Hyperparamter search and meta learning

Szu-Chi Chung

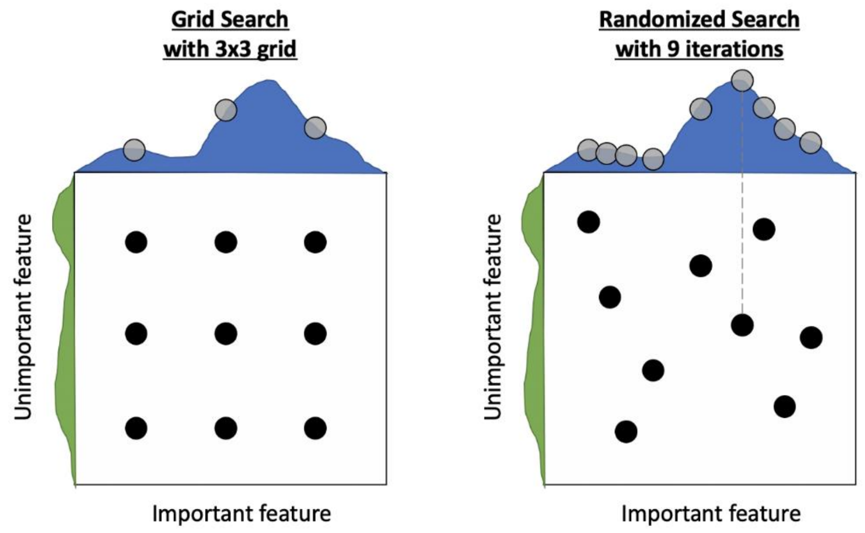
Department of Applied Mathematics, National Sun Yat-sen University

Getting the most out of your models

- You’ve come far since the beginning of this course
  - You can now train image classification models, timeseries forecasting models, text-classification models, and even generative models for images
  - The flexibility of neural networks, however, is also one of their main drawbacks: there are many hyperparameters to tweak
    - How many layers should you stack?
    - How many units or filters should go in each layer?
    - Should you use ReLU as activation, or a different function?
    - Should you use BatchNormalization after a given layer?
    - How much dropout should you use?
  - These architecture-level parameters are called *hyperparameters* to distinguish them from the parameters of a model, which are trained via backpropagation

1. Hyperparamter search

- One option is to simply try many combinations of hyperparameters and see which one works best on the validation set (or use K-fold cross-validation)
  - We can use GridSearchCV or RandomizedSearchCV to explore the hyperparameter space
  - When training is slow, however (e.g., for more complex problems with larger datasets), this approach will only explore a tiny portion of the hyperparameter space

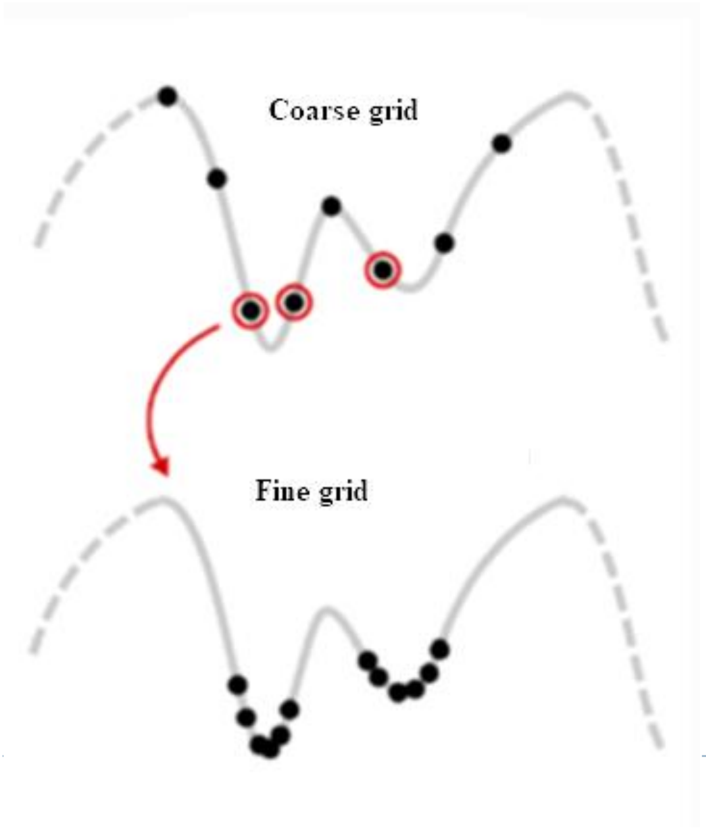


$\times 1{,}000$

https://towardsdatascience.com/gridsearch-vs-randomizedsearch-vs-bayesiansearch-cfa76de27c6b

Hyperparamter search

- You can alleviate this problem by assisting the search process manually
  - First run a quick random search using wide ranges of hyperparameter values, then run another search using smaller ranges of values centered on the best ones found during the first run, and so on. This approach will hopefully zoom in on a good set of hyperparameters
  - The core idea of most of the advance algorithm is simple: when a region of the space turns out to be good, it should be explored more. Such techniques take care of the "zooming" process for you and lead to much better solutions in much less time
  - There are plenty of hyperparameter search algorithms
    - Bayesian optimization, evolutionary optimization, early stopping-based (Hyperband) …
    - However, most of them are no longer embarrassing parallel



Why automatically hyperparamter search?

- In practice, experienced machine learning engineers build intuition over time as to what works and what doesn’t when it comes to hyperparamter search
  - But it shouldn’t be your job as a human to fiddle with hyperparameters all day—that is better left to a machine. Thus you need to explore the space of possible decisions automatically, systematically, in a principled way. The process of optimizing looks like this:

1. Choose a set of hyperparameters (Automatically)
2. Build the corresponding model
3. Fit it to your training data, and measure performance on the validation data
4. Choose the next set of hyperparameters (Automatically)
5. Repeat
6. Eventually, measure performance on a test data

Why automatically hyperparamter search?

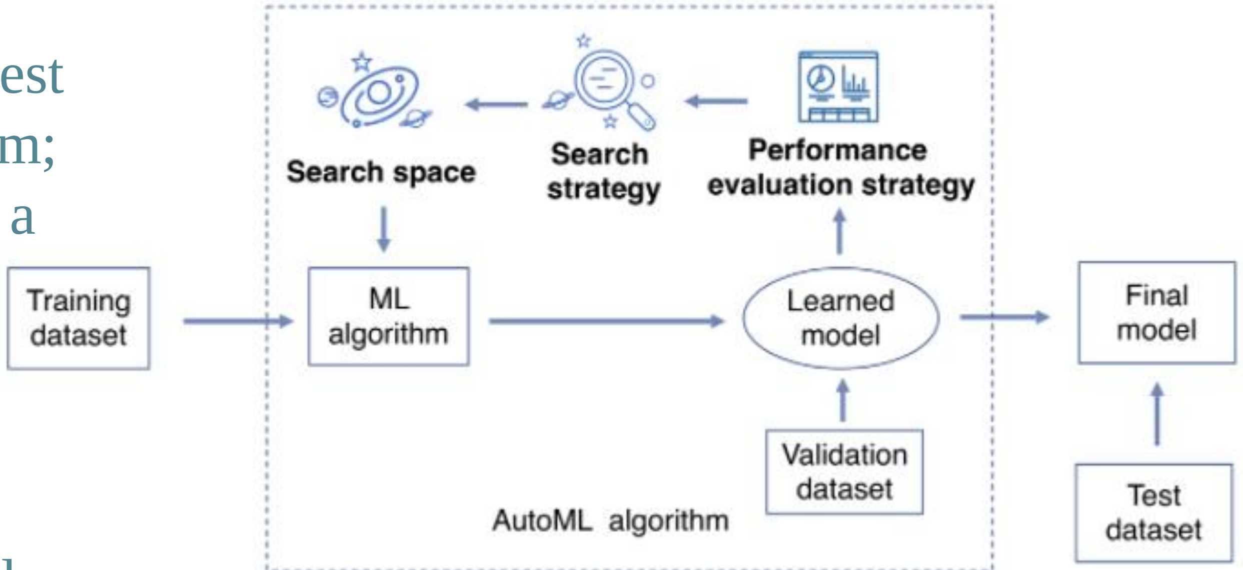
- The key to this process is various hyperparameter values to choose the next set of hyperparameters to evaluate. But it is challenging considering the fact that
    1. The hyperparameter space is typically made up of discrete decisions and thus isn't continuous or differentiable. Hence, you typically can't do gradient descent in hyperparameter space. Instead, you must rely on gradient-free optimization techniques, which naturally are far less efficient than gradient descent
    2. Computing the feedback signal of this optimization process (does this set of hyperparameters lead to a high-performing model on this task?) can be extremely expensive: it requires creating and training a new model from scratch on your dataset
    3. The feedback signal may be noisy: if a training run performs 0.2% better, is that because of a better model configuration, or because you got lucky with the initial weight values?

Why automatically hyperparamter search?

- Overall, hyperparameter optimization is a powerful technique that is an absolute requirement for getting to state-of-the-art models on any task
  - Think about it: once upon a time, people handcrafted the features that went into shallow machine learning models. That was very much suboptimal. Now, deep learning automates the task of hierarchical feature engineering—features are learned using a feedback signal, not hand-tuned, and that’s the way it should be
  - In the same way, you shouldn’t handcraft your model architectures; you should optimize them in a principled way
  - However, doing hyperparameter tuning is not a replacement for being familiar with model architecture best practices. You need to be smart about designing the right search space. Hyperparameter tuning is automation, not magic: you use it to automate experiments that you would otherwise have run by hand, but you still need to handpick experiment configurations that have the potential to yield good metrics

Automatically hyperparamter search to AutoML

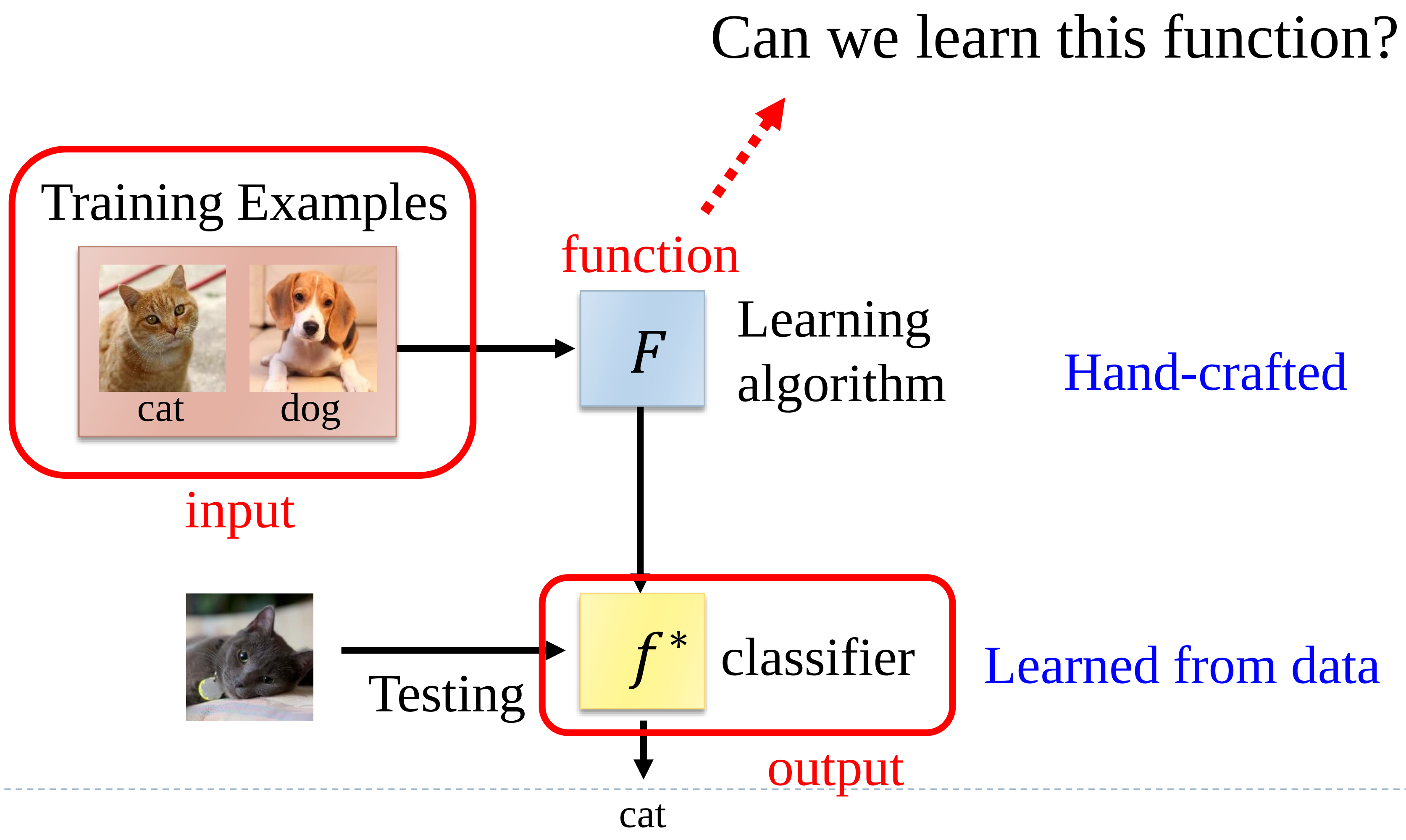
- We can also be far more ambitious and attempt to generate the model architecture itself from scratch, with as few constraints as possible, such as via reinforcement learning or evolutionary algorithms
  - For example, Google has used an evolutionary approach, not just to search for hyperparameters but also to look for the best neural network architecture for the problem; their AutoML suite is already available as a cloud service
  - In the future, entire end-to-end machine learning pipelines will be automatically generated, rather than be handcrafted by engineer-artisans. This is called automated machine learning, or AutoML



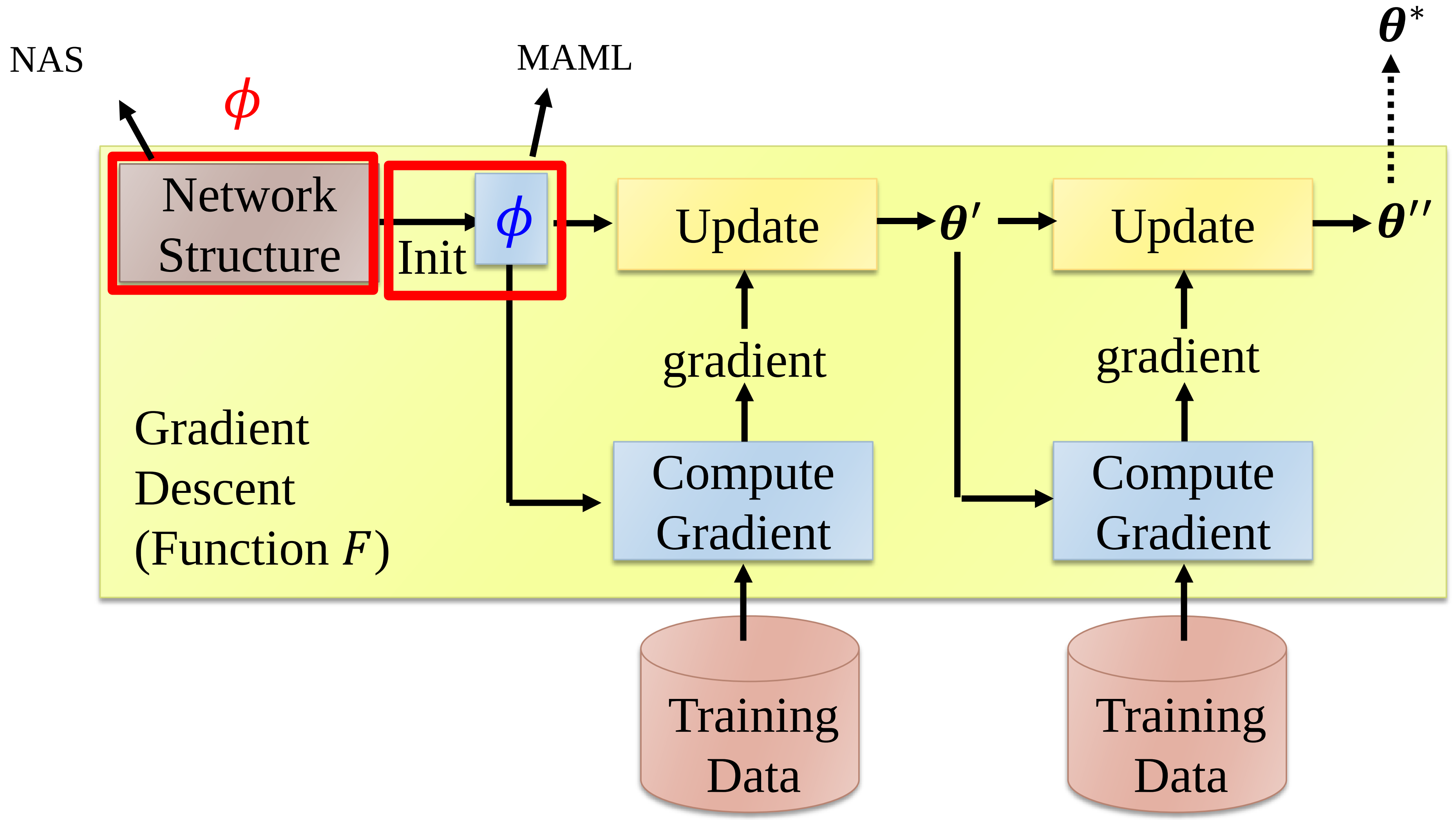
https://www.manning.com/books/automated-machine-learning-in-action

2. What is Meta Learning?

- Meta learning is one of the key component behind AutoML

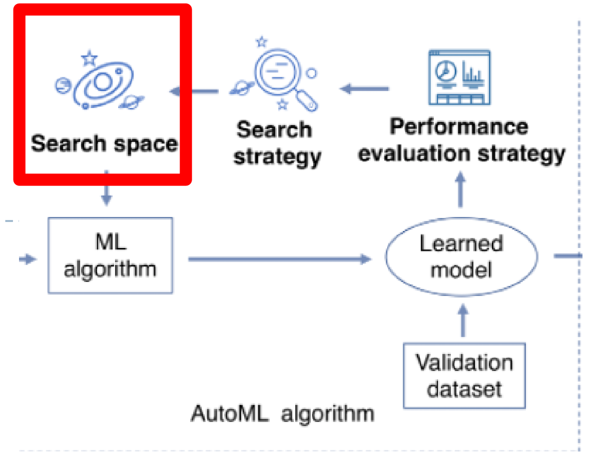


Review: Gradient descent

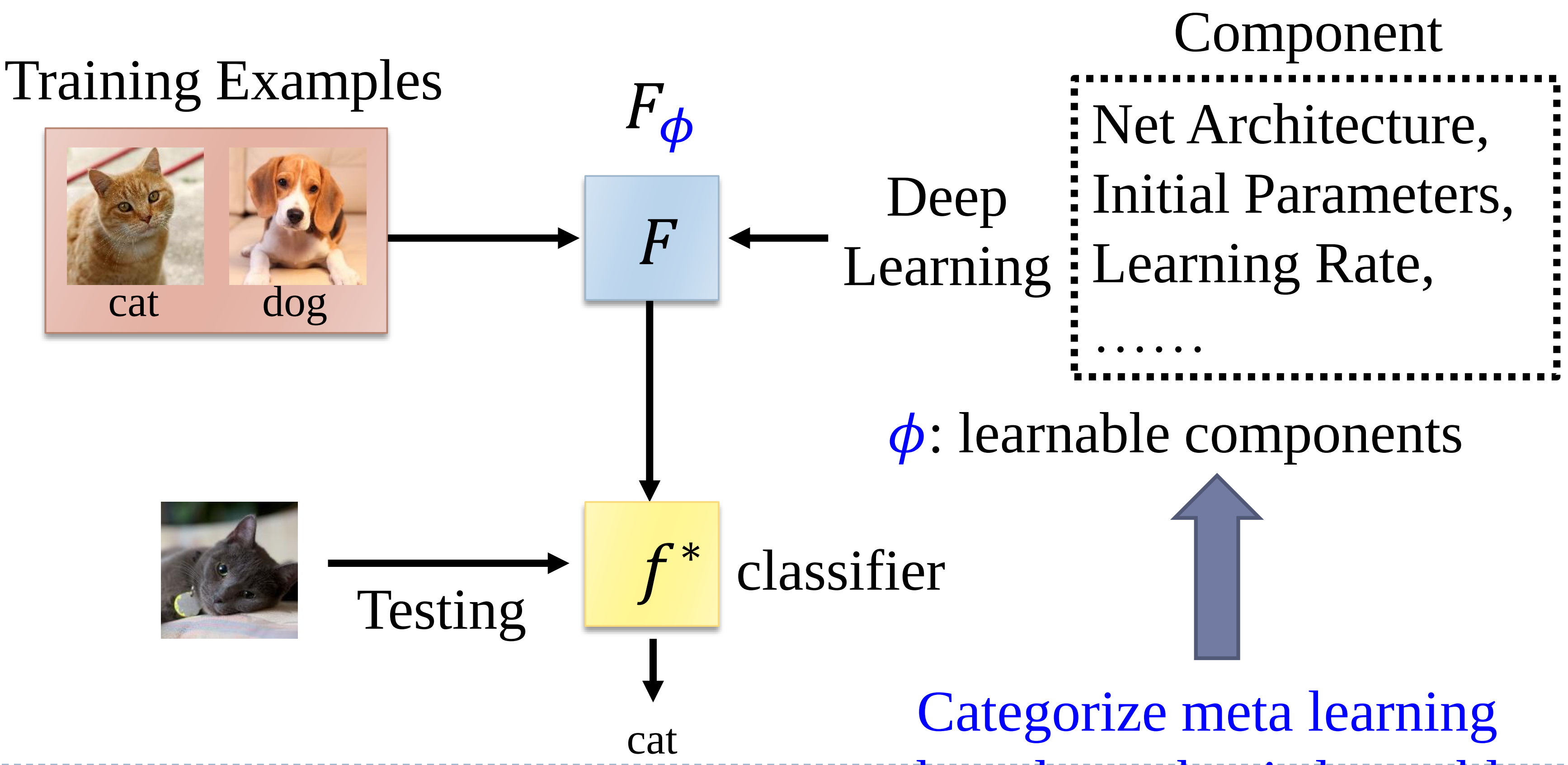


https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/meta_v3.pptx

Meta learning – Step 1

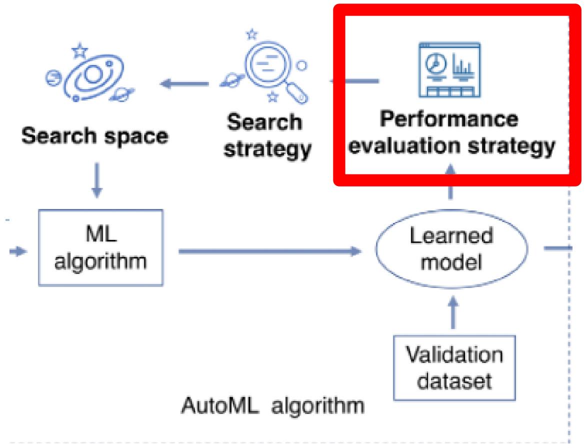


- What is ***learnable*** in a learning algorithm?



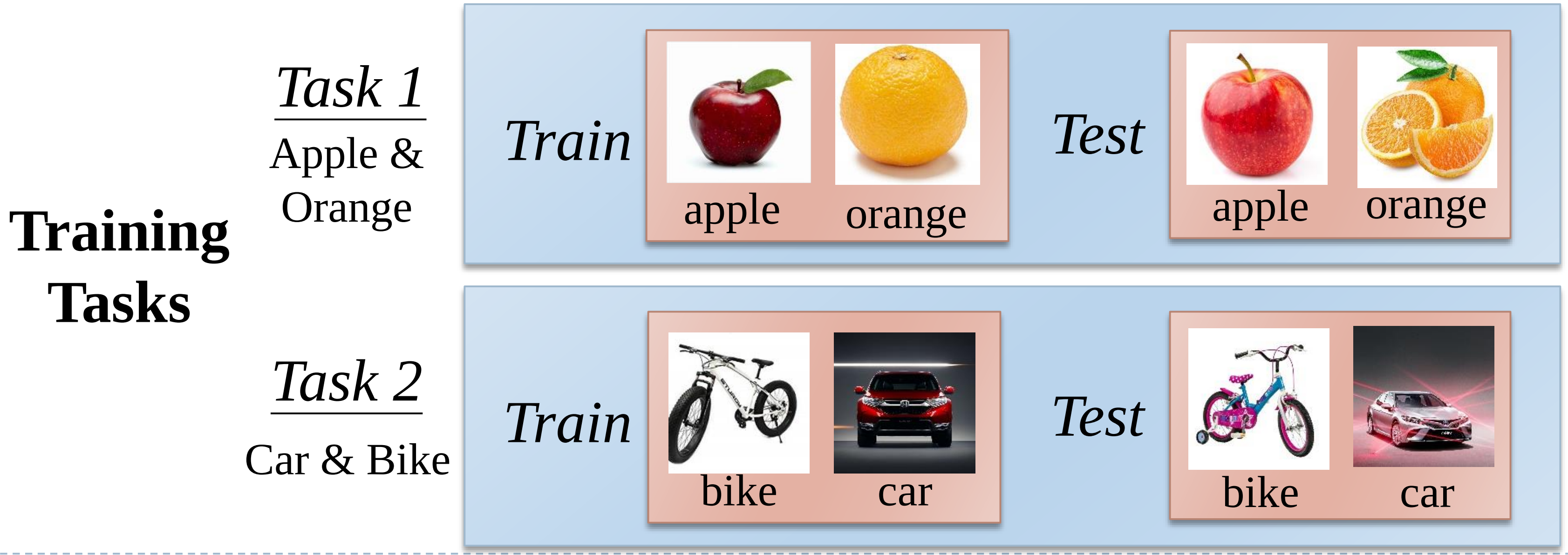
Categorize meta learning based on what is learnable

Meta learning – Step 2

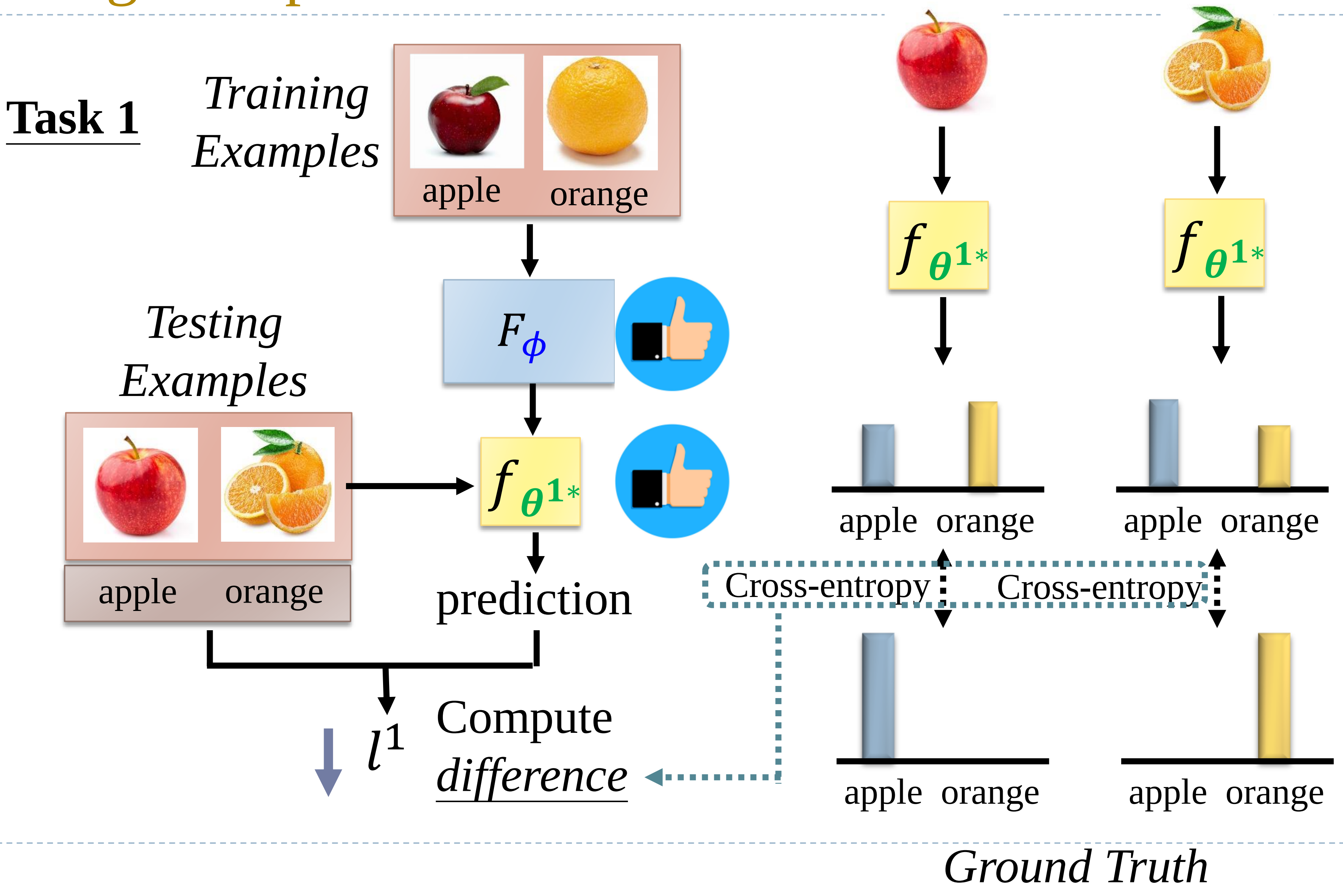


- Define ***loss function*** for ***learning algorithm*** F_ϕ
- Sample tasks from *training tasks* (Analog of training sample in supervised learning) $L(\phi)$

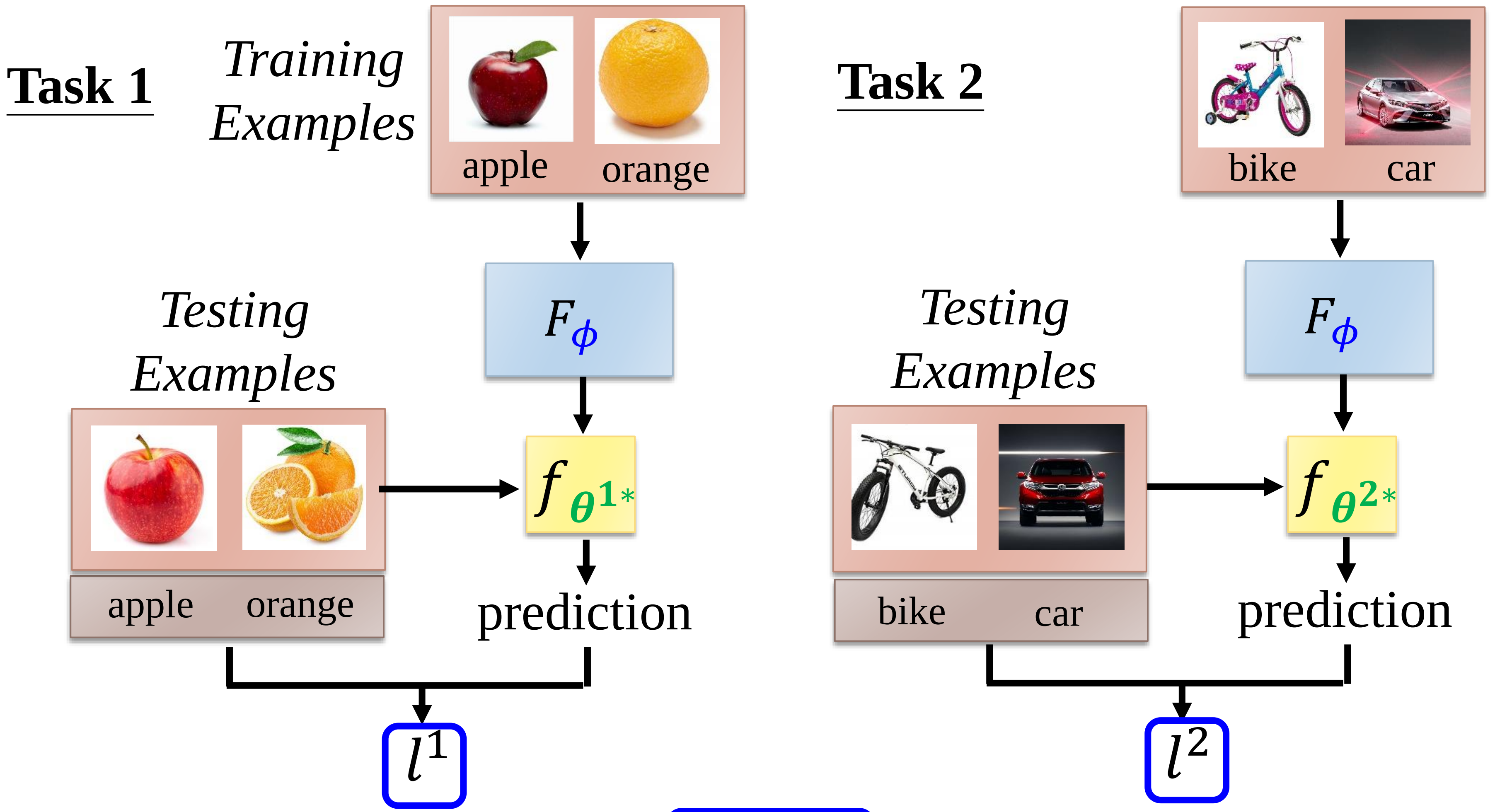
$L(\phi) \downarrow$



Meta learning – Step 2

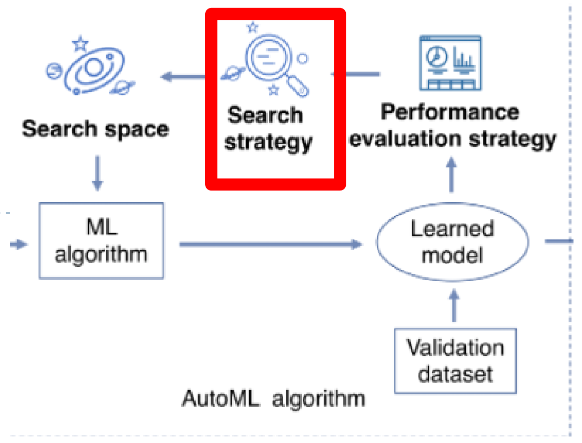


Meta learning – Step 2



Total loss: $L(\phi) = \boxed{l^1 + l^2}$ (sum over all the training tasks)

Meta learning – Step 3



- Loss function for learning algorithm where N is the number of training tasks we collect

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

- Find ϕ that can minimize $L(\phi)$

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

- Using the optimization approach you know

If you know how to compute $\partial L(\phi)/\partial\phi$

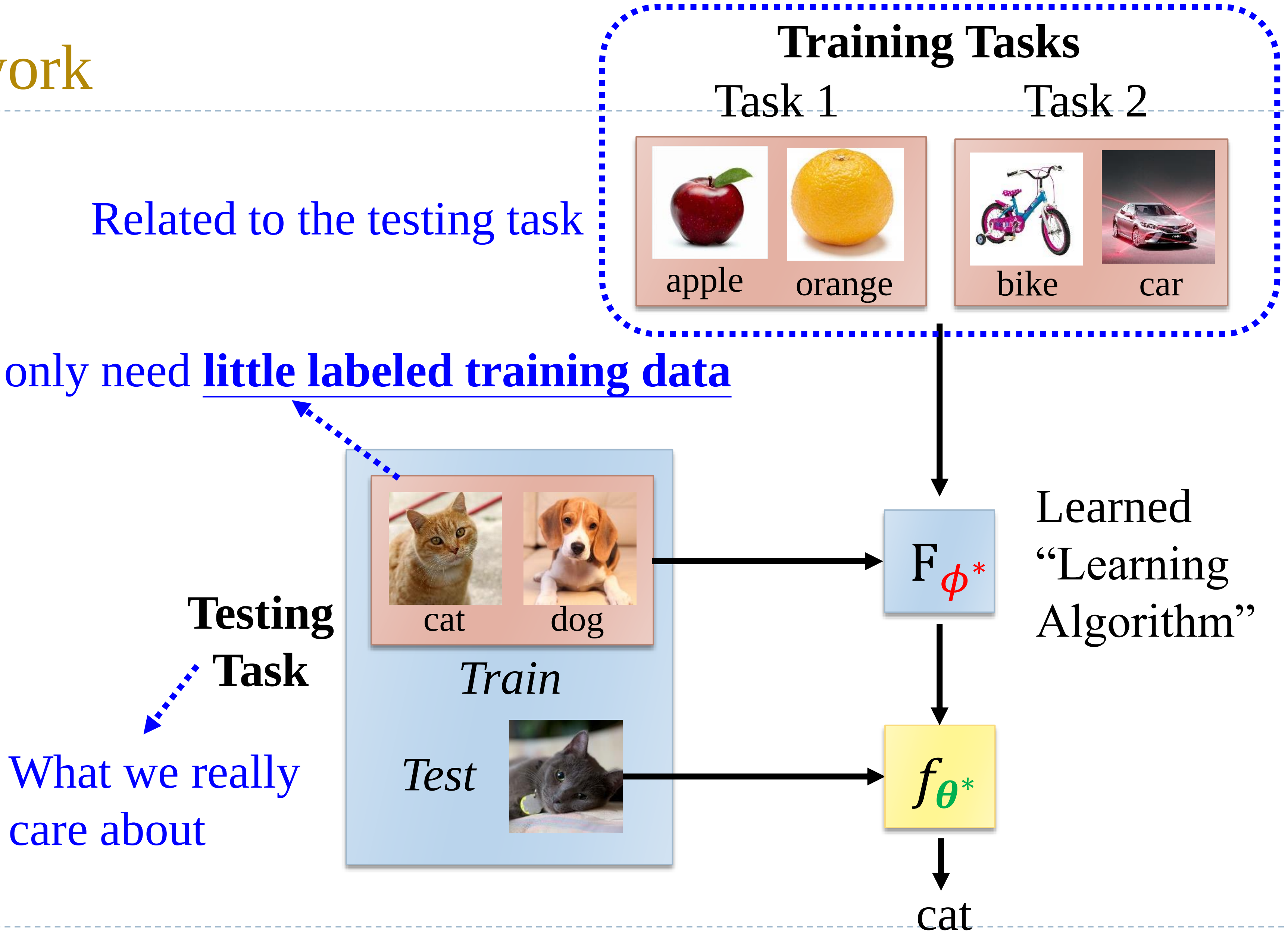
Gradient descent is your friend.

What if $L(\phi)$ is not differentiable?

Reinforcement Learning / Evolutionary Algorithm

Now we have a learned "learning algorithm" F_{ϕ^*}

Framework


Training Tasks
Task 1
Task 2
apple
orange
bike
car
Related to the testing task
only need **little labeled training data**
Testing Task
What we really care about
cat
dog
Train
Test
F_{ϕ^*}
Learned "Learning Algorithm"
f_{θ^*}
cat

Framework

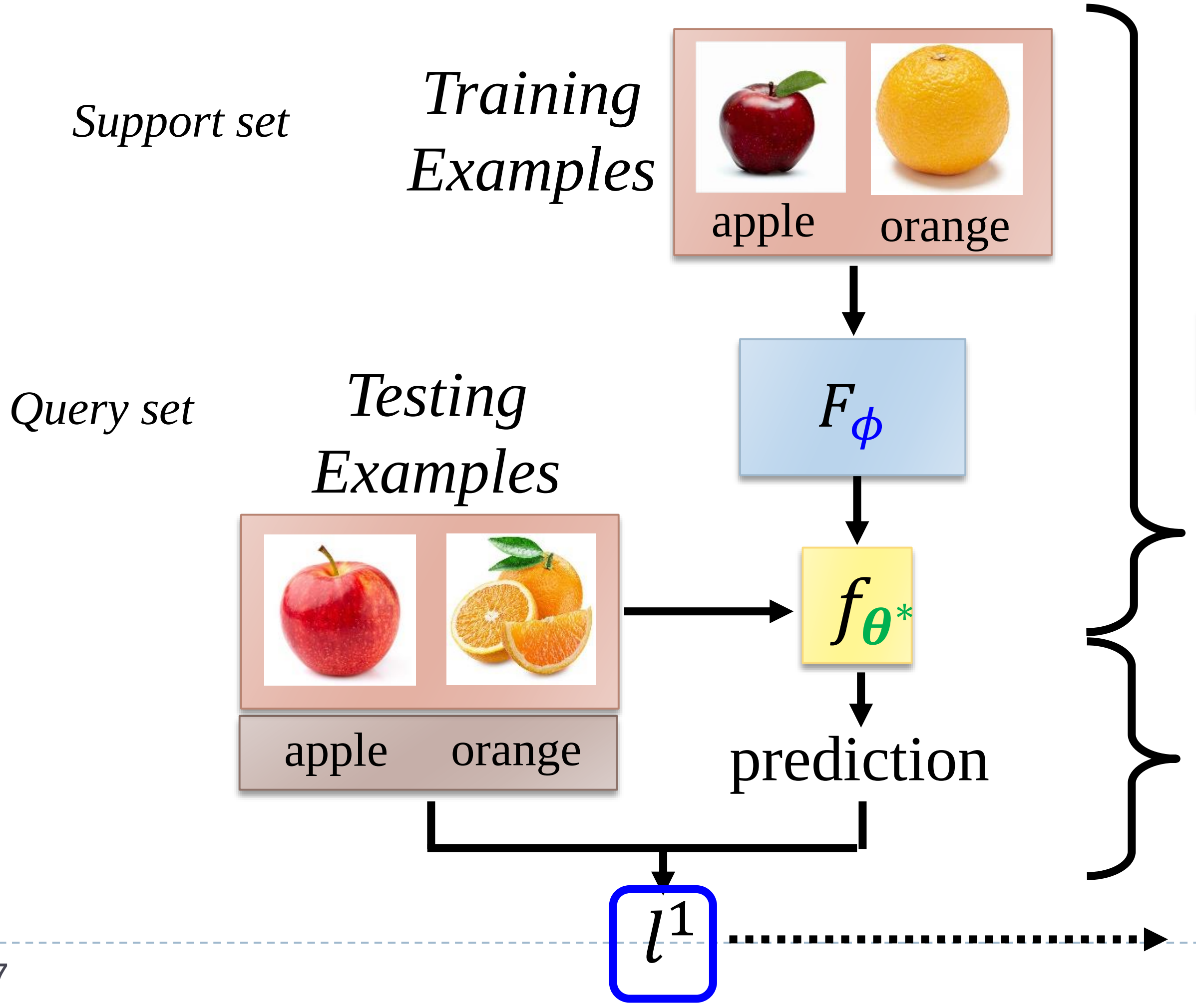
$$L(\phi) = \sum_{n=1}^{N} \boxed{l^n}$$

If your optimization method needs to compute $L(\phi)$

Outer Loop in "*Learning to initialize*"

Across-task training includes several **within-task training and testing**

Inner Loop in "*Learning to initialize*"

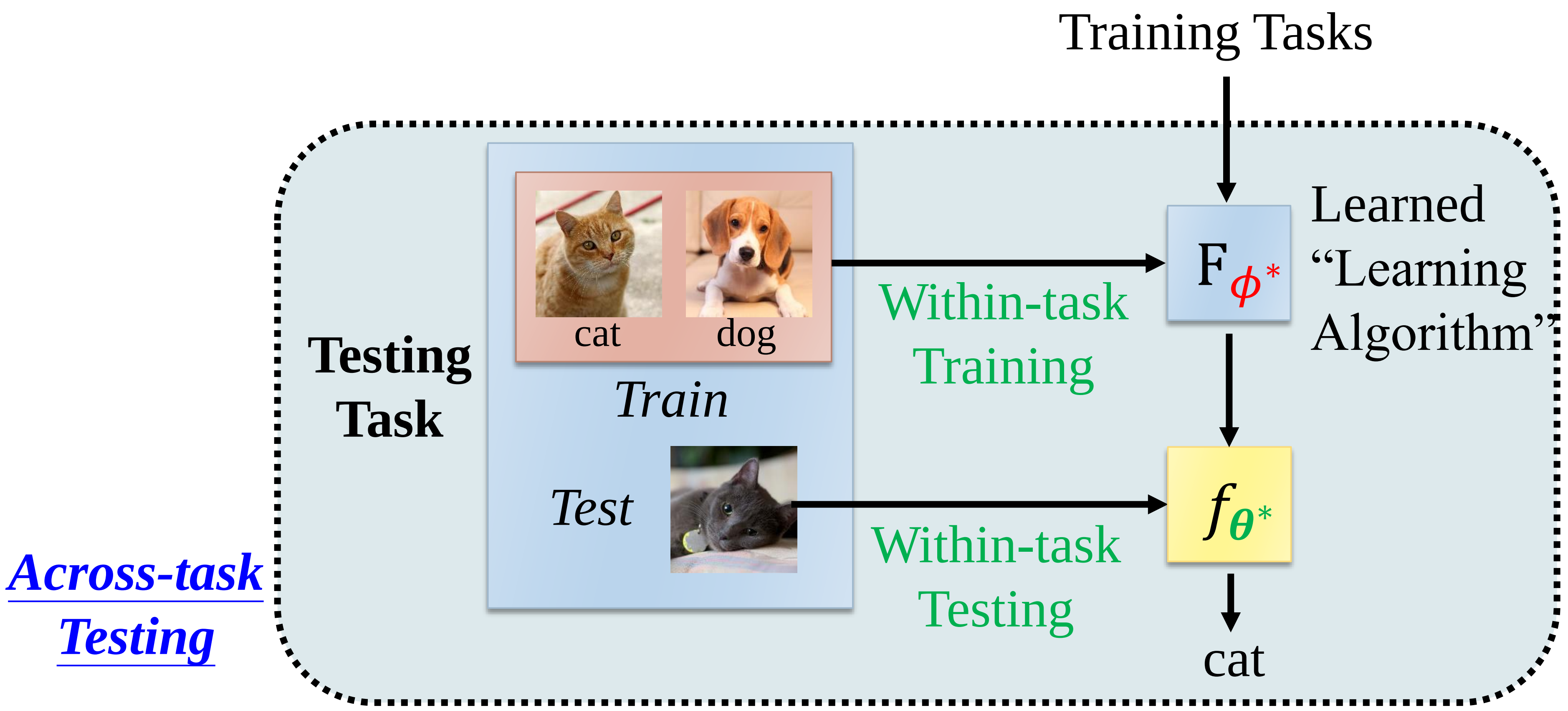


Within-task Training

Within-task Testing

To compute the loss

Framework



Training Tasks
Learned "Learning Algorithm"
F_{ϕ^*}
Testing Task
cat
dog
Train
Test
Within-task Training
Within-task Testing
f_{θ^*}
cat
Across-task Testing

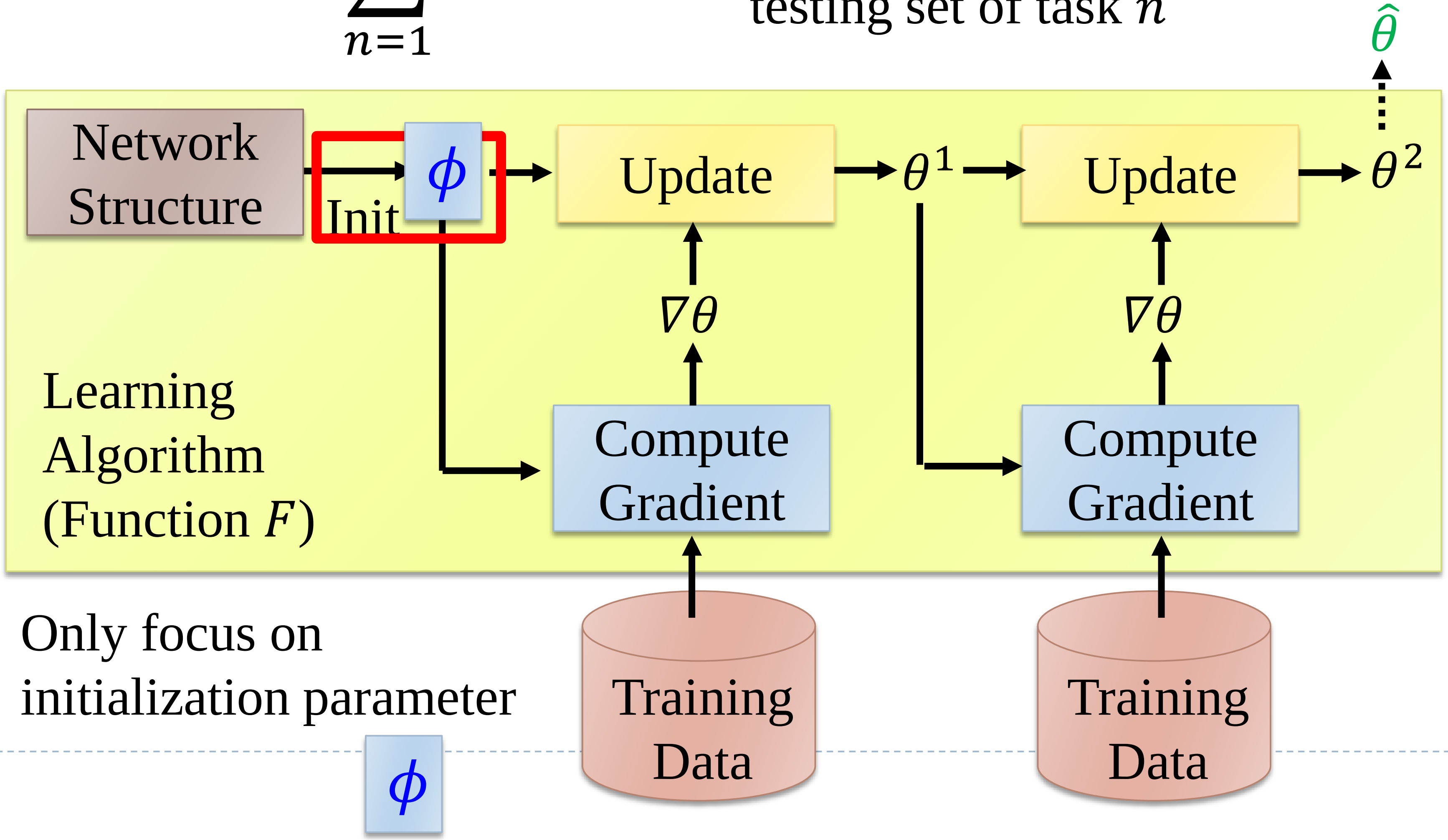
3. Model-Agnostic Meta-Learning (MAML) $\hat{\theta}^n$: model learned from task n

Loss Function:

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

$\hat{\theta}^n$ depends on ϕ

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



Only focus on initialization parameter

MAML

MAML

Loss Function:

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

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

$\hat{\theta}^n$ depends on ϕ

$l^n(\hat{\theta}^n)$: 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** | **Potential**

Model Pre-training

Loss Function:

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

Widely used in transfer learning

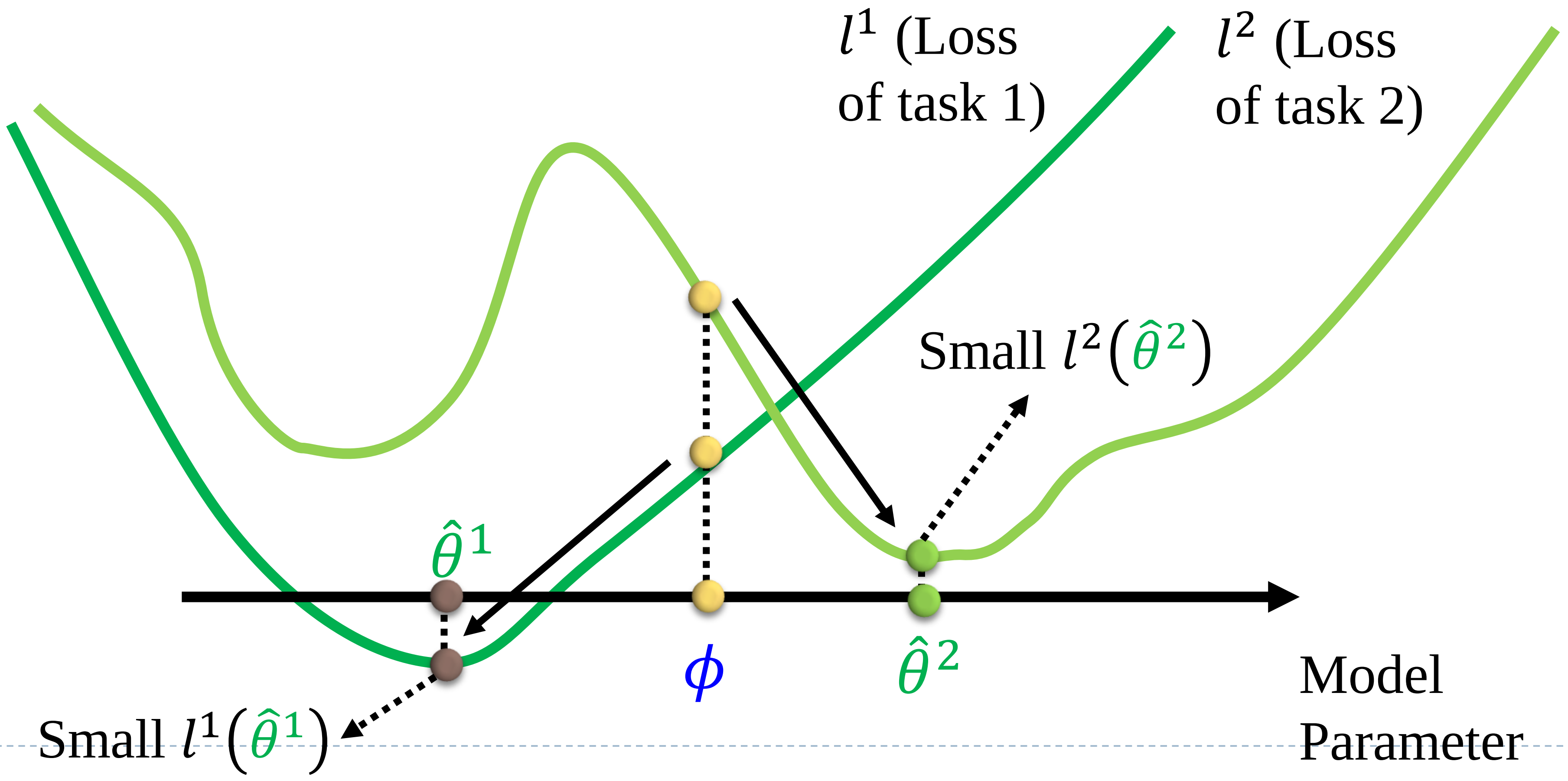
Find ϕ achieving good performance | **Current performance**

MAML

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

We don't care about the performance of ϕ on the training task

We care about the $\hat{\theta}^n$ derive from ϕ



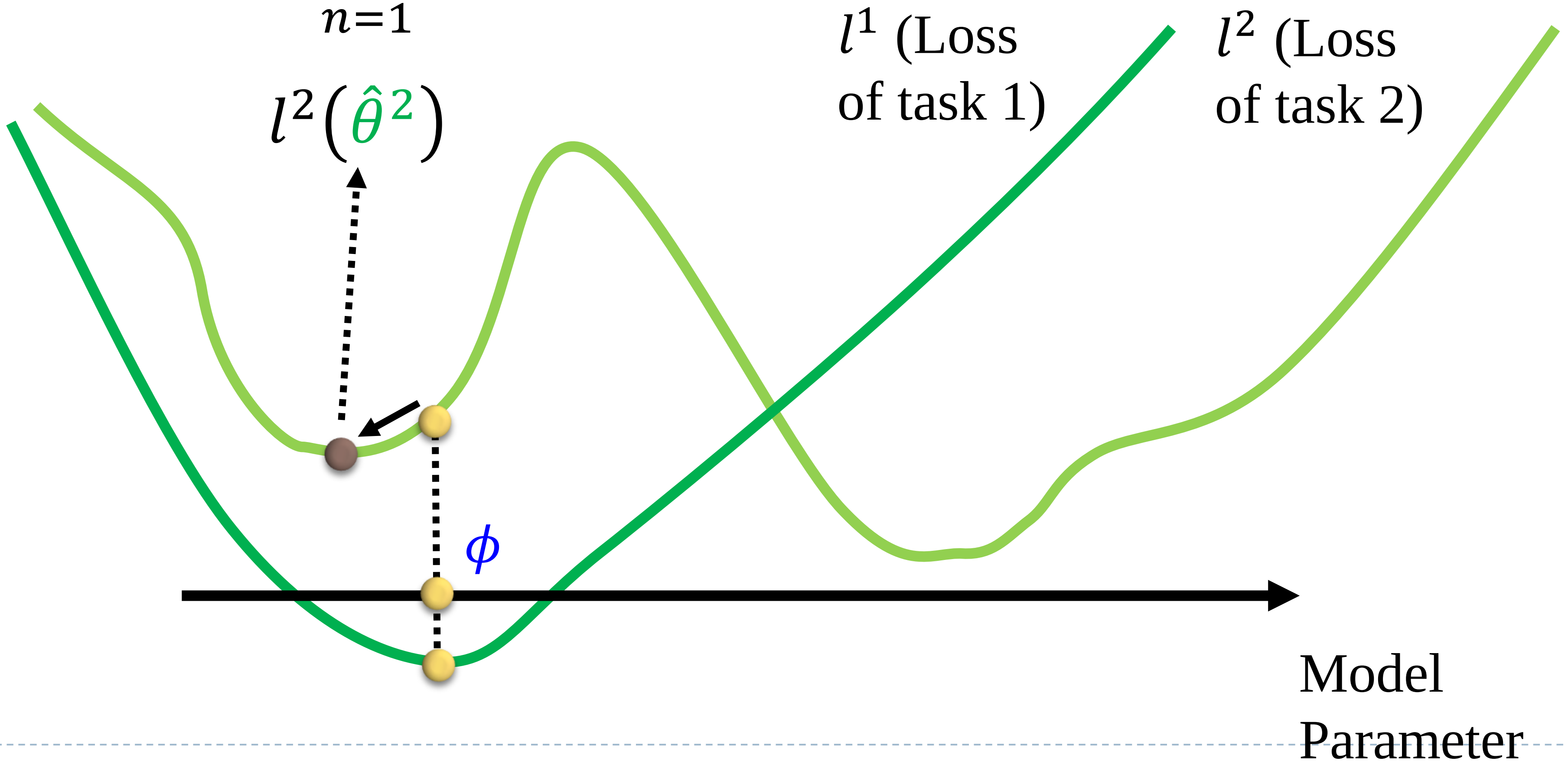
https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2019/Lecture/Meta1%20(v6).pptx

Model Pre-training

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

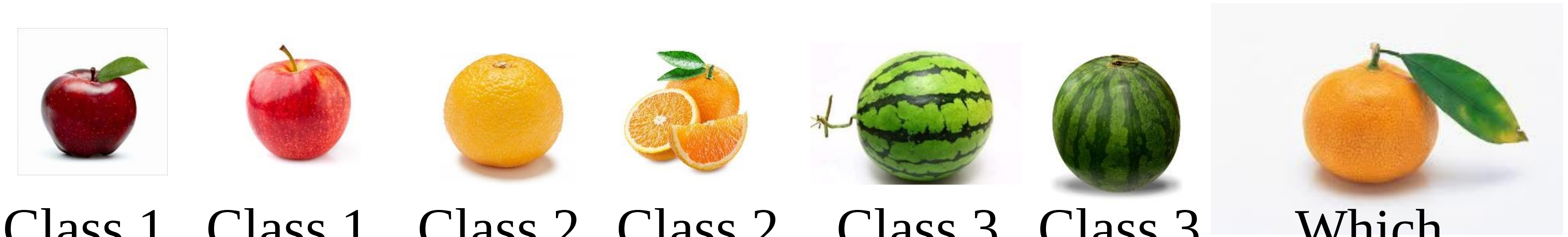
Find the ϕ that is good for all task

We do not guarantee whether ϕ will get good $\hat{\theta}^n$



Few-shot Image Classification

- Each class only has a few images



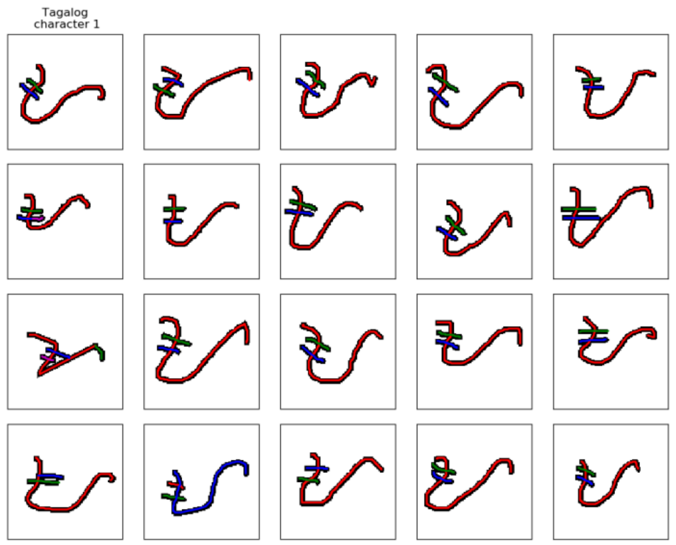
3-ways 2-shot

- N-ways K-shot classification: In each task, there are N classes, each has K examples
- In meta learning, you need to prepare many N-ways K-shot tasks as training and testing tasks

Omniglot

- 1,623 characters
- Each has 20 examples

https://github.com/brendenlake/omniglot

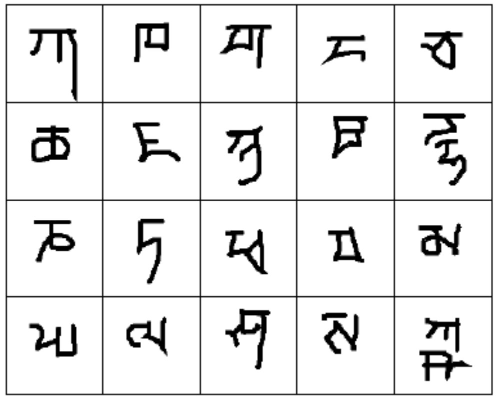


Omniglot

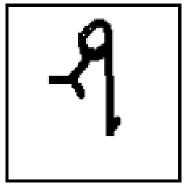
- Split your characters into training and testing characters
  - Sample *N* training characters, sample *K* examples from each sampled characters $\rightarrow$ one training task
  - Sample *N* testing characters, sample *K* examples from each sampled characters $\rightarrow$ one testing task

20 ways
1 shot

Each character represents a class



Training set
(Support set)

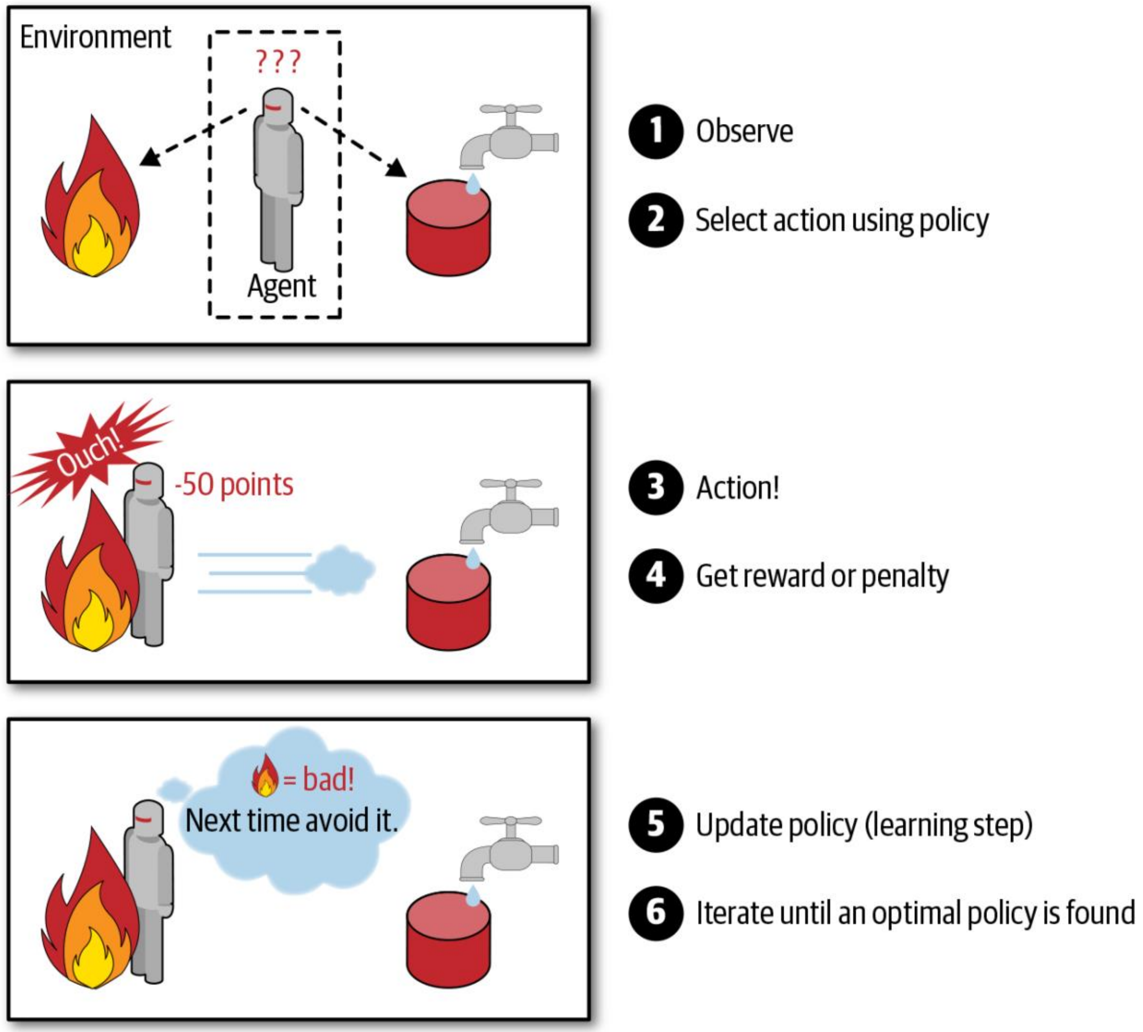


Testing set
(Query set)

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

A brief introduction to Reinforcement Learning

- Reinforcement Learning
  - *Reinforcement Learning* is very different one. The learning system, called an *agent* in this context, can observe the *environment*, select and perform *actions*, and get *rewards* in return
  - It must then learn by itself what is the best strategy, called a *policy*, to get the most reward over time
  - Robots, AlphaGo…


Environment
???
Agent
1 Observe
2 Select action using policy
Ouch!
-50 points
3 Action!
4 Get reward or penalty
= bad!
Next time avoid it.
5 Update policy (learning step)
6 Iterate until an optimal policy is found

4. Network Architecture Search (NAS)

- Evolutionary algorithm or Reinforcement learning

$$\hat{\phi} = \underset{\phi}{arg\,min}\, L(\phi) \qquad \nabla_{\phi} L(\phi) = ?$$

$\phi \searrow$ Network Architecture

An agent uses a set of actions to determine the network architecture.

ϕ: the agent's parameters

$-L(\phi)$

Reward to be maximized

Network Architecture Search (NAS)

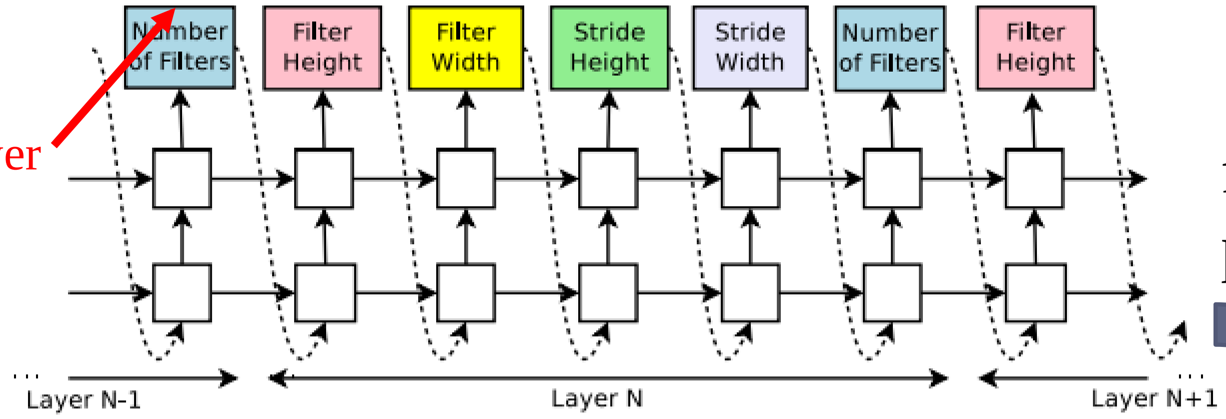
- Key idea is that we can specify the structure and connectivity of a neural network by using a configuration string
  - ["Filter Width: 5", "Filter Height: 3", "Num Filters: 24"]
- The idea is to use a RNN ("agent") to generate this string that specifies a neural network architecture
- Train this architecture ("child network") to see how well it performs on a validation set
- Use reinforcement learning to update the parameters of the agent model based on the accuracy of the child model
- The implementation is available here

Network Architecture Search (NAS)

Across-task Training

Update ϕ to maximize reward $-L(\phi)$

Softmax layer

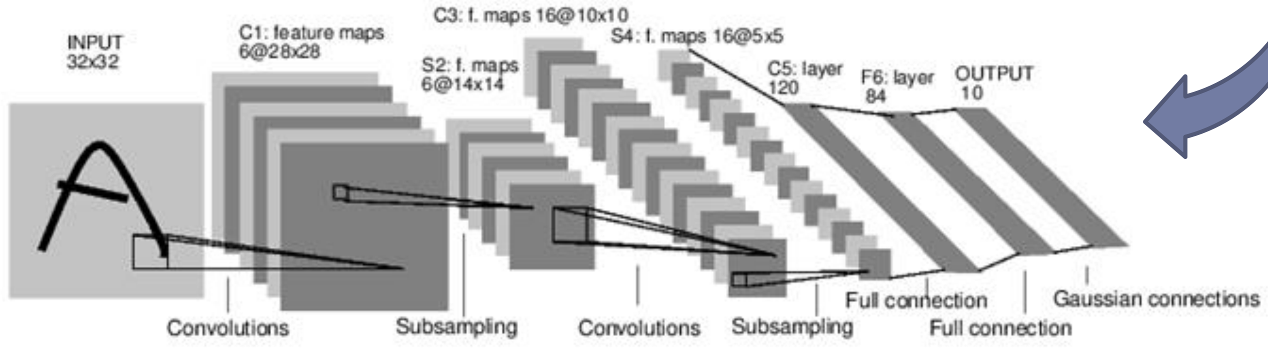


form a network with probabilities p

agent ϕ (RNN)

$-L(\phi)$

Accuracy of the network



A Full Convolutional Neural Network (LeNet)

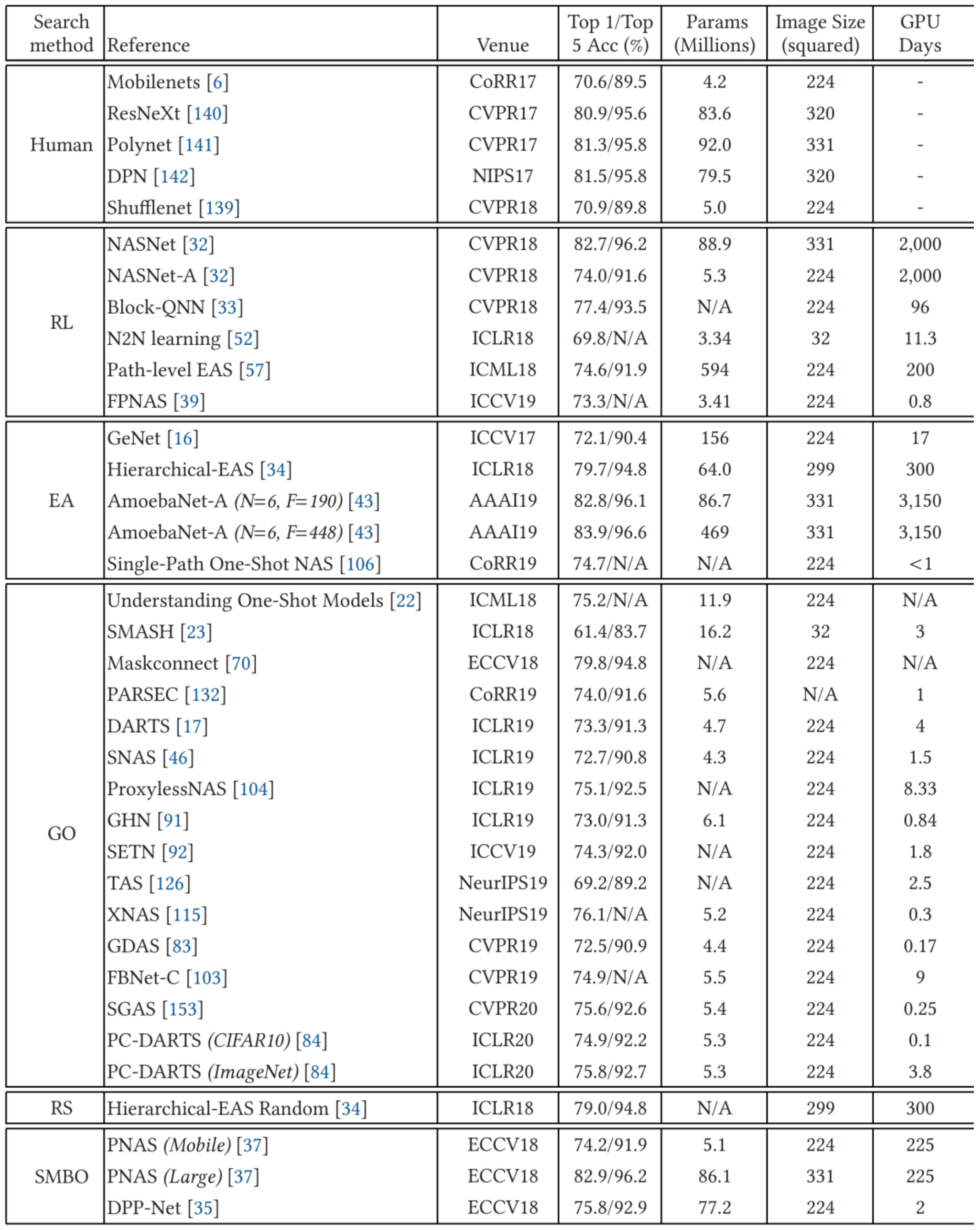
Train the network

https://arxiv.org/pdf/1611.01578.pdf

Within-task Training

NAS

- Their design may be based on reinforcement learning (RL), evolutionary algorithm (EA), gradient optimization (GO), random search (RS), and sequential model-based optimization (SMBO)…
- Some popular models
  - EfficeintNet
  - RegNet
  - MobileNet v3

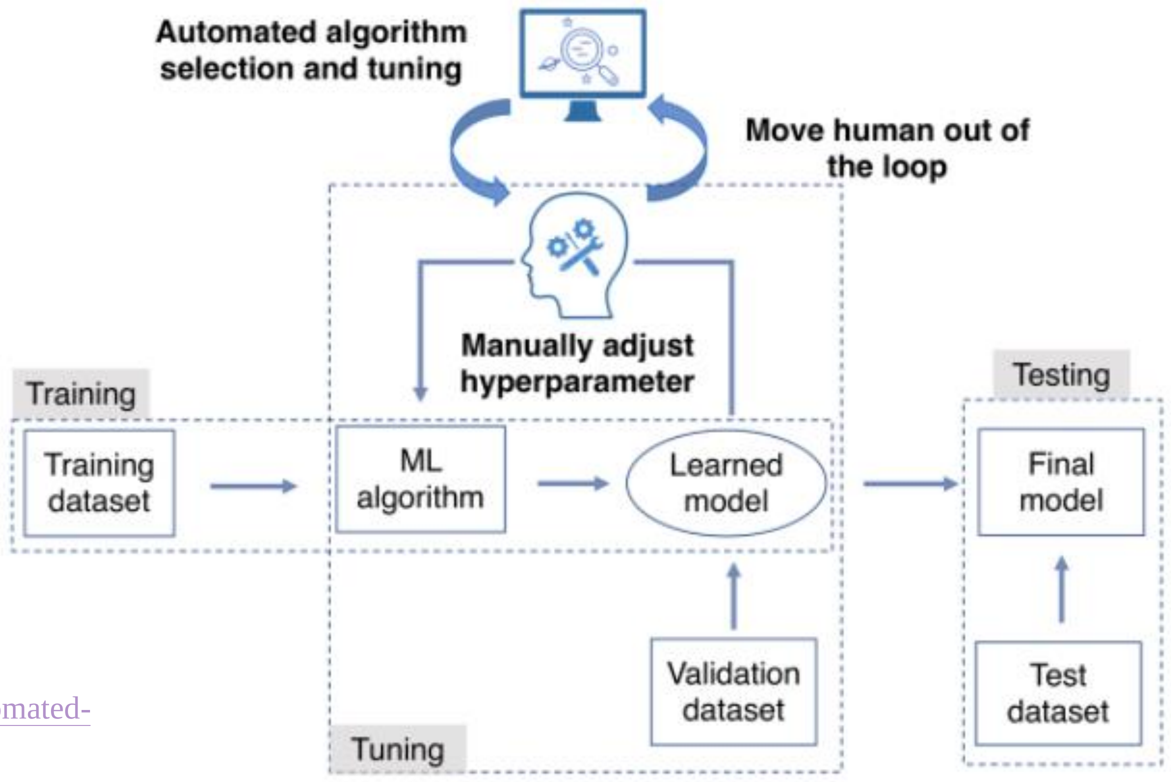


Search method	Reference	Venue	Top 1/Top 5 Acc (%)	Params (Millions)	Image Size (squared)	GPU Days
Human	Mobilenets [6]	CoRR17	70.6/89.5	4.2	224	-
	ResNeXt [140]	CVPR17	80.9/95.6	83.6	320	-
	Polynet [141]	CVPR17	81.3/95.8	92.0	331	-
	DPN [142]	NIPS17	81.5/95.8	79.5	320	-
	Shufflenet [139]	CVPR18	70.9/89.8	5.0	224	-
RL	NASNet [32]	CVPR18	82.7/96.2	88.9	331	2,000
	NASNet-A [32]	CVPR18	74.0/91.6	5.3	224	2,000
	Block-QNN [33]	CVPR18	77.4/93.5	N/A	224	96
	N2N learning [52]	ICLR18	69.8/N/A	3.34	32	11.3
	Path-level EAS [57]	ICML18	74.6/91.9	594	224	200
	FPNAS [39]	ICCV19	73.3/N/A	3.41	224	0.8
EA	GeNet [16]	ICCV17	72.1/90.4	156	224	17
	Hierarchical-EAS [34]	ICLR18	79.7/94.8	64.0	299	300
	AmoebaNet-A *(N=6, F=190)* [43]	AAAI19	82.8/96.1	86.7	331	3,150
	AmoebaNet-A *(N=6, F=448)* [43]	AAAI19	83.9/96.6	469	331	3,150
	Single-Path One-Shot NAS [106]	CoRR19	74.7/N/A	N/A	224	<1
GO	Understanding One-Shot Models [22]	ICML18	75.2/N/A	11.9	224	N/A
	SMASH [23]	ICLR18	61.4/83.7	16.2	32	3
	Maskconnect [70]	ECCV18	79.8/94.8	N/A	224	N/A
	PARSEC [132]	CoRR19	74.0/91.6	5.6	N/A	1
	DARTS [17]	ICLR19	73.3/91.3	4.7	224	4
	SNAS [46]	ICLR19	72.7/90.8	4.3	224	1.5
	ProxylessNAS [104]	ICLR19	75.1/92.5	N/A	224	8.33
	GHN [91]	ICLR19	73.0/91.3	6.1	224	0.84
	SETN [92]	ICCV19	74.3/92.0	N/A	224	1.8
	TAS [126]	NeurIPS19	69.2/89.2	N/A	224	2.5
	XNAS [115]	NeurIPS19	76.1/N/A	5.2	224	0.3
	GDAS [83]	CVPR19	72.5/90.9	4.4	224	0.17
	FBNet-C [103]	CVPR19	74.9/N/A	5.5	224	9
	SGAS [153]	CVPR20	75.6/92.6	5.4	224	0.25
	PC-DARTS *(CIFAR10)* [84]	ICLR20	74.9/92.2	5.3	224	0.1
	PC-DARTS *(ImageNet)* [84]	ICLR20	75.8/92.7	5.3	224	3.8
RS	Hierarchical-EAS Random [34]	ICLR18	79.0/94.8	N/A	299	300
SMBO	PNAS *(Mobile)* [37]	ECCV18	74.2/91.9	5.1	224	225
	PNAS *(Large)* [37]	ECCV18	82.9/96.2	86.1	331	225
	DPP-Net [35]	ECCV18	75.8/92.9	77.2	224	2

https://arxiv.org/abs/2006.02903

5. Automatic machine learning

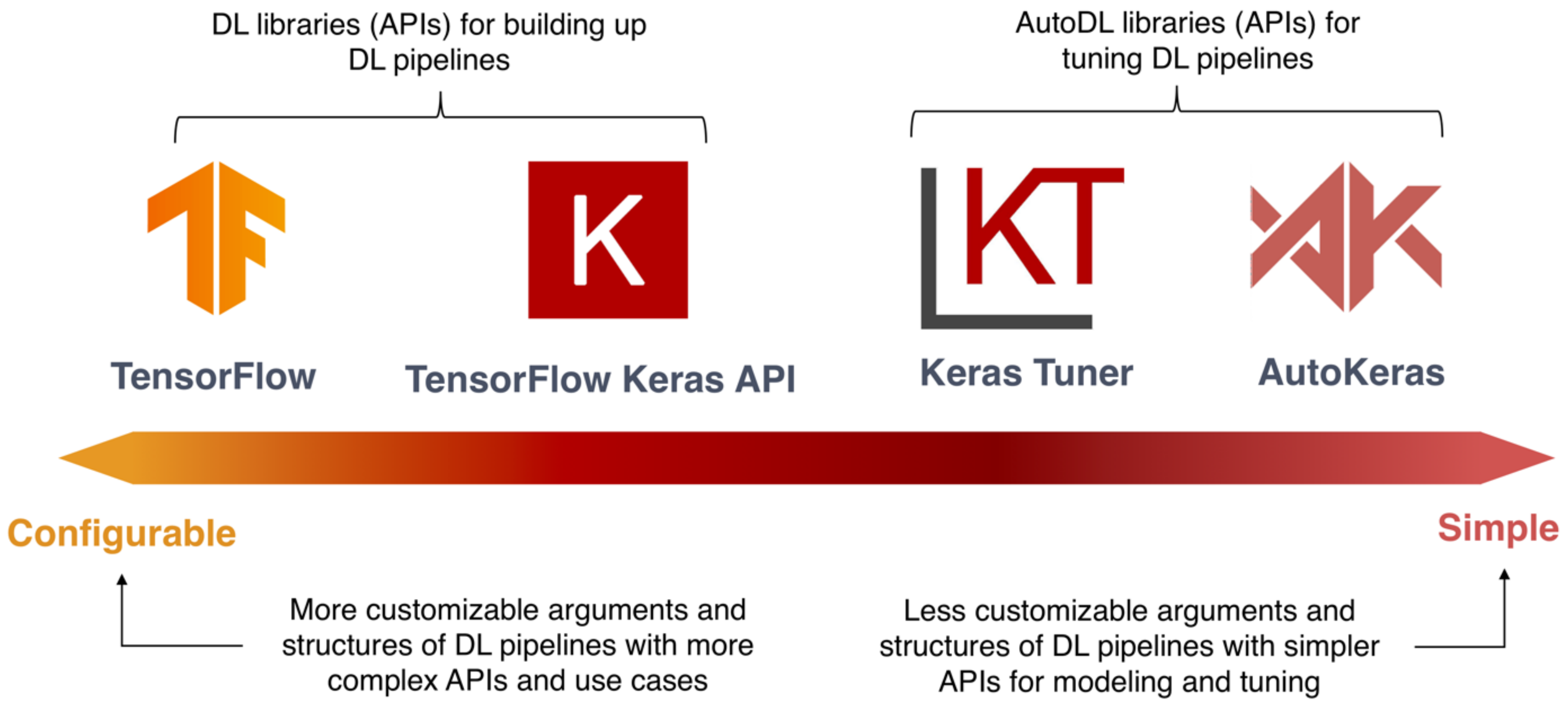
- Automated machine learning (AutoML) is the process of automating the tasks of applying machine learning to real-world problems
  - Meta-learning and automatically hyperparameter tuning are two key components
  - However, AutoML potentially includes every stage from beginning with a raw dataset to building a machine learning model ready for deployment (*data preprocessing, feature engineering, model selection and hyperparameter tuning*)



https://www.manning.com/books/automated-machine-learning-in-action

Automatic machine learning

- Users with more ML expertise can achieve more personalized solutions to meet their requirements using lower-level libraries


DL libraries (APIs) for building up DL pipelines
AutoDL libraries (APIs) for tuning DL pipelines
TensorFlow
TensorFlow Keras API
Keras Tuner
AutoKeras
Configurable
Simple
More customizable arguments and structures of DL pipelines with more complex APIs and use cases
Less customizable arguments and structures of DL pipelines with simpler APIs for modeling and tuning

https://www.manning.com/books/automated-machine-learning-in-action

Conclusion

- Hyperparameter selection is crucial for the success of your neural network architecture, since they heavily influence the behavior of the learned model
  - Automatic hyperparameter tuning is an area that studies how to efficiently search the space of possible hyperparameters
- Meta-learning can be a powerful tool for AutoML
  - Model-Agnostic Meta-Learning and Network Architecture Search are two active research fields
- Today, AutoML is still in its early days, and it doesn't scale to large problems
  - But when AutoML becomes mature enough for widespread adoption, the jobs of machine learning engineers will move up the value-creation chain
  - They will begin to put much more effort into data curation, crafting complex loss functions that truly reflect business goals, as well as understanding how their models impact the digital ecosystems in which they're deployed

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition Chapter 11,19

[2] Deep learning with Python, 2nd Edition Chapter 13

[3] https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php Lecture 15

[4] Automated Machine Learning in Action Chapter 1 and Chapter 4

Appendix

Resources

- Automatic machine learning
  - https://en.wikipedia.org/wiki/Automated_machine_learning
  - https://en.wikipedia.org/wiki/Neural_architecture_search
  - https://en.wikipedia.org/wiki/Hyperparameter_optimization
- Meta learning
  - A Survey of Deep Meta-Learning
  - AutoAugment: Learning Augmentation Policies from Data
  - Learning an Explicit Mapping For Sample Weighting
  - Learning to learn by gradient descent by gradient descent

Libraries

- Hyperas or Talos
  - Useful libraries for optimizing hyperparameters for Keras models (the former is based on Hyperopt)
- Topt
  - Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming
- Neural network inference, h2o-3 or AutoGlon
  - An open source AutoML toolkit for automate machine learning lifecycle
- Scikit-Optimize (skopt)
  - A general-purpose optimization library. The BayesSearchCV class performs Bayesian optimization using an interface similar to GridSearchCV
- Sklearn-Deap
  - A hyperparameter optimization library based on evolutionary algorithms, with a GridSearchCV-like interface

Services

- https://sigopt.com/
- https://cloud.google.com/ai-platform/training/docs/using-hyperparameter-tuning
- https://bigml.com/api/optimls

Rethinking about deep learning

- In deep learning, everything is a vector—that is to say, everything is a *point* in a *geometric space*.
  - Model inputs (text, images, and so on) and targets are first *vectorized*— turned into an initial input vector space and target vector space. Each layer in a deep learning model operates one simple geometric transformation on the data that goes through it. Together, the chain of layers in the model forms one complex geometric transformation, broken down into a series of simple ones. This complex transformation attempts to map the input space to the target space, one point at a time.
  - This transformation is parameterized by the weights of the layers, which are iteratively updated based on how well the model is currently performing. A key characteristic of this geometric transformation is that it must be *differentiable*, which is required in order for us to be able to learn its parameters via gradient descent. Intuitively, this means the geometric morphing from inputs to outputs must be smooth and continuous—a significant constraint

Rethinking about deep learning

- The entire process of applying this complex geometric transformation to the input data can be visualized in 3D by imagining a person trying to uncrumple a paper ball: the crumpled paper ball is the manifold of the input data that the model starts with. Each movement operated by the person on the paper ball is similar to a simple geometric transformation operated by one layer. The full uncrumpling gesture sequence is the complex transformation of the entire model. Deep learning models are mathematical machines for uncrumpling complicated manifolds of highdimensional data
- That's the magic of deep learning: turning meaning into vectors, then into geometric spaces, and then incrementally learning complex geometric transformations that map one space to another. All you need are spaces of sufficiently high dimensionality in order to capture the full scope of the relationships found in the original data

Rethinking about deep learning

- The whole process hinges on a single core idea: that meaning is derived from the *pairwise relationship between things* (between words in a language, between pixels in an image, and so on) and that these relationships can be captured by a distance function. But note that whether the brain also implements meaning via geometric spaces is an entirely separate question. Vector spaces are efficient to work with from a computational standpoint, but different data structures for intelligence can easily be envisioned—in particular, graphs.
- Neural networks initially emerged from the idea of using graphs as a way to encode meaning, which is why they're named neural networks; the surrounding field of research used to be called connectionism. Nowadays the name "neural network" exists purely for historical reasons—it's an extremely misleading name because they're neither neural nor networks. In particular, neural networks have hardly anything to do with the brain. A more appropriate name would have been *layered representations learning* or *hierarchical representations learning*, or maybe even ***deep differentiable models or chained geometric transforms***, to emphasize the fact that continuous geometric space manipulation is at their core

Rethinking about deep learning

- deep learning model is just a chain of simple, continuous geometric transformations mapping one vector space into another. All it can do is map one data manifold X into another manifold Y, assuming the existence of a learnable continuous transform from X to Y. A deep learning model can be interpreted as a kind of program, but, inversely, *most programs can't be expressed as deep learning models*
- For most tasks, either there exists no corresponding neural network of reasonable size that solves the task or, even if one exists, it may not be learnable : the corresponding geometric transform may be far too complex, or there may not be appropriate data available to learn it