13-Life Long Learning（終身學習）

1. Basic

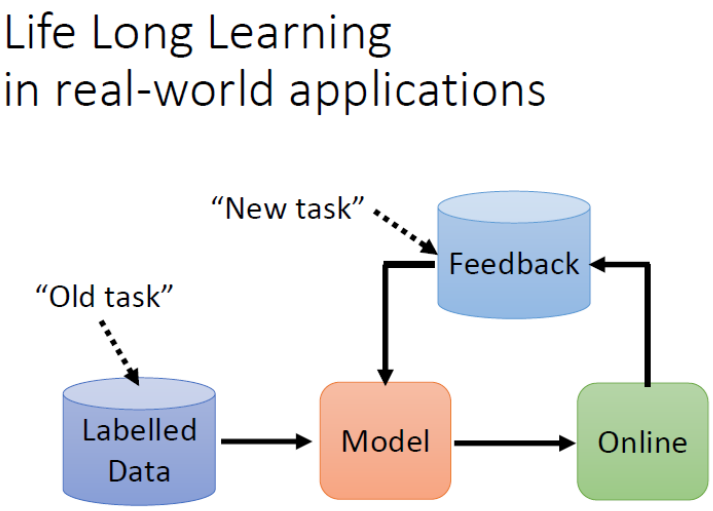
Life Long Learning（LLL），期望機器不斷學習新的任務，又稱為 **Continual Learning**、**Never Ending Learning**、 Incremental Learning

注意：

在 LLL 中，讓機器不斷學習"不同的任務"，實際上是學習類似的任務但**"不同的 domain"**，只是習慣當成不同的任務

1.1 現實世界的應用

模型上線後蒐集到新的資料，新的資料就可以更新模型的參數



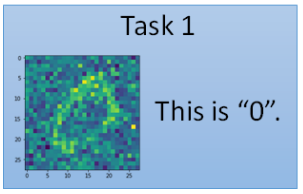
把舊有的資料想成是**過去的任務**，把新的、來自於使用者 feedback 的資料想成是**新的任務**

1.2 Catastrophic Forgetting

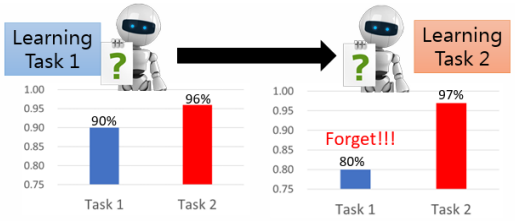
核心概念機器是**學了新的任務（資料）就忘了舊的任務（資料）**

1.2.1 Example 1：數字辨識

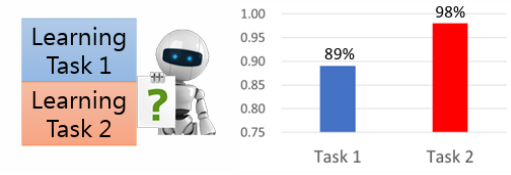
針對兩個 task，訓練一個模型對不同的數字進行識別



- **先後利用兩個任務的資料進行訓練**，以 task 2 的資料訓練模型後，機器在 task 1 的表現下降許多



- **同時利用兩個任務的資料進行訓練**，機器在兩個 task 上都表現不錯



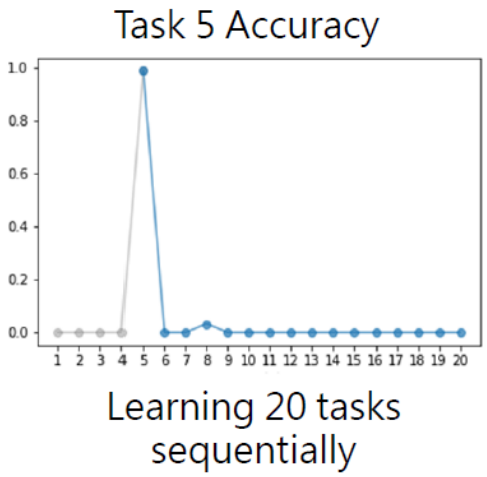
1.2.2 Example 2：QA 任務

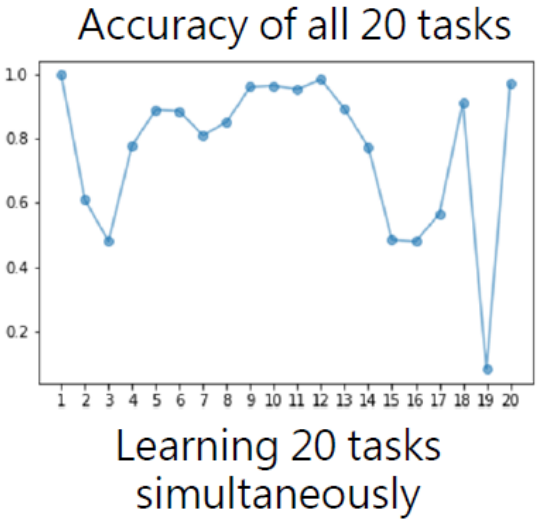
- QA: Given a document, answer the question based on the document.
- There are 20 QA tasks in bAbi corpus.

Task 5: Three Argument Relations	Task 15: Basic Deduction
Mary gave the cake to Fred. Fred gave the cake to Bill. Jeff was given the milk by Bill. Who gave the cake to Fred? A: Mary Who did Fred give the cake to? A: Bill	Sheep are afraid of wolves. Cats are afraid of dogs. Mice are afraid of cats. Gertrude is a sheep. What is Gertrude afraid of? A:wolves

- Train a QA model through the 20 tasks

- 依序學 20 個任務，學到 task 5 時的準確率提升，但自 task 6 後準確率暴跌
- 同時學 20 個任務

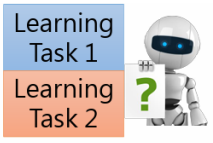




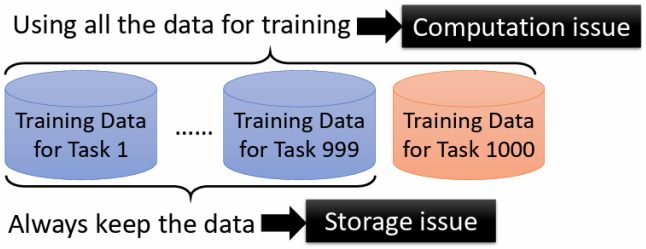
1.3 Multi-Task Training

讓機器**同時利用多個任務的資料進行學習**的訓練方法稱為 **multi-task training**

雖然某種程度上可以解決上述 catastrophic forgetting 的問題，但當機器要學新的任務時，都要**把之前學過的任務資料也都重新一起帶入訓練**，時間和空間難以負荷



- Multi-task training can solve the problem!

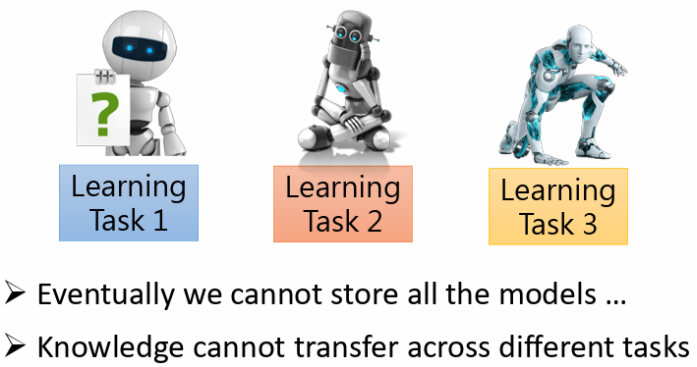


- Multi-task training can be considered as the upper bound of LLL.

multi-task training 雖不切實際，但它確實可以讓機器學會多個任務，因而**被視為 LLL 的 upper bound，是 LLL 沒有辦法超越的結果**

研究目標會希望 LLL 的技術去逼近 upper bound 的結果

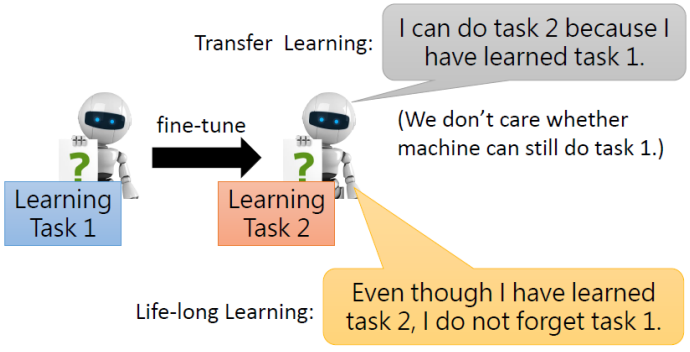
1.4 Train a model for each task



問題：

- 模型過多無法儲存
- 訊息和資料不能在不同的任務間遷移

1.5 Life-long learning vs Transfer Learning



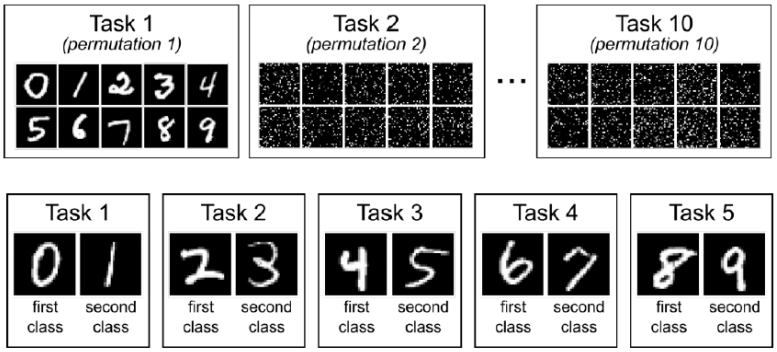
兩者的關注角度不同：

- Life Long Learning 關注的是機器學完新的任務後，還能不能夠解先前的任務
- Transfer Learning 關注的是機器學習完先前的任務後，能不能夠對新的任務有幫助

2. Evaluation

先準備一系列的任務

First of all, we need a sequence of tasks.



機器依序學習各個不同的任務，每學完一個新任務就在所有任務的測試資料上計算正確率

Evaluation

$R_{i,j}$: after training task i, performance on task j

If $i > j$,

After training task i, does task j be forgot

If $i < j$,

Can we transfer the skill of task i to task j

		Test on			
		Task 1	Task 2	.....	Task T
Rand Init.		$R_{0,1}$	$R_{0,2}$		$R_{0,T}$
After Training	Task 1	$R_{1,1}$	$R_{1,2}$		$R_{1,T}$
	Task 2	$R_{2,1}$	$R_{2,2}$		$R_{2,T}$
	⋮				
	Task T-1	$R_{T-1,1}$	$R_{T-1,2}$		$R_{T-1,T}$
	Task T	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

$$\text{Accuracy} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}$$

$$\text{Backward Transfer} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$$

$$\text{Forward Transfer} = \frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - R_{0,i}$$

$R_{i,j}$ 表示學完任務 i，在任務 j 上的表現

- 若 $i > j$，可以觀察機器學了任務 i 後，在之前學過的任務 j 上的表現
- 若 $i < j$，可以觀察機器還沒學任務 j，只學到任務 i，在任務 j 上的表現

三種評估方法：

- **Accuracy**：學完所有任務後，各個任務的準確率

		Test on			
		Task 1	Task 2	.....	Task T
Rand Init.		$R_{0,1}$	$R_{0,2}$		$R_{0,T}$
After Training	Task 1	$R_{1,1}$	$R_{1,2}$		$R_{1,T}$
	Task 2	$R_{2,1}$	$R_{2,2}$		$R_{2,T}$
	⋮				
	Task T-1	$R_{T-1,1}$	$R_{T-1,2}$		$R_{T-1,T}$
	Task T	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

- **Backward Transfer**

		Test on			
		Task 1	Task 2	.....	Task T
Rand Init.		$R_{0,1}$	$R_{0,2}$		$R_{0,T}$
After Training	Task 1	$R_{1,1}$	$R_{1,2}$		$R_{1,T}$
	Task 2	$R_{2,1}$	$R_{2,2}$		$R_{2,T}$
	⋮				
	Task T-1	$R_{T-1,1}$	$R_{T-1,2}$		$R_{T-1,T}$
	Task T	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

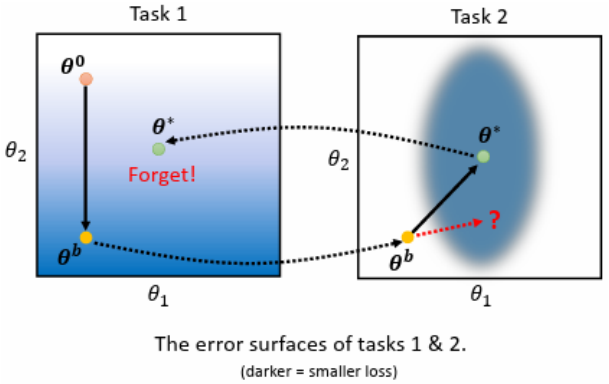
- **Forward Transfer**：不看任務 T，只看前 T-1 個任務，對在任務 T 的表現

		Test on			
		Task 1	Task 2	.....	Task T
Rand Init.		$R_{0,1}$	$R_{0,2}$		$R_{0,T}$
After Training	Task 1	$R_{1,1}$	$R_{1,2}$		$R_{1,T}$
	Task 2	$R_{2,1}$	$R_{2,2}$		$R_{2,T}$
	⋮				
	Task T-1	$R_{T-1,1}$	$R_{T-1,2}$		$R_{T-1,T}$
	Task T	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

3. Why Catastrophic Forgetting ?

關鍵：

任務間的 **error surface 不同**



The error surfaces of tasks 1 & 2.
(darker = smaller loss)

兩個任務的 error surface 如上圖，顏色越深，代表 loss 越小

1. 先用 task 1 訓練模型，隨機初始化參數 θ^0，用 gradient descent 更新參數得到 θ^b
2. 將 θ^b 作為初始參數用 task 2 訓練模型，使用 gradient descent 更新參數得到 θ^*

問題：

由於兩個任務的 error surface 不同，**擁有小的 loss 所對應的參數不同，導致 catastrophic forgetting**

解決：

嘗試讓參數往 task 2 的橢圓 error surface 下緣靠近

4. LLL 的三種解法

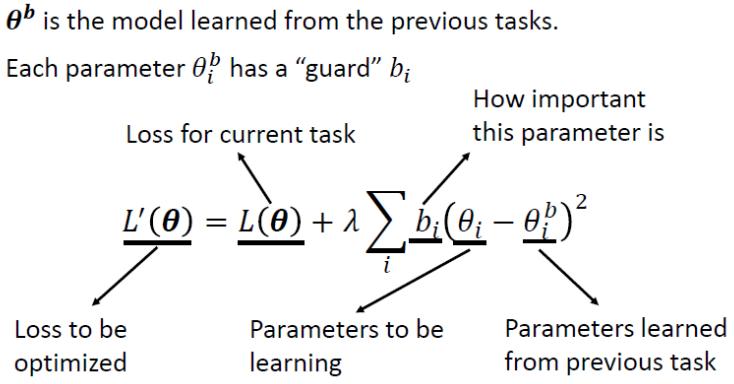
4.1 Selective Synaptic Plasticity（Regularization based Approach）

每個參數對過去學過的任務的**重要性不盡相同**

對於舊的任務所學習到的參數有"重要"與"不重要"之別，在學習新的任務時，針對**重要的參數盡量維持不變，只要改變不重要的參數**

4.1.1 改寫 Loss Function

對模型的每一個參數 θ_i^b 新增參數 guard b_i

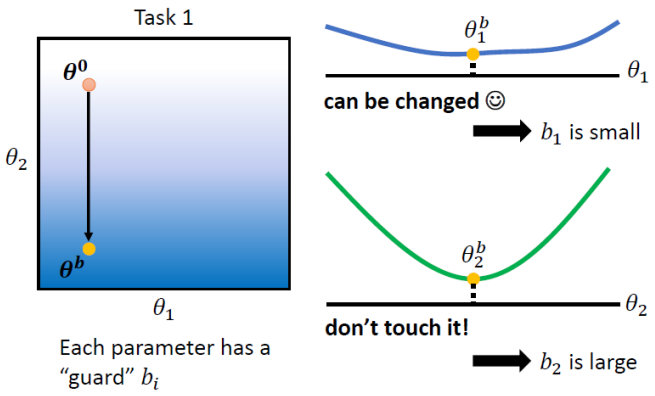


在新的任務原始的 loss $L(\theta)$ 後再加上新學到的參數 θ_i 減去過去學到參數 θ_i^b 並取平方，再乘上當前參數的 guard b_i，最後針對每一個參數的結果做加總

- 若 b_i **越大**，表示**越希望 θ_i 與 θ_i^b 值越接近**
  - $b_i = \infty$，強烈希望 θ_i 與 θ_i^b 相等，會導致 **intransigence**（在新的任務學不好）
- 若 b_i **越小**，表示**不在乎 θ_i 與 θ_i^b 接不接近**
  - $b_i = 0$，對 θ_i 沒有限制，會導致 **catastrophic forgetting**

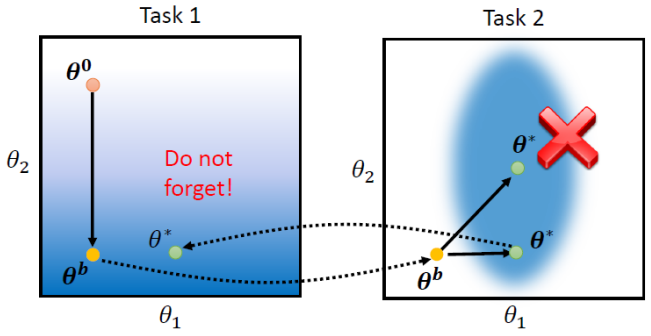
4.1.2 設定 guard b

人工設定 guard b_i，在 task 1 上訓練完後得到 θ^b，逐一調整 θ^b 的每一個參數，觀察在新任務上 loss 的變化



舉例：

- 針對 task 2 調整 θ_1，發現**對 task 1 的 loss 影響較小**，代表 θ_1 **對 task 1 較不重要**，因此**將 b_1 設較小的值**
- 針對 task 2 調整 θ_2，發現**對 task 1 的 loss 影響較大**，代表 θ_2 **對 task 1 較重要**，因此**將 b_2 設較大的值**



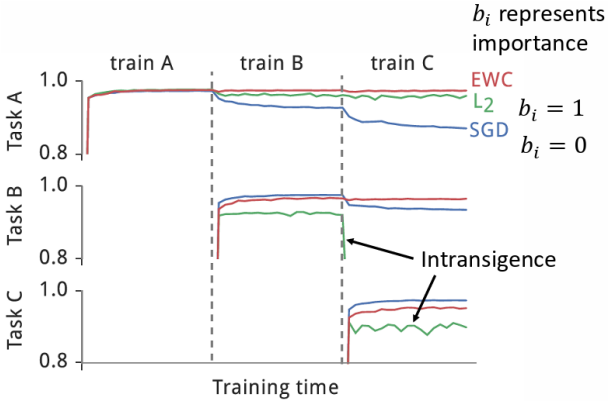
由於 b_1 較小、b_2 較大，因此參數在 task 2 上更新時，會更傾向於改變 θ_1 而不改變 θ_2，如此得到的 θ^* 在 task 1 和 task 2 上都可以有較好的結果

如何計算 b_i？

- Elastic Weight Consolidation（EWC）
- Synaptic Intelligence（SI）
- Memory Aware Synapses（MAS）
- RWalk
- Sliced Cramer Preservation（SCP）

4.1.3 實驗結果

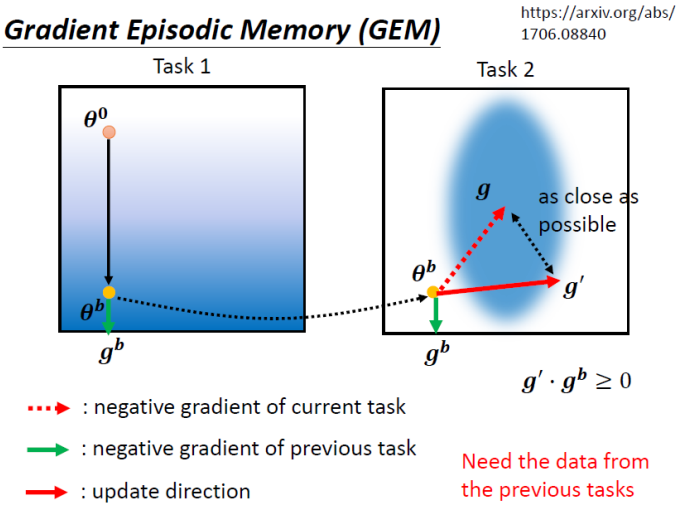
SGD 表示 b 皆為 0；L2 表示 b 皆為 1；EWC 表示根據 θ_i 的重要性有不同的 b



- SGD 方法有 **catastrophic forgetting 的問題**
- L2 方法雖然**沒有 catastrophic forgetting 的問題，但有 intransigence 的問題**
- EWC 方法實現了 LLL

4.1.4 Gradient Episodic Memory（GEM）

不在參數上做限制，而是**在 gradient 更新的方向上做限制**

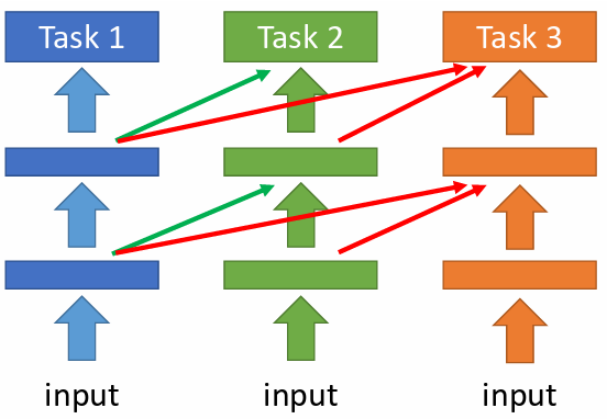


在 task 2 算出 gradient g 更新參數前，先計算該參數在 task 1上的 gradient g^b，之後兩者做內積，若 $g \cdot g^b < 0$（夾角為鈍角），則修改 g 變成 g'，讓 $g \cdot g^b > 0$（夾角為銳角），且 g' 與 g 不可以差太多

4.2 Additional Neural Resource Allocation Memory

改變使用在每一個任務里面的 neural 的 resource

4.2.1 Progressive Neural Network



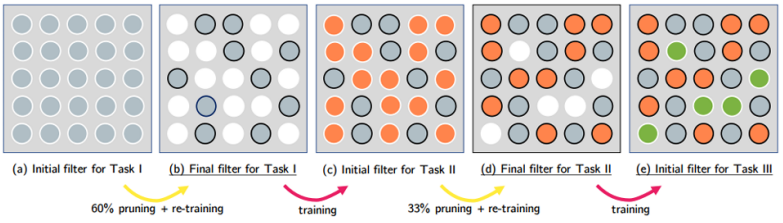
每一個任務都重新訓練一些**額外的 neural**，以保證**過去的任務的模型參數不會被改變**

問題：

當模型大小增長過快時，空間上仍難以負荷

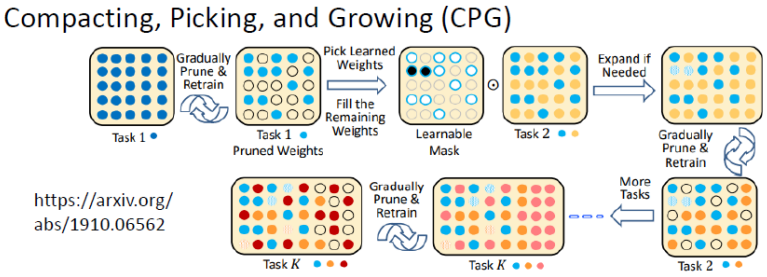
4.2.2 PackNet

先分配一個較大的模型，限制每個任務只允許使用某些參數



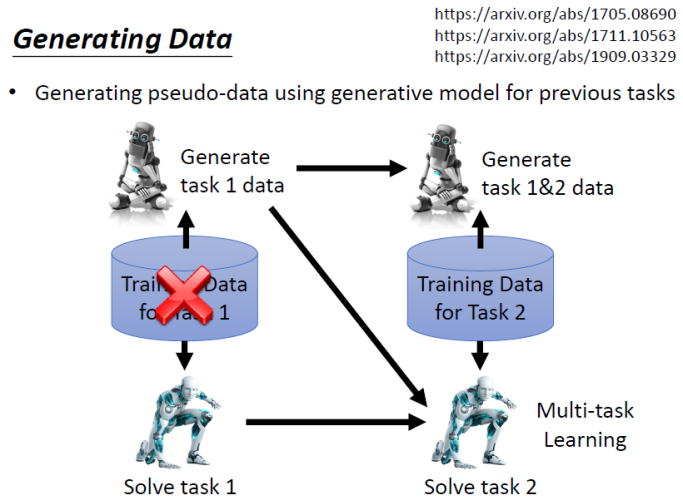
4.2.3 CPG（Progressive Neural Network + PackNet）

模型既可以增加新的參數，且每訓練一個新的任務，又都只保留部分的參數可以拿來做訓練



4.3 Memory Reply

針對每個新任務，額外訓練一個**可以產出該任務即過去所有任務的資料的 generator**



問題：

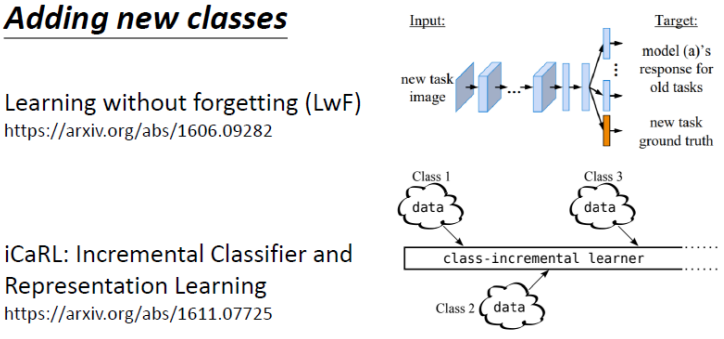
額外訓練 generator 同樣會佔用空間，但如果這個空間比儲存訓練資料佔用的空間小，那就是一個有效的方法

實際上，**此方法往往都可以逼近 LLL 的 upper bound，可以做到跟 multi-task learning 差不多的結果**

5. Adding new classes

問題：

不同的任務間的 class 數目不一樣



解決：

Learning without Forgetting（**LwF**）

iCaRL: Incremental Classifier and Representation Learning

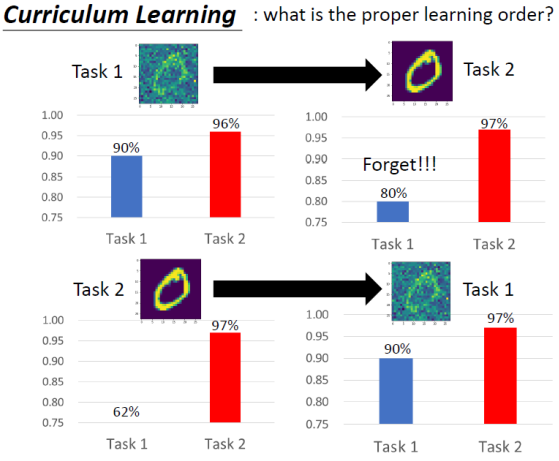
6. Three scenarios for Continual Learning（LLL）

LLL 有三種 scenarios，課程只介紹其中一種，其他兩種可參考論文：*Three scenarios for continual learning*

7. Curriculum Learning

兩個實驗：

- 訓練模型 task 1 有雜訊的圖片，再到 task 2 沒有雜訊的圖片，發生 catastrophic forgetting
- 順序反過來，訓練模型 task 1 沒有雜訊的圖片，再到 task 2，沒有發生 catastrophic forgetting



結論：

任務的順序也是關鍵