Homework 14
Regularization-based Lifelong Learning

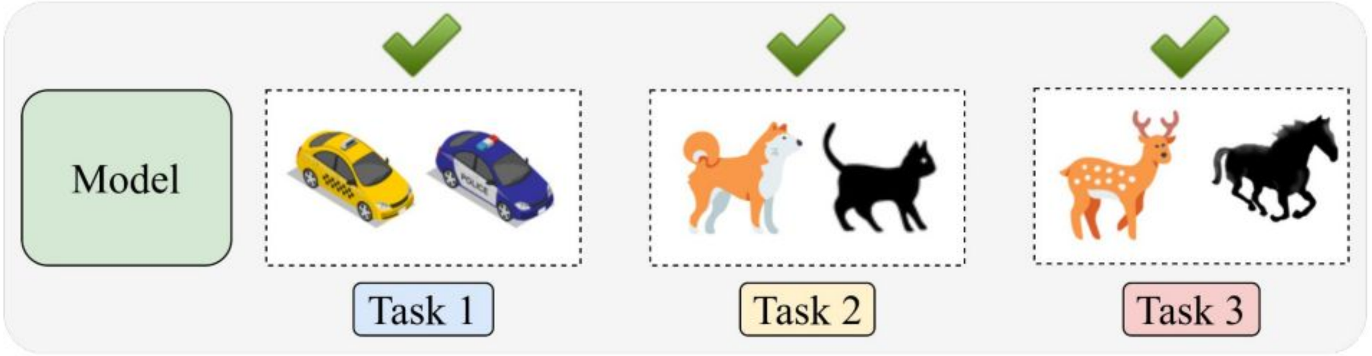
ML TAs
mlta-2023-spring@googlegroups.com

Outline

- Introduction
- Dataset
- Sample Code
- Grading
- Submission

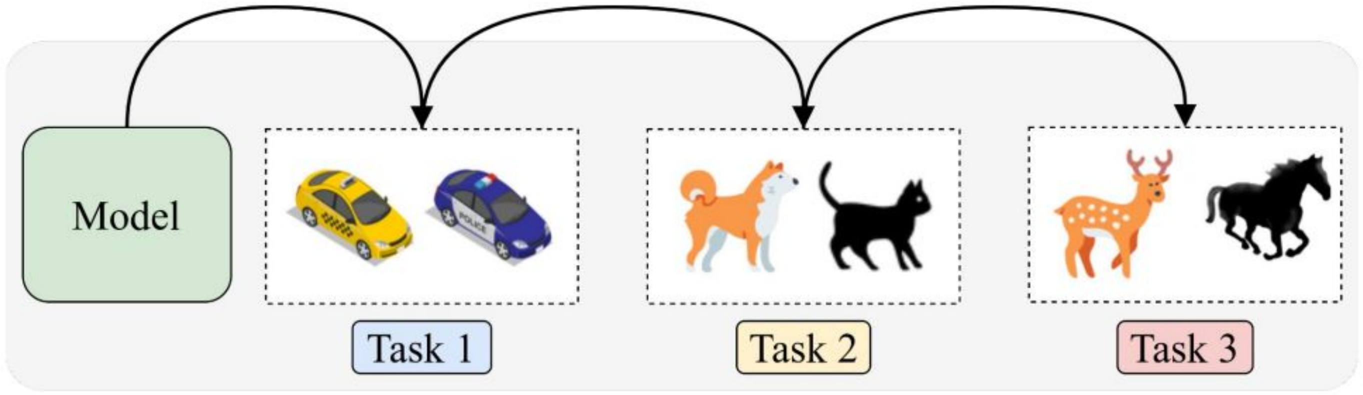
Introduction - LifeLong Learning

Goal: A model can beat all tasks!



Introduction - LifeLong Learning

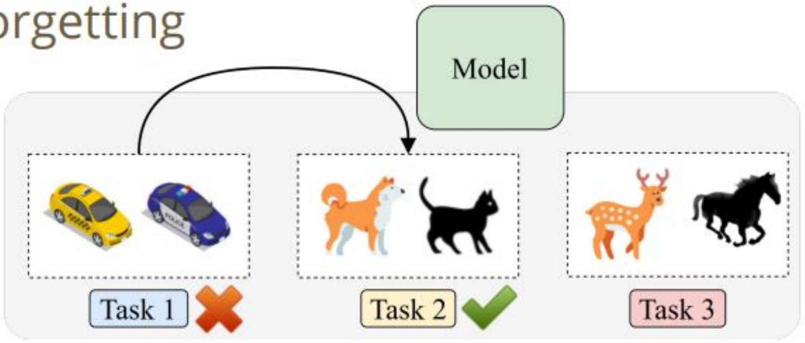
Condition: Model Sequentially Learn Different Tasks! (In Training Time)



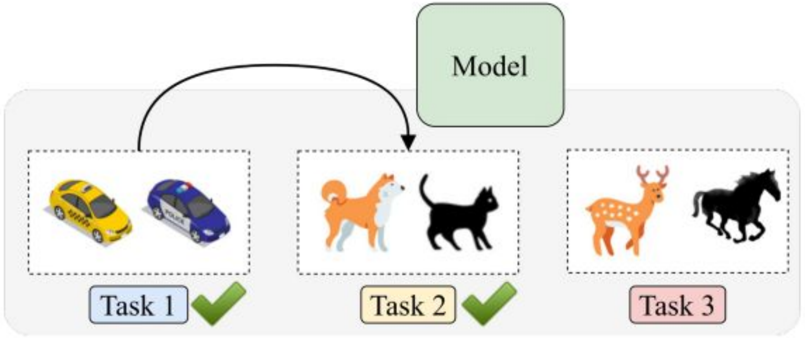
Introduction - LifeLong Learning

KeyPoint: Avoid Catastrophic Forgetting

Catastrophic Forgetting



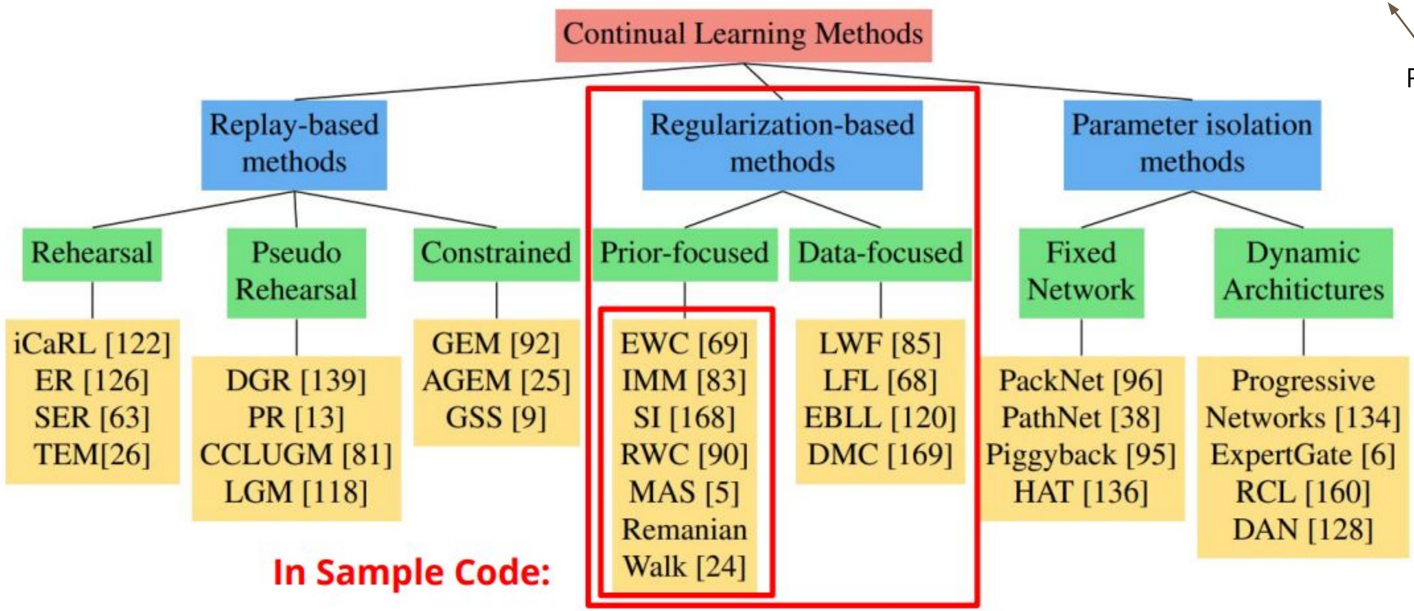
Avoid Catastrophic Forgetting



Introduction

$$L'(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda \boxed{\sum_i b_i \left(\theta_i - \theta_i^b\right)^2}$$

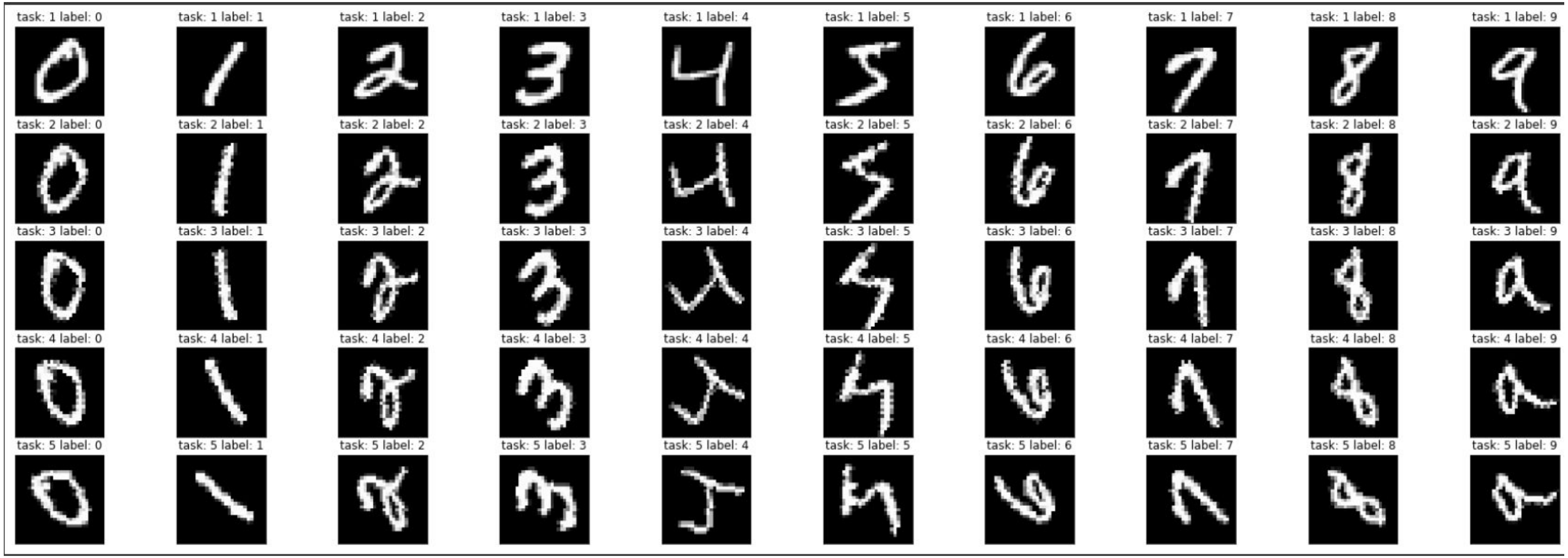
Regularization term



In Sample Code:

Dataset

Rotated MNIST (Generated by TAs)



Sample Code - Training Details

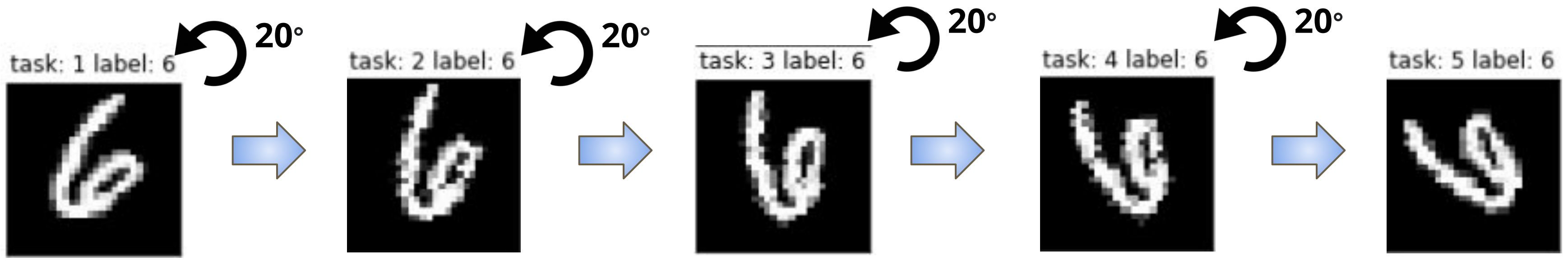
- 5 tasks / Each task has 10 epoches for training.
- Each method cost ~10 minutes for training model. (Tesla P100)
- Each method cost ~20 minutes for training model. (Tesla T4)
- Each method cost ~60 minutes for training model. (Tesla K80)
- Sample Code

Sample Code - Guideline

- Utilities
- Prepare Data
- Prepare Model
- Train and Evaluate
- Methods
- Plot function

Sample Code - Prepare Data

- Prepare Data
    - Rotation and Transformation
    - Dataloaders and Arguments
    - Visualization

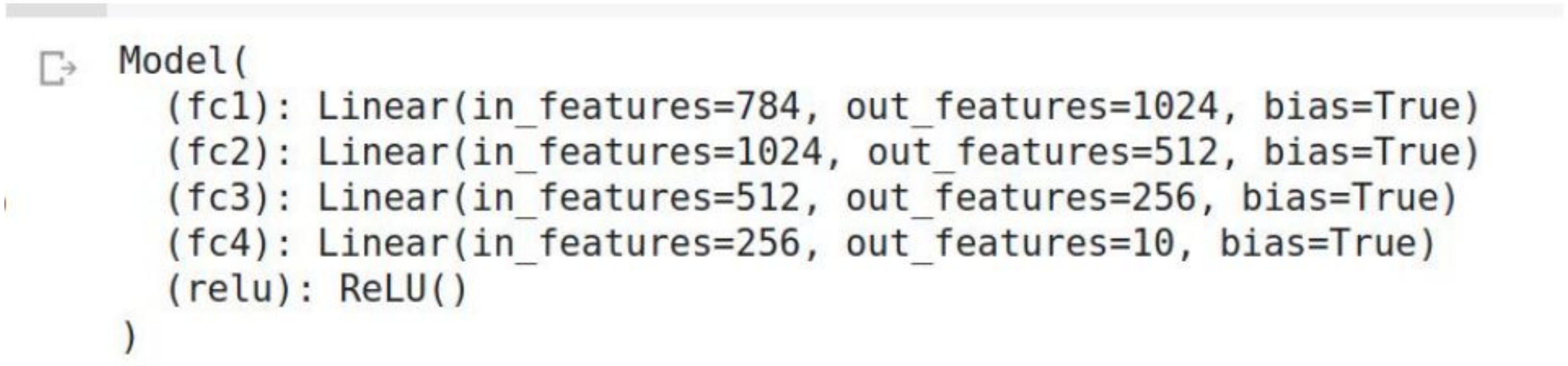


5 tasks

Sample Code - Prepare Model

- Prepare Model
  - Model Architecture

Fixed model size!



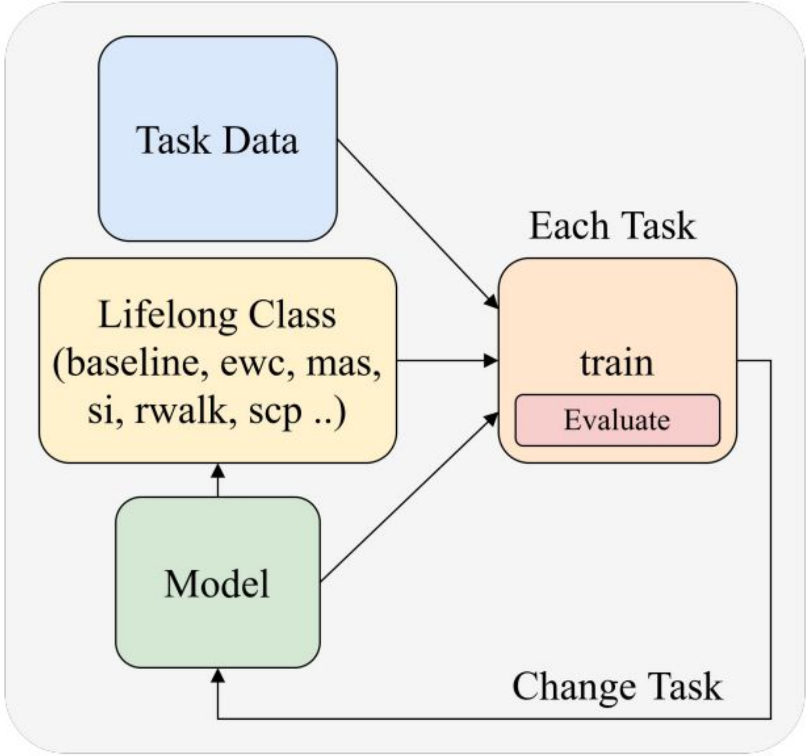
```
Model(
  (fc1): Linear(in_features=784, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=256, bias=True)
  (fc4): Linear(in_features=256, out_features=10, bias=True)
  (relu): ReLU()
)
```

Sample Code - Train and Evaluate

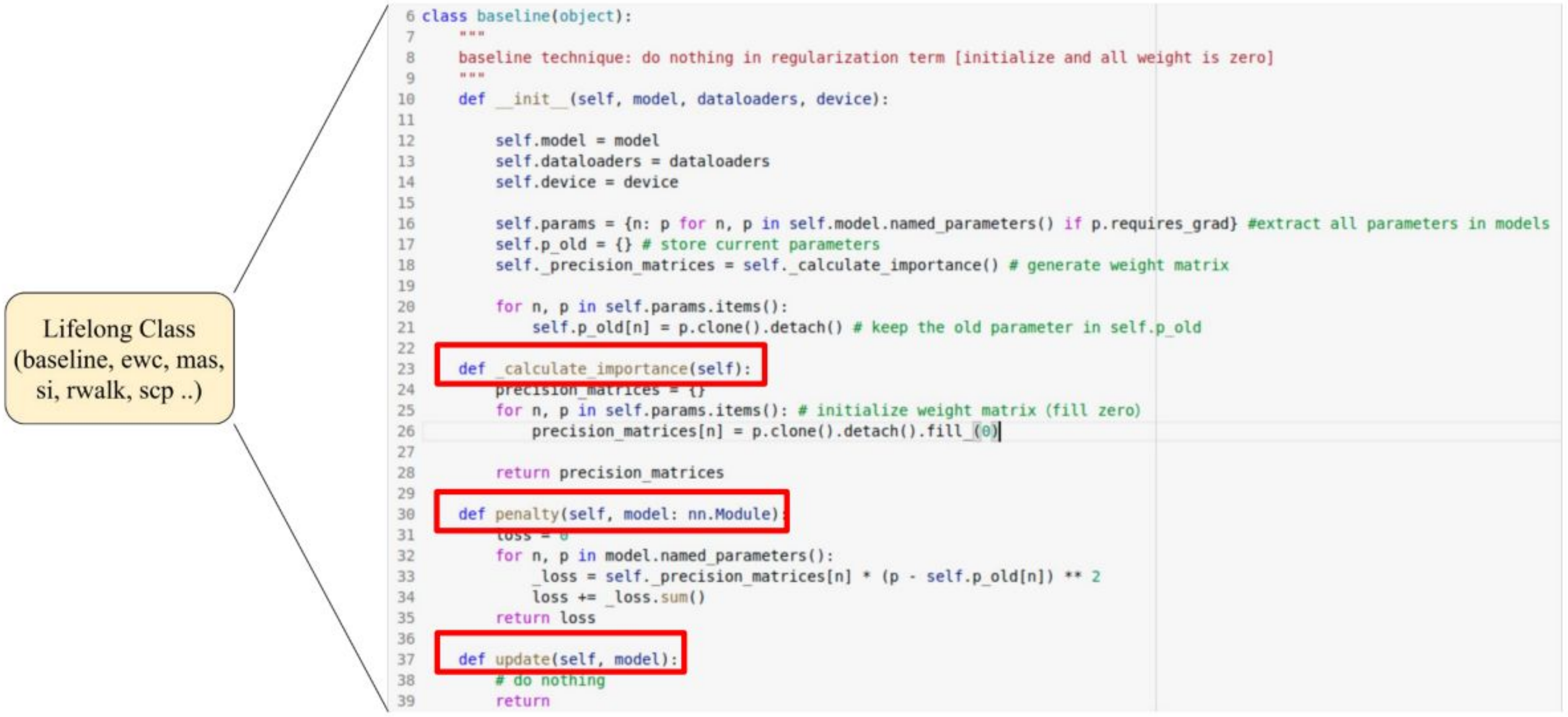
- Train:
  - Sequentially train.
  - Add regularization term and update it.
- Evaluate:
  - Evaluate by using a special metric.
  - (Please read sample code and describe it in your report.)

Sample Code - Before Going to Methods

Training Pipeline:

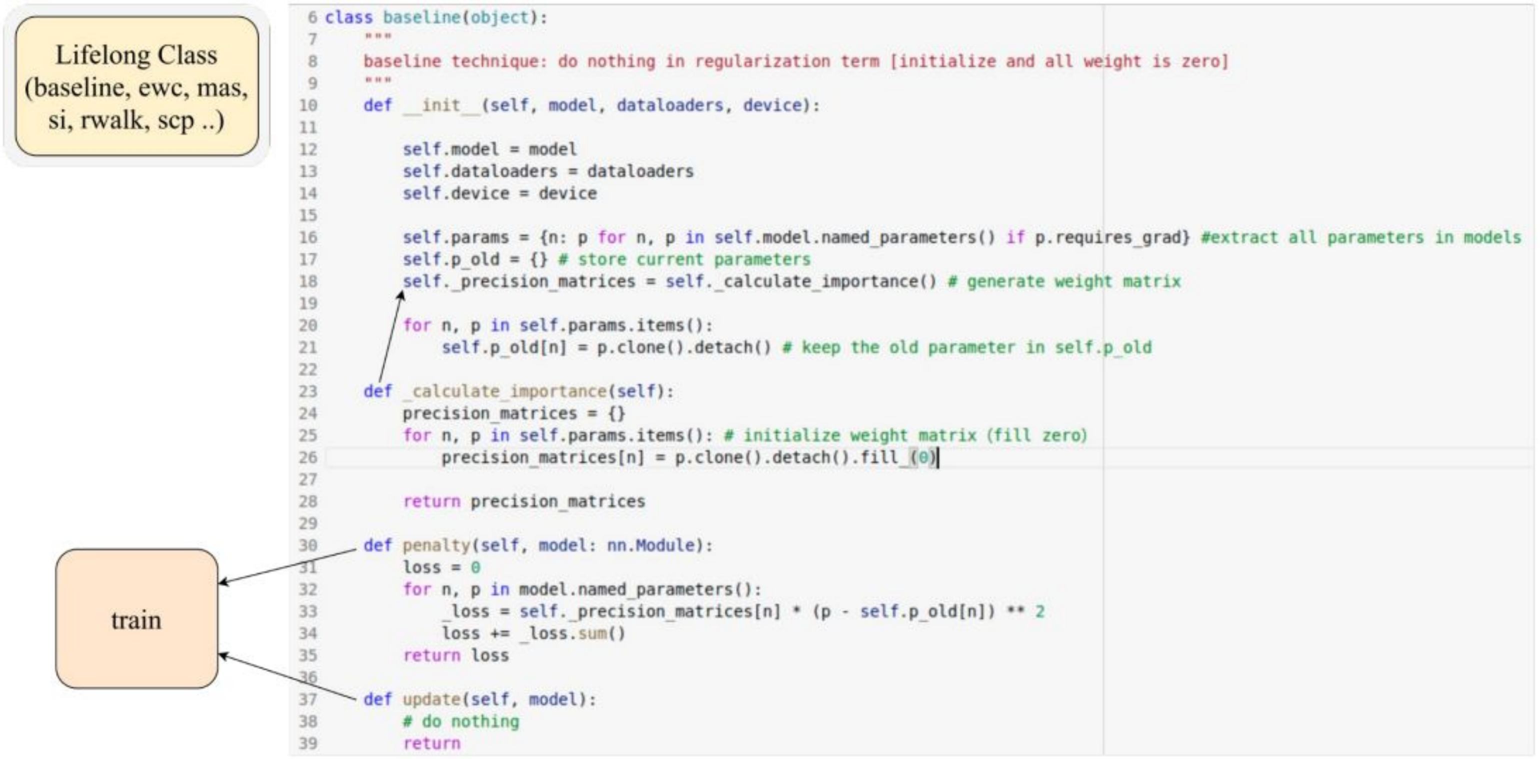


Sample Code - Before Going to Methods



```python
class baseline(object):
    """
    baseline technique: do nothing in regularization term [initialize and all weight is zero]
    """
    def __init__(self, model, dataloaders, device):

        self.model = model
        self.dataloaders = dataloaders
        self.device = device

        self.params = {n: p for n, p in self.model.named_parameters() if p.requires_grad} #extract all parameters in models
        self.p_old = {} # store current parameters
        self._precision_matrices = self._calculate_importance() # generate weight matrix

        for n, p in self.params.items():
            self.p_old[n] = p.clone().detach() # keep the old parameter in self.p_old

    def _calculate_importance(self):
        precision_matrices = {}
        for n, p in self.params.items(): # initialize weight matrix (fill zero)
            precision_matrices[n] = p.clone().detach().fill_(0)

        return precision_matrices

    def penalty(self, model: nn.Module):
        loss = 0
        for n, p in model.named_parameters():
            _loss = self._precision_matrices[n] * (p - self.p_old[n]) ** 2
            loss += _loss.sum()
        return loss

    def update(self, model):
        # do nothing
        return
```

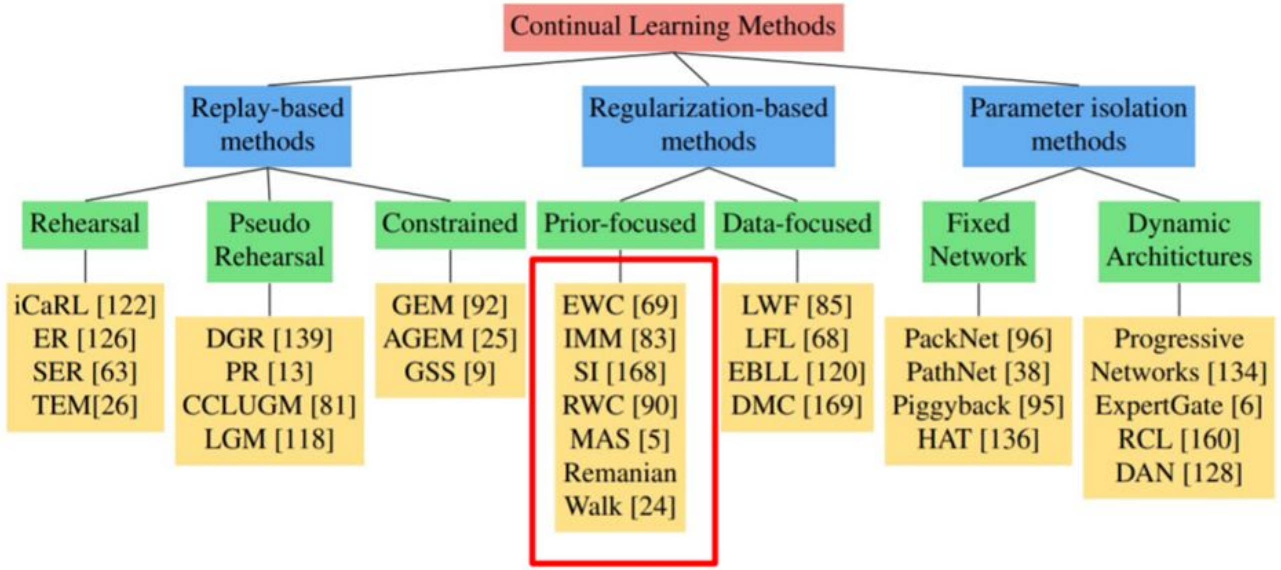
Sample Code - Before Going to Methods



```python
class baseline(object):
    """
    baseline technique: do nothing in regularization term [initialize and all weight is zero]
    """
    def __init__(self, model, dataloaders, device):

        self.model = model
        self.dataloaders = dataloaders
        self.device = device

        self.params = {n: p for n, p in self.model.named_parameters() if p.requires_grad} #extract all parameters in models
        self.p_old = {} # store current parameters
        self._precision_matrices = self._calculate_importance() # generate weight matrix

        for n, p in self.params.items():
            self.p_old[n] = p.clone().detach() # keep the old parameter in self.p_old

    def _calculate_importance(self):
        precision_matrices = {}
        for n, p in self.params.items(): # initialize weight matrix (fill zero)
            precision_matrices[n] = p.clone().detach().fill_(0)

        return precision_matrices

    def penalty(self, model: nn.Module):
        loss = 0
        for n, p in model.named_parameters():
            _loss = self._precision_matrices[n] * (p - self.p_old[n]) ** 2
            loss += _loss.sum()
        return loss

    def update(self, model):
        # do nothing
        return
```

Sample Code - Methods

- Baseline (Do nothing in regularization term)
- EWC
- MAS
- SI
- RWalk
- SCP


Continual Learning Methods
Replay-based methods
Regularization-based methods
Parameter isolation methods
Rehearsal
Pseudo Rehearsal
Constrained
Prior-focused
Data-focused
Fixed Network
Dynamic Architictures
iCaRL [122] ER [126] SER [63] TEM[26]
DGR [139] PR [13] CCLUGM [81] LGM [118]
GEM [92] AGEM [25] GSS [9]
EWC [69] IMM [83] SI [168] RWC [90] MAS [5] Remanian Walk [24]
LWF [85] LFL [68] EBLL [120] DMC [169]
PackNet [96] PathNet [38] Piggyback [95] HAT [136]
Progressive Networks [134] ExpertGate [6] RCL [160] DAN [128]

EWC - Elastic Weight Consolidation

1. You need to know how to generate Guardiance weight from EWC!

2. Do this method need to use label ?

3. Hint: (Trace the class ewc and its calculate_importance function)

Paper Link: https://arxiv.org/abs/1612.00796

MAS - Memory Aware Synapse

1. You need to know how to generate Guardiance weight from MAS!
2. Do this method need to use label ?
3. We want you to implement Omega Matrix for MAS! Please read page 21 carefully and paste your code (only TODO block) in report.

Please do not modify any part of the sample code except the TODO block.

Paper Link: https://arxiv.org/abs/1711.09601

MAS - Memory Aware Synapse

- The method proposed in the paper is the local version by taking squared L2-norm outputs from each layer of the model.
- Here we only want you to implement the global version by taking outputs from the last layer of the model.
- Hint: (It is similar to the way you generate the Fisher matrix for EWC, the only difference is the calculation of the important weight.)

$$\mathcal{L}_B = \mathcal{L}(\theta) + \sum_i \frac{\lambda}{2}\Omega_i(\theta_i - \theta^*_{A,i})^2$$

Paper Link: https://arxiv.org/abs/1711.09601

$$\Omega_i = ||\frac{\partial \ell_2^2(M(x_k;\theta))}{\partial \theta_i}||$$

SI - Synaptic Intelligence

1. You need to know how to generate Guardiance weight from SI!
2. Do this method need to use label ?
3. Hint: (Accumulated loss change in each update step)

Paper Link: https://arxiv.org/abs/1703.04200, Talk Slide

SI - Main Idea

$$L(\theta) = L_2(\theta) + \boxed{c\sum_i \Omega_i \left(\theta_i - \boxed{\theta^*_{1,i}}\right)^2}$$

From learning trajectory

Parameter importance on-line from learning trajectory!

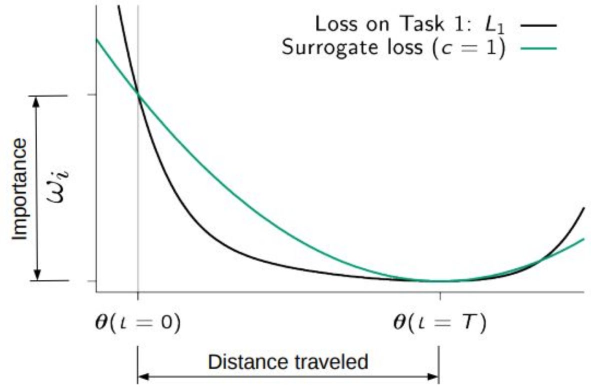
Picture comes from: Talk Slide

SI - Main Idea

Leveraging per-parameter importance for continual learning

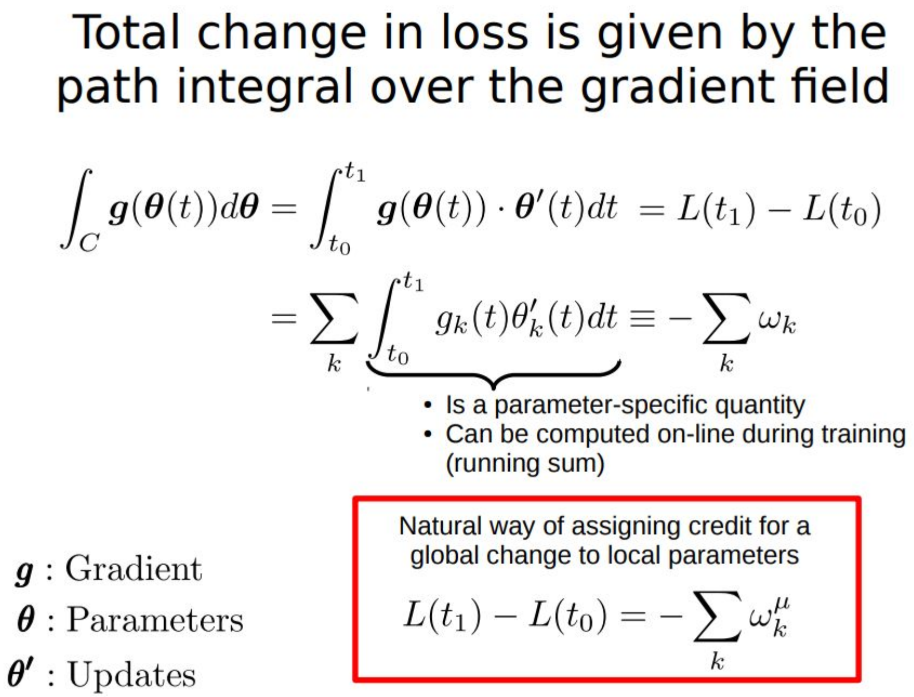
$$L(\theta) = L_2(\theta) + c \sum_i \Omega_i \left(\theta_i - \theta^*_{1,i}\right)^2$$

$$\boxed{\Omega_i \equiv \frac{\omega_i}{(\Delta_i)^2 + \epsilon}}$$



Picture comes from: Talk Slide

SI - Main Idea



Total change in loss is given by the path integral over the gradient field

$$\int_C \boldsymbol{g}(\boldsymbol{\theta}(t))d\boldsymbol{\theta} = \int_{t_0}^{t_1} \boldsymbol{g}(\boldsymbol{\theta}(t)) \cdot \boldsymbol{\theta}'(t)dt = L(t_1) - L(t_0)$$

$$= \sum_k \underbrace{\int_{t_0}^{t_1} g_k(t)\theta_k'(t)dt}_{} \equiv -\sum_k \omega_k$$

- Is a parameter-specific quantity
- Can be computed on-line during training (running sum)

Natural way of assigning credit for a global change to local parameters

$$L(t_1) - L(t_0) = -\sum_k \omega_k^\mu$$

$\boldsymbol{g}$: Gradient

$\boldsymbol{\theta}$: Parameters

$\boldsymbol{\theta}'$: Updates

Picture comes from: Talk Slide

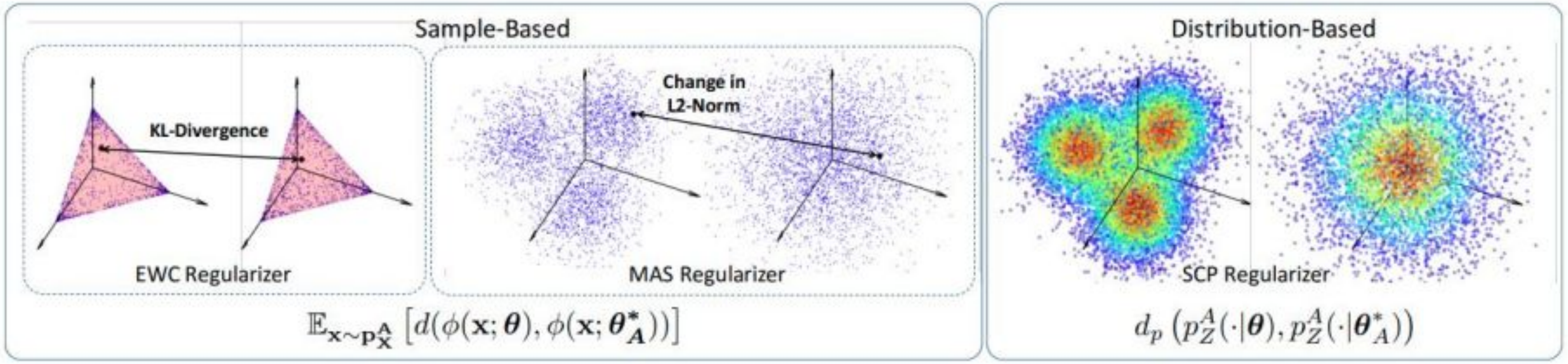
RWalk - Remanian Walk

1. Trace Rwalk class and its update function!
2. Do this method need to use label ?
3. Hint: (The code is similar to two method which mentioned in sample code)

Paper Link: https://arxiv.org/abs/1801.10112

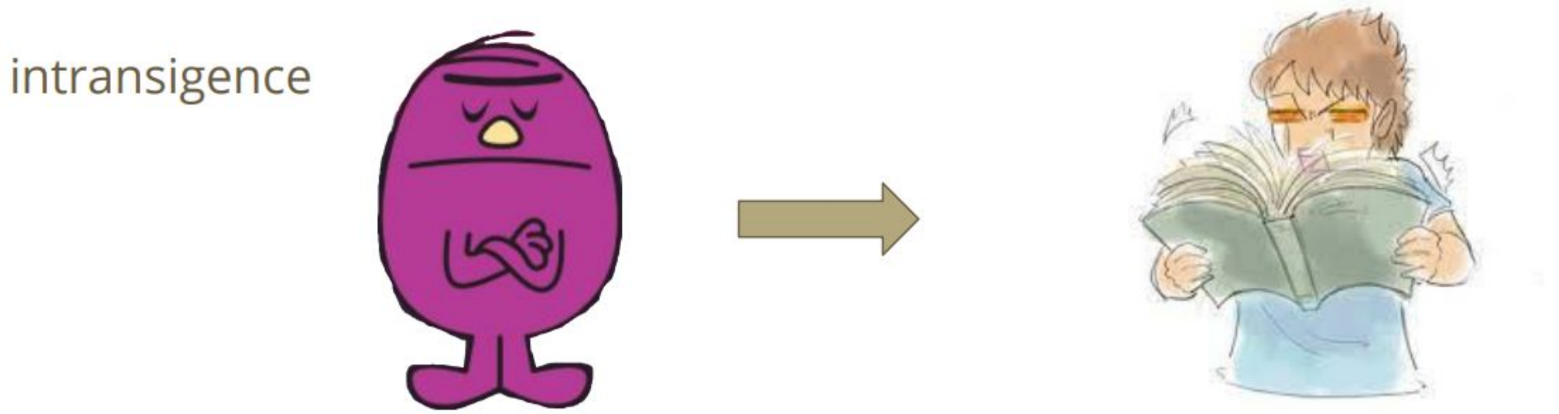
SCP - Sliced Cramer Preservation

1. Paper Link: https://openreview.net/pdf?id=BJge3TNKwH
2. Do this method need to use label ?


Sample-Based
KL-Divergence
EWC Regularizer
Change in L2-Norm
MAS Regularizer
$\mathbb{E}_{\mathbf{x}\sim \mathbf{p}_{\mathbf{X}}^{\mathbf{A}}}\left[d(\phi(\mathbf{x};\boldsymbol{\theta}),\phi(\mathbf{x};\boldsymbol{\theta}_A^*))\right]$
Distribution-Based
SCP Regularizer
$d_p\left(p_Z^A(\cdot|\boldsymbol{\theta}),p_Z^A(\cdot|\boldsymbol{\theta}_A^*)\right)$

SCP - Main Idea

- Propose Distributed-based Distance to prevent fast intransigence and avoid overestimating the importance of parameters.



Model do not want to learn new task, and it just keep old task performance

Other Methods and Scenarios

- Only in Multiple Choice Questions
- iCaRL (https://arxiv.org/abs/1611.07725)
- LwF (https://arxiv.org/abs/1606.09282)
- GEM (https://arxiv.org/abs/1706.08840)
- DGR (https://arxiv.org/abs/1705.08690)
- Three scenarios for continual learning (https://arxiv.org/abs/1904.07734)

Grading

- 25 multiple choice questions **(7.5pts, 0.3pt each)**
- Report **(2.5pts)**
- You have to choose ALL the correct answers for each question
- No leaderboards are needed!!

Grading - Multiple Choice Questions

- 25 multiple choice questions **(7.5pts, 0.3pt each)**
  - Basic Concept: 5 Questions
  - EWC: 3 Questions
  - MAS: 3 Questions
  - SI: 3 Questions
  - RWalk: 3 Questions
  - SCP: 3 Questions
  - Other Methods & scenarios: 5 Questions
    - ICaRL, LwF, GEM, DGR
    - Three Scenarios

Grading - Report

- Plot the learning curve of the metric with MAS method. (**The Plotting function is provided in the sample code, you can answer all the methods' learning curve, but it should include MAS.**) **(0.5pt)**
- Describe the metric. **(1pt) (You should mention how to measure the previous and current test set.)**
- Paste the code that you implement Omega Matrix for MAS. **(1pt) (Your code should able to be reproduced by TA when pasting back to MAS block.)**

Please do not modify any part of the sample code except the TODO block.

Please just paste the TODO block.

If you plot the right learning curve, you will still get the point of the first part no matter whether you implement Omega Matrix for MAS or not.

Submission

- The questions are on **gradescope**
- Submit your report to **gradescope**
- Running the code may need some time!
- You can answer the questions unlimited times
- The length of anwsering time of the assignment is unlimited
- We will consider the latest submission as the final score
- You will see the scores after the deadline only!
- **No late submission!**
- **Remember to save the answer when answering the questions!**
- Deadline: **2023/06/30 23:59**

Link

- Code: Kaggle 、Colab

If any questions, you can ask us via...

- NTU COOL (Recommended)
- Email
  - mlta-2023-spring@googlegroups.com
  - The title should begin with “[hw14]”
- TA hour
  - Monday 19:00 - 21:00
  - Friday 19:00 - 20:00