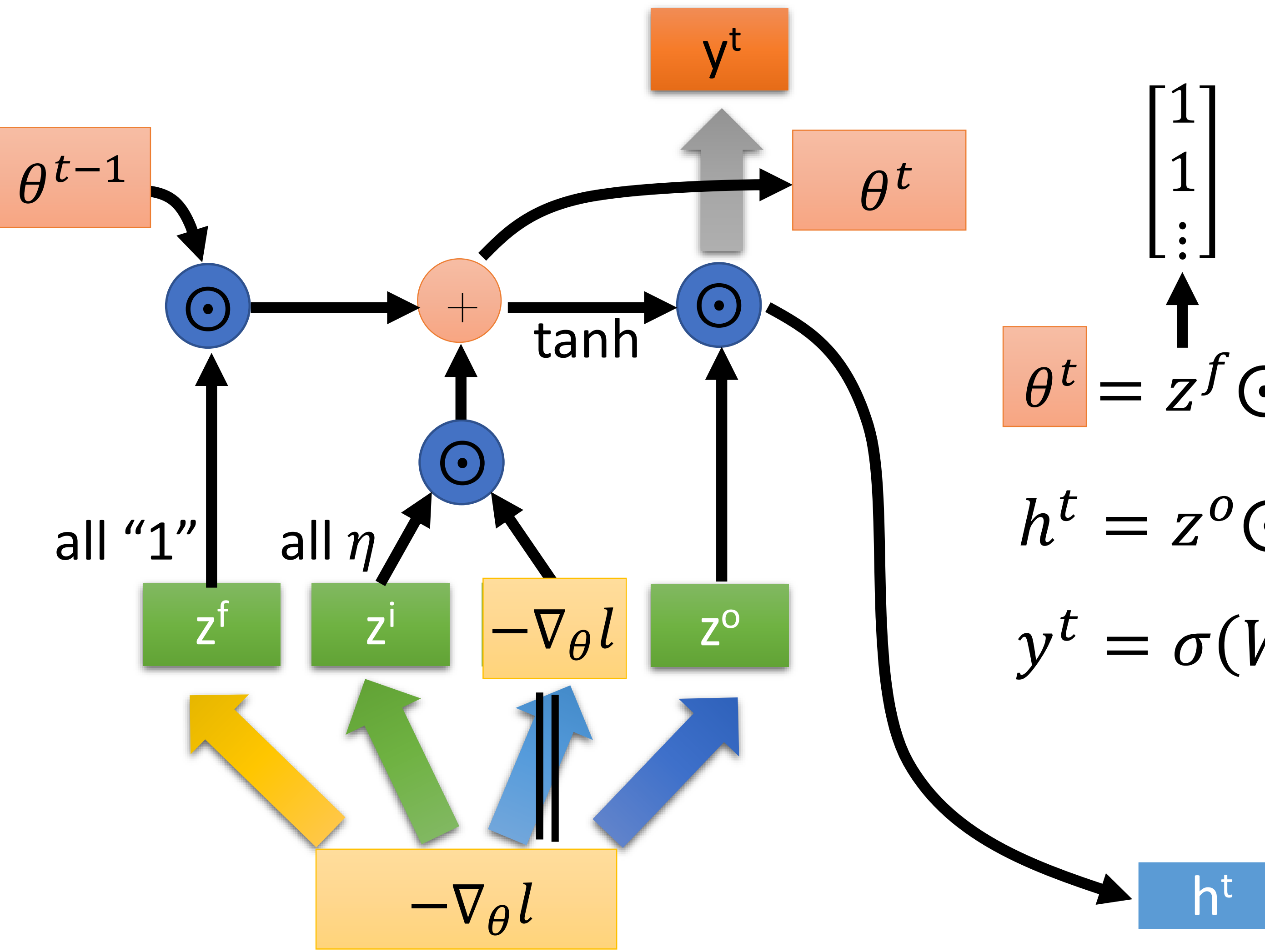
$$h^t = z^o \odot tanh(c^t)$$

$$y^t = \sigma(W' h^t)$$

Review: LSTM



Similar to gradient descent based algorithm



$$\theta^t = \theta^{t-1} - \eta \nabla_\theta l$$

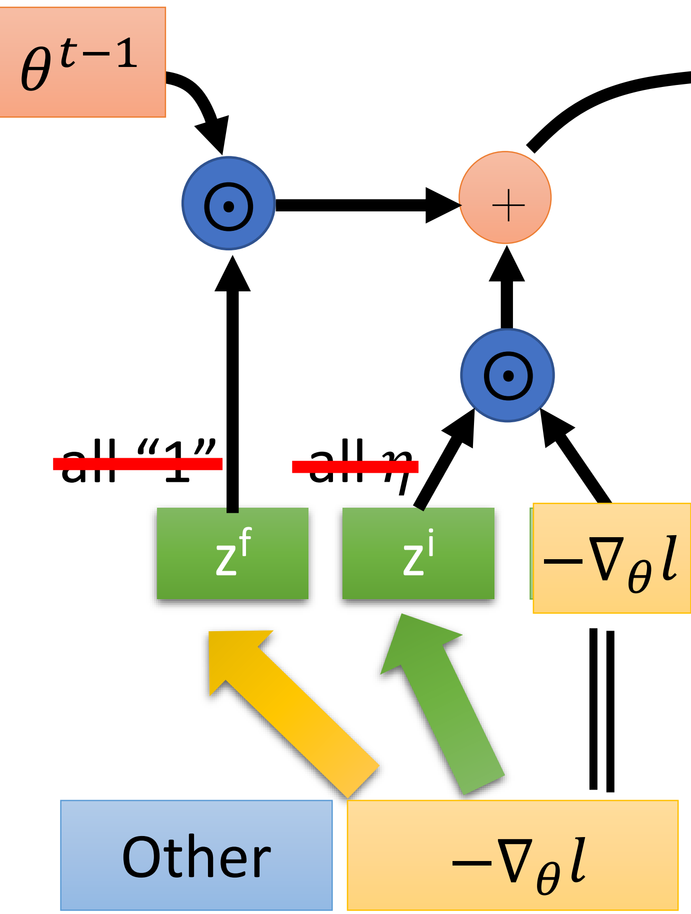
$$\theta^t = \underbrace{z^f}_{\begin{bmatrix}1\\1\\\vdots\end{bmatrix}} \odot \theta^{t-1} + \underbrace{z^i}_{\begin{bmatrix}\eta\\\eta\\\vdots\end{bmatrix}} \odot -\nabla_\theta l$$

$$h^t = z^o \odot tanh(c^t)$$

$$y^t = \sigma(W' h^t)$$

Similar to gradient descent based algorithm

$$\theta^t = \theta^{t-1} - \eta\nabla_\theta l$$



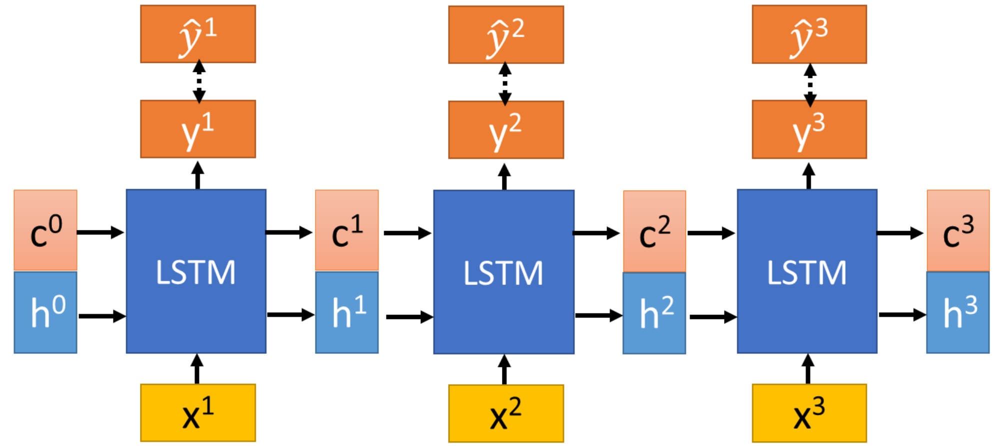
Something like regularization

$$\theta^t = z^f \odot \theta^{t-1} + z^i \odot -\nabla_\theta l$$

Dynamic learning rate

How about machine learn to determine z^f and z^i from $-\nabla_\theta l$ and other information?

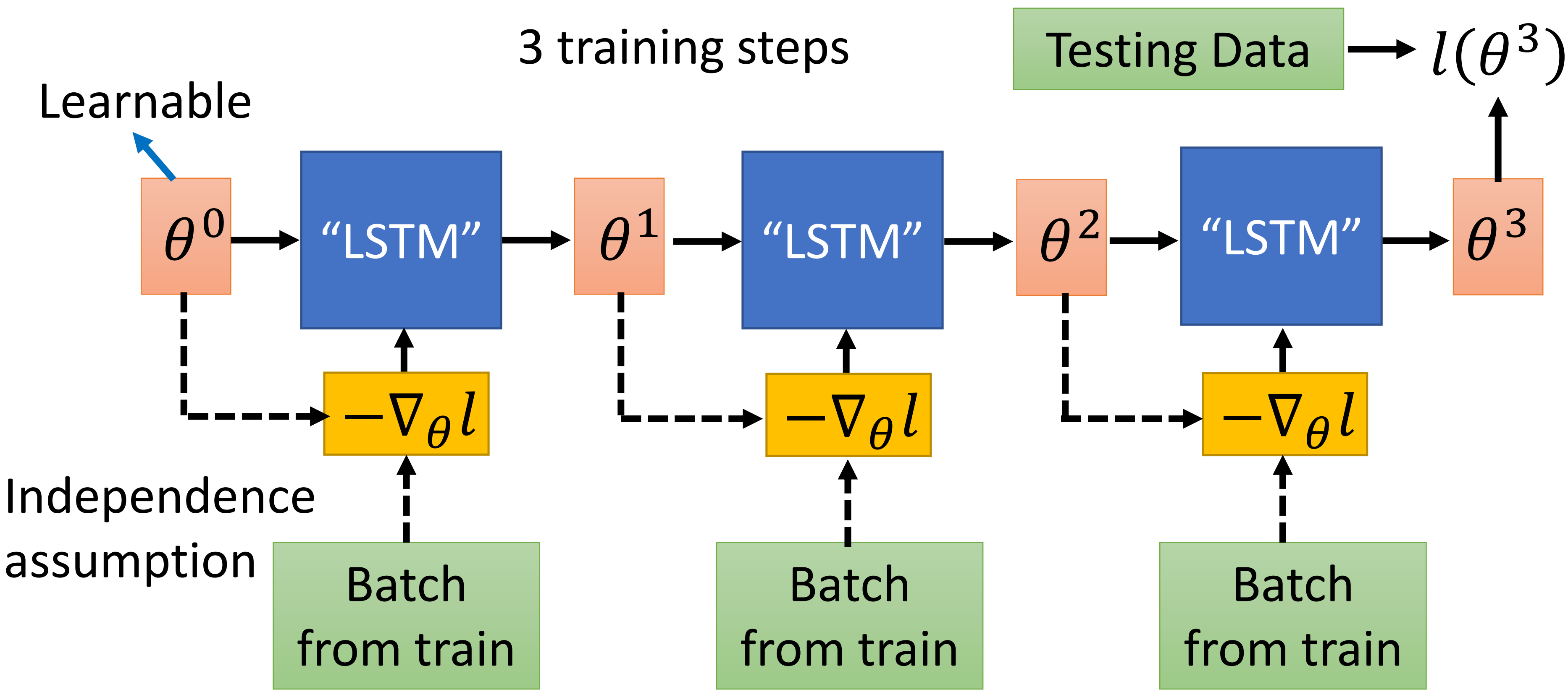
Typical LSTM



LSTM for Gradient Descent

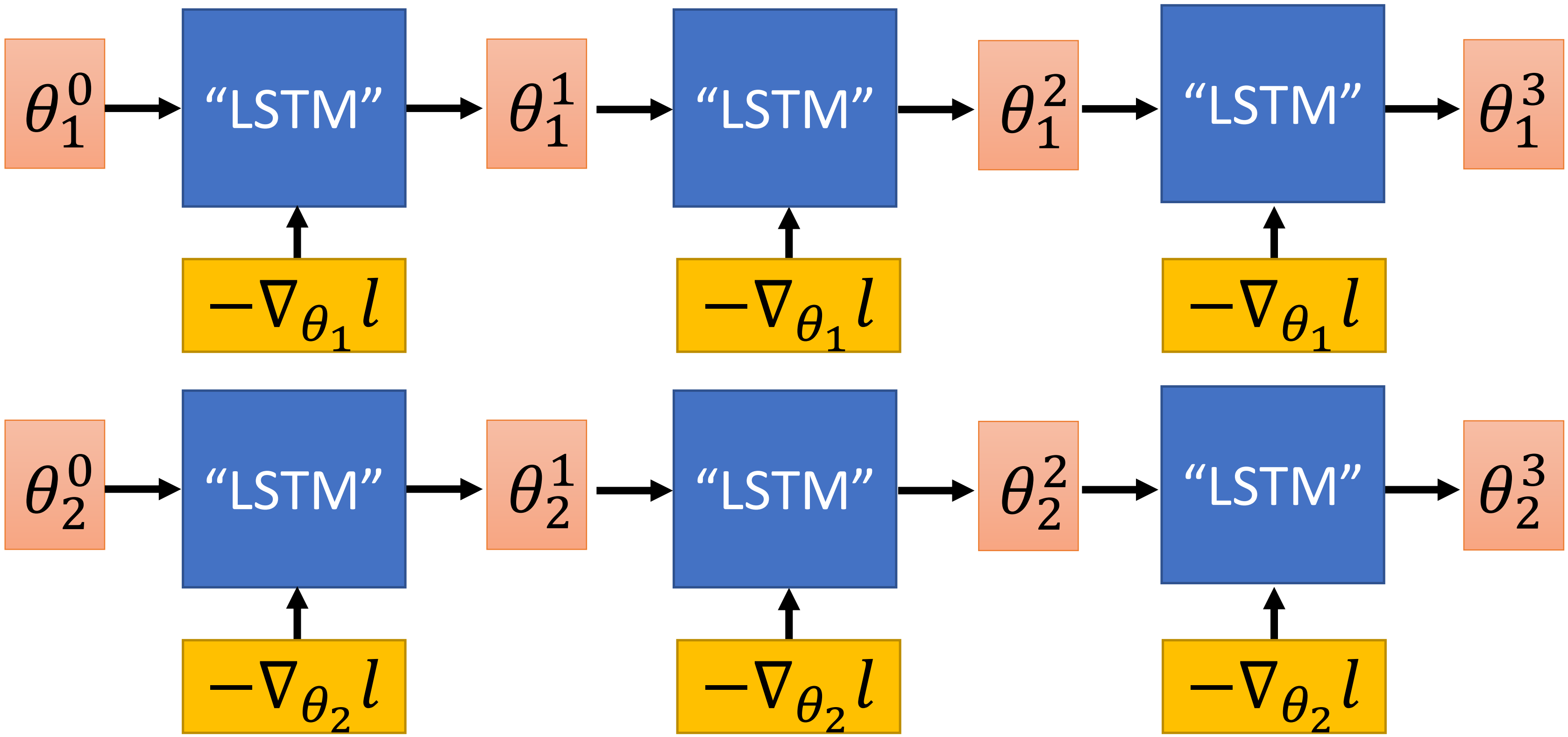
$$\theta^t = z^f \odot \theta^{t-1} + z^i \odot -\nabla_\theta l$$

Learn to minimize



Real Implementation

The LSTM used only has one cell. Share across all parameters

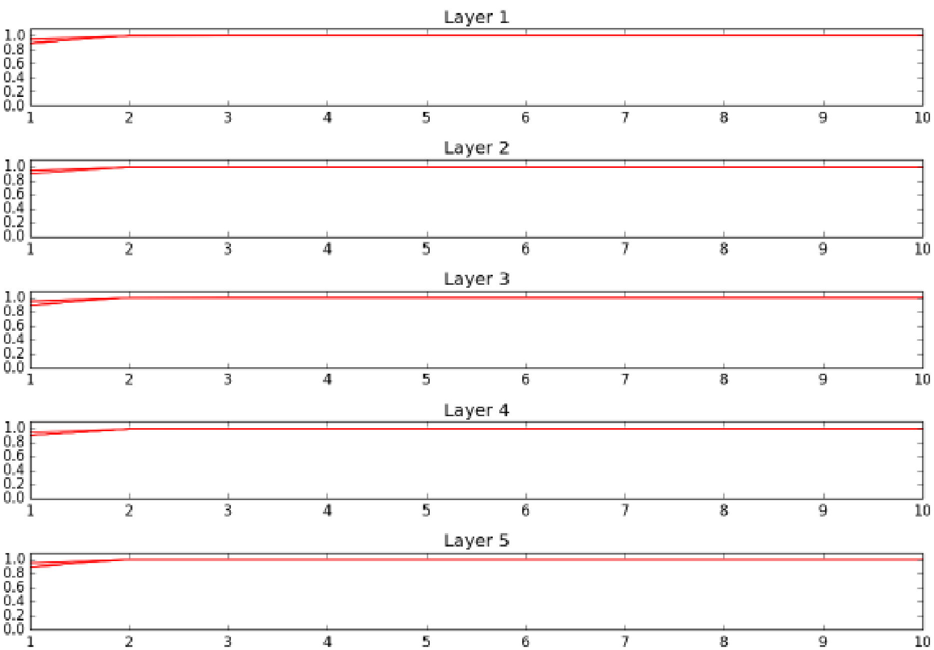


- Reasonable model size
- In typical gradient descent, all the parameters use the same update rule
- Training and testing model architectures can be different.

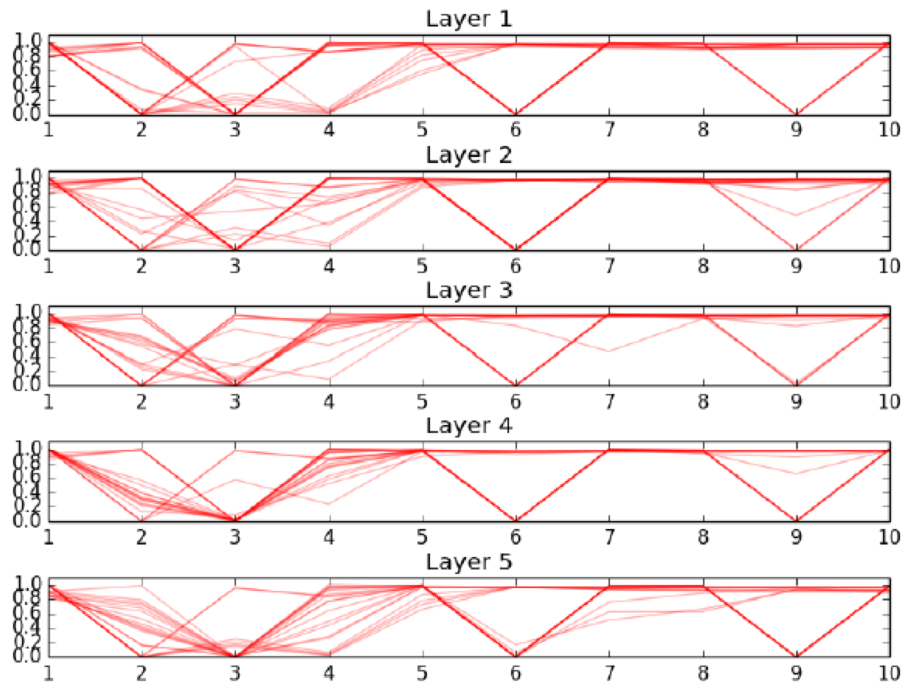
Experimental Results

https://openreview.net/forum?id=rJY0-Kcll&noteId=ryq49XyLg

$$\theta^t = z^f \odot \theta^{t-1} + z^i \odot -\nabla_\theta l$$

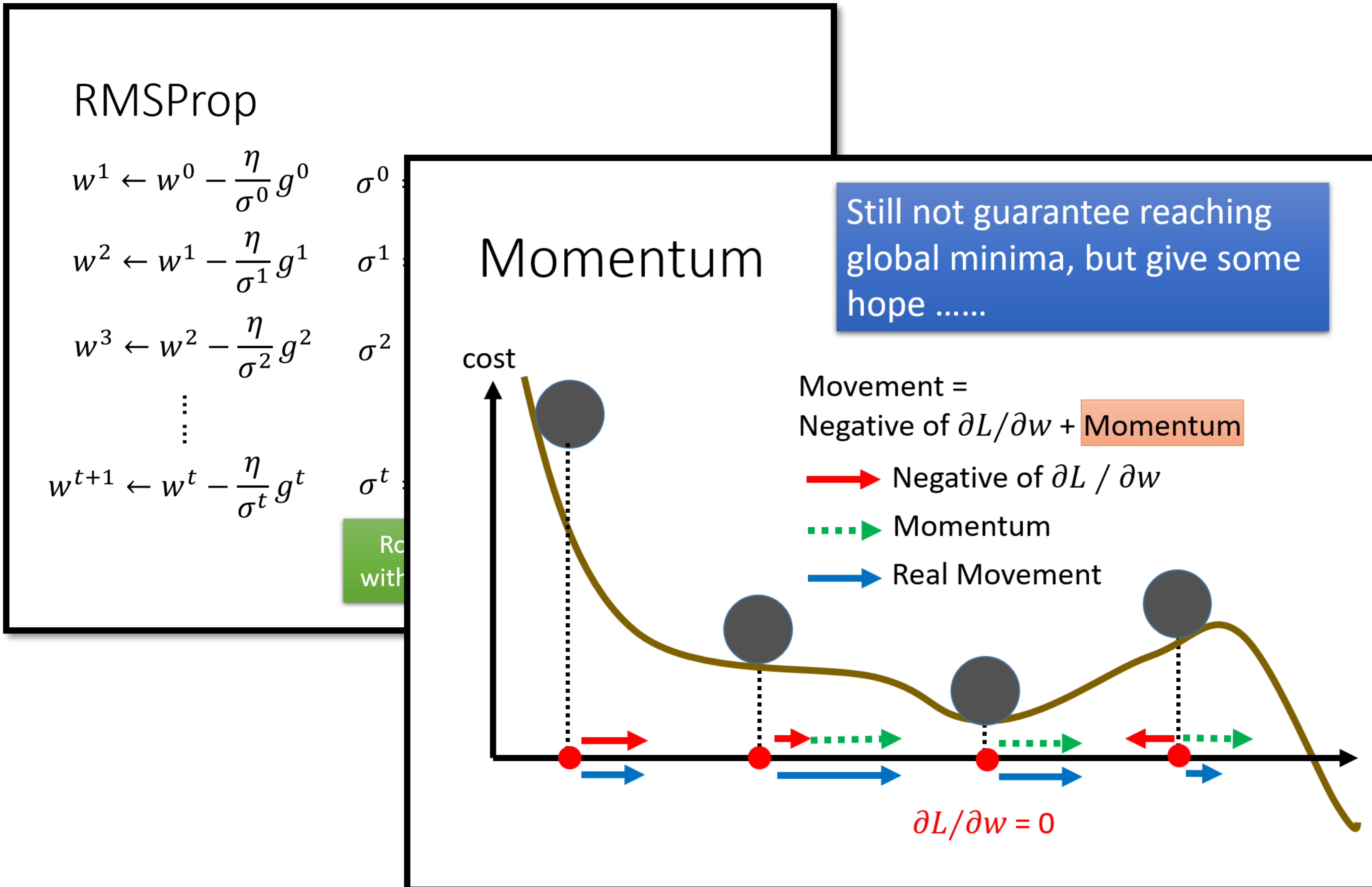


(a) Forget gate values for 1-shot meta-learner



(b) Input gate values for 1-shot meta-learner

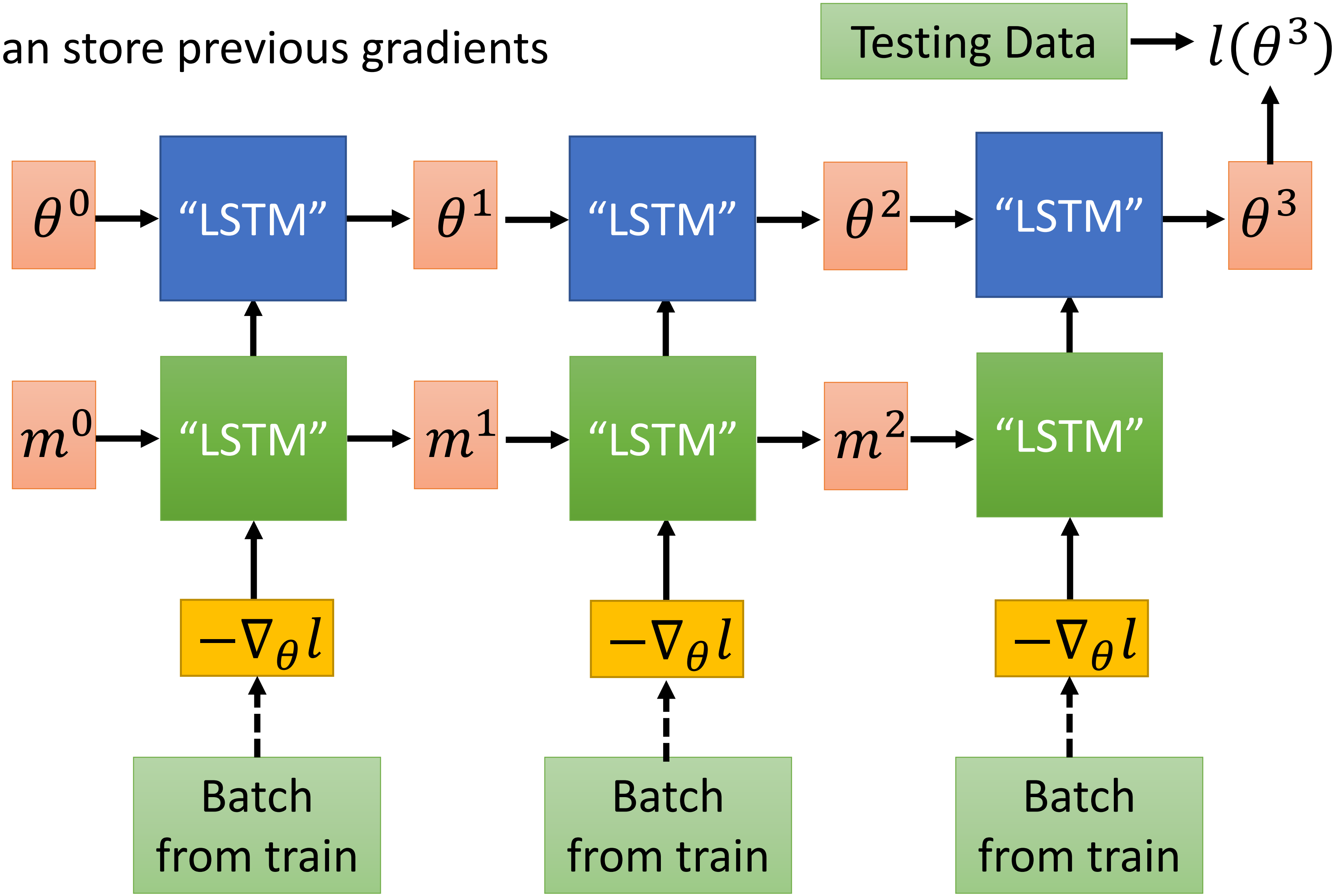
Parameter update depends on not only current gradient, but ***previous gradients***.



LSTM for Gradient Descent (v2)

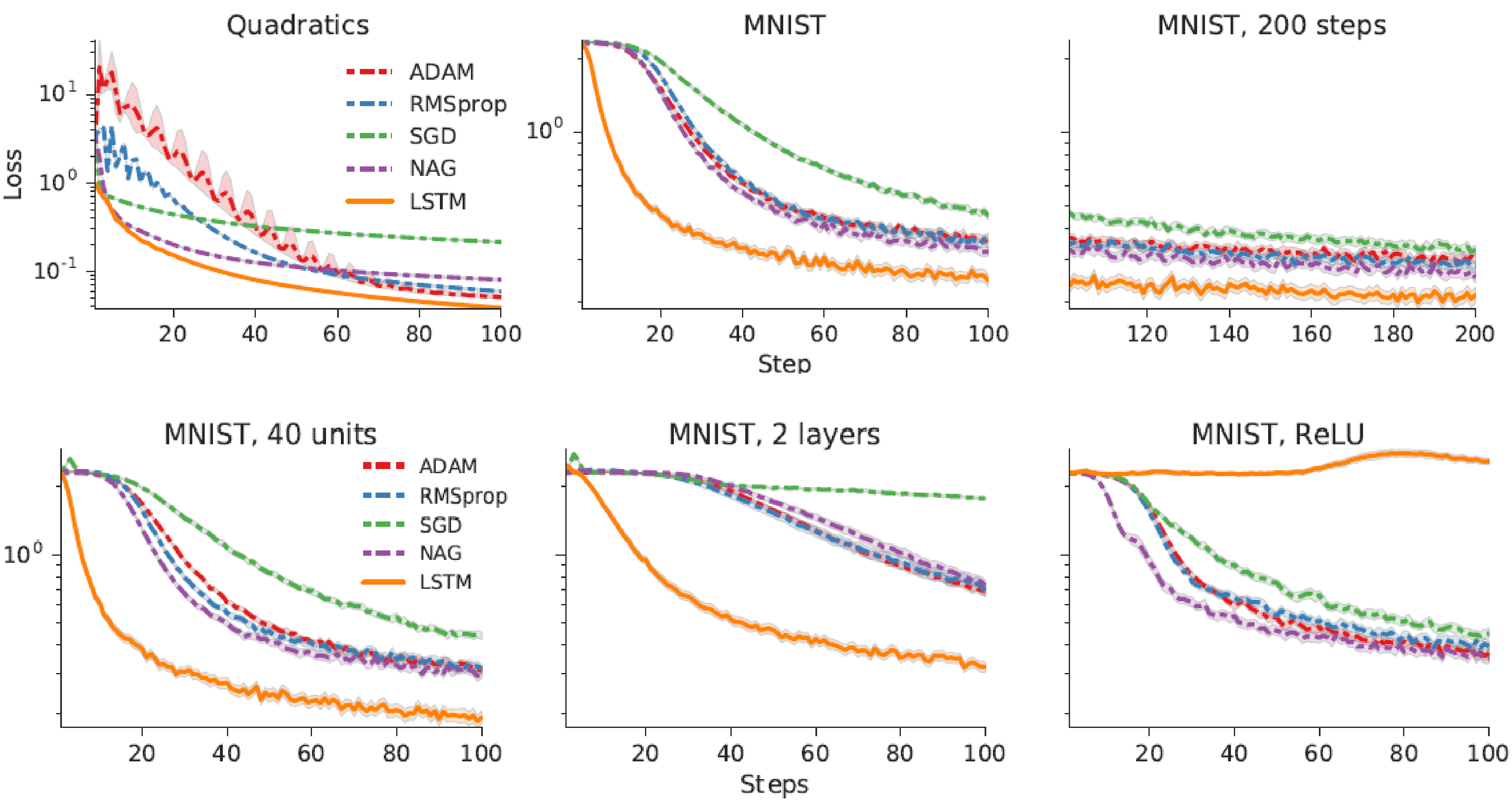
3 training steps

m can store previous gradients



https://arxiv.org/abs/1606.04474

Experimental Results

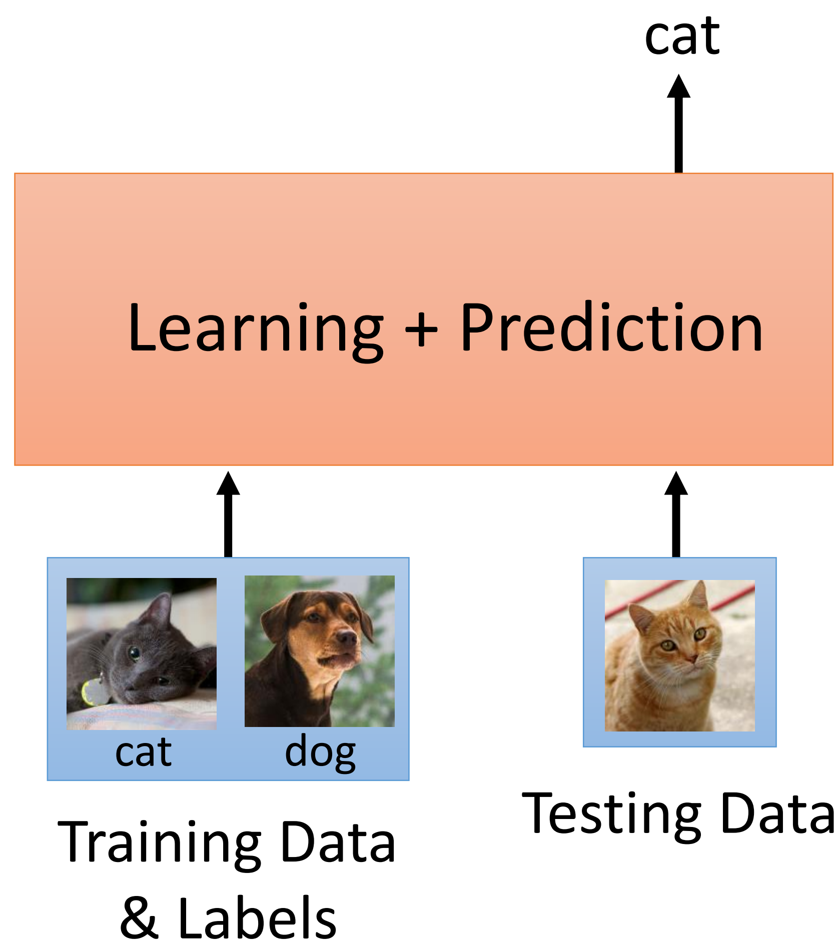


Meta Learning (Part 3)

Hung-yi Lee

Even more crazy idea …

- Input:
  - Training data and their labels
  - Testing data
- Output:
  - Predicted label of testing data



Face Verification

In each task:

Training

Few-shot Learning

Registration (Collecting Training data)

Testing

Unlock your phone by Face

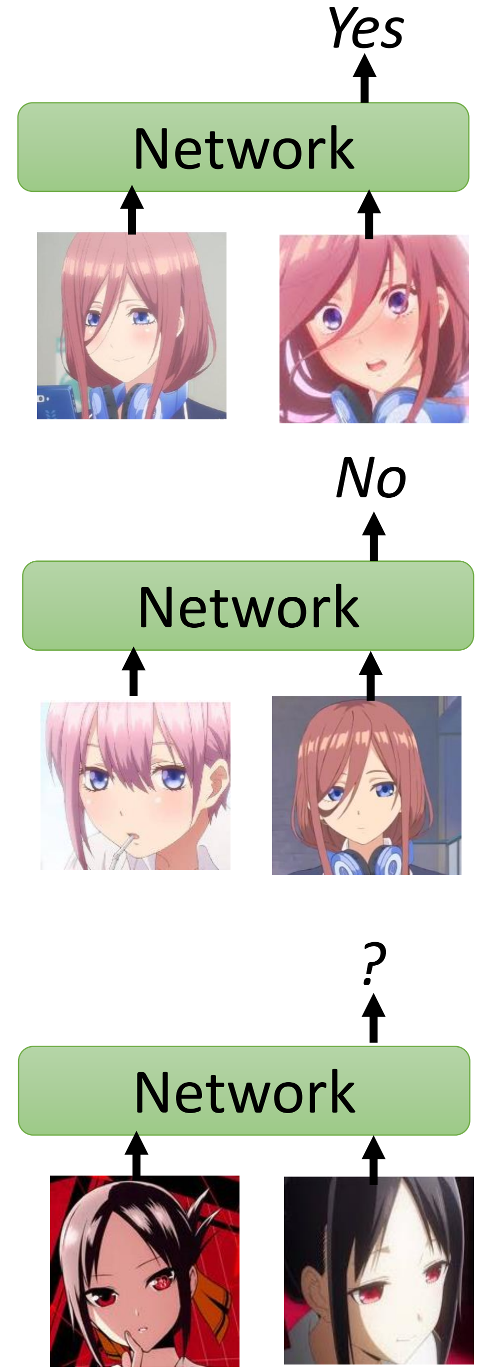
https://support.apple.com/zh-tw/HT208109

Meta Learning

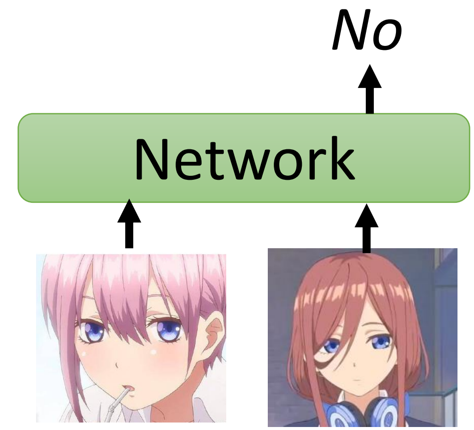
Same approach for Speaker Verification

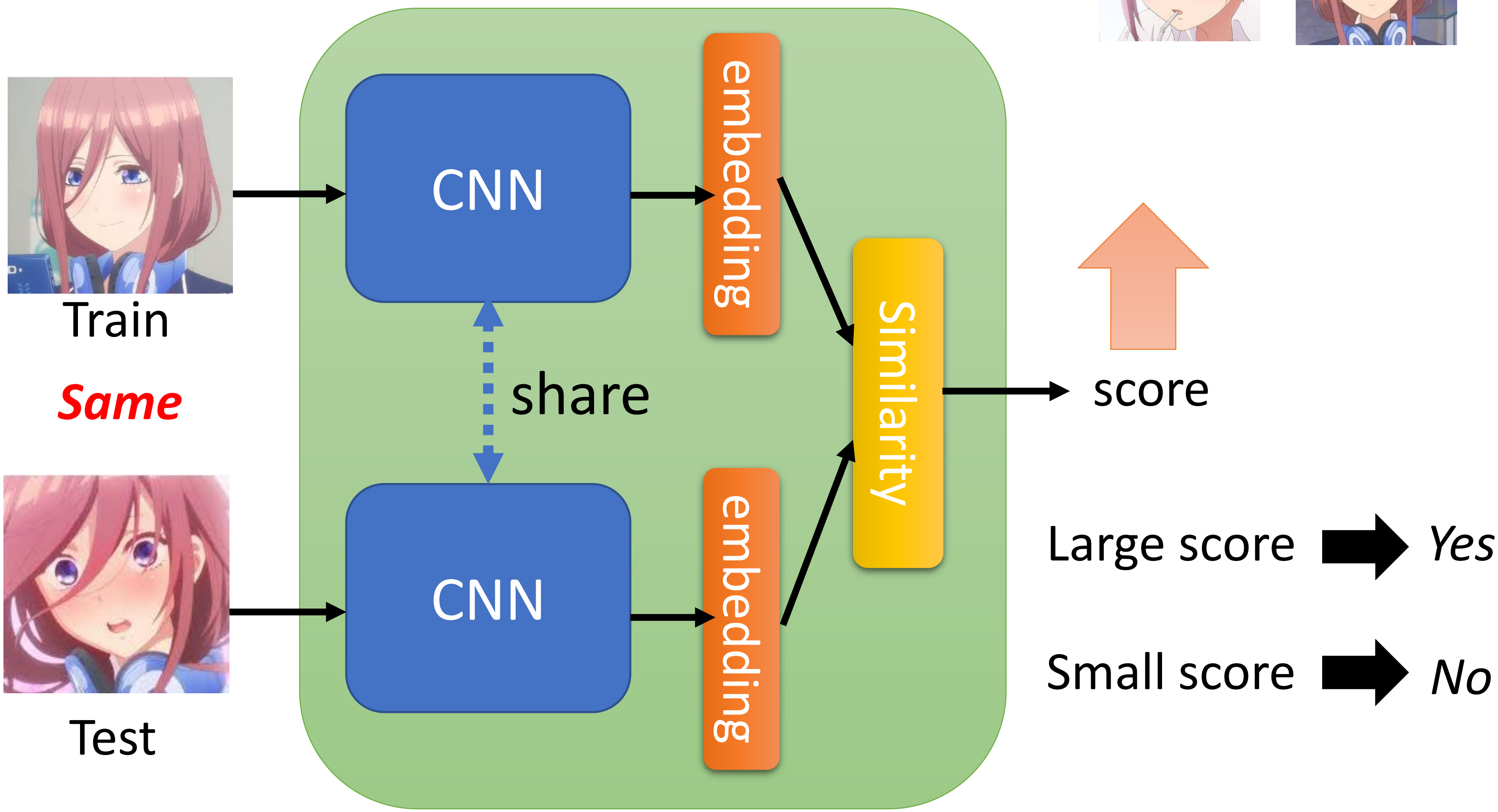
Training Tasks

Testing Tasks

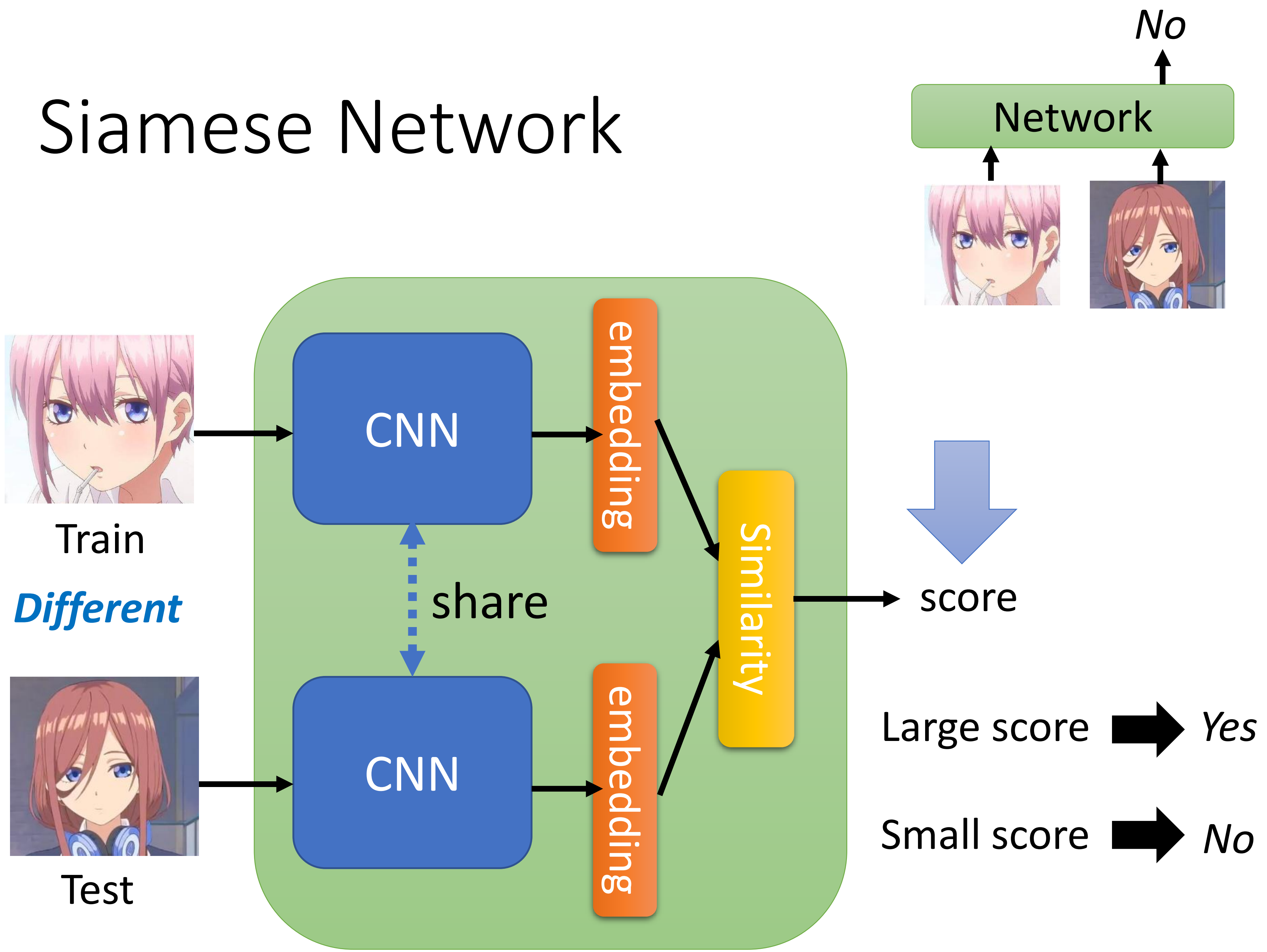


Siamese Network



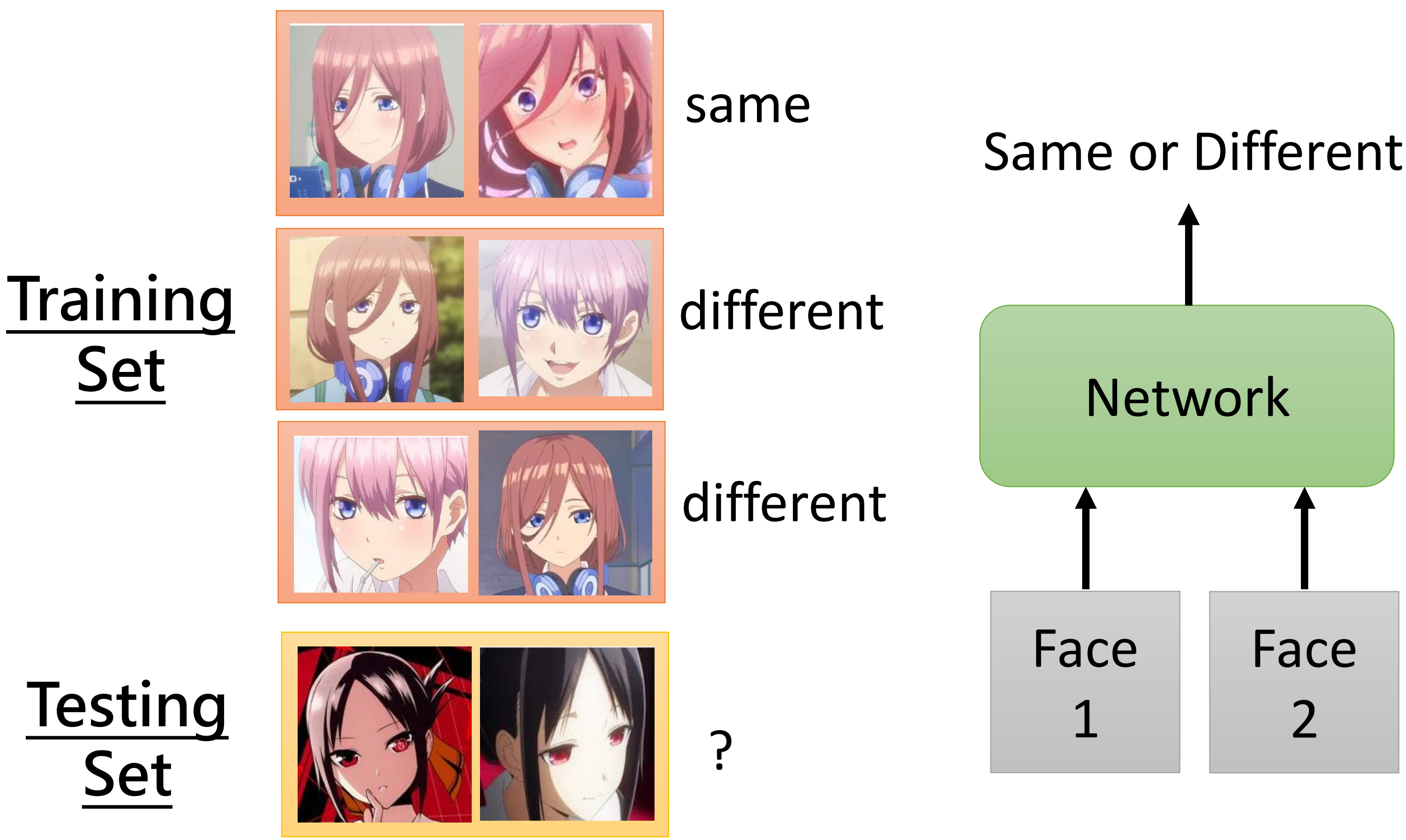


Siamese Network



Siamese Network
- Intuitive Explanation

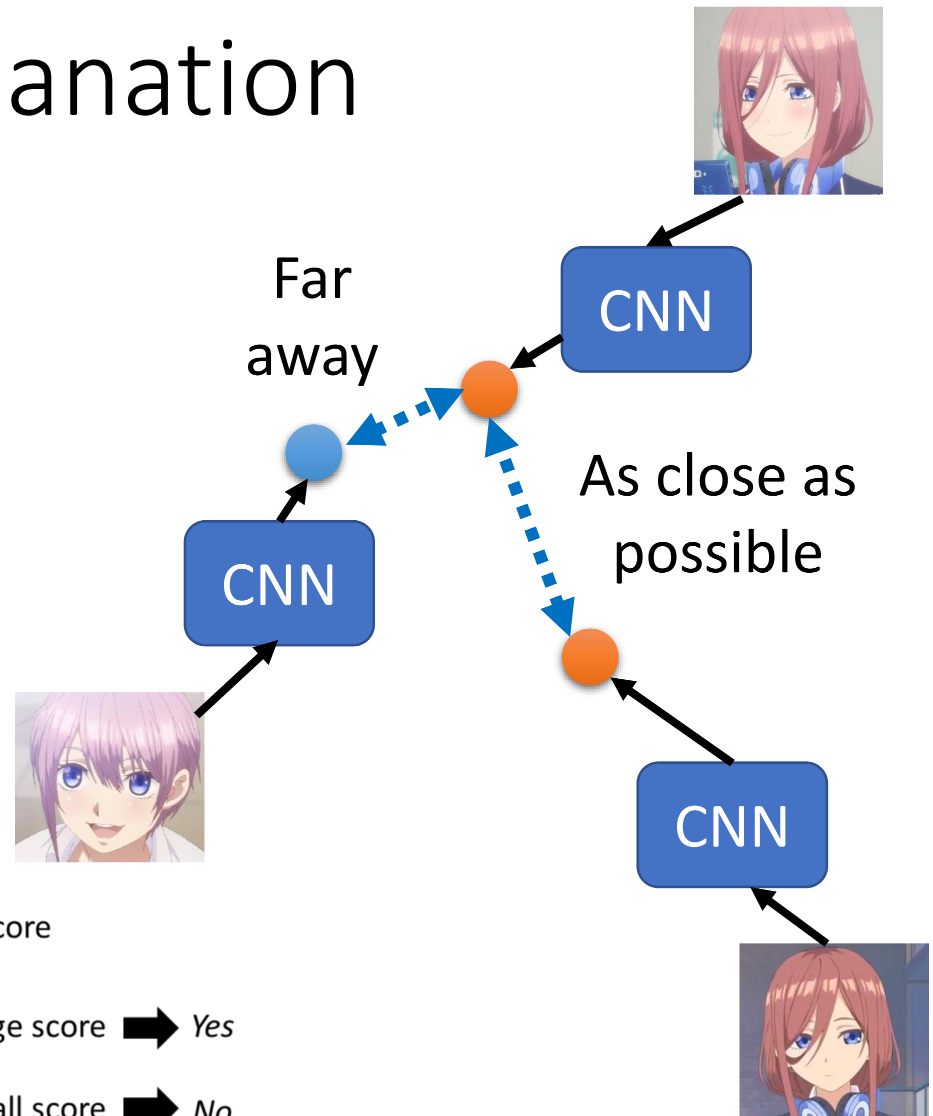
- Binary classification problem: “Are they the same?”

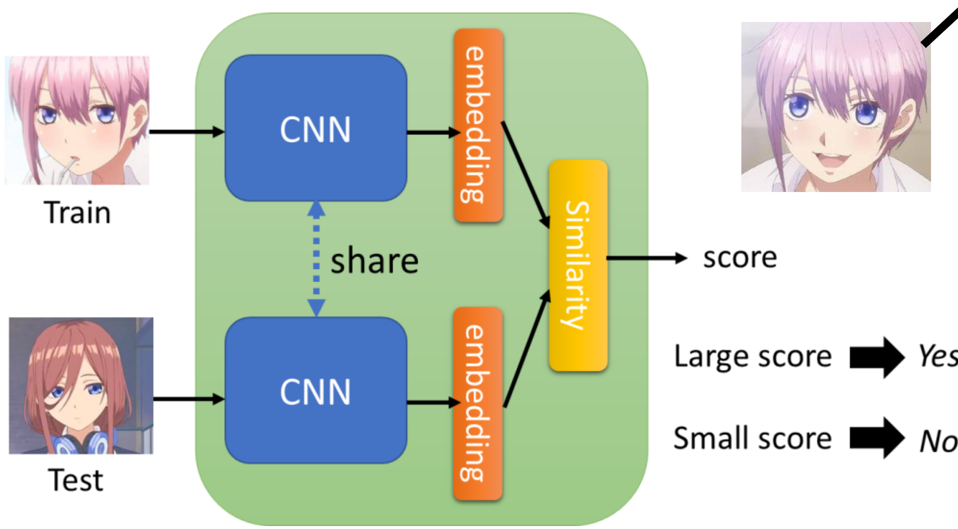


Siamese Network
- Intuitive Explanation

Learning embedding for faces

e.g. learn to ignore the background



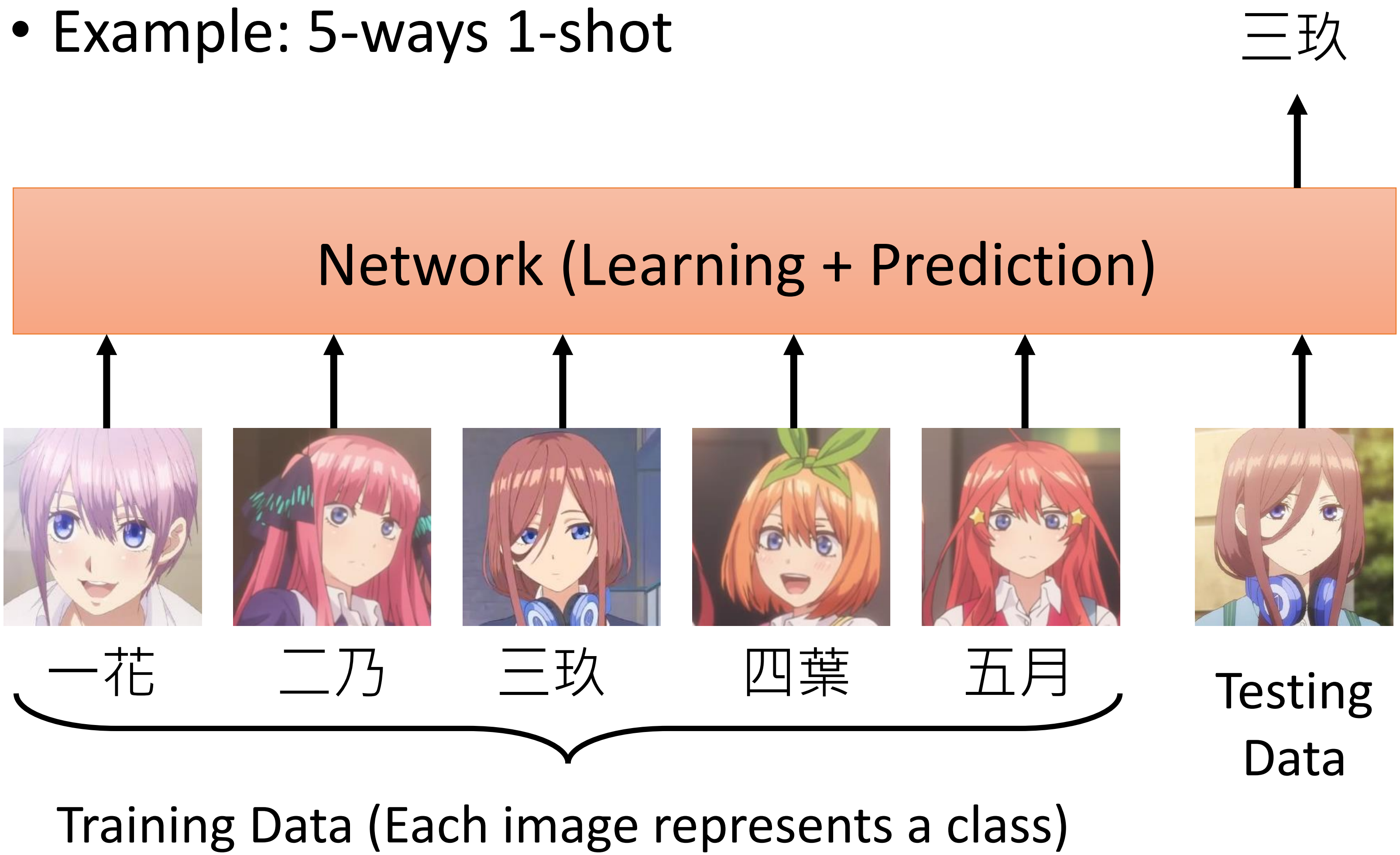


To learn more …

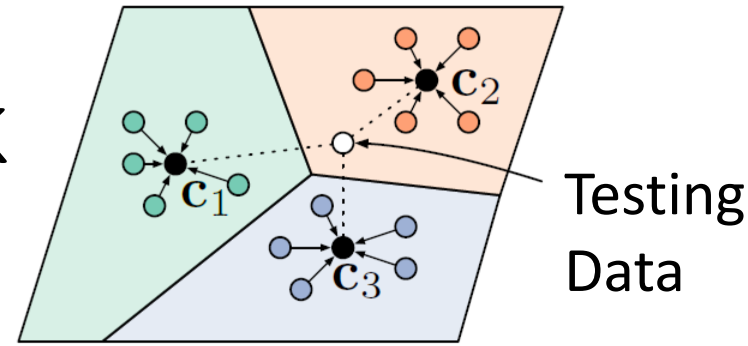
- What kind of distance should we use?
  - SphereFace: Deep Hypersphere Embedding for Face Recognition
  - Additive Margin Softmax for Face Verification
  - ArcFace: Additive Angular Margin Loss for Deep Face Recognition
- Triplet loss
  - Deep Metric Learning using Triplet Network
  - FaceNet: A Unified Embedding for Face Recognition and Clustering

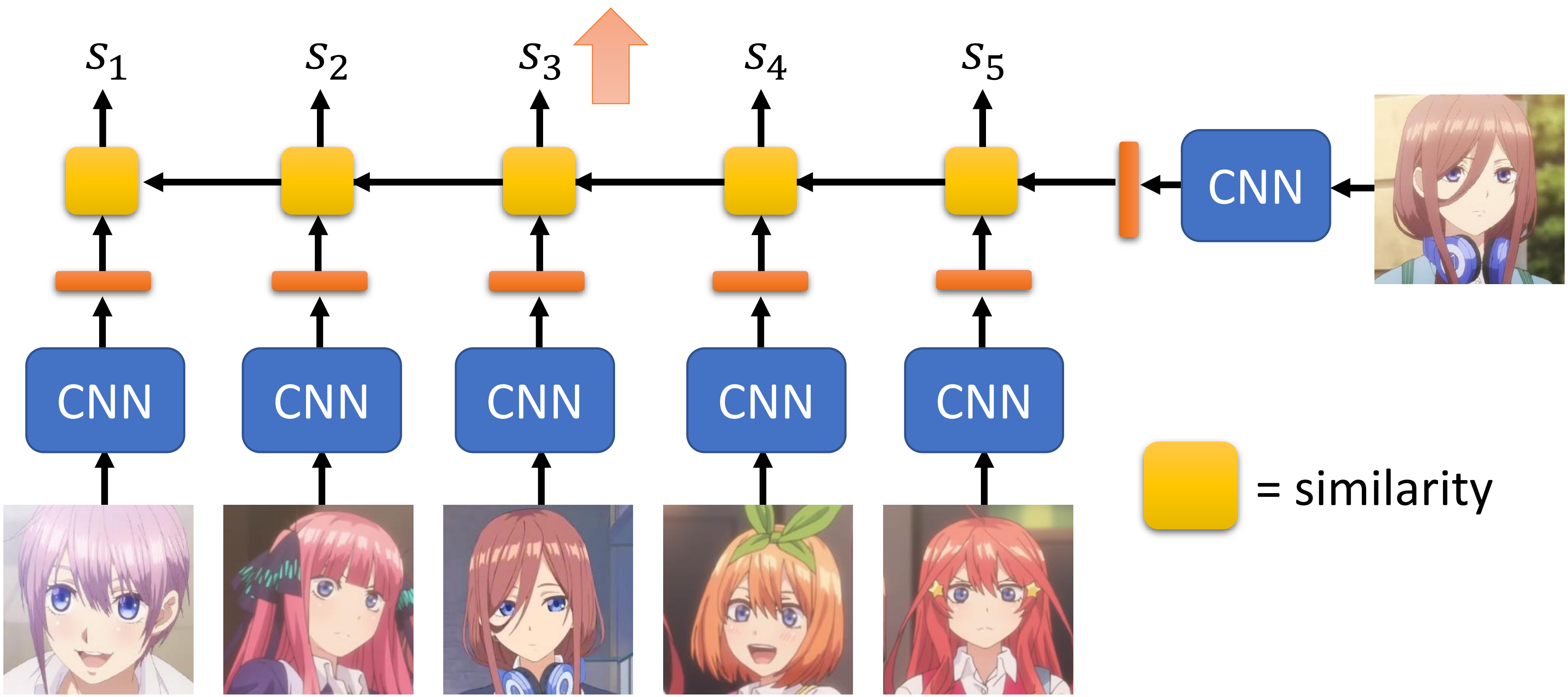
N-way Few/One-shot Learning

- Example: 5-ways 1-shot



Prototypical Network

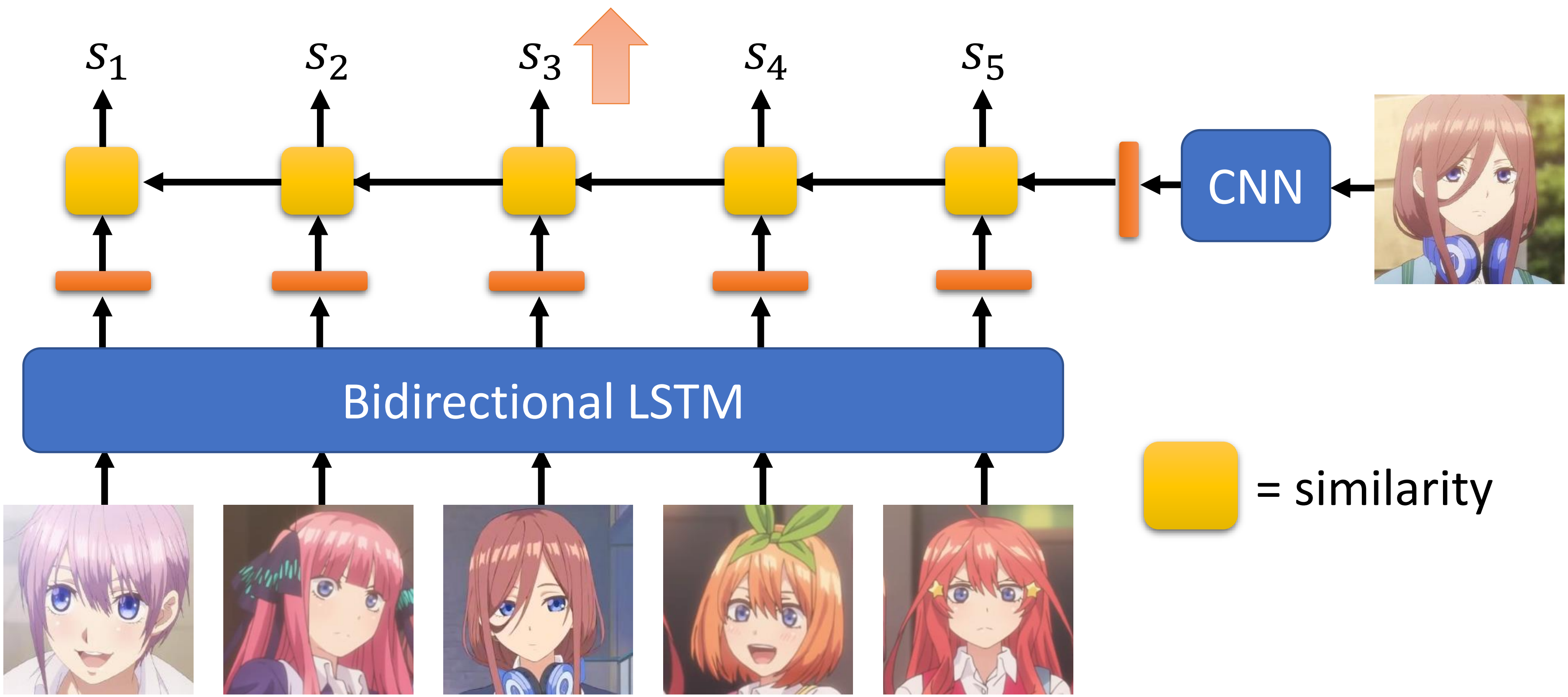




https://arxiv.org/abs/1703.05175

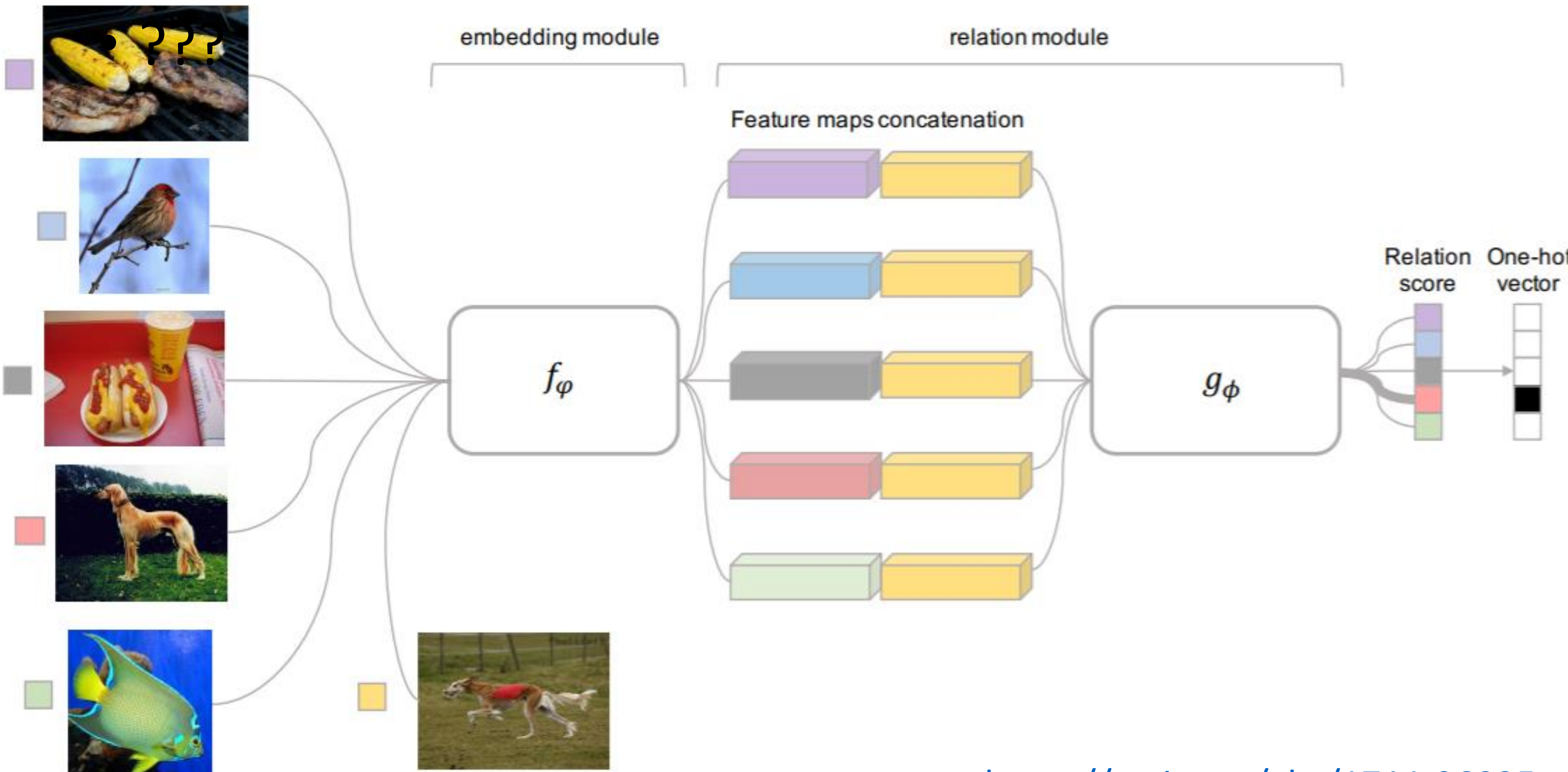
Matching Network

Considering the relationship among the training examples



https://arxiv.org/abs/1606.04080

Relation Network

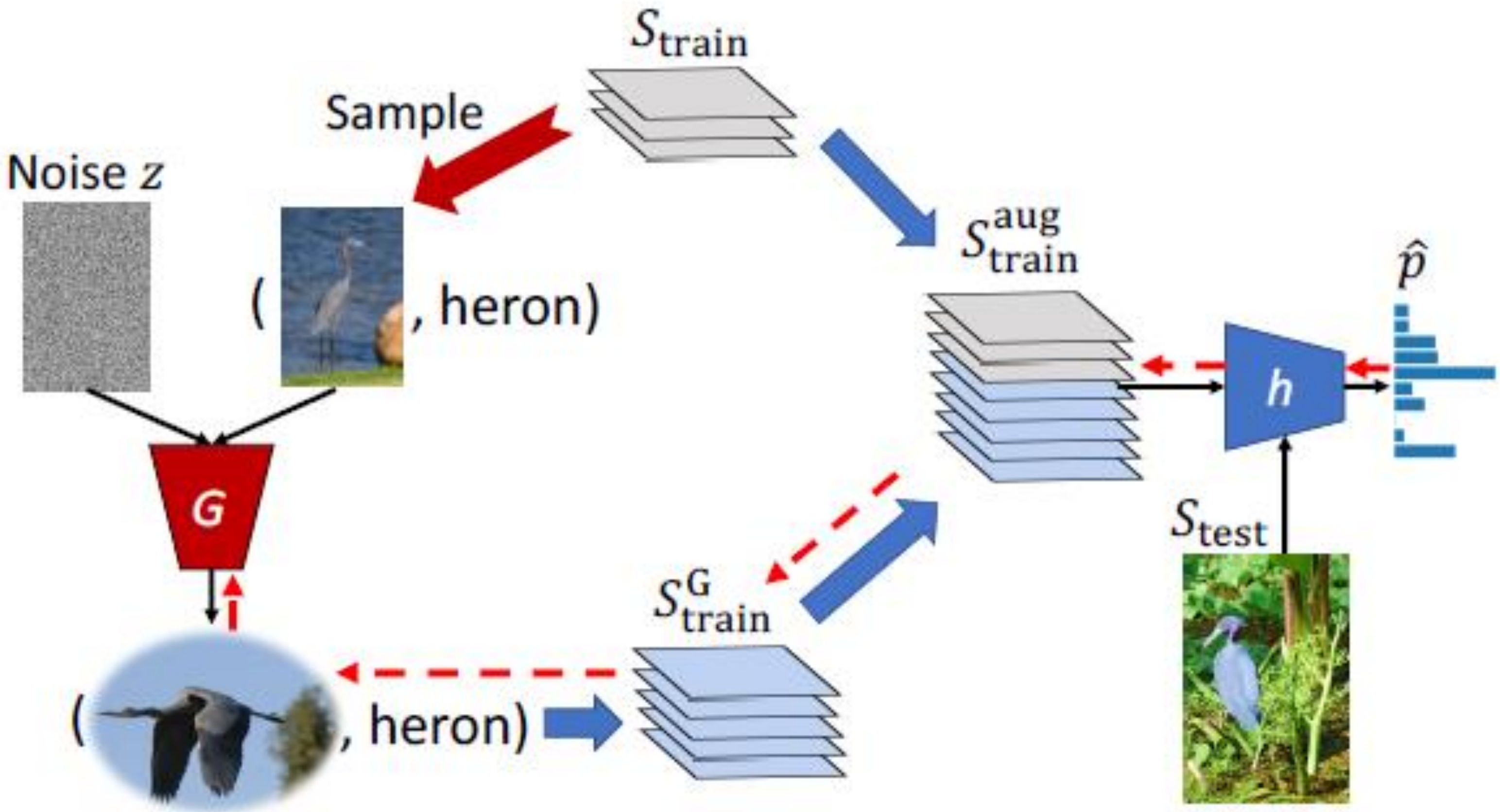


https://arxiv.org/abs/1711.06025

Few-shot learning for imaginary data

https://arxiv.org/abs/1801.05401

Few-shot learning for imaginary data



https://arxiv.org/abs/1801.05401