Hyperparamter search and experiment management

Szu-Chi Chung

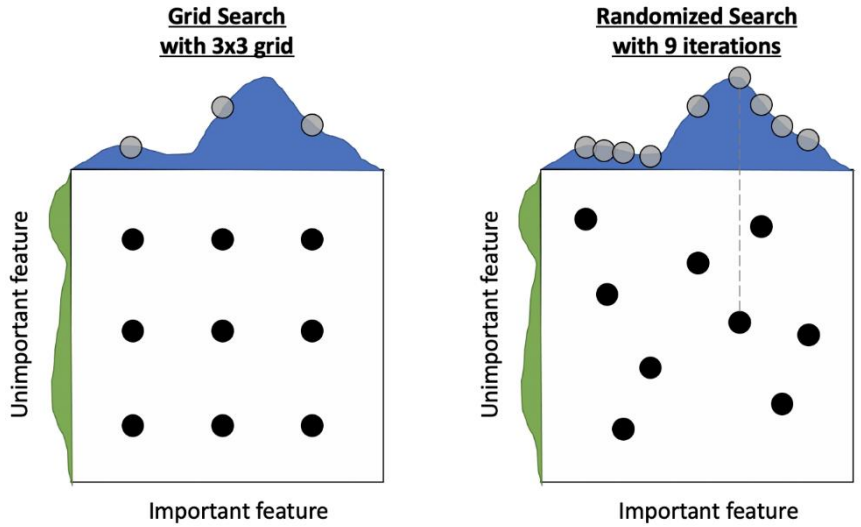
Department of Applied Mathematics, National Sun Yat-sen University

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 an activation or a different function?
    - Should you use BatchNormalization after a given layer?
    - How many dropouts should you use?
  - These architecture-level parameters which can be set in advance are called *hyperparameters* to distinguish them from the parameters of a model, which are trained via backpropagation

1. Hyperparamter search – Grid and randomized

- 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 Grid Search or Randomized Search 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



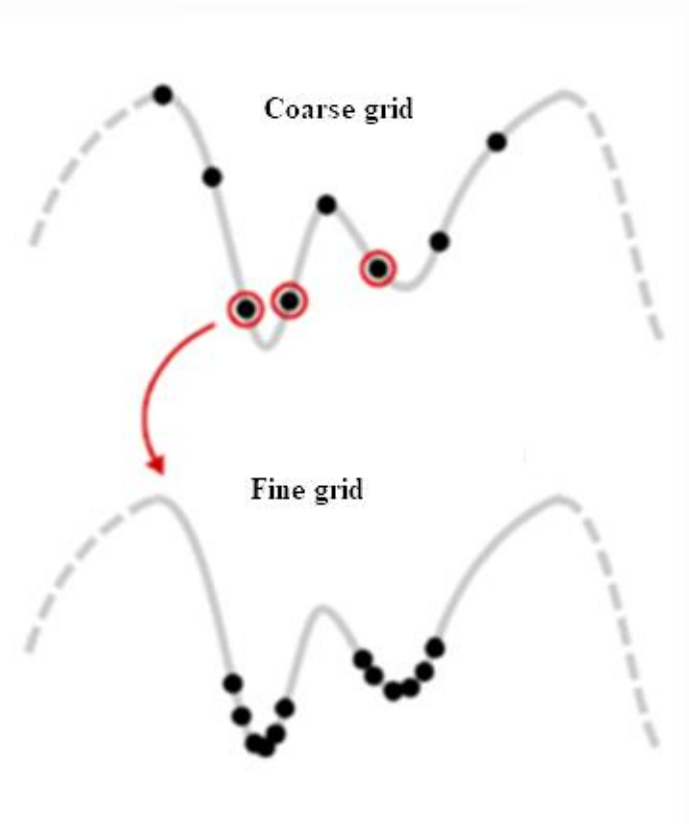
embarrassingly parallel

$\times$ 1,000

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

Hyperparamter search

- You can alleviate this problem by designing the search process
  - 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 is simple: when a region of space turns out to be good, it should be explored more. Such techniques lead to much better solutions in much less time
  - However, we should also balance exploration (testing new regions) and exploitation (focusing on known "good" regions)!



Hyperparamter search

- The key to hyperparamter search is the next set of hyperparameters to evaluate. But it is challenging considering the fact that
  1. The search space is typically made up of discrete decisions and thus isn't continuous or differentiable. Hence, you typically can't do gradient descent in search 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?

Hyperparamter search as an optimization problem

Machine/Deep Learning ≈ find a function f

Dog-Cat Classification

$f($  $) =$ "cat"

Meta Learning

≈ find a function F that finds a function f

Optimization Algorithm

$F($  $) = f$

cat dog cat dog

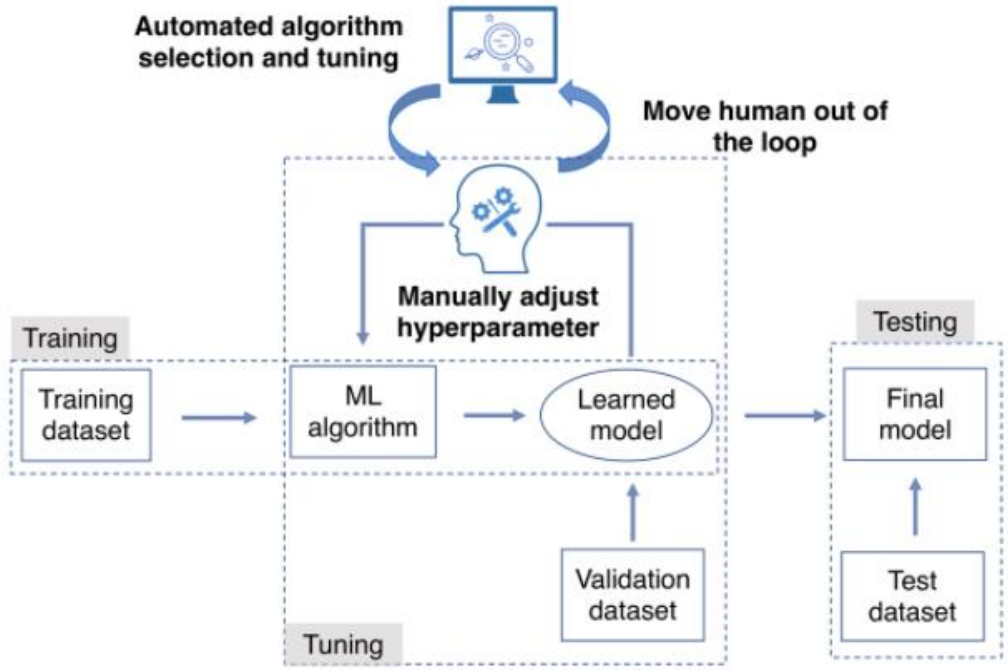
Training Examples

1. Choose a model f_θ
2. Choose a quality measure (objective function, loss function) for fitting
3. Optimization (fitting) to chose best θ

https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php

Hyperparamter search as an optimization problem

1. Choose a set of hyperparameters (Choose *search space*)
2. Fit model to your training data, and measure performance on the validation data
3. Choose the next set of hyperparameters (Automatically by *search strategy*)
4. Repeat
5. Eventually, measure performance on a test data

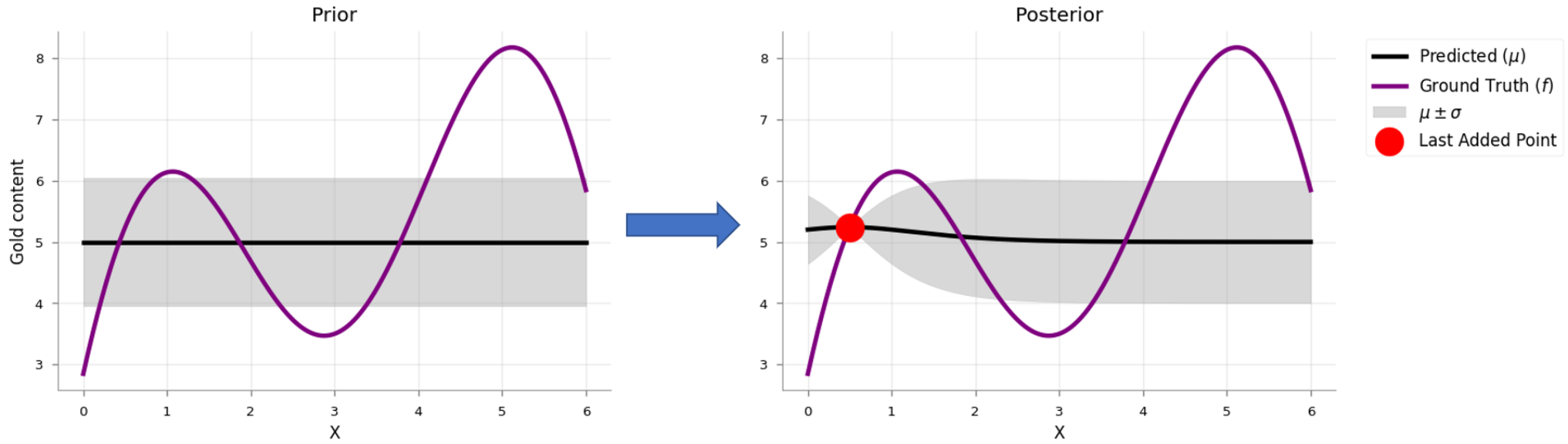


Hyperparamter search - Bayesian optimization

- Treat it as an optimization problem: hyperparameter search algorithms
  - Bayesian optimization, gradient-based, evolutionary optimization …
  - However, most of them are no longer embarrassing parallel…
- In Bayesian optimization, we train a function called the *surrogate function* to approximate the objective function
  - Once we have a surrogate model, we can sample a new hyperparameter combination for evaluation
  - To proceed with the sampling, we need to design another function, called the *acquisition function*, based on the surrogate function. This function specifies the criteria for comparing the ML models (determined by the hyperparameters) so that we can pick the most promising one to train and evaluate

Hyperparamter search - Bayesian optimization

1. We choose a surrogate model for the true function f and define its **prior**
2. Given the set of observations, use the Bayes rule to obtain the **posterior**
3. Use an acquisition function $\alpha(x)$, which is a function of the posterior, to decide the next sample point argmax $x_t = argmax_x \alpha(x)$
4. Add newly sampled data to the set of observations and go to step 2 till convergence or budget elapses



Hyperparamter search - Bayesian optimization

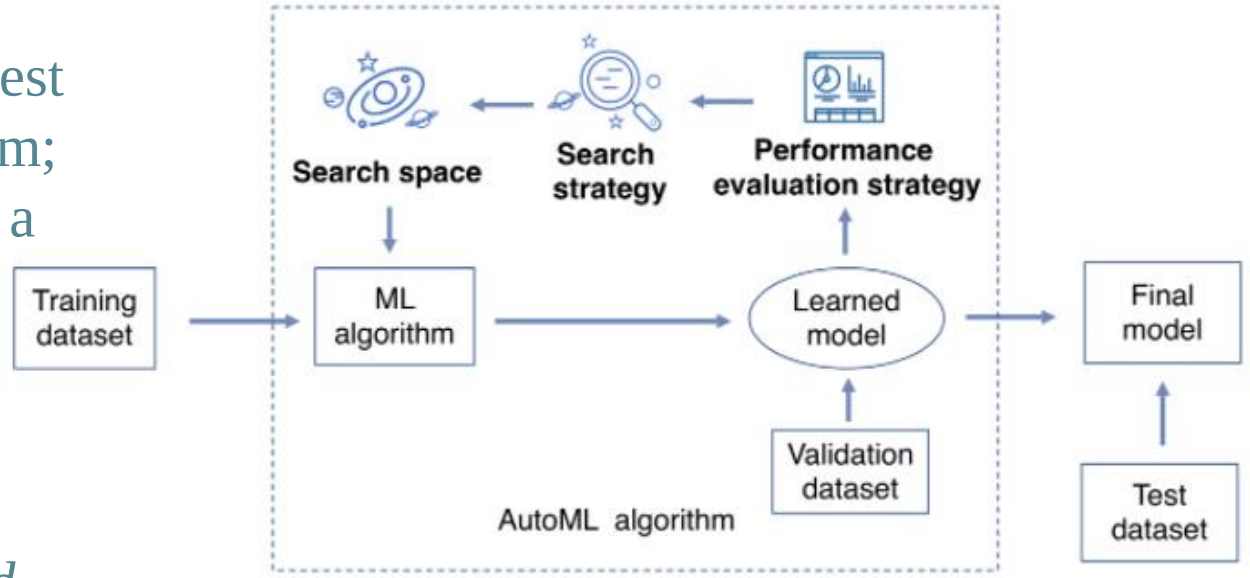
- The surrogate function is often chosen as Gaussian Process (GP) that contains μ and σ at each t
- Some popular acquisition functions are below:
  - Probability of Improvement (PI)
    - $x_{t+1} = argmax_x(P(f(x) \geq (f(x^+ + \varepsilon)))$ where $x^+ = argmax_{x_i \epsilon x_{1:t}} f(x_i)$
  - Expected Improvement (EI)
    - $x_{t+1} = argmax_x E(\max\{0, h_{t+1}(x) - f(x^+)\}|D_t)$ where D_t is the past training data and $h_{t+1}(x)$ is the posterior mean of the surrogate
  - Thompson Sampling
    - At every step, we sample a function from the surrogate's posterior and optimize it
- Bayesian optimization tries to make smart decisions about where to sample next, thereby reducing the number of evaluations needed. The acquisition functions try to balance the trade-off between exploration and exploitation!

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
  - We need to be smart about designing the right search space. Hyperparameter tuning is automation, not magic: we still need to handpick experiment configurations that have the potential to yield good metrics
- 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!

Automatically hyperparamter search to AutoML

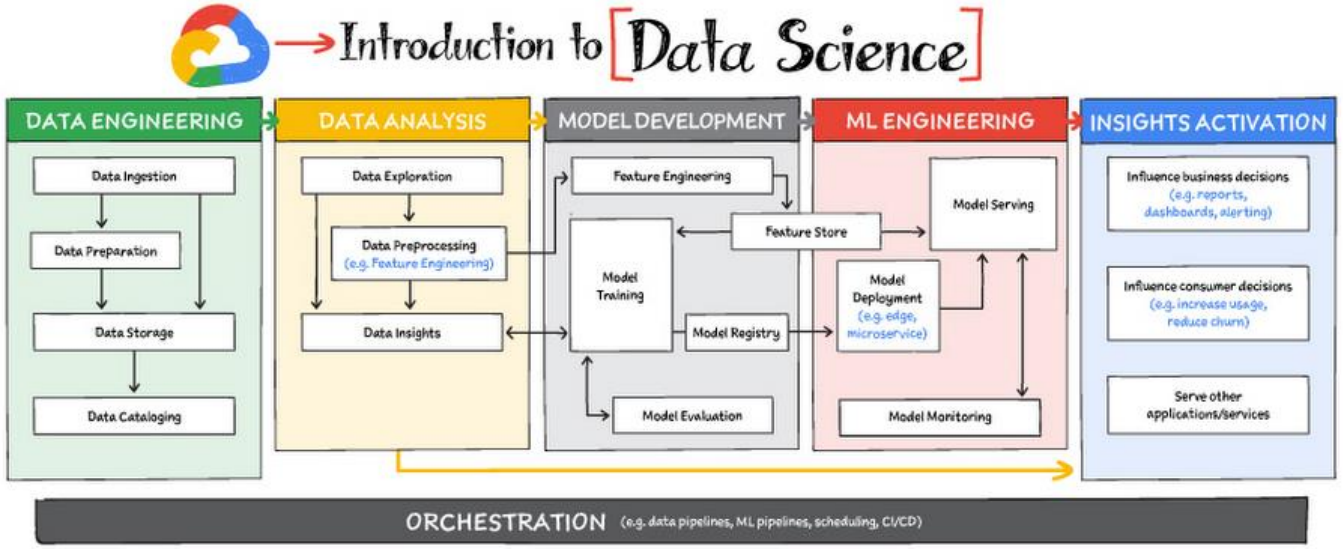
- We can also be far more ambitious and attempt to generate the model architecture itself from scratch, with as few constraints as possible
  - For example, Google has used an evolutionary approach, not just to search for hyperparameters but also to look for the best neural network architecture for the 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. 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 automatic 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, model ensembling*)

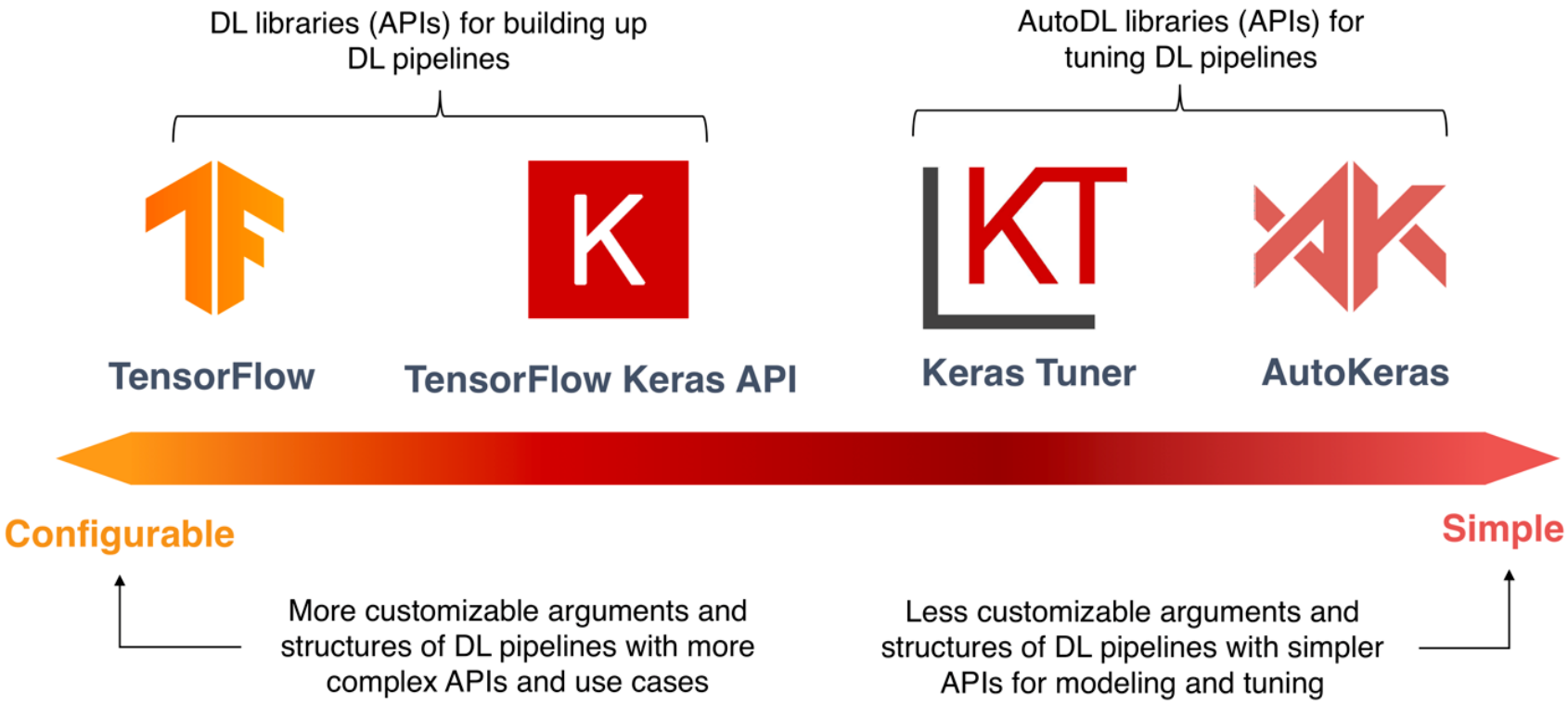


Automatic machine learning

- We still have a lot of challenges and limitations:
  - *The automation of collecting and cleaning data* — AutoML still requires people to collect, clean, and label data. These processes are often more complicated in practice than the design of ML algorithms, and, for now, they still cannot be automated. For AutoML to work today, it has to be given a clear task and objective with a high-quality dataset.
  - *The costs of selecting and tuning the AutoML algorithm*—The "no free lunch" theorem tells us that there is no omnipotent AutoML algorithm that fits any hyperparameter tuning problem. The effort you save on selecting and tuning an ML algorithm may be amortized or even outweighed by the effort you need to put into selecting the AutoML algorithm
  - *Resource costs* — AutoML is a relatively costly process in terms of both time and computational resources. Existing AutoML systems often need to try more hyperparameters than human experts to achieve comparable results

Automatic machine learning

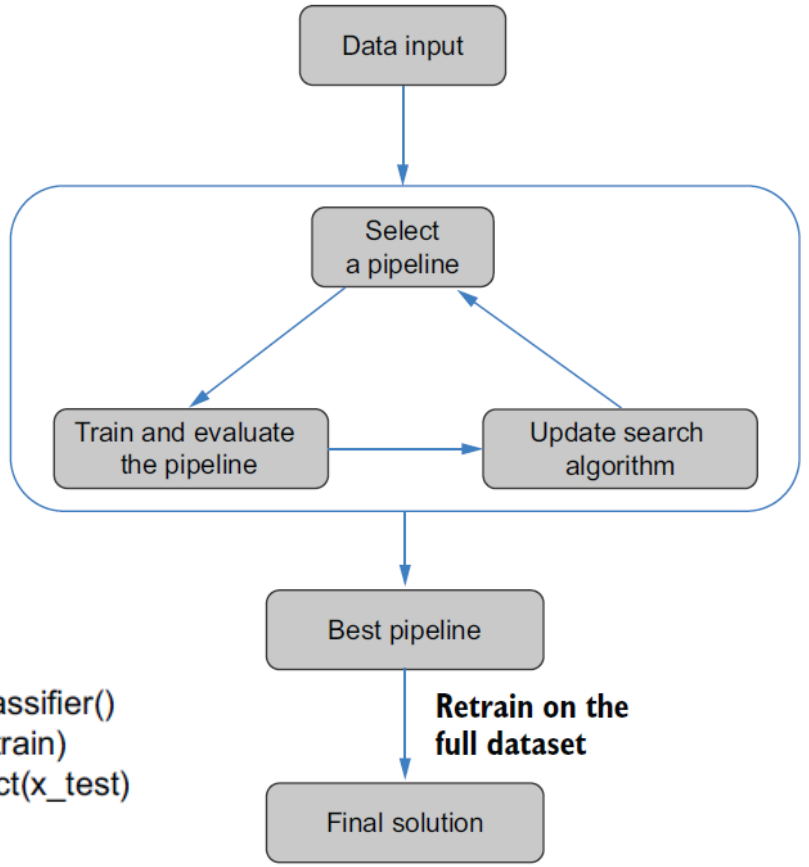
- Users with more ML expertise can achieve more personalized solutions to meet their requirements using lower-level libraries



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

Automatic machine learning - AutoKeras

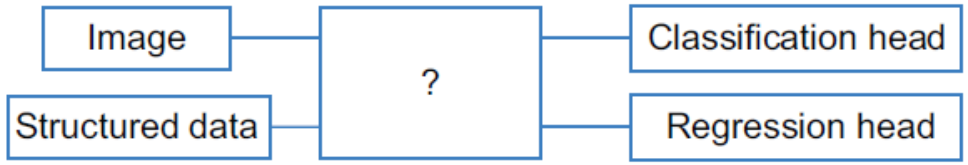
- The task APIs help you generate an end-to-end deep learning solution for a target ML task, such as image classification
  - These are the most straightforward AutoKeras APIs because they enable you to achieve the desired ML solution with only one step: feeding in the data. Six different task APIs support six different tasks in the latest release of AutoKeras, including classification and regression for image, text, and structured data




```
# Task API
auto_model=ak.ImageClassifier()
auto_model.fit(x_train,y_train)
result =auto_model.predict(x_test)
```

Automatic machine learning - AutoKeras

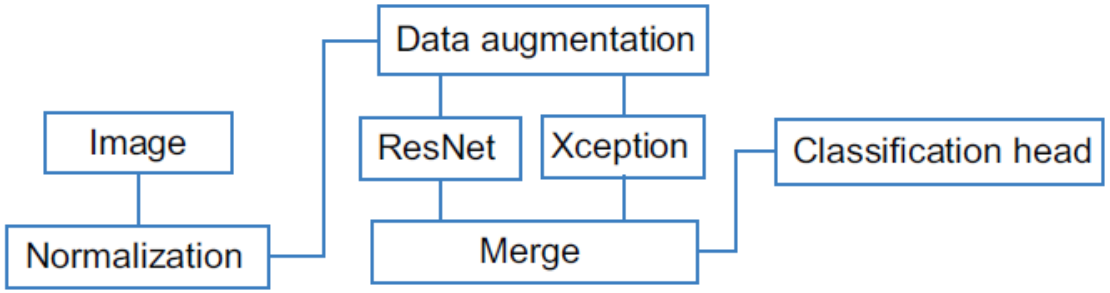
- Real-world problems can have multiple inputs or outputs. For example, we can use both visual and acoustic information to detect actions in a video. We may also want to predict multiple outputs, such as using the consumption records of customers to predict their shopping interests and income levels
  - To address these tasks, we can use AutoKeras's IO API
  - The search space is usually tailored to different tasks. AutoKeras provides a default search space for each task to save you effort on search space design. To customize the search space for personalized use cases, you need to use the functional API




```
# Input/Output API
auto_model = ak.AutoModel(
  inputs=[ak.ImageInput(), ak.StructuredDataInput()],
  outputs=[ak.ClassificationHead(), ak.RegressionHead()])
auto_model.fit(x=[x_image, x_struct], y=[y_clf, y_reg])
```

Automatic machine learning - AutoKeras

- The functional API is mainly for advanced users who want to *tailor the search space to their needs*
  - It allows us to build a deep learning pipeline by wiring some AutoKeras building blocks
  - A building block often represents a specific deep learning model *composed of multiple Keras layers* such as a CNN, meaning that we don't have to specify these models layer by layer. The search space of the hyperparameters for each block is also designed and set up for us so that we can focus on the hyperparameters that concern us without worrying about the rest

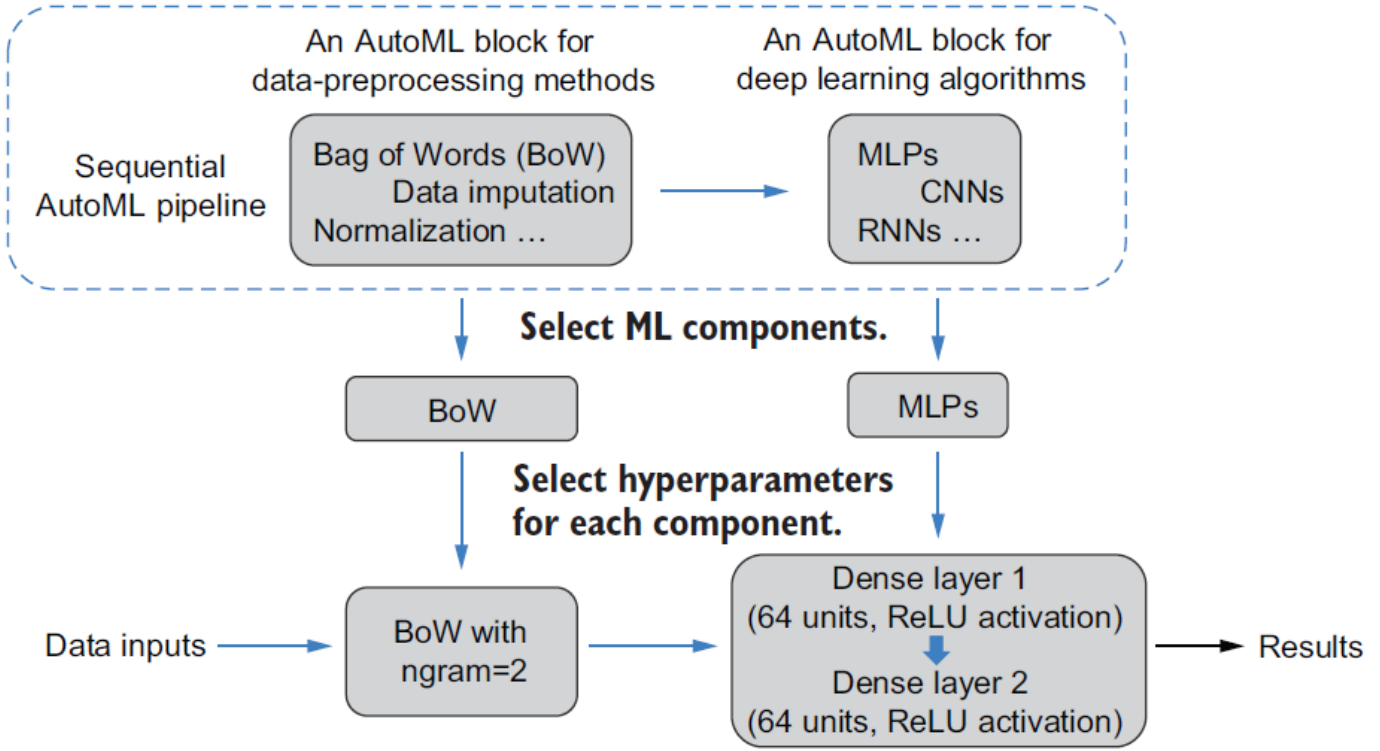



```
# Functional API
input_node = ak.ImageInput()
output_node = ak.Normalization()(input_node)
output_node = ak.ImageAugmentation()(output_node)
output_node_1 = ak.ResNetBlock(version='v2')(output_node)
output_node_2 = ak.XceptionBlock()(output_node)
output_node = ak.Merge()([output_node_1, output_node_2])
output_node = ak.ClassificationHead()(output_node)
auto_model = ak.AutoModel(input_node, output_node)
auto_model.fit(x_train, y_train)
```

Automatic machine learning - AutoKeras

- *Automated hyperparameter tuning* - The model type and preprocessing methods are chosen. We want to tune the hyperparameters of each specified ML component in the pipeline. The search space will include only the relevant hyperparameters for each fixed component

- *Automated pipeline search* — In some situations, we may not know which model or data preparation method to adopt ahead of time. In this case, one or more of the AutoML blocks will contain multiple components. To do this, we can include blocks for each model architecture. The preprocessing method can be either fixed or selected and tuned along with the model



AutoML with a fully customized search space - KerasTuner

- KerasTuner is a library for selecting and tuning both deep learning and shallow ML models. Besides the tasks that can be solved by AutoKeras, it tackles the following three scenarios that are difficult or introduce an extra burden for AutoKeras to handle:
  - Pipelines in the search space have different training and evaluation strategies, such as shallow models implemented with scikit-learn and deep learning models implemented with TensorFlow Keras
  - You need to perform tasks other than supervised learning tasks
  - There are no built-in AutoML blocks in AutoKeras that are appropriate for use

AutoML with a fully customized search space

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_model(hp):
    input_node = keras.Input(shape=(20,))
    units = hp.Int('units', min_value=32,
                   max_value=512, step=32)
    output_node = layers.Dense(units=units, activation='relu')(input_node)
    output_node = layers.Dense(units=units, activation='relu')(output_node)
    output_node = layers.Dense(units=1, activation='sigmoid')(output_node)
    model = keras.Model(input_node, output_node)

    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
    model.compile(
        optimizer=optimizer,
        loss='mse',
        metrics=['mae'])
    return model
```

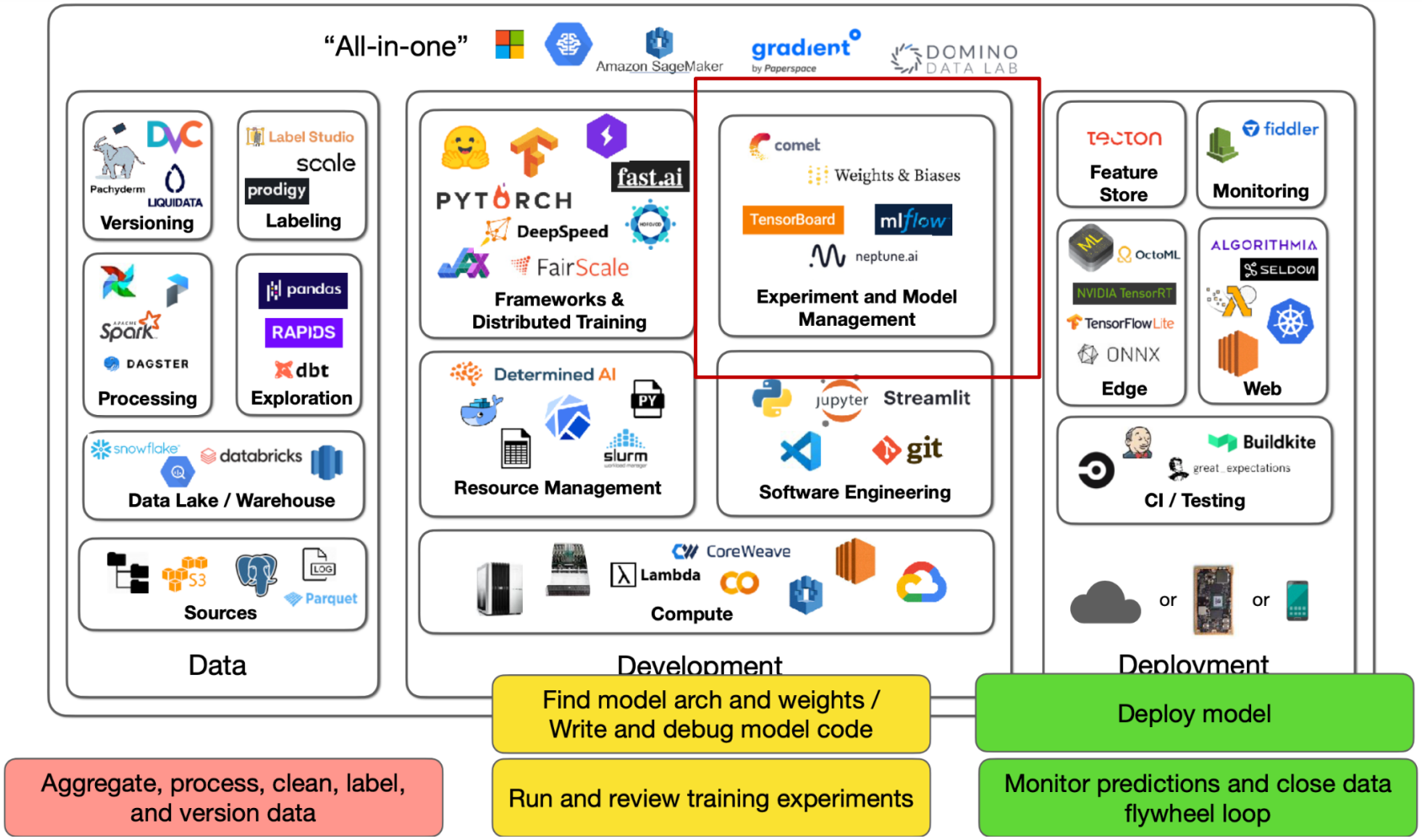
Creates the build function, whose input is an hp container instance

Defines the search space for the number of units in the two hidden layers

Compiles the model with the Adam optimization algorithm and MSE loss, and calculates the MAE metric for the model

The returned model is a Keras model.

3. Experiment Management



Experiment Management

- As you run numerous experiments to refine your model, it's easy to lose track of code, hyperparameters, and artifacts. Model iteration can lead to lots of complexity and messiness
  - For example, you could be monitoring the learning rate's impact on your model's performance metric. *With multiple model runs, how will you monitor the impact of the hyperparameter?*
  - Experiment management refers to tools and processes that help us keep track of code, model parameters, and data sets that are iterated on during the model development lifecycle
  - A low-tech way would be to manually track the results of all model runs in a spreadsheet. Without great attention to detail, this can quickly spiral into a messy or incomplete artifact. Dedicated experiment management platforms are a remedy to this issue

Experiment Management

- There are several solutions here:
  - **TensorBoard**: A non-exclusive Google solution effective at one-off experiment tracking. It is difficult to manage many experiments
  - **MLflow**: A non-exclusive Databricks project that includes model packaging and more, in addition to experiment management. It must be self-hosted
  - **Weights and Biases**: An easy-to-use solution free for personal and academic projects! Logging starts simply
- Other options include **Neptune AI**, **Comet ML**, and **Determined AI**, all of which have solid experiment-tracking options

Experiment Management

- Many of these platforms also offer *intelligent hyperparameter optimization*, which allows us to control the cost of searching for the right parameters for a model
  - For example, Weights and Biases has a product called **Sweeps** that helps with hyperparameter optimization. It's best to have it as part of your regular ML training tool; there's no need for a dedicated tool
  - With a YAML file specification, we can specify a hyperparameter optimization job and perform a "sweep," during which W&B sends parameter settings to individual "agents" (our machines) and compares the performance

Conclusion

- Hyperparameter selection is crucial for success 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
- 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, 3rd Edition Chapter 11,19

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

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

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

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
  - https://distill.pub/2020/bayesian-optimization/
  - https://www.automl.org/book/
- AutoML
  - PyCaret
  - Neural network inference
  - AutoGlon
  - h2o-3
  - Auto-sklearn

Resources

- Hyperparameter tuning
  - Optuna
  - hyperopt
  - Talos
  - Topt
  - Scikit-Optimize
  - Sklearn-Deap
  - SigOpt
  - Ray Tune
- Data/model management
  - https://fullstackdeeplearning.com/course/2022/lecture-2-development-infrastructure-and-tooling/
  - https://fullstackdeeplearning.com/spring2021/lecture-6/

Resources

- Guide
  - Empirical guidelines
  - Deep dive guidelines
- Service
  - https://sigopt.com/
  - https://cloud.google.com/ai-platform/training/docs/using-hyperparameter-tuning
  - https://bigml.com/api/optimls
- 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

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

Compute Hardware

- https://lambdalabs.com/gpu-benchmarks
- https://www.aime.info/blog/en/deep-learning-gpu-benchmarks-2022/

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit
K80	2014	Kepler	Server	24	5	N/A	No
P100	2016	Pascal	Server	16	10	N/A	Yes
V100	2017	Volta	Server	16 or 32	14	120	Yes
RTX 2080 Ti	2018	Turing	PC	11	13	107	Yes
RTX 3090	2021	Ampere	PC	24	35	285	Yes
A100	2020	Ampere	Server	40 or 80	19.5	312	Yes
A6000	2022	Ampere	Server	48	38	?	Yes

- **RAM**: should fit your model + meaningful batch of data
- **32bit vs Tensor TFlops**
  - Tensor Cores are specifically for deep learning operations (mixed precision)
- **16bit** ~doubles your effective RAM capacity

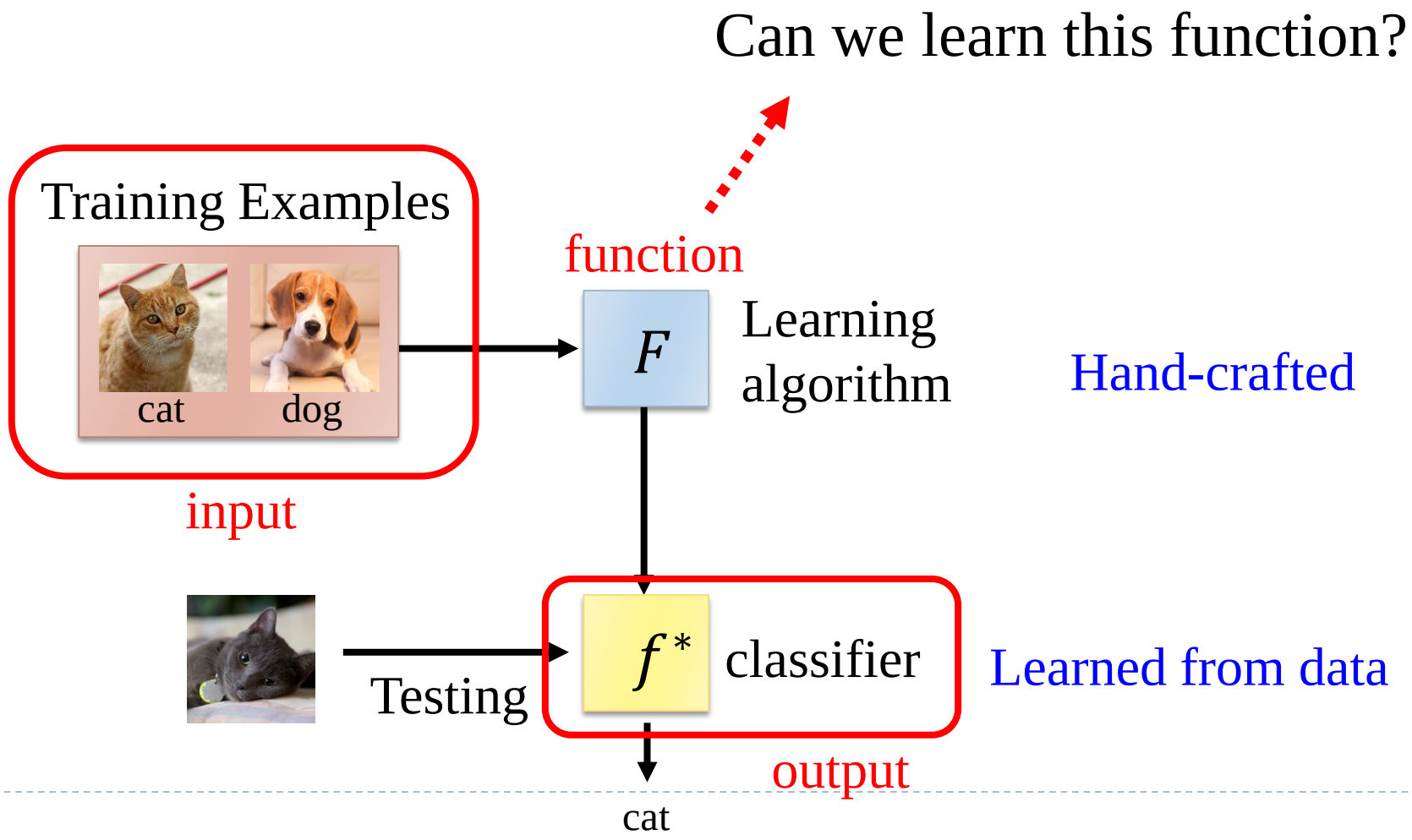
Compute Hardware

- https://fullstackdeeplearning.com/cloud-gpus/
- https://github.com/the-full-stack/website
- Some tips on on-prem vs. cloud use:
  - It can be useful to have your own GPU machine to shift your mindset from minimizing cost to maximizing utility.
  - To truly scale-out experiments, you should probably just use the most expensive machines in the least expensive cloud.
  - TPUs are worth experimenting with for large-scale training, given their performance.
  - Lambda Labs is a sponsor, and we highly encourage looking at them for on-prem and cloud GPU use!

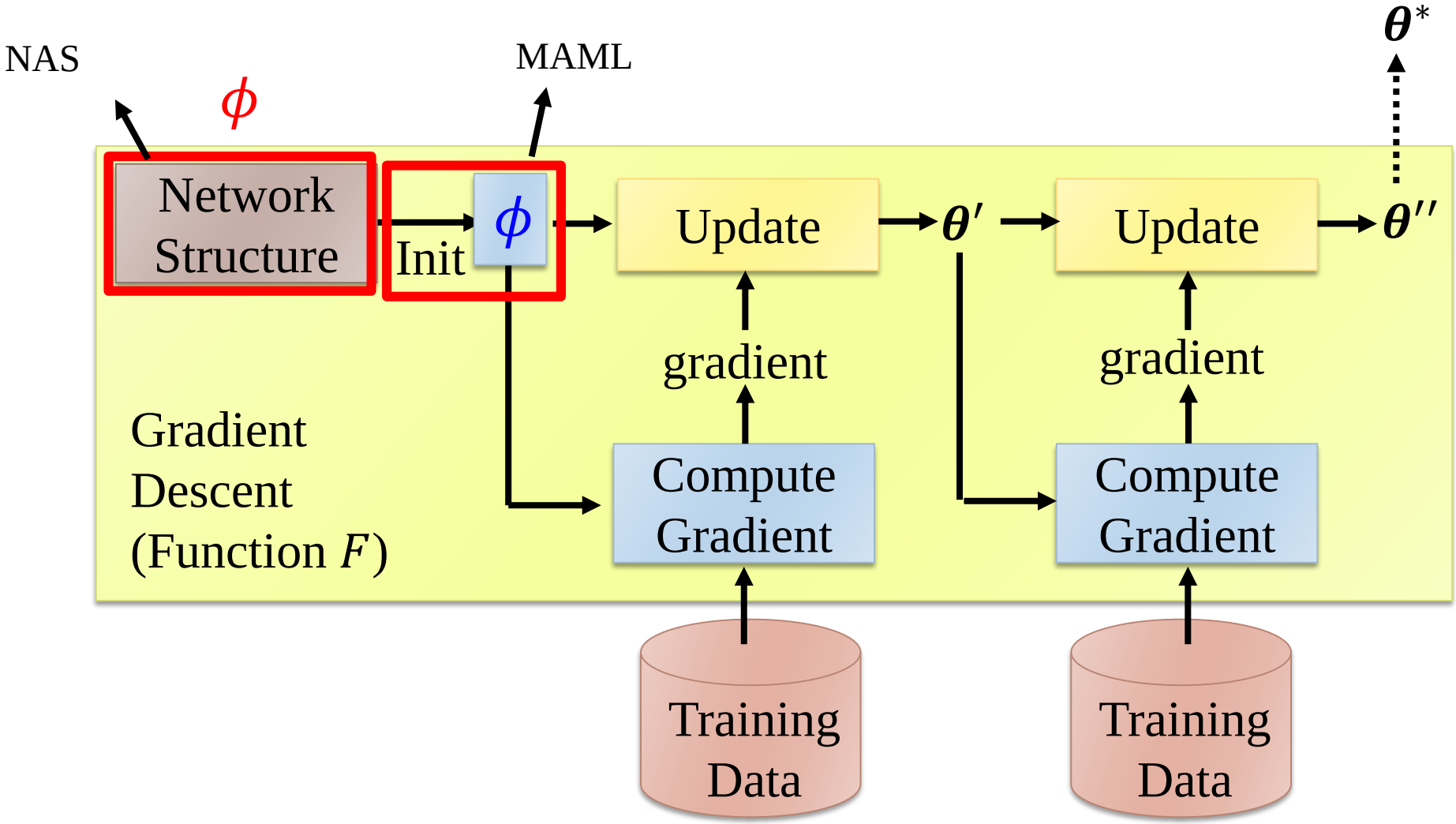
	W&B	Paperspace Gradient	Floyd	Determined.ai	Domino Data Lab	Amazon SageMaker	GC ML Engine
Hardware	N/A	Paperspace	GCP	Agnostic	Agnostic	AWS	GCP
Resource Management	No	Yes	Yes	Yes	Yes	Yes	Yes
Hyperparam Optimization	Yes	No	No	Yes	Yes	Yes	Yes
Storing Models	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reviewing Experiments	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Deploying Models as REST API	No	No	Yes	No	Yes	Yes	Yes
Monitoring	No	No	No	No	Yes	Yes	Yes

What is Meta Learning?

- Meta learning is one of the key component behind AutoML



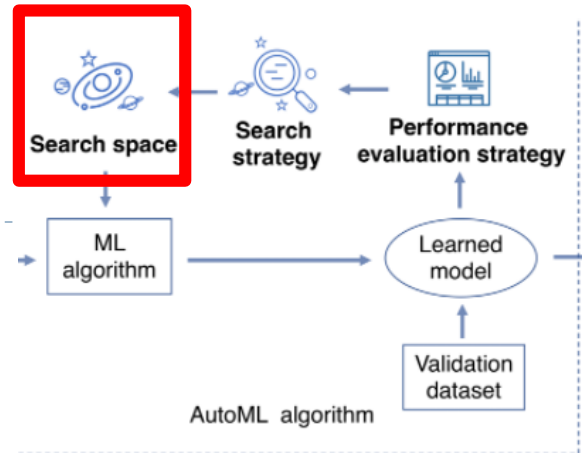
Review: Gradient descent



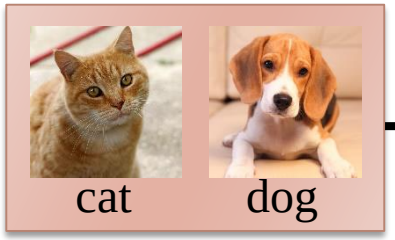
https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/meta_v3.pptx

Meta learning – Step 1

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



Training Examples



F_ϕ

F

Deep Learning

Component

Network Architecture,
Initial Parameters,
Learning Rate,
……

ϕ: learnable components

Testing

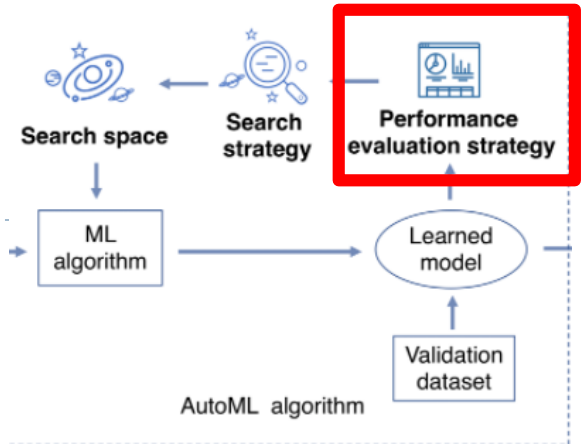
f^*

classifier

cat

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) $L(\phi)$

$L(\phi)$ ↓ 👍

Training Tasks

Task 1
Apple & Orange

Train: apple, orange

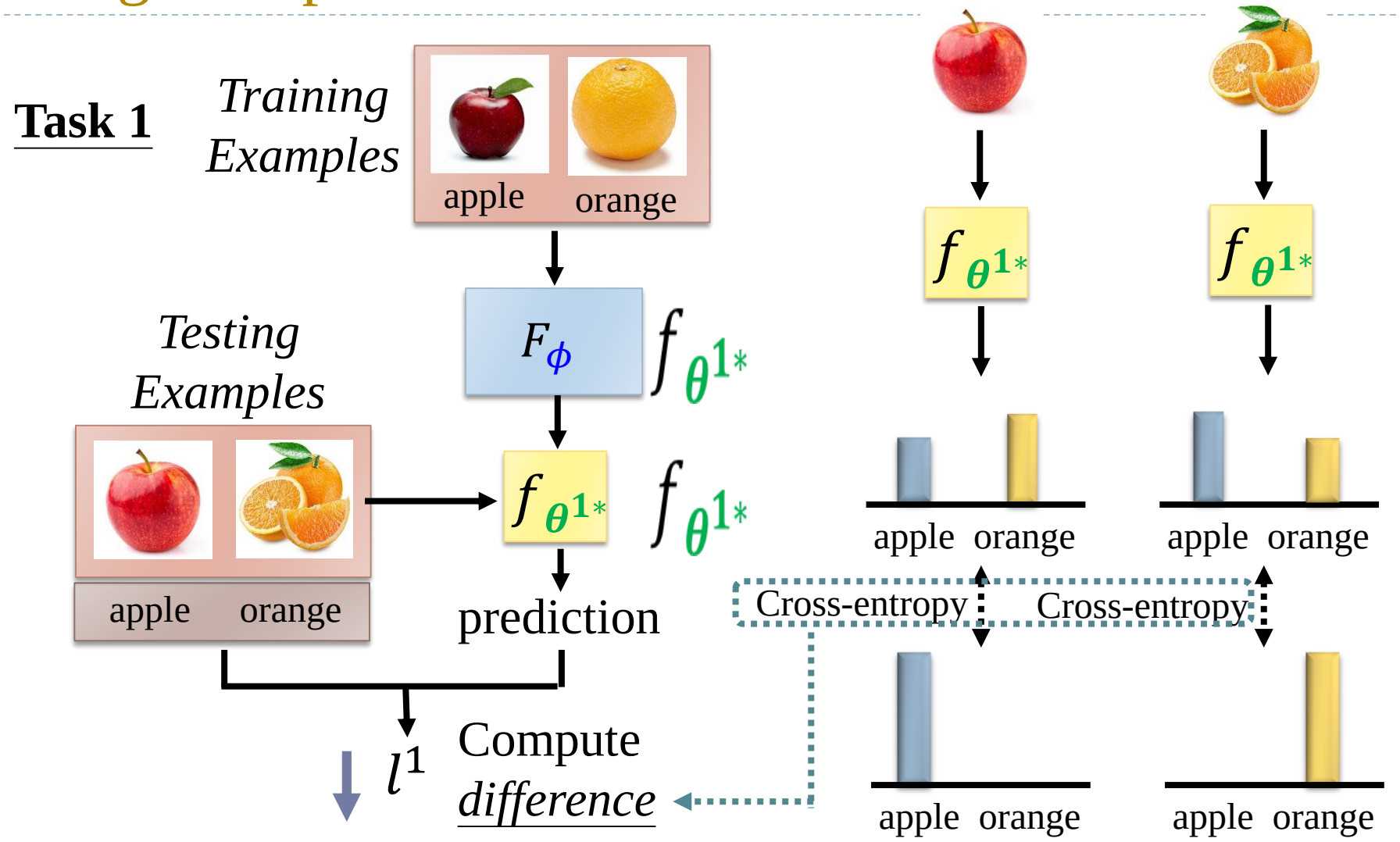
Test: apple, orange

Task 2
Car & Bike

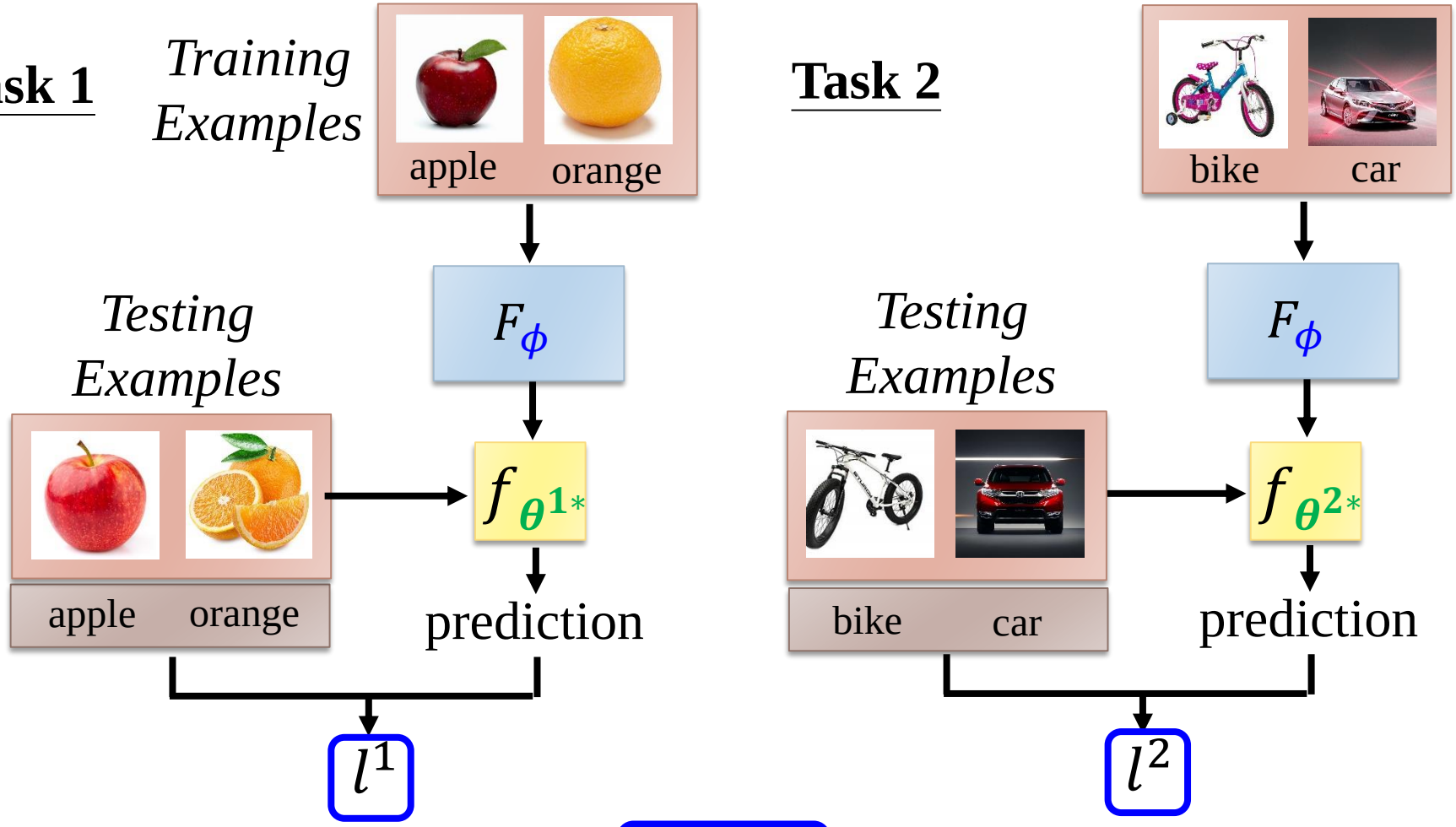
Train: bike, car

Test: bike, car

Meta learning – Step 2

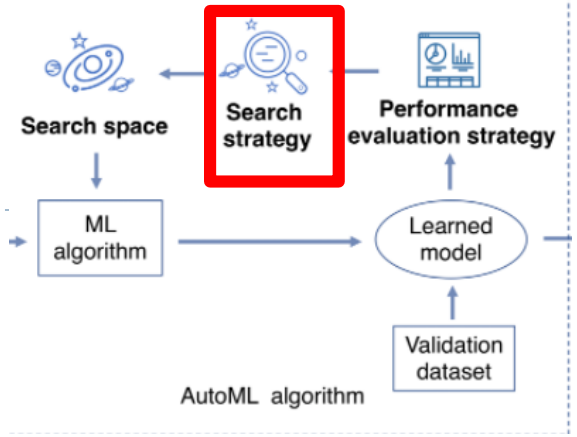


Meta learning – Step 2



Total loss: $L(\phi) = l^1 + l^2$ (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 $\partial L(\phi)/\partial\phi$

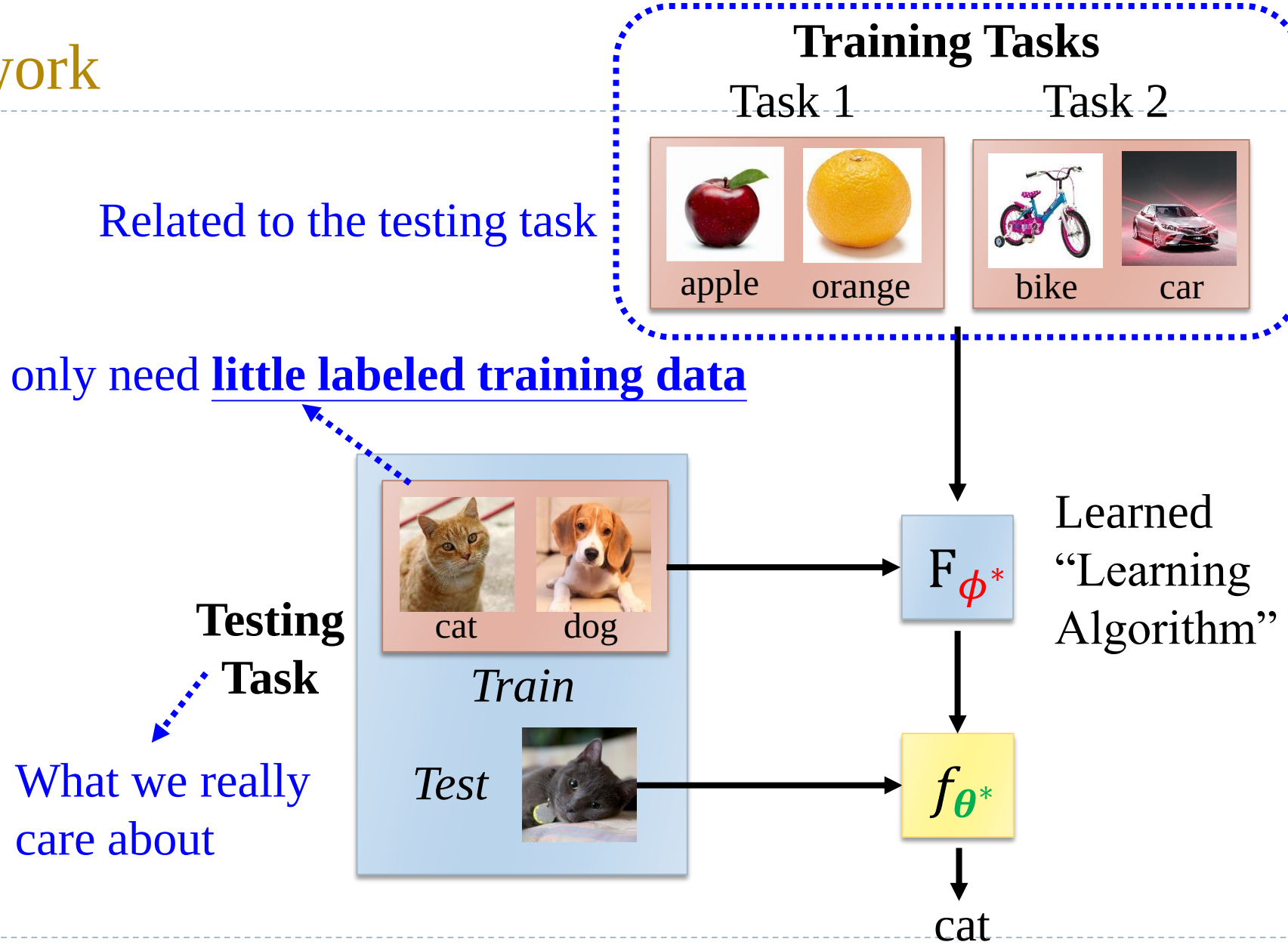
Gradient descent is your friend.

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

Framework

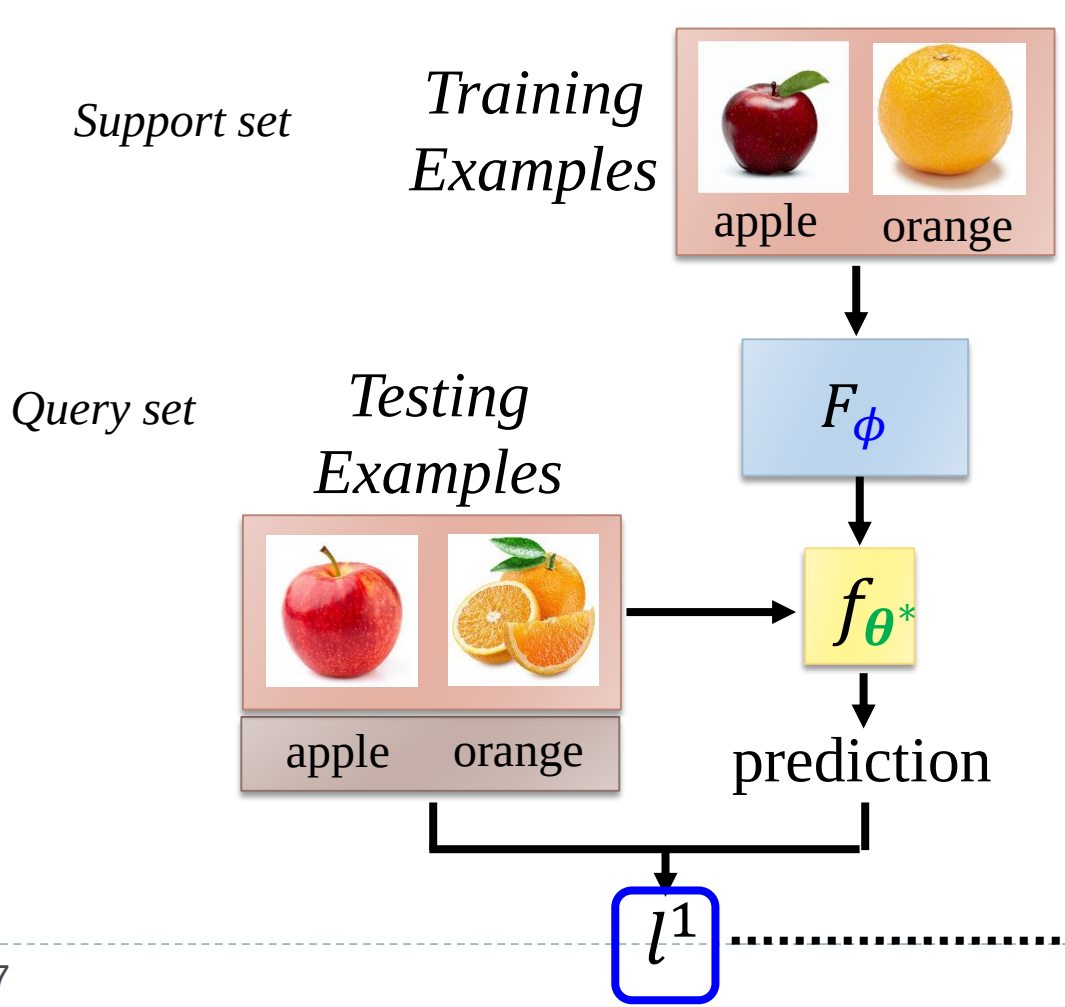
$$L(\phi) = \sum_{n=1}^{N} l^n$$

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

Outer Loop in "*Learning to initialize*"

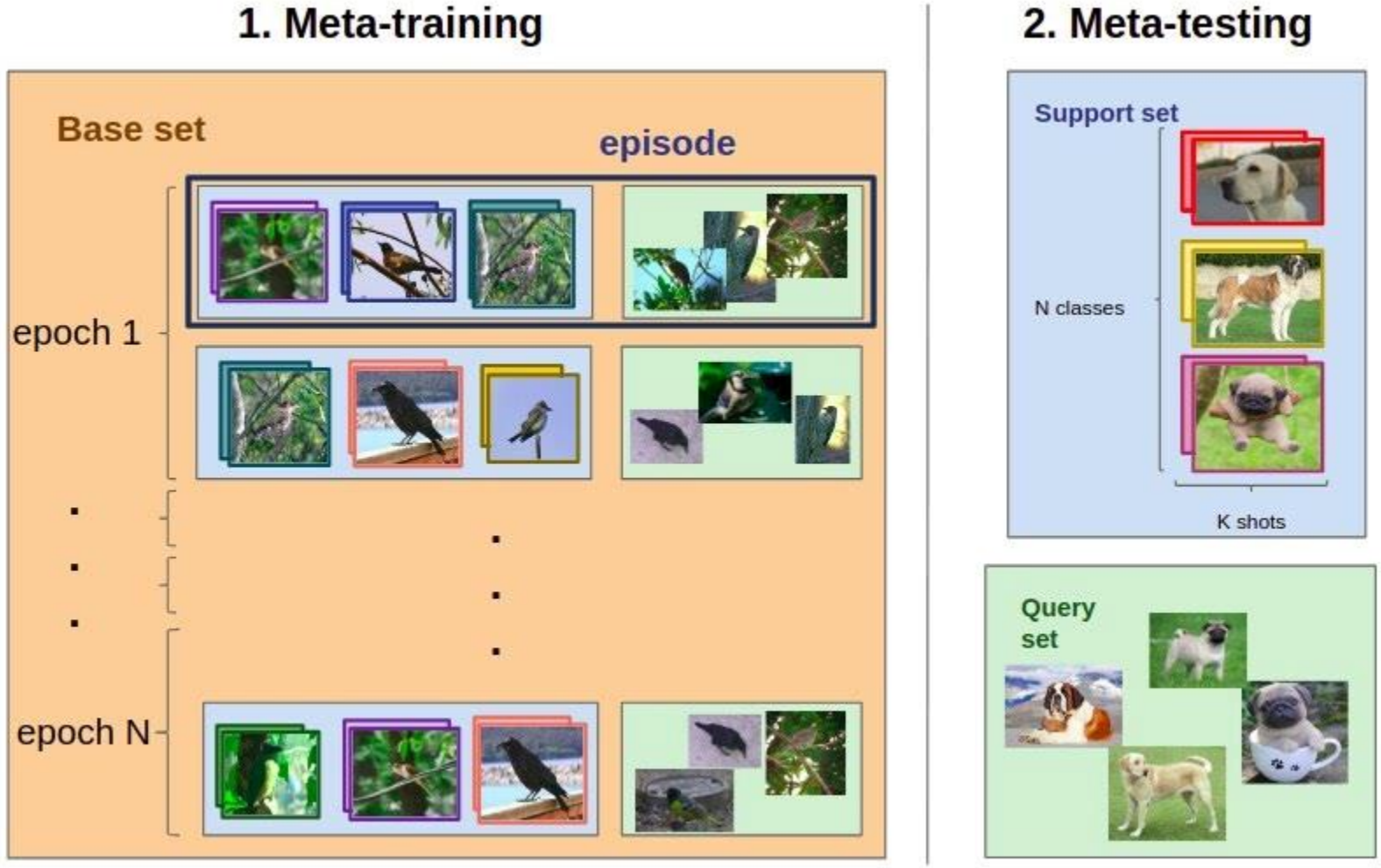
Across-task training (Meta-training) includes several **within-task training and testing**

Inner Loop in "*Learning to initialize*"



Framework

Across-task Testing



https://neptune.ai/blog/understanding-few-shot-learning-in-computer-vision