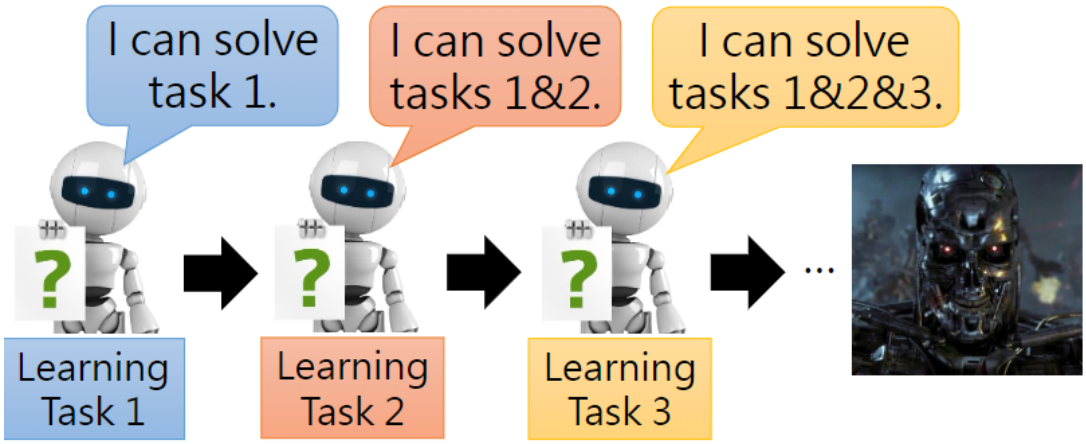
Lecture 11：机器终生学习 Lifelong Learning

> Lectured by HUNG-YI LEE (李宏毅)
> Recorded by Yusheng zhao（yszhao0717@gmail.com）

什么是lifelong learning——活到老学到老，这里的主体是人类。类似的，机器也可以做lifelong learning，机器的终身学习非常符合人类对AI的想象。

先教机器做task 1，接下来教它做task 2，然后教 task 3，.....AI可以不断学习新任务，这个构想就是Life Long Learning。

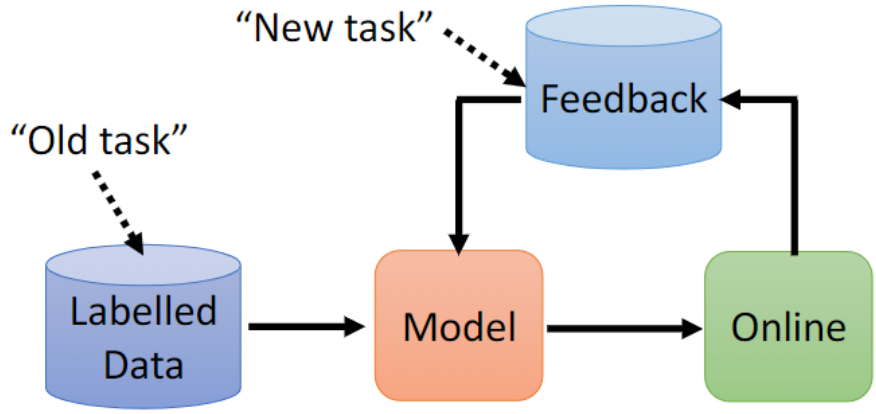


Life Long Learning(LLL)介绍

> 别称：Continuous Learning、Never Ending Learning、Incremental Learning（增量学习）

LLL的应用意义

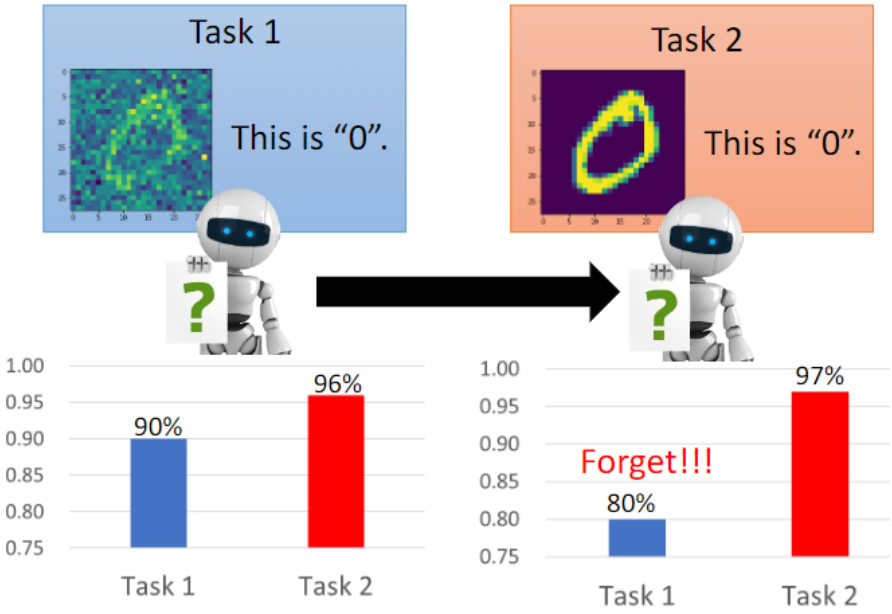
Real-world Applications：开发出一款机器学习模型，模型上线后（online），会得到用户的反馈（feedback）；这时数据集的扩充成为一个新的循环。



机器不断搜集线上的资料，再用线上的资料来更新模型，这本质上就是一个LLL的问题。

LLL的难点

- **E.g. 1**：以两个任务举例来说：

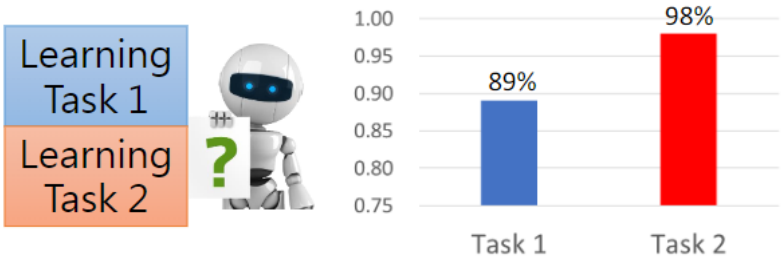


手写数字识别，其中task 1中图像里边有杂讯；而task 2比较简单，图片没有杂讯。（可以说是一个任务，两个domain）实做中，终身学习的不同任务也是类似于这种情况（任务目标一致，只是分属于不同的domain，在终身学习中被看作是不同的任务）。

以一个3层（每层50个神经元）的网络为例，（如左图）先学任务一，在任务一的正确率达90%，此时即便没看过任务二，也能在任务二上达到96%。

先学任务一，再让同一个模型学习任务二；即同一个模型用任务二来更新，此时任务2的正确率会更高；但是此时模型在任务一上的表现变差。

另外一个实验，把任务一和任务二的资料合并一起，同时取训练一个network。结果如下



对于一个network而言，其想同时学会两个任务是完全办得到的。

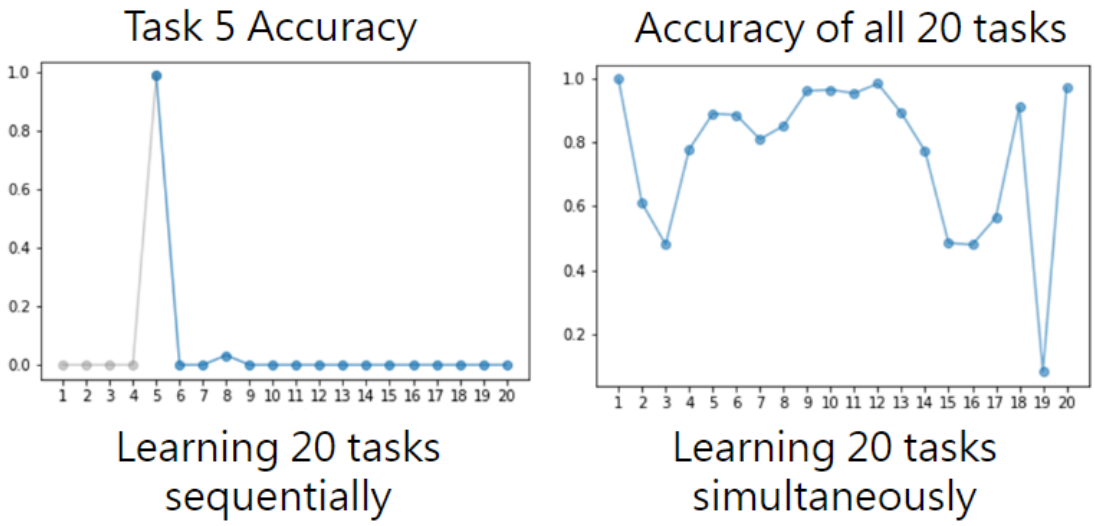
总结：对于network而言，其完全有能力在多任务上表现良好（合并数据集训练）；然而，再LLL的pattern下，先学一个任务，再由另外任务更新模型，会出现在现有任务表现好，但之前任务表　现糟糕的现象。

- **E.g. 2**：以NLP为例，QA任务：给定一篇文本，基于文本回答问题。（Given a document, answer the question based on the document)

  There are 20 QA tasks in bAbi corpus.（bAbi是比较古早的QA任务）

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

  20个任务就train20个QA模型。一字排开，按序学习。

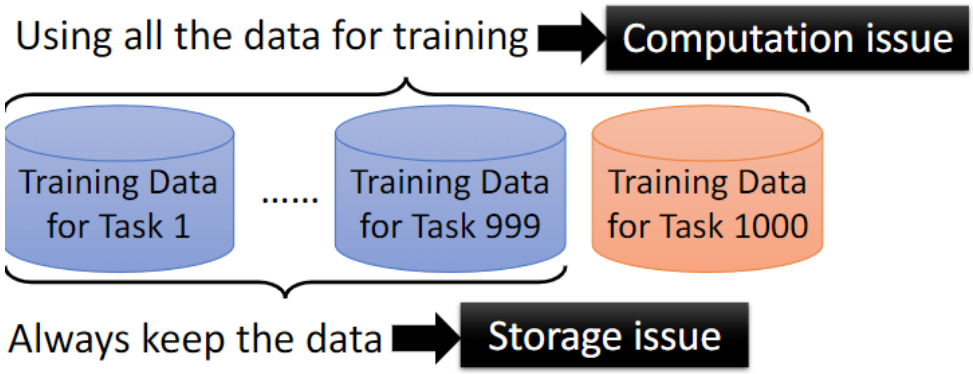


一学新的任务，旧的任务就忘得精光。右边图是机器同时学20个任务的时候，事实上机器可以同时学会多个任务。而让机器依序学习任务时，Machine forget what it has learned when it is learning the new.

（我的脑袋和机器的一样┬┬﹏┬┬）——这种现象称之为**Catastrophic Forgetting**

思考

- 多任务学习（Multi-task training）好像比终身学习效果好，那为什么要做终身学习？Multi-task training需要使用所有数据来完成训练，这表明如果任务数量比较多的时候，Multi-task training会导致存储资源压力过大而且加剧运算资源的消耗。（Computation issue & Storage issue）



- 因此，Multi-task training不是做终身学习的最佳选择方式；反之，multi-task learning可以看作是终身学习的upper bound。

在实操过程中，我们在做一个LLL之前，会跑一个相对应的multi-task learning康康LLL的大概上限。我们在LLL上的改进优化，就是为了逼近这个上限结果。

- LLL的主旨在于“不准复习”的情况下，可不可以减少甚至不遗忘，达到其自身的upper bound。

为什么不让一个模型学一个任务呢?

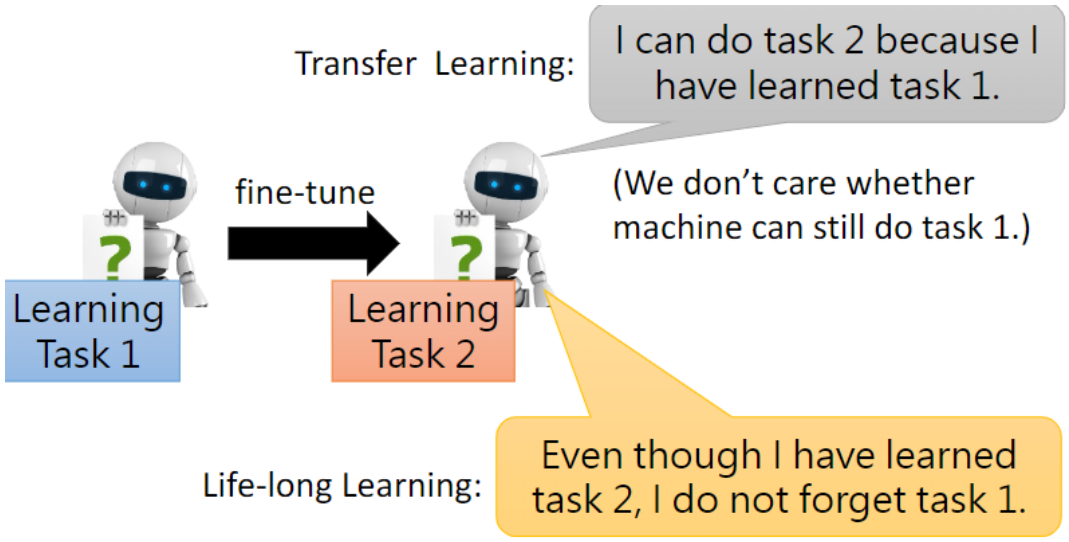
> Train a model for each task.

局限性：

- 最终我们无法存储下所有模型（Storage Issue）
- 不同的任务间知识难以迁移。（Knowledge cannot transfer across different tasks.）

Lifelong v.s. Transfer

> 有人觉得LLL和迁移学习很像嘛，实际上有区别的。

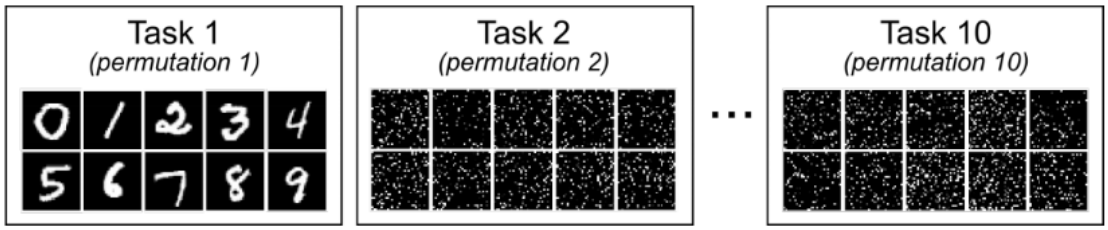


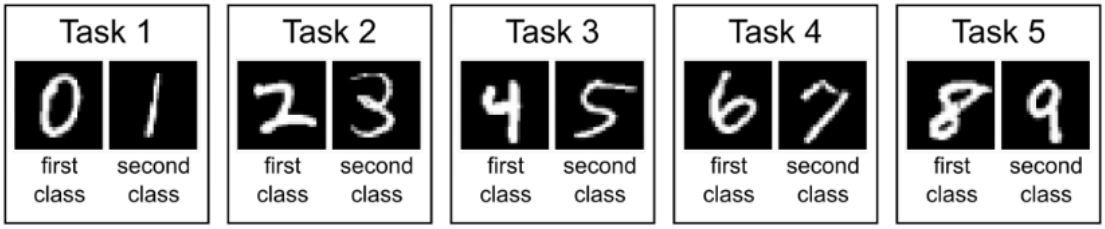
区别：（关注点不同）LLL还要关注原任务上的性能；而迁移学习只关注在新任务上的performance

如何评估Life Long Learning? （Evaluation）

以https://arxiv.org/pdf/1904.07734.pdf为例：

- 首先，我们需要一串tasks，以MNIST为例，这些任务通常都是相关的，但属于不同的domain。





把每一个数字用某种固定的规则打乱，形成另一个任务数据集的domain；每一个打乱规则就对应一个task的domain。

- 评估方法：我们有T个任务。

训练前，随机初始化参数，在T个任务上计算正确率。

每在第$i(i = 1, 2, \ldots, T)$个任务上训练后得到的NN在T个任务上test下，计算得出正确率。最终我们就会得到一个表格👇

		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	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

$R_{i,j}$:在task i上训练后，在task j上的表现。

- 如果$i > j$:假设$R_{i,j}$:表现不错，表示在未来的任务上训练后在过去任务性能表现良好
- 如果$i < j$:这时候，对于$R_{i,j}$机器还没有学会task j，想知道机器是否具有"无师自通"的能力。

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

这个量度就是把最后一行加起来，做一个平均；来评估一个LLL系统的好坏。

另外一种评估方法👇

$$Backward\ Transfer = \frac{1}{T-1}\sum_{i=1}^{T-1}(R_{T,i} - R_{i,i}) \tag{2}$$

这个值通常是负值。相对应的另一种评估方式，再没学时任务T，计算学到任务T-1时任务T上的表现和训练前任务T上的表现之差值。

$$Forward\ Transfer = \frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - R_{0,i} \tag{3}$$

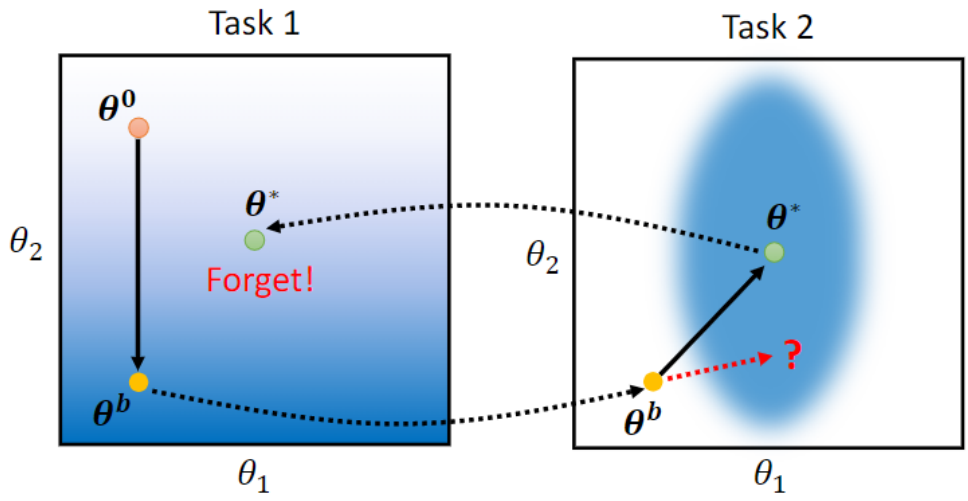
		Test on			
		Task 1	Task 2	.....	Task T
Rand Init.		$R_{0,1}$	$R_{0,2}$		$R_{0,T}$
After Training	Task 1	$R_{1,1}$	$R_{1,2}$		$R_{1,T}$
	Task 2	$R_{2,1}$	$R_{2,2}$		$R_{2,T}$
	⋮				
	Task T-1	$R_{T-1,1}$	$R_{T-1,2}$		$R_{T-1,T}$
	Task T	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

Life Long Learning的几种可能解法（Research Directions）

Selective Synaptic Plasticity（最为完整的一种方法）

> 可选择的 · 突触的 · 可塑性： 让NN中部分的联结具备可塑性或者弹性（这种方法也称之为 **Regularization Based Approach**）

为什么Catastrophic Forgetting会发生?



The error surfaces of tasks 1 & 2.

如上图，（颜色：蓝-->白说明loss从小到大。）Task 1和2依次训练：任务一从θ^0到θ^b；在任务二上，从θ^b到θ^*。把θ^*拿回到task 1上，变现差了（表现出forget的特点）。

克服难点的做法：需要对后续任务的梯度下降做一个限制；一方面自身在朝loss下降的方向迭代的时候，另一方面也要考虑到前序任务在这个参数组下的loss的表现。

Basic Idea： ome parameters in the model are important to the previous tasks. Only change the unimportant parameters.

当我们在学习新的任务时候，希望哪些在旧任务上比较重要的参数尽量不要变；只去改那些不太重要的参数。

已知θ^b是模型从前序任务中learned到的参数；每个参数θ_i^b都有一个"保镖（guard）"称之为b_i，我们这样描述loss函数

$$L'(\theta) = L(\theta) + \lambda \sum_i b_i(\theta_i - \theta_i^b)^2 \tag{4}$$

这个"保镖"b_i代表这个参数对过去的任务来说重不重要。以上公式中$L(\theta)$是LLL目前正在学习的任务的loss（Loss for Current task）；二我们真正要去minimize的是$L'(\theta)$（Loss to be Optimized）；多加的这项$\lambda \sum_i b_i(\theta_i - \theta_i^b)^2$中$\theta_i$是正在learning的参数，$\theta_i^b$是从前序任务中学习完（learned）的参数（下标i就是第i个参数）。这个b_i我们对表示θ_i, θ_i^b足够接近的期望（即参数的前序任务表现的重要性）。

并不是所有参数都需要这个约束规定：（有以下两种极端情况）

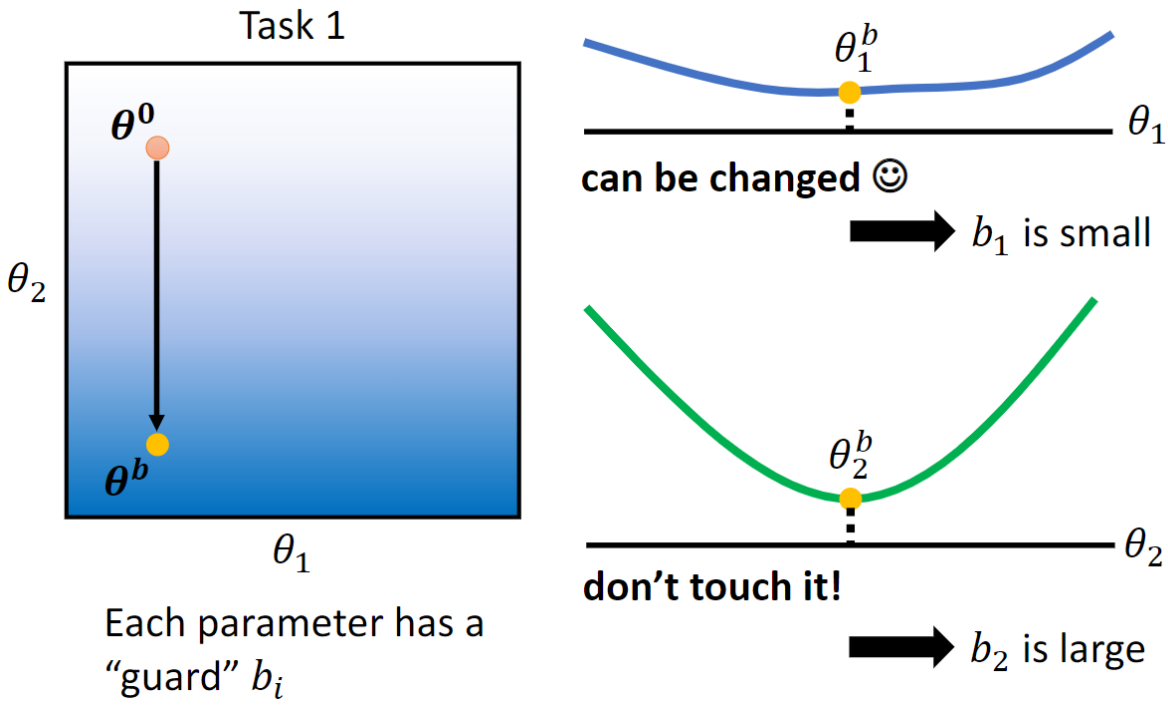
- 如果$b_i = 0$，约束失效，即这里对参数θ_i而言没有限制，最终导致Catastrophic Forgetting
- 如果$b_i = \infty$，所有参数都受到约束，换言之θ_i总是与θ_i^b相接近甚至相等，最终导致Intransigence（不妥协、不肯让步），实际上没有在新任务上learn的必要了，尽管旧任务上表现良好，但新任务上永远学不好。

在LLL的技术里面，关键的技术就是我们如何设定b_i，在许多文献当中，b_i都是*人为设定*的。如上任务中b_i就不太可能是learn得到的，为了让公式（4）的loss值越小越好，最终learn出来的结果就是$b_i = 0$，此时loss就最小了。

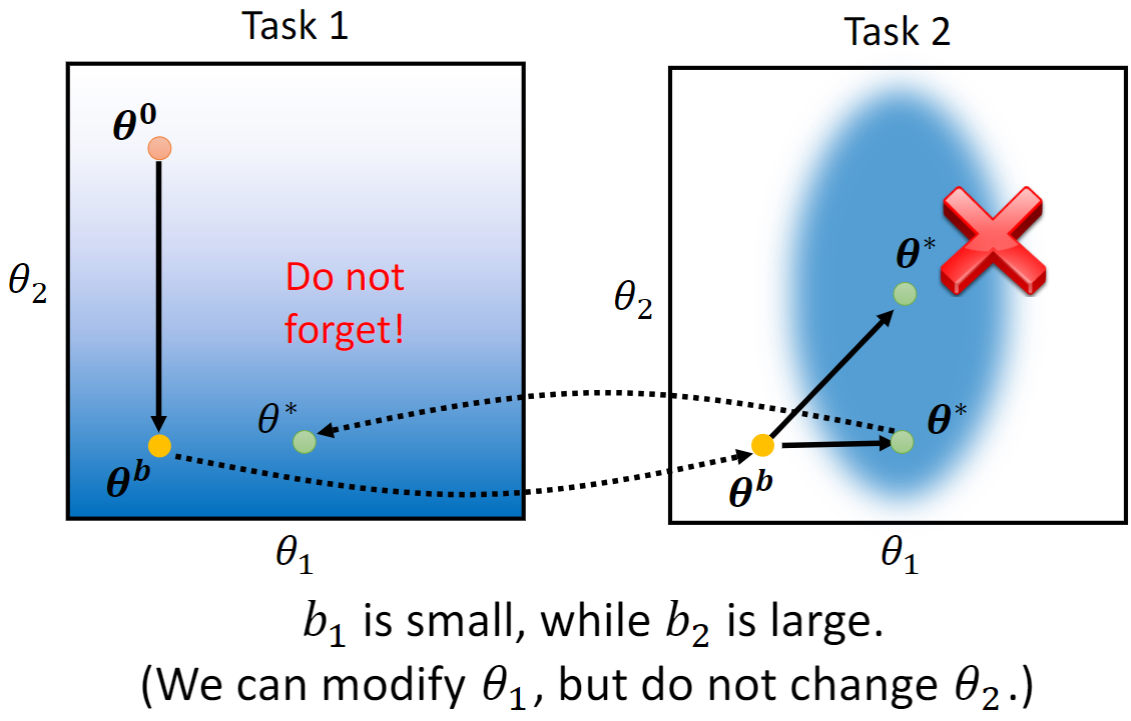
如何找到每个参数的重要性度量b_i

> 研究的重点

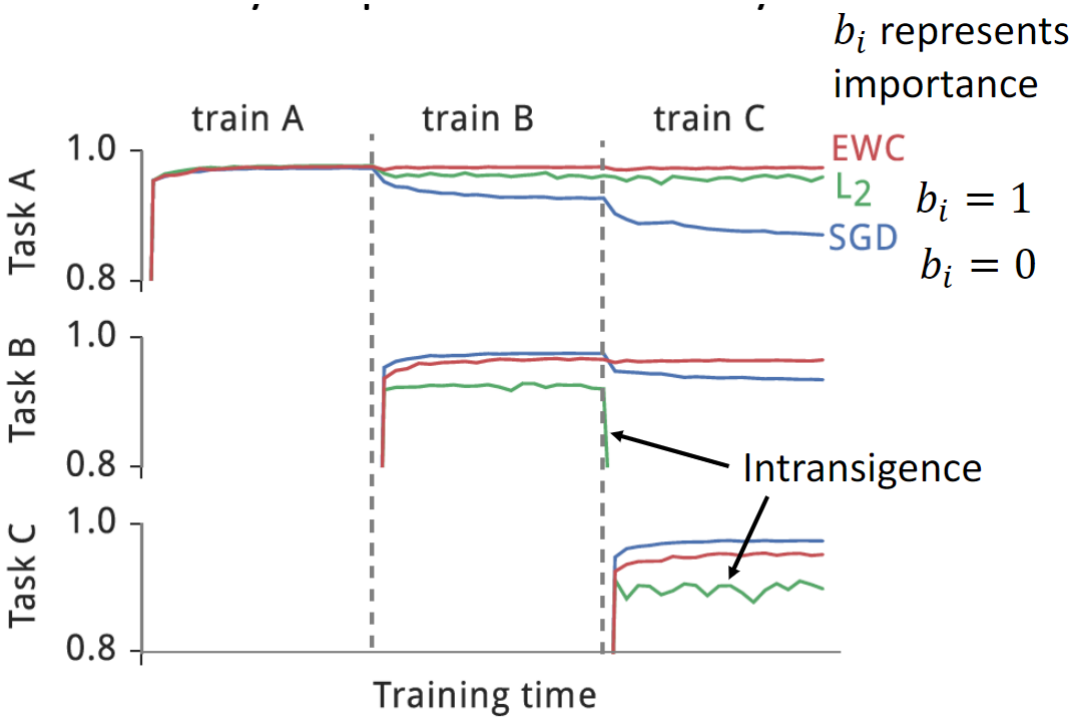
以下图为例，针对任务1，我们从参数集θ^0学习得到参数集θ^b（颜色越深loss越小）。我们可以针对学习得到的参数集θ^b的每个维度的参数做一个“对照实验”，看看参数的变化是否影响到loss的变化。下图中参数集以θ_1, θ_2为例，在θ_1上移动，对loss无影响，而在θ_2上移动则会使loss急剧增大。



由此，我们可以给θ_1比较小的b_1值，因为它对模型性能重要性较小；而b_2则设的较大些。从而对模型训练迭代方向产生一定影响。



E.g.来自EWC实验真正的结果，task为MNIST permutation：SGD中$b_i = 0$；L2中$b_i = 1$；而EMC则给b_i人为设置大小。



reference

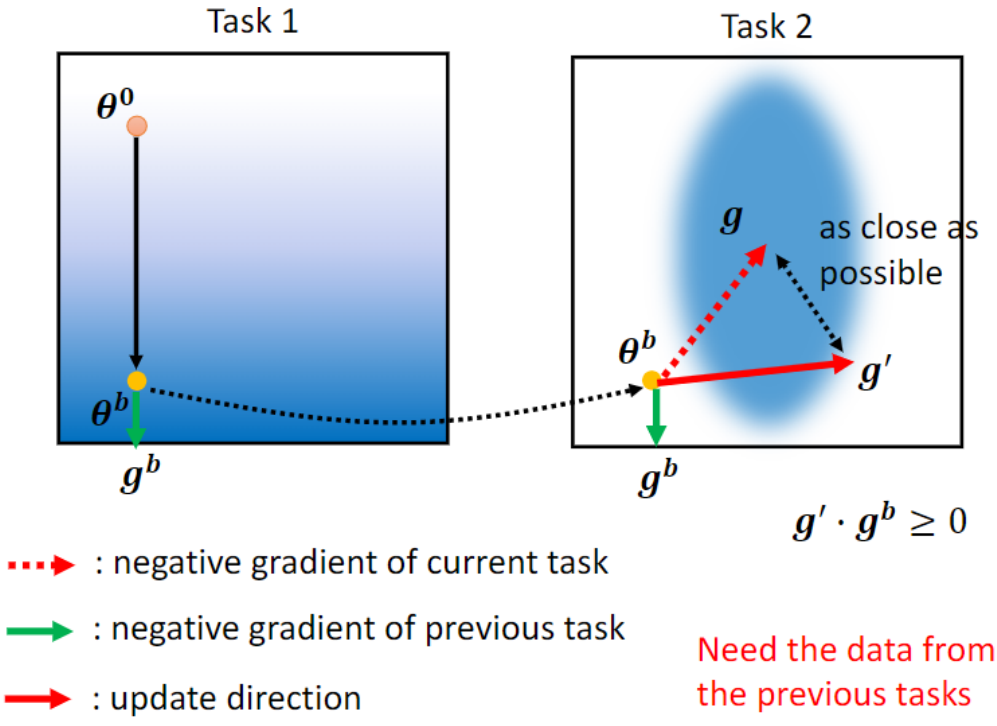
- Elastic Weight Consolidation (EWC)
- Synaptic Intelligence (SI)
- Memory Aware Synapses (MAS)
- RWalk
- Sliced Cramer Preservation (SCP)

以上都是文献中如何设置b_i的方法。

改变任务的顺序会影响方法的正确率

Gradient Episodic Memory (GEM)

> 在梯度上而不是参数上做文章



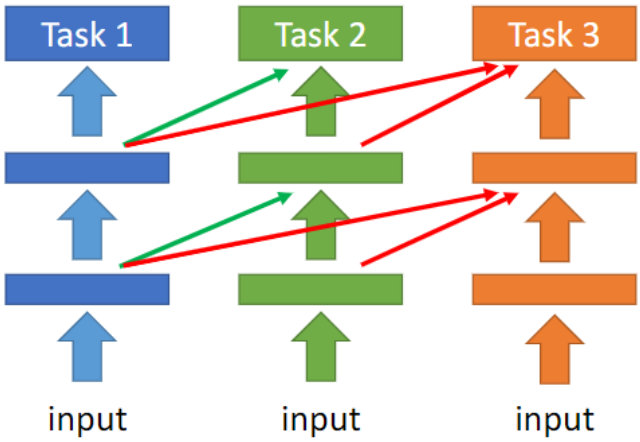
如果在新任务（如task 2）上梯度下降方向和前序任务的梯度下降方向不一致（内积小于0）；则修改该任务上的梯度方向，修改的criterion则是新修改的$g' \cdot g^b$（inner product）大于0，即两个方向角为锐角（上图所示有点问题）

劣势：需要把过去任务的资料存下来，因为梯度方向的修正需要之前的梯度方向g^b；所存资料占小部分，只需要计算得到原梯度方向即可

Additional Neural Resource Allocation

> 改变使用在每个neuron中的resource

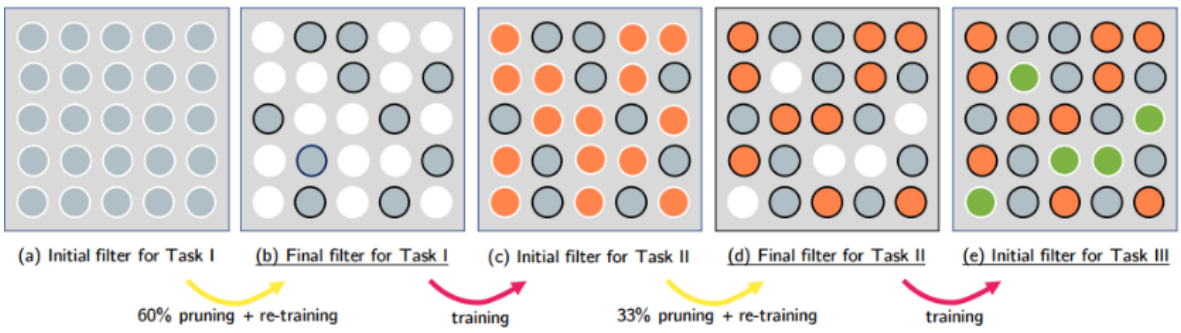
最早的做法：Progressive Neural Networks



学完任务一，就不再改动模型；另开一个模型，基于任务一中学到的参数，新增一些额外的参数，在任务二上train这些额外的参数。任务三同理。

劣势：每一次做一个新的task，模型就会长大；当任务不断新增下去，memory终会耗尽。不过当任务量较少时，这个方法还是合适的。

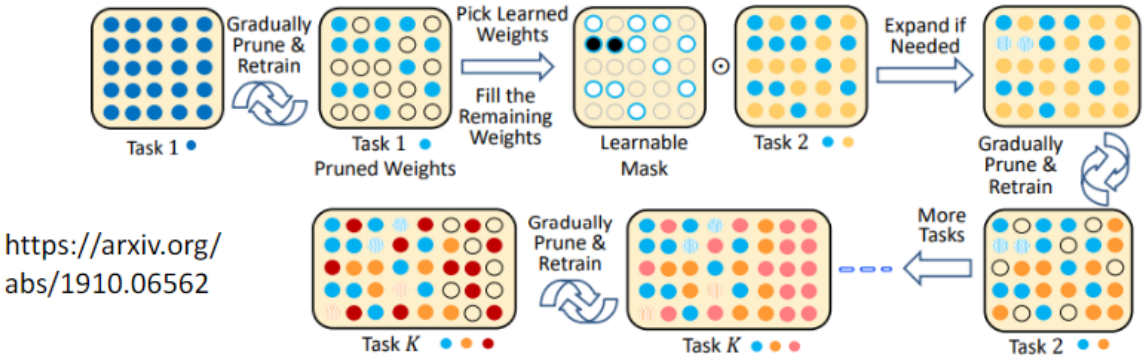
PackNet



> Progressive Neural Networks的逆用

先开一个较大的network，对于任务一进来只准使用其中一部分参数（灰色的）；任务二进来只准使用另一部分参数（橘色的）。任务三同理。这个方法局限性也很大，和Progressive Neural Networks一样半斤八两。

两种方法的结合：Compacting, Picking, and Growing (CPG)



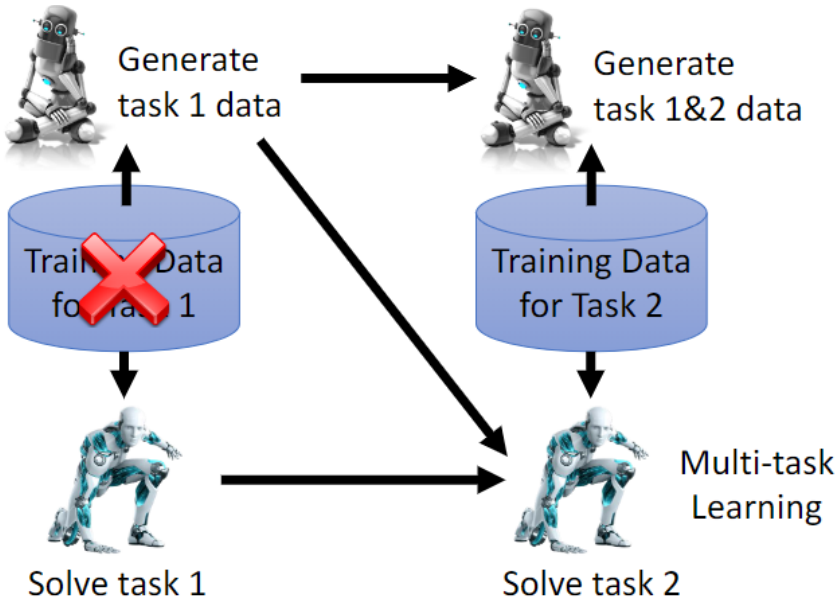
既可以增加新的参数，在每一次的task中又只使用部分的参数。

Memory Reply

> 非常符合直觉的一种做法

既然我们不能存储过去的资料，那么为何不做一个**generative model**，来生成previous tasks的"数据"（这里称之为pseudo-data），整个做法就是**Generating Data**

> https://arxiv.org/abs/1705.08690；https://arxiv.org/abs/1711.10563；https://arxiv.org/abs/1909.03329



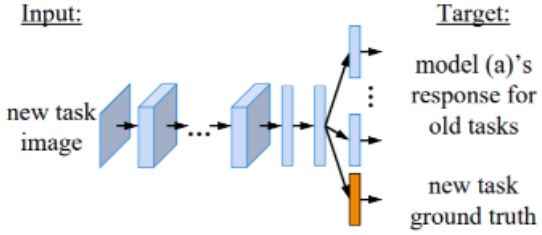
用generative model产生任务一的伪资料，在solve task 2阶段，同task 2的资料一起，喂给模型train下去。老师表示，这种generate data方法是非常有效的，甚至可以接近model的upper bound...

化解task不对称（class数目不一致）的方法

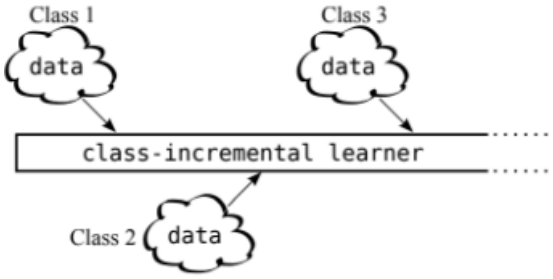
> 第一、二、三个任务的class数目假如是10、20、100个；遇到这种情况，我们需要对class数目较少的任务**adding new classes**

（没有展开讲）罗列相关文献如下：

- Learning without forgetting (LwF)



- [iCaRL: Incremental Classifier and Representation Learning]



Life Long Learning其他不同的情景

> 以上只是LLL中的一小块

参考[Three scenarios for continual learning].LLL中又三个情景，以上所述只是其中之一最简单的Scenario。

LLL调换task顺序：Curriculum Learning

如上实验，好像顺序会对LLL产生影响啊。研究顺序产生影响的LLL称之为Curriculum Learning。

Elastic Weight Consolidation（EWC）——可塑权重巩固

不同的task data对model的重要性会不会冲突？如何merge

文献的做法：每一个task都会计算一个重要性，然后统统加起来就行。