

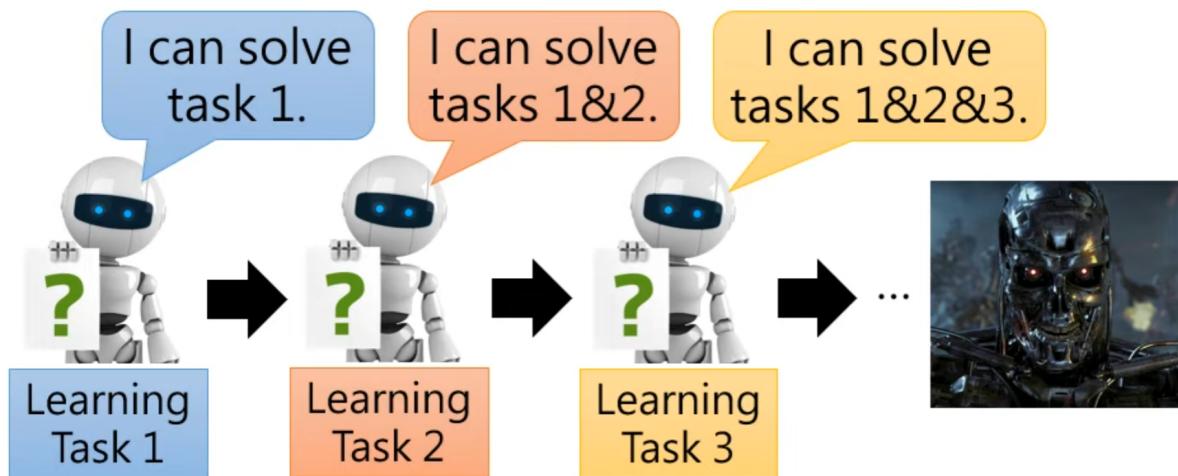
Life Long Learning

好 那接下来的课程呢，要跟大家讲 Life Long Learning，那什么是 Life Long Learning 呢？

如果你 Google Life Long Learning 这个东西，你最有可能找到的，不是跟 Machine Learning 相关的内容。Life Long Learning，从它字面上的翻译，你可以知道 Life Long Learning 指的就是，终身学习，也就是活到老学到老，所以如果你用 Life Long Learning 做关键字，去 Google 的话，你通常找到的都是有关人类的，Life Long Learning，人类怎么活到老学到老。

What people think about AI

那虽然今天大家关注的主题，比较偏向人类怎么做 Life Long Learning，但是其实机器也需要做 Life Long Learning，机器可以做 Life Long Learning 这件事情呢，非常接近人类对 AI 的想像。你知道在还没有修这门课之前，在还没有接触，任何 Machine Learning 的内容之前，也许你对 AI 的想像是这个样子的，我们先教机器做某一件事情，比如说学会做语音辨识，那它就会做语音辨识了，接下来你再教它第二个任务，教它做影像辨识，它就会做影像辨识了，接下来你再教它做第三个任务，也许是翻译，它就会做影像辨识加语音辨识加翻译了，你不断教它新的技能，等它学会上百万 上千万个技能以后，它就变成天网，然后就可以统治人类。我们一般人对 AI 的想像是，AI 可以不断地学习新的任务，最终越来越厉害，直到人类不能企及的程度，那这个构想 这个目标就是，Life Long Learning，那 Life Long Learning 呢，常常缩写成三个 L，LLL，Triple L，不是 LoL 是 LLL，那 Life Long Learning 呢，也有很多其他很潮的名字，比如说有的人叫它 Continuous Learning，有人叫它 Never Ending Learning，听起来都很潮，有一个比较不潮的名字是，Incremental Learning。

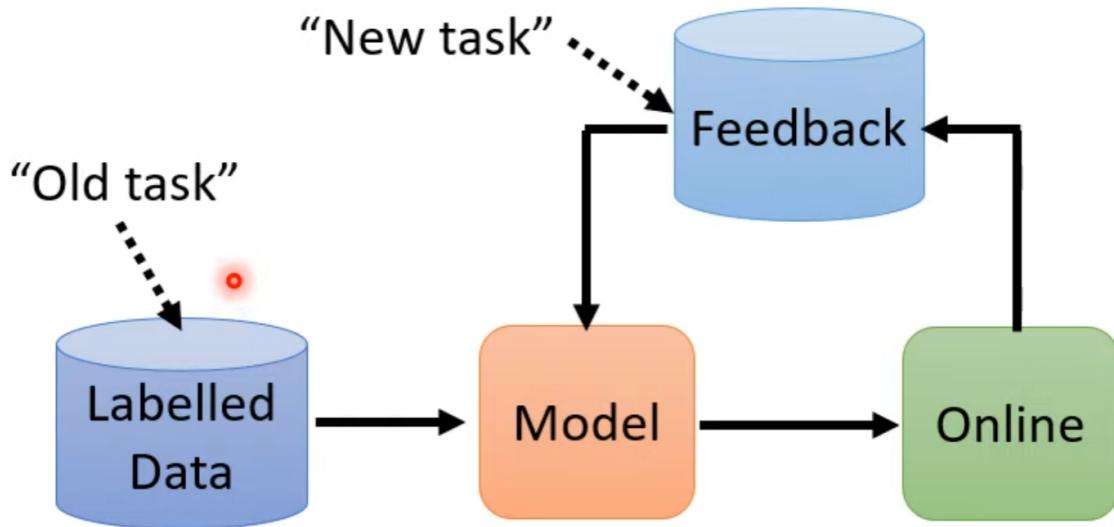


Life Long Learning (LLI), Continuous Learning,
Never Ending Learning, Incremental Learning

好 但是你可能会想说，Life Long Learning 这个目标太过远大，我今天又没有要做天网，那 Life Long Learning 对我有什么意义呢，在真实的 Application 里面，Life Long Learning 也是派得上用场的。

Life Long Learning in real-world applications

举例来说啊，今天你在实验室里面开发出某一个模型，你在实验室里面搜集一些资料，把这些资料进行训练，训练出一个模型，模型上线以后，它会取得来自使用者的 Feedback，这时候我们都希望，搜集资料这件事情可以变成一个循环，我们模型上线以后搜集到新的资料，新的资料就可以让我们来更新，我们模型的参数，模型的参数更新以后，又可以搜集更多的资料，搜集更多的资料，模型的参数又可以再次更新，不断更新模型的参数，最终我们的系统就会越来越厉害。那你可以把旧有的资料想成是过去的任务，把新的资料，来自于使用者 Feedback 的资料，想成是新的任务，所以这样子的情境，也可以看作是 Life Long Learning 的问题，机器不断地在线上搜集资料，用线上搜集的资料来更新模型，这本质上就是一个 Life Long Learning 的问题。那 Life Long Learning 有什么样的难点呢？我们不就是让机器不断地看新的资料，不断地去 Update 它的参数，就做到 Life Long Learning 了吗，那为什么 Life Long Learning，会是一个值得研究的问题呢？

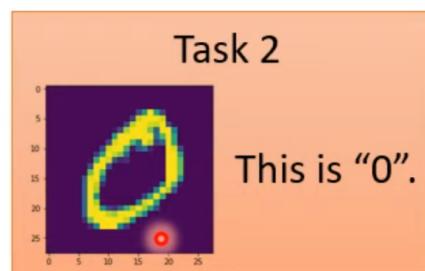
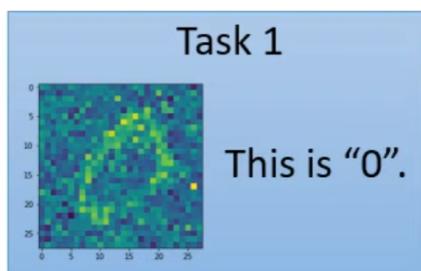
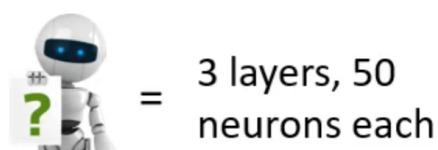


Difficulty of LLL

Example in CV

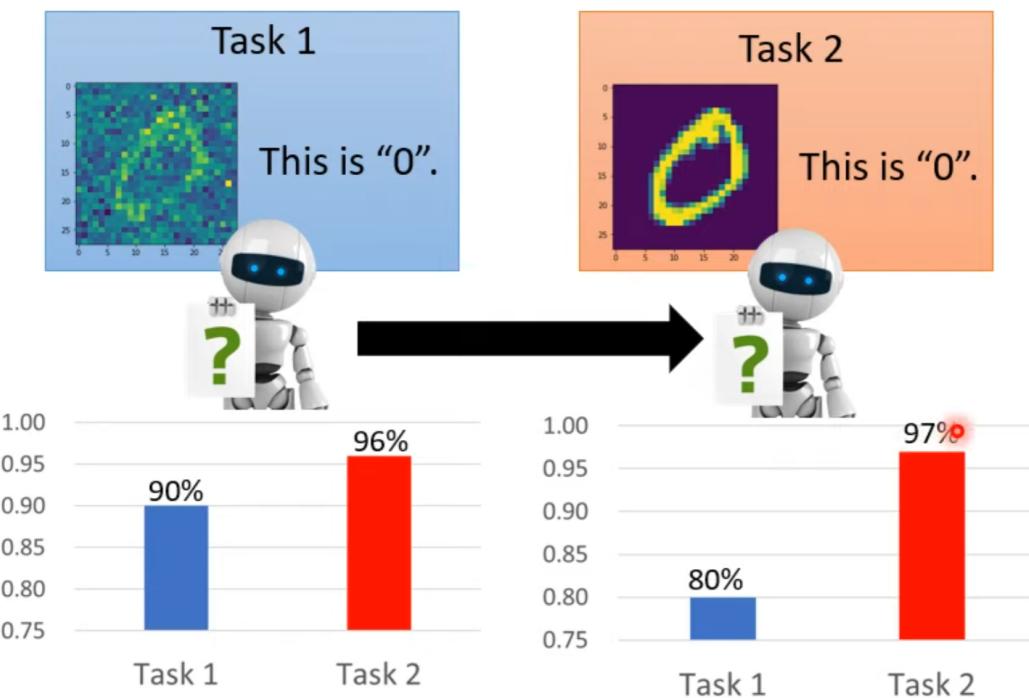
以下举一个简单的例子告诉你说，Life Long Learning 的难点出在什么样的地方。

Example

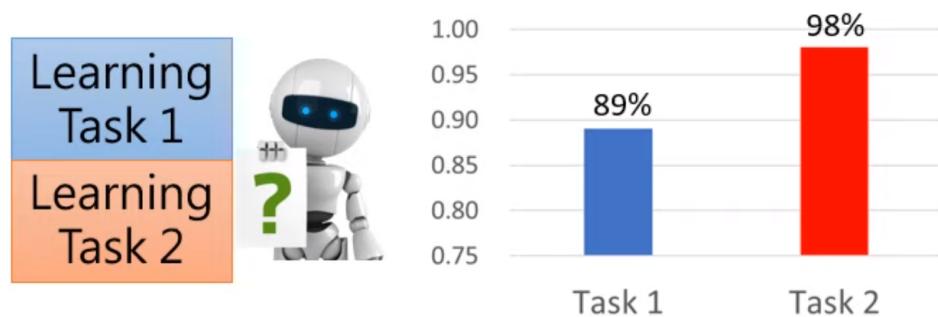


假设呢，我们现在有两个任务，那第一个任务是要做手写数字辨识，那给它一张非常 Noisy 的 Image，机器要判断说这里面是图片 0，任务二也是手写数字辨识，那只是现在是比较简单的任务，你的图片里面是没有任何杂讯的，我们要让机器学会这两个任务。那讲到这边有人可能会说，啊 老师这不算是两个任务啦，这个算是同一个任务 不同的 Domain，你要这样想也没问题，你也可以想成是同一个任务 不同的 Domain，但是其实啊，在这个，我知道说，当我说我们要让机器学一连串的任务的时候，在你心里的想像也许是，机器先学个语音辨识，再学个影像辨识，再学个翻译，但其实今天 Life Long Learning，都还没有做到那个程度。一般在 Life Long Learning 的文献上，所谓的不同任务指的差不多

就是，我这边这种等级，通常比较像是不同的 Domain，而不是不同的任务。只是我们在这边，我们把它当做不同的任务来看待，但就算是非常类似的任务，你在做 Life Long Learning 的时候，也会有以下的问题，等下就是来看看有什么样的问题。



好那我这边，我就训练一个非常简单的 Network，它只有三层，每层五十个 Neuron，先在任务一上学一下，在任务一上学完以后，我们得到的结果是这个样子的，任务一正确率 90%，就算还没有看过任务二，任务二也已经得到 96% 的正确率了。这个 Transfer 的结果非常地好，能够解任务一，其实就能够解任务二了，好那任务一学完之后，我们再让同一个模型继续去学任务二，所谓同一个模型继续去学任务二的意思就是，我们在学任务二的时候，并不是让机器从头学起，并不是让机器从一组，Random Initialize 的参数学起，而是用任务一的资料更新完模型之后，同一个的模型接下来，继续用任务二的资料来更新。好所以听清楚哦，是同一个模型继续用任务二的资料来更新。好那所以同一个已经学完任务一的模型，再学任务二会发生什么事呢？这是我们得到的结果，任务二正确率变更高了，因为之前根本没看过任务二的资料就有 96% 了，看过任务二的资料当然更厉害，变成 97%。但糟糕的事情是，机器忘了怎么做任务一了，它本来任务一有 90% 的正确率，在它学会任务二以后，任务一变成只有 80% 的正确率，它忘记了它过去已经学到的技能。那有人看到这边可能会觉得说，欸老师这有什么好奇怪的呢，你这个 Network 就是一个小小的 Network 啊，那你叫它学任务二嘛，那任务一是之前学的嘛，那当然就，它的能力有限啊，它脑容量有限啊，学完任务二以后当然就忘了任务一嘛。



The network has enough capacity to learn both tasks.

但是我接下来告诉你另外一个实验，假设我们把任务一跟任务二的资料，直接倒在一起，会发生什么事呢，假设我们把任务一跟任务二的资料倒在一起，同时去训练这一个 Network，我们得到的结果是这样的，任务一可以得到 89% 的正确率，任务二可以得到 98% 的正确率。也就是对这个 Network 而言，要同时学好任务一跟任务二，它是办得到的，虽然它每一层只有五十个 Neuron，但五十个 Neuron 已经足以让它，同时在任务一上 任务二上，得到这样子的正确率。那不知道为什么，如果不是同时学任务一任务二，而是先学任务一 再学任务二的话，它在学任务二的时候，就会忘记任务一学过的东西了。它有足够的能力把两个任务都学好，但是当你让它依序学习的时候，它没办法记住旧的任务。

Example in NLP

我刚才举的例子呢，是影像辨识的例子，那我接下来再举一个，这个自然语言处理的例子。告诉你说刚才看到的那个状况，它不是一个特例，它是一个非常一般 非常常见的现象。

这边要举的例子啊，是 QA Question Answering，也就是我们的作业七，在作业七里面，你知道你可以让机器读一篇文章，问它一个问题，然后它可以回答你的问题。那我们在这边用的并不是作业七的资料，而是更简单的 QA 的任务，这个 QA 的任务叫做 bAbi，通常就念成 bAbi，那 bAbi 是一个非常早的 QA 的任务，在人们刚开始研究 QA，在人类刚开始想要用，这个 Deep Learning 的技术，解 QA 的问题的时候，一开始人们觉得 QA 的问题太难了，我们要用 Neuron Network，用 Deep Learning 的技术解它，感觉非常地困难，所以 Facebook 呢，就先定义了二十个简单的 QA 的任务，它们叫做 bAbi。那这些 QA 的任务不是真实的 QA 的任务，你如果打开你的作业七的资料，跟 bAbi 的资料进行比较的话，你会发现你作业七的资料是，远比 bAbi 难得多的，bAbi 里面的文章都是用某种规则生成的，用某种 Template，固定的句型生成的，问题也是用固定的句型生成的。

Example

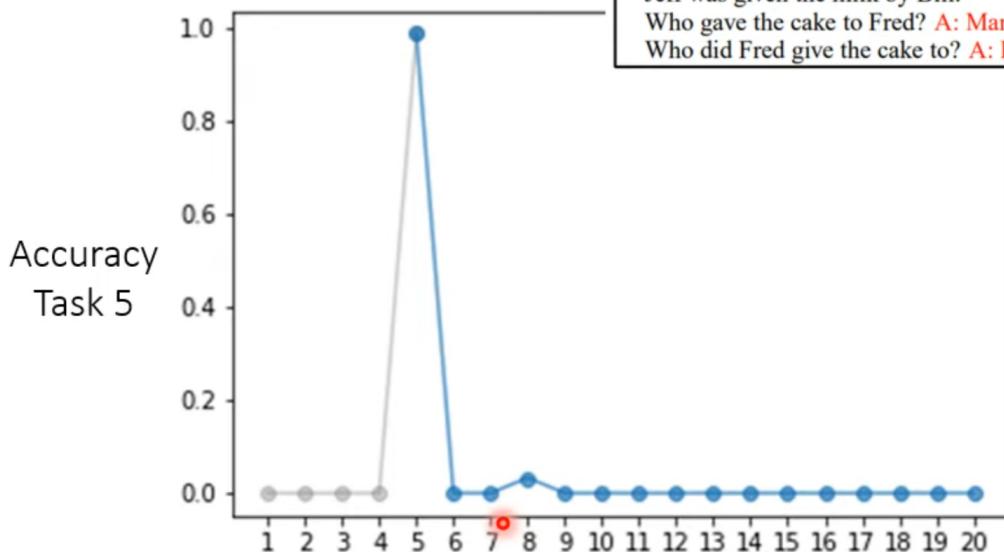
- QA: Given a document, answer the question based on the document.
- There are 20 QA tasks in bAbi corpus.

Task 5: Three Argument Relations
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

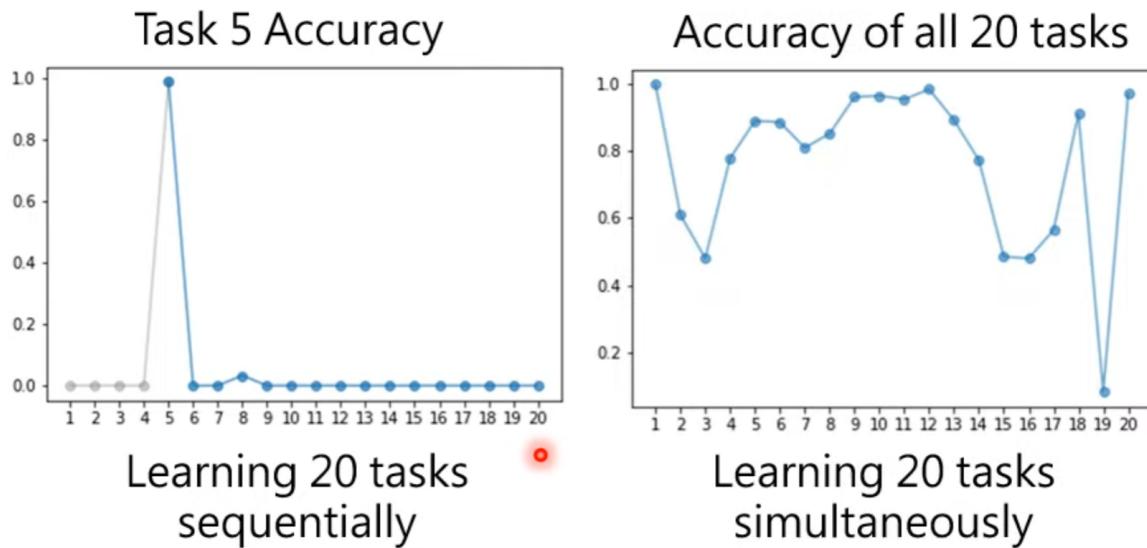
Task 15: Basic Deduction
Sheep are afraid of wolves.
Cats are afraid of dogs.
Mice are afraid of cats.
Gertrude is a sheep.
What is Gertrude afraid of? A:wolves

所以 bAbi 是一个非常简单的任务，它里面看起来就像这样，Mary 把蛋糕给了 Fred，然后 Fred 把蛋糕给了 Bill，然后 Bill 把牛奶给了 Jeff，然后问你说谁把蛋糕给了 Fred，答案是 Mary，就这么简单。或者是第十五个任务里面是，羊会怕狼，猫会怕狗，然后老鼠会怕猫，然后呢 Gertrud 它是一个羊，那它怕什么，它怕狼就这样，都是这么简单，这么简单的问题，今天你一定会觉得说，这个问题对 Network 来说不是问题，作业七都学得起来，这么简单的问题没有学不起来的道理，不过在当年啊，能够让机器学会这么简单的问题，人们已经觉得非常震惊了。好 那接下来我们要做的事情是，让机器依序学过这二十个 QA 的任务。那一般 bAbi 的使用方法是把，二十个任务通通倒在一起，让机器一次学二十个任务，或者是二十个任务就 Train 二十个模型，那二十个模型各自有不同的技能。

Example



Machine learns 20 tasks sequentially



Learning 20 tasks sequentially

Learning 20 tasks simultaneously

那我这边想要做的事情是，把二十个任务一字排开，让机器从第一个任务学起，学到第二个任务，那看看它能不能够把二十个任务都学会，好那这边的结果是这个样子，我们这边只看任务五的正确率，所以这个纵轴呢，是任务五的正确率，横轴呢，是依序学二十个任务的过程，好那任务五就是这么简单，就是我们刚才看过的那个蛋糕的例子，好先让机器学任务一，再学任务二，再学任务三，再学任务四，把这四个任务依序学完。你发现说，欸任务五都是，正确率都是0，把任务一到四都学完，任务五正确率都是0，这件事没什么好惊讶的，因为它还没学到任务五啊，所以它当然不知道要怎么解任务五的问题啊，不教而杀谓之虐嘛，所以它不会解任务五的问题，得到正确率0%，不能怪你的Model。而接下来学完任务五以后，会发生什么事呢，学完任务五以后，正确率直接爆冲变成100%，因为看过任务五的训练资料了嘛，机器知道怎么解任务五，所以任务五正确率变100%。但是当我们继续学剩下的任务的时候，会发生什么事呢，你发现正确率一阵暴跌，当机器学完任务六之后，任务五的正确率就变0%了，学完任务七再去测试任务五就变0%了，机器只要一学新的任务，旧的任务马上就忘个精光。

那你可能会以为说，会不会是因为机器就是没有能力，多学好几个任务呢？其实不是的，刚才在左边这个图是任务五的正确率，让机器依序学这二十个任务，如果我们把二十个任务的资料通通倒在一起，让机器同时学二十个任务的话会发生什么事呢？右边这张图是机器同时学二十个任务的结果。那这边的纵轴啊，并不是某一个任务的正确率，而是这个对应到1的这个正确率，对应到1这个坐标点的正确率，就代表第一个任务的正确率，第二个任务的正确率，第三个任务的正确率，一直到第二个任务的正确率

Task 5: Three Argument Relations

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

以此类推，你发现让机器同时学二十个任务的时候，当然有些任务很难，比如任务十九机器学不会，但是机器是可以同时学会多个任务的，有好几个任务机器都可以得到非常高的正确率，但是这是同时学的状况，那你让机器依序学的时候，它就是学了新的东西就忘了旧的东西。

而右边这个实验告诉我们说，机器明明有能力学多个任务，但是你让它依序学一个一个任务的时候，它就是不肯把多个任务都学会，所以它明明可以精通多个任务却不肯做到，它不是做不到，它是是不为也非不能也。

Not because machine are not able to do it, but it just didn't do it.
是不為也 非。不能也

Catastrophic Forgetting

所以你会发现说，当机器依序学多个任务的时候，它就好像是一个脑袋有洞的人，它就像左边这个人一样，**新的任务进来，旧的东西就掉出去了，它永远学不会多个技能，而这个状况叫做 Catastrophic Forgetting。**

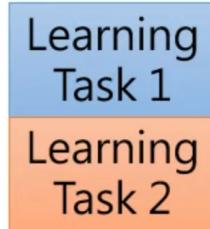
在 Forgetting 前面呢，特别加上 Catastrophic 这个形容词，因为 Forgetting 遗忘可能，人类也会遗忘啊，所以机器会遗忘也许不是什么特别惊人的事，但是机器遗忘的程度也未免太过分了，基本上它根本就学不会新的技能，所以这种遗忘前面加上了一个形容词，叫 Catastrophic 灾难性的，告诉我们说，这个遗忘不是一般的遗忘，它是灾难性的遗忘。



Problems of Multi-task Training

好讲到这边啊，我们等一下就会继续来看说，怎么解灾难性遗忘的问题，怎么让机器有办法依序学习多个任务，但在我们继续讨论技术之前，也许你会有一个问题，也许你会问说，等一下，刚才我们不是看到说，只要把多个任务的资料通通倒在一起，机器就可以学多个任务了吗，**把多个任务的资料倒在一起同时学，这个叫做 Multi-Task 的 Training。**

Wait a minute



- Multi-task training can solve the problem!

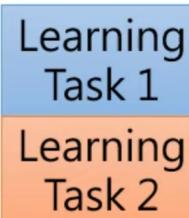
Using all the data for training



Always keep the data

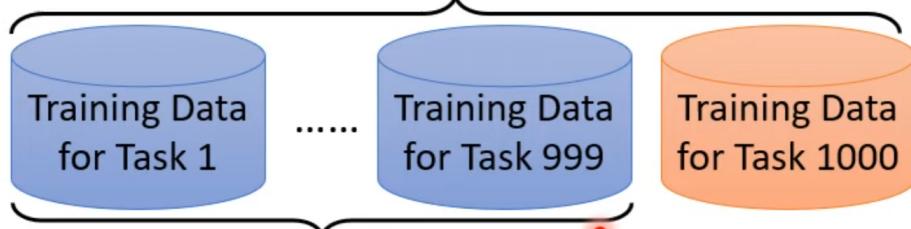
Multi-Task Training, 就可以让机器学会多个任务了，那这个 Life Long Learning 的问题，有什么好研究的，但是你想想看喔，假设现在我们要让机器学，第 1000 个任务，你要让机器避免遗忘前面 999 个任务，你必须要把前面 999 个任务的资料，通通都拿出来，然后把它跟第 1000 个任务倒在一起，把这 1000 个任务所有的资料通通倒在一起，一起做训练，机器才能同时学会，同时具有 999 个，同时具有这 1000 个任务，我们要它学的技能，但是在实务上这可能会是有问题的，因为如果我们要让机器学第 1000 个技能，需要前面 999 个任务的资料，那意味着机器需要把它一辈子看过的资料，通通都背在身上，它必须要把一辈子看过的资料，通通都存下来，你可能根本没有那么大的空间，可以储存所有的资料。

Wait a minute



- Multi-task training can solve the problem!

Using all the data for training → Computation issue



Always keep the data → Storage issue

而另外一方面呢，Computation 也是一个问题，如果我们今天需要把 1000 个任务的资料，通通倒在一起，才能进行训练，那这个训练的时间可能太长了，1000 个任务的资料全部倒在一起，可能太多了，那你训练的时间可能会太长，那你就没有办法让机器学习多个任务。

所以假设今天机器一定要做，Multi-Task Learning，才能够学习多个任务的话，那就好像是说有一个人，假设我们要叫他上一门新的课，他必须要把他这辈子所有学过的课，所有读过的教材，通通都再读一遍，他才有办法学习新的任务，那这样显然非常没有效率嘛，而且随着你要学的任务越来越多，你训练的时间就会越来越长，你需要储存的资料也会越来越多。所以 Multi-Task Training，虽然可以让机器学习多个任务，但是它不是解决 Life Long Learning，最终的 Solution。

那在文献上啊，通常会把 Multi-Task Training，看作是 Life Long Learning 的 Upper Bound。

就是如果我们把所有的资料通通倒在一起，虽然当任务多的时候，这是一个不切实际的做法，但是它可以让机器学会多个任务，这把所有的任务倒在一起一次做训练，Multi-Task Learning，往往视为是，Life Long Learning 的 Upper Bound，是 Life Long Learning 没有办法超越的结果。所以你在做 Life Long Learning 研究的时候，你往往会，比如说先跑个 Multi-Task Training 的结果，告诉我们说 Upper Bound 在哪里，然后再看看你的 Life Long Learning 的技术，能不能够逼近这个 Upper Bound 的结果。

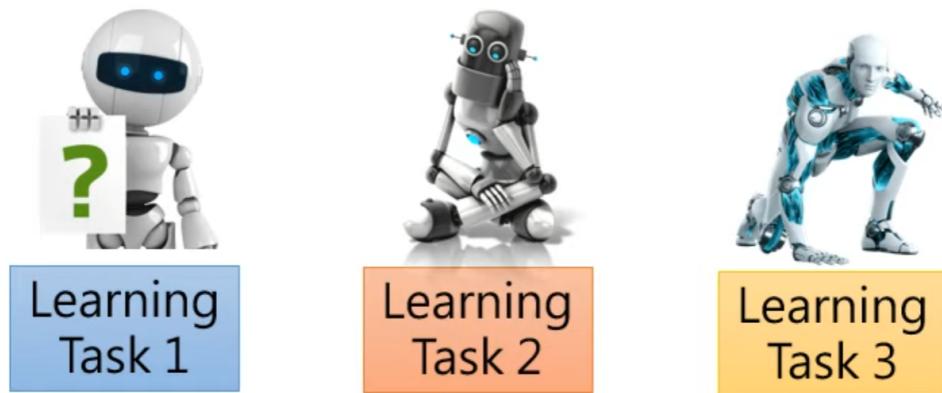
那我们怎么不每一个任务，就学一个模型就好了呢？干嘛要那么执着于，Life Long Learning 的问题呢，为什么一定要让一个模型学多个任务呢，为什么不一个任务，一个任务学一个模型就好了呢？

如果我们只让，让每一个任务都分开学一个模型，确实就没有 Catastrophic Forgetting 的问题，但是我们会遇到的第一个问题是，假设我们要叫机器学的技能，非常非常地多，我们要学个天网，天网总是要会上亿个技能吧，我们总不能每个技能都有一个模型吧，这样子你可能没有办法把所有的模型储存下来。

另外一方面，如果我们是不同的任务就用不同的模型，不同任务的资料间就不能够互通有无，它没有办法从其他的任务里面，汲取单一个任务所没有办法学到的资讯。而且你想看，对人类来说我们只有一个脑，但这个脑却可以学会多种不同的任务，不断学会新的技能，我们并不需要每一个任务，都用一个独立的脑来储存，但是为什么机器不能够做到一样的事情呢，这个就是 Life Long Learning 想要探讨的问题：能不能一个模型学多个任务。

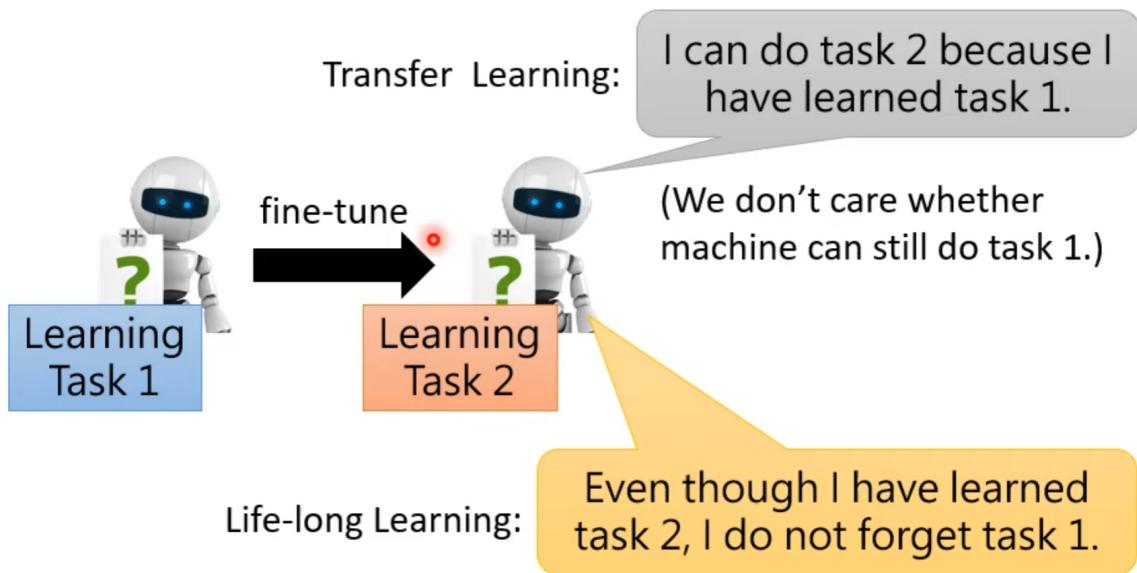
- Multi-task training can be considered as the upper bound of LLL.

- Train a model for each task



- ➤ Eventually we cannot store all the models ...
- Knowledge cannot transfer across different tasks

Life-Long v.s. Transfer



好 那讲到这边有的同学可能会说，这个听起来跟 Transfer Learning 挺像的，Transfer Learning 就是让机器在任务一上学习，希望任务一上学到的技能，可以 Transfer 到任务二上面去。

但是 Life Long Learning 跟 Transfer Learning，虽然它们都是叫机器学多个任务，但它们的关注点是不一样的。在 Transfer Learning 里面，我们在意的事情是，机器在第一个任务上面学到的技能，能不能够对第二个任务有帮助，所以我们只在意在第二个任务上，机器做的好不好。但 Life Long Learning 的关注点是不一样的，Life Long Learning 的关注点是，当机器学完第二个任务的时候，回过头去再看第一个任务，它到底还能不能够解第一个任务。

所以虽然 Life Long Learning，跟 Transfer Learning，都会 Involve 两个任务，都要考虑两个任务。但 Transfer Learning 在意的是，新的任务做得怎么样，而 Life Long Learning 它是在意，旧的那个任务做得怎么样。

Evaluation

好 那在讲 Life Long Learning 的技术之前，我们来讲一下，怎么评估一个 Life Long Learning 的技术，做得好不好。

当然要做 Life Long Learning 之前，你得先有一把任务，让机器可以依序做学习，但是其实今天，**如果你看那些 Life Long Learning 的文献的话，所谓这个一把任务，往往都是比较简单的一些任务**，一个常见的 Setting 是这个样子，你的任务一 是做手写数字辨识，任务二其实还是手写数字辨识，但是这些点点，这些看起来像是星，这些看起来像是星星的图，到底是在做什么呢，这些是我们把每一个数字，用某一种特殊的，用某一种固定的规则把它打乱，那每一个任务，就是把数字做不同的打乱，就成为不同的任务。那 Permutation 呢，还算是比较难的，还有看过更简单的是，把所有的数字转，往右转 15 度算是新的任务，往左转 15 度也算是新的任务，那所以每一个任务都还是数字，只是呢图片的角度不太一样而已，那这样你也可以来研究 Life Long Learning，或者是另外一种状况是，你的任务一是要让机器分辨 0 跟 1，任务二是要让机器分辨 2 跟 3，任务三是要让机器分辨 4 跟 5，以此类推。但是对机器来说，0 就是第一个 Class，1 就是第一个 Class，然后 2 就是第一个 Class，3 就是第二个 Class，4 就是第一个 Class，5 就是第二个 Class，所以今天你给它一张 5，它是要判断 5 属于第二个 Class，判断 3 属于第二个 Class，1 属于第一个 Class，然后 0 2 4 属于第一个 Class，然后依序进行训练。

Evaluation

First of all, we need a sequence of tasks.

Task 1 (permutation 1)	Task 2 (permutation 2)	Task 10 (permutation 10)
...		

那这个 Life Long Learning, 这个作业是选择题, 那助教也有提供一些程式, 但是这些程式你能够跑过是最好的, 但我们要的就不是程式的结果, 是根据程式的内容问大家一些问题, 那其中一个送分的问题就是会问你说, 在助教的程式里面, 所谓的不同的任务是怎么样定义的。

看到同学有些问题, 那我来回答一下, 有方法可以让 NN 加一些约束, 让它和原本的参数不要差太多, 让它记得旧的任务怎么做吗, 太好了 其实就是这样解的。

好 那我们就继续吧, 那这边呢是有关任务 Sequence 的定义, 那再来要讲, 我们如果有一堆任务的话, 我们怎么评估一个 Life Long Learning 的演算法, 做得好不好呢?

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

你评估的方法是这个样子的, 好 你有一排任务, 有一排任务, 然后呢你先有一个随机初始化的参数, 把这随机初始化的参数, 用在这大 T 个任务上, 得到大 T 个正确率, 你有这大 T 个任务的 Casting Set, 把随机初始化的参数用在这大 T 个任务上, 得到大 T 的 Accuracy, 大 T 个 Accuracy. 然后接下来, 你让 Model 先学第一个任务, 学完第一个任务, 拿第一个任务的训练资料出来, 学完第一个任务以后在这 T

个任务上，再去量一次正确率，然后学完第二个任务以后，再去这 T 个任务上再量一次正确率，学到第 $T-1$ 个任务以后，在这 T 个任务上再学一次正确率，学完最后一个任务以后，在这 T 个任务上再算一次正确率，你会得到这样一个表格。接下来呢你会用这个表格呢，来判断一个 Life Long Learning 的 Model，做得怎么样。那在这个表格里面，每一个数值指的就是，某一个任务它的测试资料的正确率，那这个表格里面的每一个数值呢，都有两个下标 i 跟 j ，第一个下标代表说呢，这个是训练完第 i 个任务以后的正确率。第二个下标指的是，这是在第 j 个任务上的正确率。

比如说 $R_{2,1}$ 的意思就是说，现在呢你的 Model 刚学完任务 2，它在任务一的正确率上怎么样， $R_{T-1,2}$ 的意思就是说呢，你的模型刚学完大 $T-1$ 这个任务，在任务二上表现的怎么样。

好 所以如果我们今天看的，是 i 大于 j 的那些正确率的话，那意味着什么呢， i 是比较后面的任务， j 是比较前面的任务，就我们要知道，我们想要知道的事情是，今天让模型训练完任务 i 以后，它在过去已经训练过的任务 j 上，到底表现得怎么样，它有没有忘记任务 j 过去学过的东西。如果今天，我们看的是 i 小于 j 的那些 R 的话，那代表什么，代表是说我们刚学完任务 i ，还没去学任务 j ，但机器会不会就无师自通，已经会解任务 j 了呢，这边要看的是机器 Transfer 的能力，在新的没有看过的任务上，Transfer 的能力怎么样。

好 那最常见的，评估一个 Life Long Learning 系统的方法，就是把最后这个 R 的正确率加起来就结束了，你就让你的模型依序学过所有的任务，到最后的任务都学完以后，去之前所有的任务上面，都算一遍正确率，平均起来就代表，你的 Life Long Learning 的方法的好坏。当然 $R_{T,T}$ 可能会是最高的，因为刚学完任务大 T ，在任务大 T 上当然表现最好。在前面的任务，那你的模型就会逐渐忘记，第一个任务可能就是忘记得很惨了，完全忘记了，可能正确率是趋近于零的，第二个任务可能稍微好一点，正确率 1~2% 等等，那把这些所有的正确率平均起来，就是评估一个 Life Long Learning 系统的好坏，常见的用法。

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
After Training	Rand Init.	$R_{0,1}$	$R_{0,2}$		$R_{0,T}$
	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	$R_{T,1}$	$R_{T,2}$		$R_{T,T}$

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

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

(It is usually negative.)

那其实还有其他的评估方法，有一个评估方法叫做 Backward 的 Transfer，这个 Backward 的 Transfer 呢，是拿两个数值出来相减，它是拿 R_{T-1} 去减掉 $R_{1,1}$ ，拿 R_{T-2} 去减掉 $R_{2,2}$ ，以此类推，然后把所有的任务都加起来做平均，那这个值呢就是 Backward 的 Transfer，那这个 $R_{T,1}$ 减 $R_{1,1}$ ，这边是拿 $R_{T,1}$ 减 $R_{1,1}$ ，或者是 $R_{T,2}$ 减 $R_{2,2}$ ， $R_{T,2}$ 减 $R_{2,2}$ ，它们到底是什么意思呢？它们的意思是说，当你的模型先学完任务一的时候，在任务一上的正确率，跟学到大 T 个任务完以后，在任务一上的正确率差多少，当学完任务一的时候记忆犹新，这个时候正确率是最高的。那随着学的任务越来越多，那你的正确率就会

不断地递减，那到底会减少多少呢，所以我们把这两个 R 进行相减，可以评估说现在遗忘的程度有多严重。那因为呢机器每次看到新的任务以后，旧的任务就会不断遗忘，所以你可以想见说，RT-1 通常是比较 R1,1 小的了，RT-2 通常是比 R2,2 小的了，所以如果你是拿 RT,1 减 R1,1，RT,2 减 R2,2 的话，你通常得到的值是负的。所以 Backward Transfer，通常算出来的值是负的是小于 0 的，如果你今天可以提出一个，Life Long Learning 的方法，它很厉害，它的 Backward Transfer 算出来是正的，那你就很厉害了。因为通常 Backward Transfer 都是负的，如果说，机器学了新的任务以后，它还可以触类旁通，把原来的任务一做得更好，它学完新的任务以后触类旁通，把原来的任务二做得更好，那你就，你提出来的 Life Long Learning 就很厉害了，一般 Life Long Learning 做不到这件事，通常只要这个值不要负得太严重，就已经很厉害了。

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

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

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

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

好那还有一种评估方式，叫做 Forward Transfer，那 Forward Transfer，通常就比较不是 Life Long Learning 的重点，Forward Transfer 想要问的问题是说，在还没有看过某个任务，只看过其他任务的时候，机器到底已经学到什么样的程度。

所以你在做 Forward Transfer 的时候，你就是拿 RT-1,T 去减 R0,T，这个 RT-1,T 去减 R0,T，到底意思呢，它的意思是说，今天在还没有看到任务大 T 的时候，只看到任务 T1 到 T-1，只看到任务 T1 到 T-1 的时候，你的模型到底可以学出什么样程度的结果，这个是 Forward Transfer，好那接下来就是准备要进入，这个 Life Long Learning 的解法。

Research Directions

好那我们就继续讲吧，我们就继续讲吧，好那我们接下来呢，就是要讲三个 Lifelong Learning 的可能解法。

Research Directions

Selective Synaptic Plasticity

Additional Neural Resource Allocation

Memory Reply

Selective Synaptic Plasticity

那第一个解法，叫做 Selective Synaptic Plasticity，那从字面上，你可能一下子没有办法 Get 到说，这个方法到底想要做什么。这个 Synaptic 是突触的意思，就是我们脑神经中这个，神经跟神经之间的连结，这个叫做突触，Plasticity 呢，是可塑性的意思。所以简单来说，这个方法想要做的事情就是，我们只让我们的这个类神经网路中，某一些神经元，或某一些神经元间的连结，具有可塑性，Selective 的意思就是说，只有部分的连结是有可塑性的，有一些连结必须被固化，它必须不能够再移动，不能够再改变它的数值，那像这样的方法又叫做，Regularization-based 的方法。

突觸的 可塑性

Regularization-based Approach

Selective Synaptic Plasticity

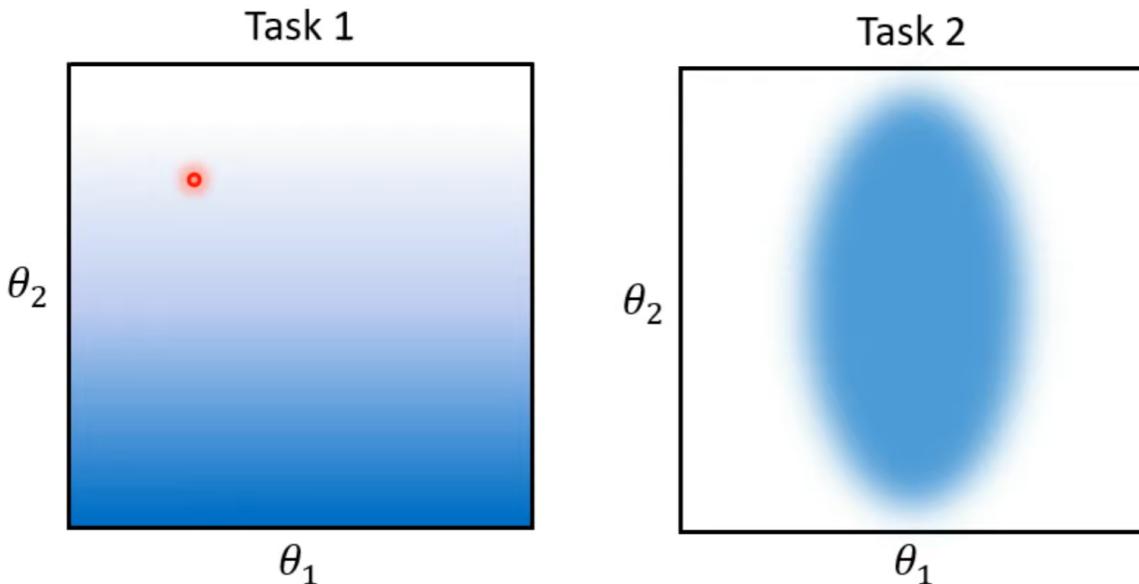
那这个面向，这个研究的面向，在 Lifelong Learning 的领域里面，我觉得是发展得最完整的，所以等一下我们会花比较多的时间，来讲 Selective Synaptic Plasticity，那另外两个面向呢，我们都只用一两页投影片很快地带过。

那你会发现作业里面主要的问题，也都集中在，跟 Regularization-based 有关的方法上面，

Why Catastrophic Forgetting?

好那我们先来想一下，为什么 Catastrophic Forgetting 这件事情，会发生呢？我们假设有任务一跟任务二，这两个任务，而这两个任务呢，我们假设我们的模型只有两个参数， θ_1 跟 θ_2 。那当然一个模型通常有上，上百万 上亿个参数，不过我们假设只有两个参数，好那这个投影片上这两张图，代表的是任务一跟任务二的 Loss Function，也就是在任务一上面，如果你的 θ_1 跟 θ_2 ，设不一样的值，你就会有不一样的 Loss，那我们用颜色来代表 Loss 的大小。如果颜色越偏蓝色，代表 Loss 越小，颜色越偏白色，代表 Loss 越大。好所以左右两张图分别就是任务一，跟任务二的 Loss Function，也就是他们的 Error Surface。

Why Catastrophic Forgetting?

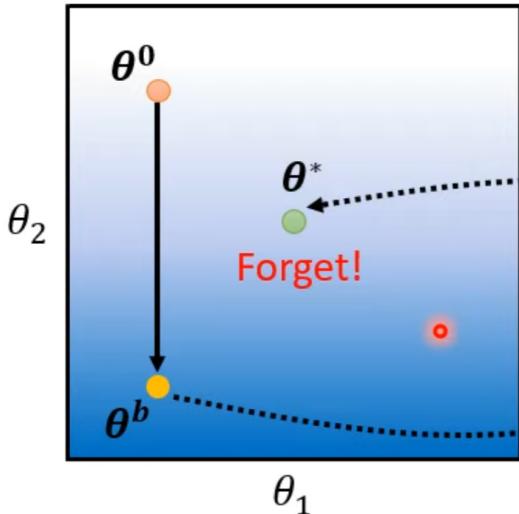


The error surfaces of tasks 1 & 2.
(darker = smaller loss)

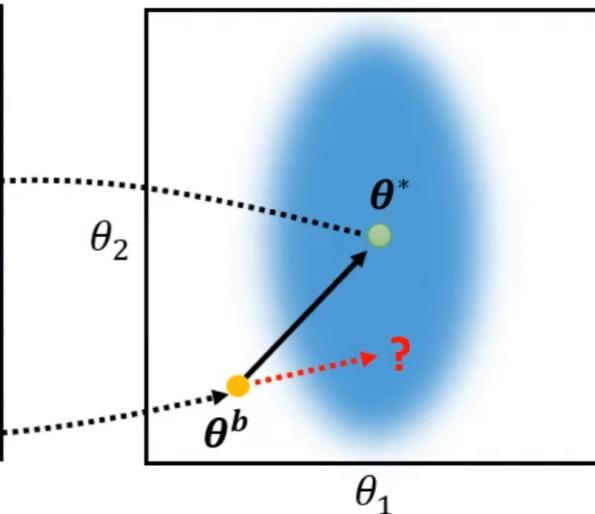
好那我们现在先让模型训练任务一，那模型怎么训练任务一呢，你要有一个随机初始化的参数，我们这边叫它 θ^0 ($\theta^0 = [\theta_1^0, \theta_2^0]^T$ ，这是一个二维向量)。然后我们会用 Gradient Descent 的方法，去调整 θ^0 的参数，那你就按照 Gradient 的方向呢，去 Update θ^0 的参数，得到 θ^0 ，好那假设 Update 够多次数，你觉得 Loss 降得够低了，那你就等于是把任务一学完了，那假设任务一学完后，我们得到的参数是 θ^b 。

接下来我们得继续解任务二，你就把 θ^b ，同样的参数拷贝过来，拷贝到任务二的这个 Error Surface 上面。注意一下，虽然左右两边 Error Surface 是不一样的，但是 θ^b 我们这边指的是同一组参数， θ^b 是用任务一训练出来的参数，我们现在把它用在任务二上，我们现在把 θ^b 放在任务二上，继续去做训练。那在任务二上，我们有另外一个不一样的 Error Surface，根据这个任务二的 Error Surface，去再 Update 参数。那我们可能会把 θ^b 往右上角移，那得到 θ ， θ 是训练完任务一，接下来又训练完任务二，依序训练两个任务以后所得到的参数，现在用 θ^* 来代表，依序训练完两个任务以后所得到的参数。这个 θ^* ，它在任务二上，是在一个 Error Surface 比较低，所以是在一个这个 Loss 比较低的位置，所以它在任务二上会得到好的表现，但如果你回头再把 θ^* ，拿回到任务一上去做使用，你会发现你并没有办法得到好的结果，因为 θ^* 只是在任务二上好，它在任务一上不见得会有低的 Loss。那这个就是 Forget 这件事情产生的原因。

Task 1



Task 2



The error surfaces of tasks 1 & 2.

(darker = smaller loss)

那要怎么解决 Forget 这个问题呢？对一个任务而言，也许有很多不同的地方，也许有很多组不同的参数，都可以给某一个任务低的 Loss，对任务二而言，也许在这个蓝色椭圆形的范围内，结果都算是够好的，也许在这个蓝色椭圆形范围内，Loss 都算是够低的，如果我们 θ^b 移动的方向，不要往右上移，而是只往左边移，那会不会把新的参数放到任务一上，就不会有 Forget 的情形呢。

Basic Idea

那这个就是我们等一下要跟大家分享的做法，好那怎么做呢，所以这边的基本的想法是说，每一个参数，对我们过去学过的任务，它的重要性是不一样的，有一些参数，也许对我们过去看过的任务特别重要，我们就希望在学新的参数的时候，那些旧的参数，那些重要的参数，它的值尽量不要变，新的任务只去改那些，对过去的任务不重要的参数就好。好我们现在假设 θ^b ，是在前一个任务所学出来的参数，所以 θ^b 在前一个任务上是好的，那我们会让 θ^b 在第二个任务上继续做学习，那在这个 Selective Synaptic Plasticity，这样的做法里面，我们会给每一个参数，一个保镖，一个守卫，我们这边用 b_i 来表示那个守卫。对每一个参数 θ_i ，我们都应该有一个守卫 b_i ，这边所谓的每一个参数就是，Neural 里面的每一个 Weight 跟它的 Bias，如果你的 network 有，100 万个参数的话，那就有 100 万个 b_i 的值，那它们每一个参数的 b_i 都是不一样，每一个参数都有一个各自的守卫。这个守卫代表什么，这个守卫代表说这个参数，对过去的任务而言，到底重不重要。

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

θ^b is the model learned from the previous tasks.

Each parameter θ_i^b has a “guard” b_i

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

The diagram illustrates the components of the loss function $L'(\boldsymbol{\theta})$. The first term $L(\boldsymbol{\theta})$ is labeled 'Loss for current task'. The second term $\lambda \sum_i b_i (\theta_i - \theta_i^b)^2$ is labeled 'How important this parameter is'. The third term $(\theta_i - \theta_i^b)^2$ is labeled 'Parameters to be learning'.

好 所以我们今天呢，在新的任务上，我们在 Update 我们的参数的时候，我们会改写 Loss Function，原来的 Loss Function，假设写成 $L(\boldsymbol{\theta})$ ，但我们不会直接去 Minimize $L(\boldsymbol{\theta})$ ，如果我们直接 Minimize $L(\boldsymbol{\theta})$ ，就会发生 Catastrophic Forgetting 的情形，就会发生灾难性的遗忘。所以我们要做的事情呢，是更改我们的 Loss Function，我们有一个新的 Loss Function，叫做 L' ，这个 L' 才是我们真正要去 Minimize 的对象，那这个 L' 呢，是原来的 Loss L 后面再多加了一项，这一项是什么东西，这一项是，我们先 Σ over，这边有个 Σ ，这边有个 Σ over 所有的 i ， Σ over 所有的参数。那我们把我们要 Learn 的那个参数，那个 θ_i 代表我们要 Optimize，我们 unknown 的那个参数，要找出来的那个参数，去减掉从过去的任务 Learn 出来的参数 θ_b ， θ_{bi} ， θ 上标 b 下标 i ，是 θ_b 就是过去的任务 Learn 出来的那个模型，下标 i 就是第 i 个参数，好那我们要让 θ_i 跟 θ 上标 b 下标 i ，越接近越好，所以我们呢，把 θ_i 跟 θ 上标 b 下标 i 相减，然后取它们的平方，那我们在前面呢，会乘上一个数值叫做 b_i 。这个 b_i 就是要告诉我们说，到底我们有多强烈地希望， θ_i 跟 θ 上标 b 下标 i ，越靠近越好，如果 b_i 的值很大，就代表说我们希望 θ_i 跟 θ_{bi} 非常靠近，如果 b_i 的值很小，代表我们认为 θ_i 没有跟 θ_{bi} 很靠近，也无所谓，好那这个呢，就是我们要 Optimize，这个 L' 呢，就是我们真正要 Optimize 的对象，它里面有两项，一个是原来的新的任务的 Loss，另外一项呢，就是要让 θ_i 跟 θ_{bi} 越接近越好。

但是要注意一下，我们并不是，平等地去看待，所有的参数要不要接近这件事，我们其实只要求 θ 跟 θ_{bi} ，在某些参数上接近就好，并不需要所有参数都接近，只要某些参数接近就好，那哪些参数要接近，就由 b_i 来控制。如果某一个参数 i ，第 i 个参数它的 b_i 很大，就代表我们希望第 i 个参数，它跟旧的参数，之前的任务 Learn 出来的那个参数，要非常接近，反之 b_i 等于 0 就代表，我们根本不 care 新的参数跟旧的参数，到底要不要接近。

θ should be close to θ^b in certain directions.

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

If $b_i = 0$, there is no constraint on θ_i  Catastrophic Forgetting

If $b_i = \infty$, θ_i would always be equal to θ_i^b  Intransigence

好 所以如果今天呢, b_i 设为 0, 所有的参数它的 i , 所有的参数它的 b_i 我们都设为 0, 那意味着什么, 那意味着就是, 我们没有给我们的 θ_i 任何限制, 我们完全没有要求新的 Learn 出来的参数, 跟过去学出来的参数有什么样的关系, 那这个时候就是一般的 Training, 就会有 Catastrophic Forgetting 的问题。但是你可能会想说既然 b_i 设 0 是不好的, 那我们就把 b_i 给它设一个非常大的值, 所有的 i , 所有的参数都给它一个非常大的 b_i , 那这样就不会有 Forgetting 的问题, 但是你会进入另外一个极端, 这个极端叫做 Intransigence。Intransigence 的意思就是, 这个 不肯妥协 不肯让步, 顽固的意思。所以 Intransigence 的意思是说, 假设你现在 b_i 非常地, 非常地大, 那你最后 Learn 出来的结果, θ_i 跟 θ^b 就会非常地接近, 你的新的参数跟旧的参数会非常接近, 那你的模型, 可能在旧的任务上不会遗忘, 但新的任务它学不好, 它没有能力去把新的任务学好, 那这种状况就叫做 Intransigence。

Q: 好 有同学问说, b_i 是要人为设定, 还是可以作为参数的一部分给机器训练,

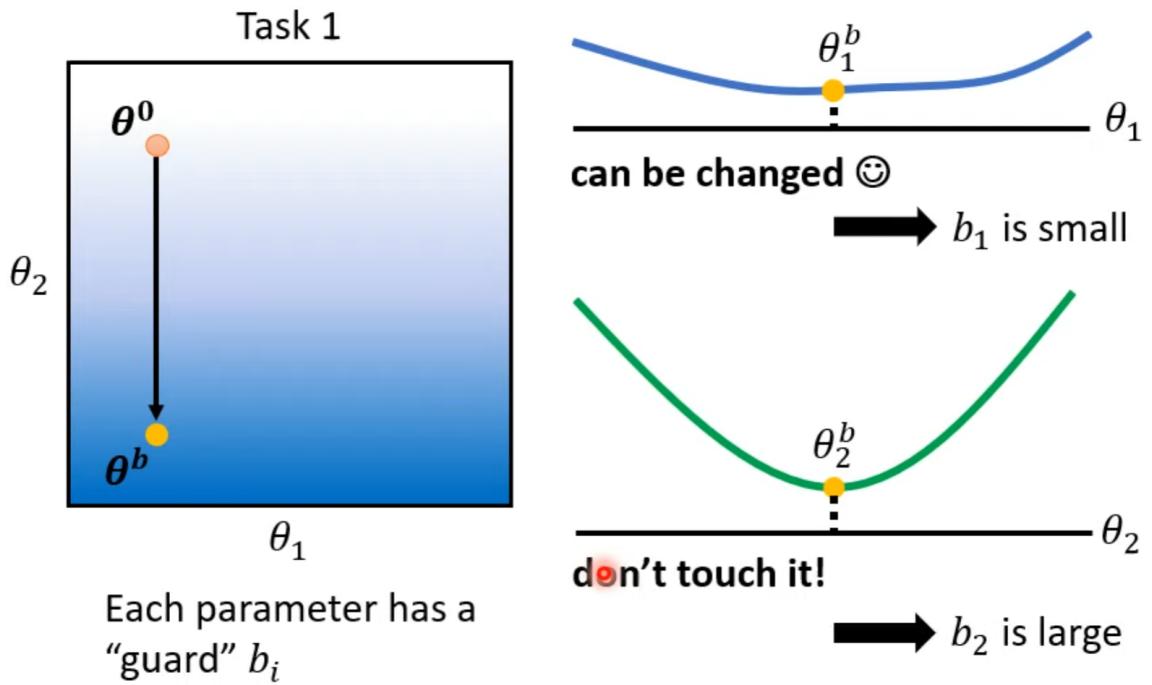
A: 好 那再等一下, 在文献上, 这个 b_i 都是人为设定的, 在 Lifelong Learning 的研究里面, 关键的技术就在于, 我们怎么设定这个 b_i , 那如果 b_i 用 Learn 的到底行不行呢, 你可以想见说在这个任务里面, 你恐怕不能让 b_i 用 Learn 的, 如果你让 b_i 自己学, 它会学出什么, 对它来说它要让 Loss 越小越好嘛, 那怎么让 Loss 最小, 直接 b_i 等于 0, Loss 就最小了, 所以如果你让 b_i 直接用 Learn 的, 你并没有办法达到 Lifelong Learning 的效果, 所以 b_i 是人为设的。

How to Design b_i

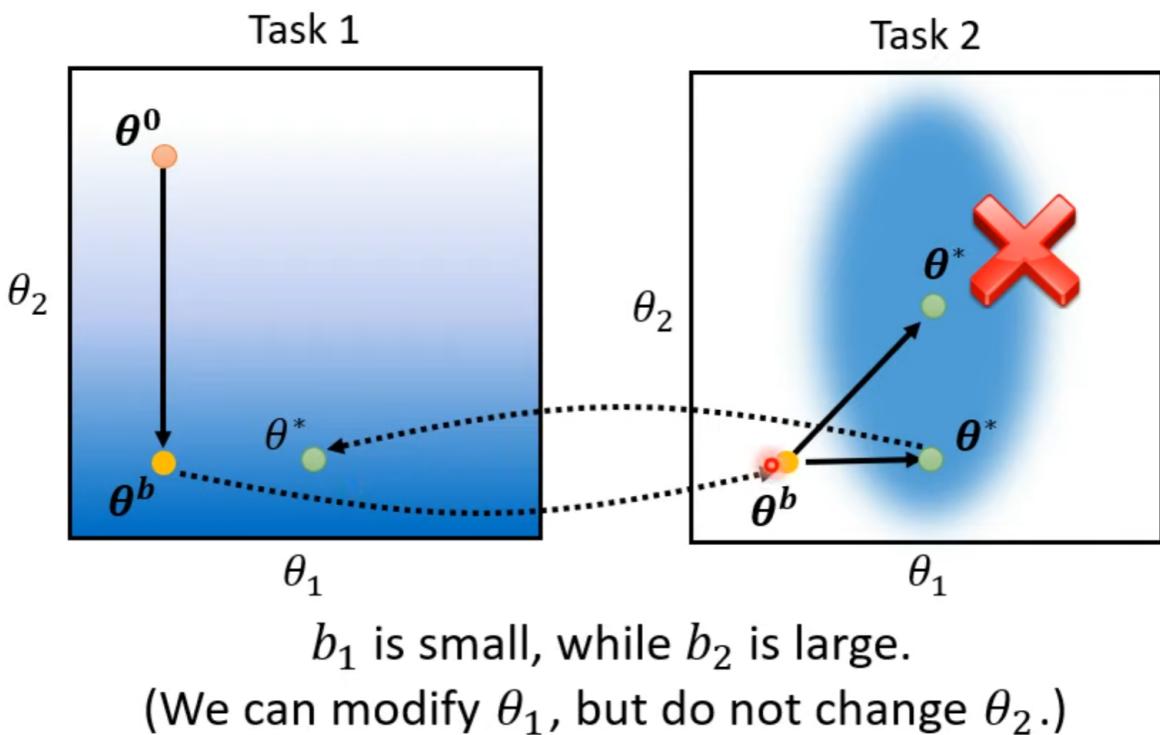
但是接下来最关键的研究问题就是, b_i 到底要怎么找到呢, 我们怎么知道, 哪些参数对旧的任务是重要的, 哪些参数对旧的任务是不重要的呢? 这个就是研究的重点。

我们这边呢 只跟大家用提示的方式, 简单地跟大家说, b_i 设计的大概念是什么。好 那怎么知道某一个参数, 对某一个任务到底重不重要呢? 那你可以在训练完一个模型之后, 我们得到 θ^b 之后, 看看 θ^b 里面每一个参数, 对这个任务的影响。举例来说, 你把 θ^b 在 θ_1 这个方向做一下移动, 发现说, 在 θ_1 这个方向上做移动, 好像对 Loss 没有什么影响, 那我们就知道说, θ_1 没有很重要, θ_1 对任务一没有很重要, 可以随便给它一个值, 那既然 θ_1 对任务一没有很重要, 在新的任务上, θ_1 就可以任意改变, 所以我们就可以给 θ_1 比较小的 b_i 的值, 也就是 b_1 的值就会比较小。

反过来说, 如果我们观察 θ_2 这个方向, 你会发现说, 当我们改变 θ_2 的值的时候, 对 Loss 的影响是大的, 我们改变 θ_2 的值的时候, 对 Loss 的影响是大的, 代表说 θ_2 是一个很重要的参数, θ_2 对 Task 1 是重要的参数, 所以你就不要去动它, 所以你要把 θ_2 的 b 设得大一点, 你要把 b_2 设得大一点。



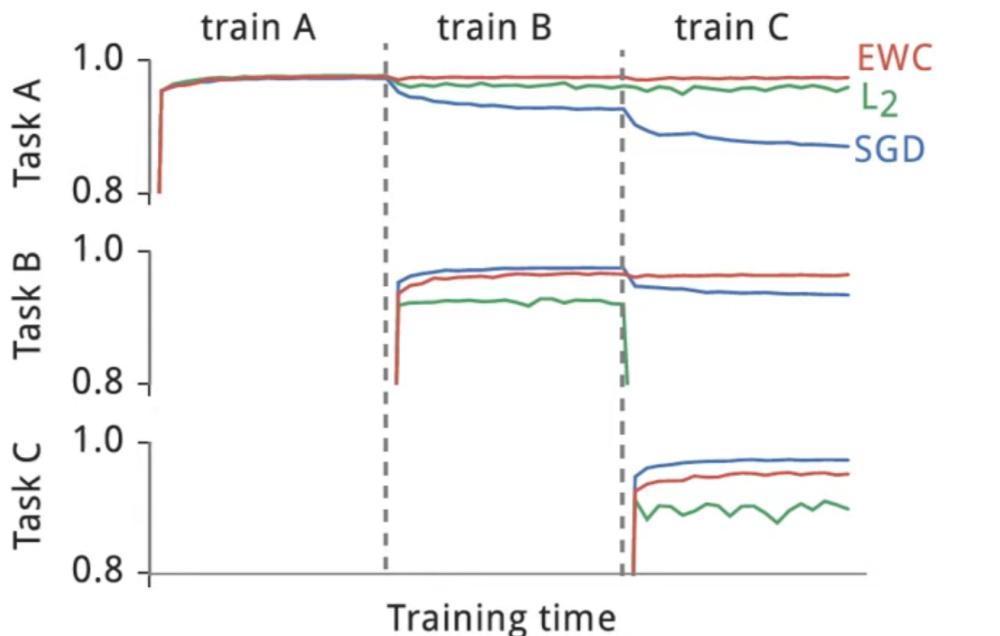
好 这个就是，Selective Synaptic Plasticity 的基本概念，而如果我们今天把 b_1 呢 设小一点， b_2 设大一点，那如果在 Task 2 训练的时候，会发生什么事呢？因为 b_1 比较小，代表说我们可以把这个模型在这个方向上，自由地移动，而 b_2 比较大，代表说在这个方向上自由地移动，是没办法的，所以如果我们把 b_1 设小一点， b_2 设大一点，那你把 θ_b 做 Update 的时候，它就不会往这个方向走，它就会倾向于往这个方向走，因为我们只希望模型去更新这个 θ_1 就好，尽量不要动到 θ_2 ，那你就可能把你的这个 Gradient 的方向呢，本来是这样 Update 的，那就变成这样子 Update，得到 θ^* 然后再把 θ^* 拿回来原来的任务一，那因为任务一呢 在这个方向上移动，对 Loss 的影响是小的，所以你在任务二上，假设只有在这个方向上移动，那对任务一的 Loss 的影响就小了，那所以新的 θ^* 用在这个地方，它的对 Task 1 的伤害就不大，也许你就可以借此做到，避免 Catastrophic Forgetting 的问题.



Examples

好那接下来这个呢，是一个文献上真正的实验结果，那这个结果呢，是来自于 EWC 这篇 Paper。EWC Paper 的连结，也有放在下一页投影片里面，如果你有兴趣的话再自己参考，好那这个图怎么看呢，你在看那个 Lifelong Learning 的文献的时候，这种类似的图是非常常出现的，常常 Paper 几乎都会放类似这样子的图。

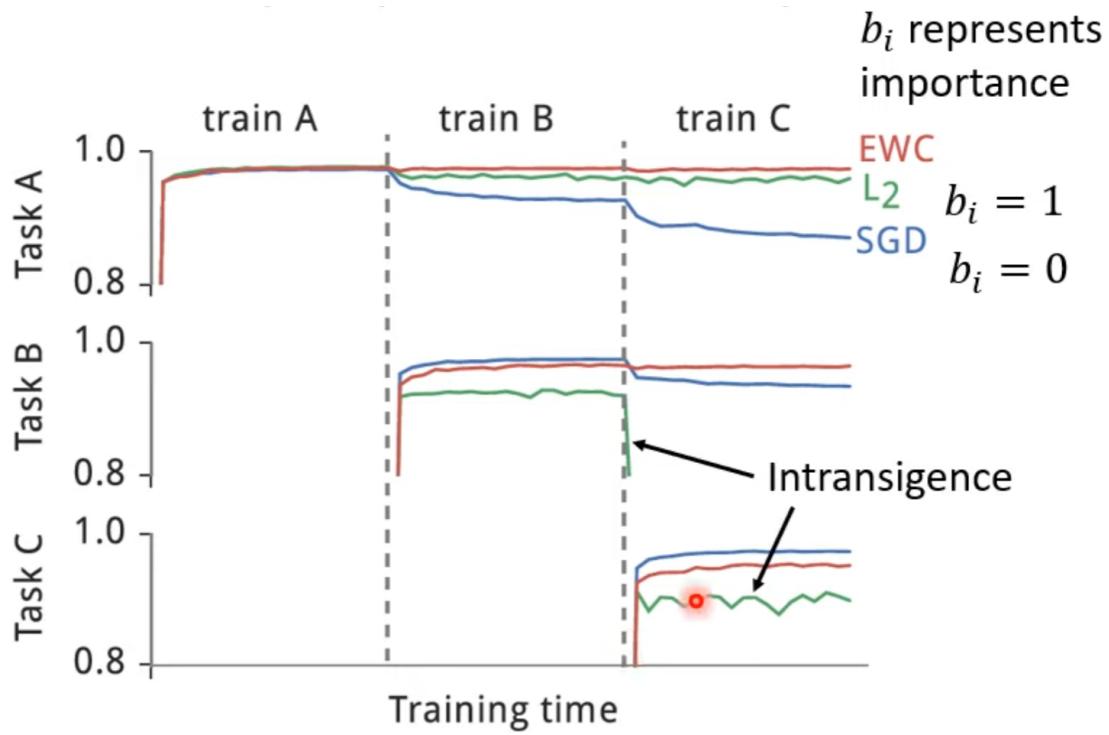
这个横轴是什么，横轴代表依序训练的过程，在第一个虚线左边，这个指的是训练任务 A，就我们有 A B C 三个任务，我们先训练任务 A，然后呢再训练任务 B，再训练任务 C，这三个任务依序训练的结果。那纵轴呢这边有三个纵轴，第一个纵轴代表任务 A 的正确率，第二个纵轴是任务 B 的正确率，第三个纵轴是任务 C 的正确率。那画这样一个图你就可以看出说，当我们依序学 A B C 三个任务的时候，任务 A B C 这三个任务，它的正确率会怎么样变化。



MNIST permutation, from the original EWC paper

我们先看任务 A 吧，我们先看任务 A 的变化情形，好我们先看蓝色这一条线，蓝色这一条线是什么呢，蓝色这一条线就是我们完全不管，Catastrophic Forgetting 的问题，就做一般的 Training，也就是 b_i 永远都设为 0，如果 b_i 永远都设为 0 会发生什么状况呢？你会发现说，我们看任务 A 的正确率，一开始刚学任务 A 的时候没有问题，正确率很高，接下来开始学任务 B 了，任务 A 的正确率就掉下来，接下来开始学任务 C 了，任务 A 的正确率又再掉下来，这个就是 Catastrophic Forgetting。

那 L2 呢，L2 这个实验是， b_i 不管哪个参数通通设 1，如果 b_i 不管哪个参数通通设 1，你看 Task A，确实有达到防止，Catastrophic Forgetting 的效果，举例来说你看绿色这条线，在任务 B 的时候没有下降很多，在任务 C 的时候，也没有下降很多，在训练任务 B 的时候，任务 A 的正确率，没有掉很多，在训练任务 C 的时候，任务 A 的正确率，也没有掉很多，但是 b_i 永远都设 1，你得到一个新的问题。这个问题呢，就是我们刚才提过的 Intransigence，我们来看一下，任务 B 跟任务 C 学习的状况，绿色这一条线，当 b_i 永远等于 1 的时候，我们在学任务 B 的时候，任务 B 的正确率，却没有升得足够高，这边这个第二个纵轴呢，代表是任务 B 的正确率，然后这边代表是学任务 B 的时候，我们在学任务 B 的时候，照理说任务 B 的正确率就应该飙升，但是没有，绿色这一条线没有飙升，学不起来，任务 B 学不起来，任务 C 更惨，更学不起来。横轴是这边是训练任务 C 的时候，纵轴是任务 C 的正确率，你发现任务 C 的正确率，没有其他方法高，代表任务 C 学不起来，这个就是 Intransigence。所以如果你给所有的参数一样的限制，那这个对你的模型来说限制太大了，会导致它新的任务学不起来。



好那如果我们给不同的参数，不同的 b_i ，就是有的参数 b_i 大，有的参数 b_i 小，只固定某些参数，某些参数可以任意更动，那你就得到红色这条线。那红色这条线，在每一个任务上表现都是最好的，看任务 A 的话，依序学三个任务，正确率没有掉，看任务 B 的话，在学任务 B 的时候，正确率跟蓝色这条线比起来只掉了一点点，然后任务 C 也不会再掉，那如果看任务 C 的话，任务 C 的正确率，其实相较蓝色这条线还是有，还是比较低一点，所以你有设这个 b_i 的时候，就是会有一些影响，新的任务就是比较难学，但是没有 b_i 都设 1 的时候结果那么惨，还是学得比 b_i 都设 1 的时候，结果还要更好的。

好那其实在课堂上，我们就没有真的告诉你 b_i 怎么算了，我们只讲了概念，那在助教的程式里面呢，助教实作了各种不同的方法，实作了各种不同算 b_i 的方法，那在选择题里面你要回答的就是，每一种方法，它的 b_i 是怎么求出来的，那你可以选择看文献，来知道 b_i 是怎么设的，你也可以选择直接读助教的程式，看看 b_i 是怎么被设出来的。那我们这边呢，就列了一大堆方法，有 EWC，有 SI，有 MAS，有 RWalk，有 SCP，那这边是按照那个年代设的，这边是按照年代，这边是按照年代放的，由最旧的方法到最新的方法，那每一个方法，都有它自己的特色，还有它想要解决的问题，它想要考量的点，那这个就是留在作业里面，让大家自己去发掘。那这个部分我们就不在课堂上讲，因为假设你对 Lifelong Learning 没有特别有兴趣的话，那每一个方法都讲一遍，你会觉得特别冗，但是假设你对 Lifelong Learning 有兴趣的话，那你把作业的选择题要好看一下，那你其实可以学到很多东西。

- Elastic Weight Consolidation (EWC)
 - <https://arxiv.org/abs/1612.00796>
- Synaptic Intelligence (SI)
 - <https://arxiv.org/abs/1703.04200>
- Memory Aware Synapses (MAS)
 - <https://arxiv.org/abs/1711.09601>
- RWalk
 - <https://arxiv.org/abs/1801.10112>
- Sliced Cramer Preservation (SCP)
 - <https://openreview.net/forum?id=BJge3TNKwH>

Q: 看来 b_i 可以直接用算的

A: 对, b_i 是直接用算的 算出来的, 但是 b_i 怎么算, 每一个方法都不一样, 而且每一个方法用的资料不一样, 有的方法, 只需要 Model 的 Input 就好, 有的方法要 Input 加 Output, 也就是假设是影像分类的问题的话, 有的方法只要 Image, 过去任务的 Image, 就可以算出 b_i , 有的方法是需要过去任务的 Image 加 Label, 才能算出 b_i , 那助教每一个方法都会问你说, 这个方法有没有用到 Label, 你再自己看一下助教的 Code , 看看有没有用到 Label

Q: 有同学问说, 改变训练 Task 的顺序, 训练出来的结果会不会差很多

A: 这个问题太棒了, 简单的回答就是会, 然后等一下会举一个例子告诉你说, 改变任务的顺序, 结果就会差很多, 所以你会发现说, 在这些 Paper 里面, 他们在做实验的时候, 都不是只做一种任务的顺序, 他们会穷举所有任务的顺序出来做实验, 然后再取它的平均值, 也就是假设你有三个任务, 你不能说, 你只把任务 A B C 跑完, 得到平均, 就说你的方法, 是得到的平均值是这个样子。在这些 Paper 里面, 常见的做法就是, 你会穷举所有的顺序, A B C C B A, B C A 通通都跑过一遍, 得那个平均值, 才能够代表说呢, 你的 Lifelong Learning 的方法, 做得好不好, 好, 那这个是有关 Regularization Based 的方法。

Gradient Episodic Memory(GEM)

那其实在 Regularization Based 的方法, 还有一个早年的做法呢, 叫做 GEM。

Gradient Episodic Memory, 这个方法呢, 也是一个挺有效的方法, 但它不是在参数上做限制, 而是在 Gradient Update 的方向上做限制。

那这个 GEM 是怎么做的呢, 这个 GEM 的做法是这个样子, 在任务二上, 我们会计算任务二的 Gradient。但是我们要小心一下, 我们到底要不要按照, 这个任务二算出来的 Gradient 的方向, 去 Update 我们的参数, 那在 Update 参数之前呢, 我们会先回去任务一上面, 算一下说, 这一个参数, 如果在任务一上做 Update 的时候, 它的方向到底是哪一个方向, 我们用蓝色的箭头, 代表 θ_b 这个参数, 在任务一上面的时候, 它 Update 的方向, 如果 θ_b 跟 g , 它们的方向不一致, 那这边所谓方向不一致就是, 它们的 Inner Product 小于 0 , 那这个时候怎么办呢, 就修改一下你的 g , 把它变成 g' , 那这边修改的条件, 修改的 Criterion 是, 你希望找到一个新的 g' , 这个 g' 呢, 跟 gb 呢, 它们做 Inner Product 以后, 要大于等于 0 , 而且 g' 跟 g 呢, 不能够差太多。

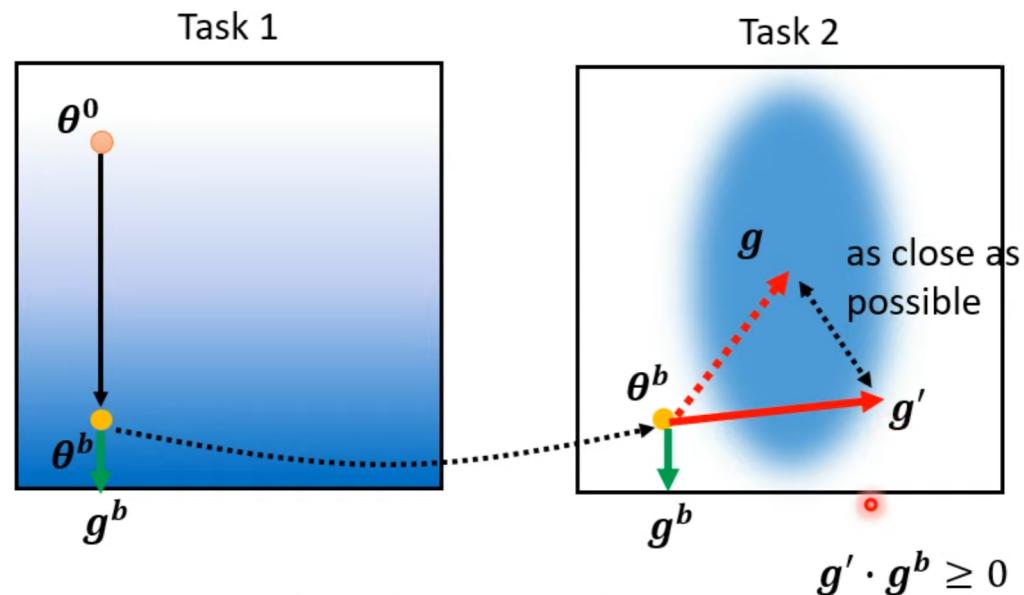
所以本来，你往这个方向 Update，可能会产生 Catastrophic Forgetting 的情形，但是我们刻意去修改 Update 的方向，从 g 变成 g' ，这样就可以减轻，Catastrophic Forgetting 所造成的问题，但讲到这边，你有没有发现这个方法有一点猫腻呢，有没有一点什么奇怪的地方呢，你仔细想想， gb 是怎么算出来的，我们要算 gb ，我们要算 Task 1 的 Gradient，意味着我们有存有 Task 1 的资料，如果我们没有存 Task 1 的资料，那我们根本就没有办法把 gb 算出来，所以 GEM 这个方法的一个劣势就是，它需要把过去方法的资料存下来。

那这个跟 Lifelong Learning 想要追求的，是有点不一致的，因为我们一开始就有说，Lifelong Learning 就是不希望，把过去的资料都存下来呀，如果过去的资料都存下来的话，资料累积得越来越多，那，那你最终会没有办法把过去的资料都存下来，所以 GEM，有点违反 Lifelong Learning 的最初的精神，它是有偷偷存过去的资料的，但是也许这个问题并没有特别严重。

为什么，因为 GEM 这个方法，只需要存非常少量的资料就好，因为这个 gb ，它最重要的工作，只是去修改一下 g 的方向，所以也许算 gb 的时候，我们不需要非常大量的资料，只要存一点点的资料就好，所以 GEM 想要做的事情是，希望透过只存一点点资料，来达到避免 Catastrophic Forgetting 的效果，所以 GEM 比较于其他方法，比如说我们刚才看到的 EWC 等等，有点不公平，因为它有偷存额外的资料。但是其实你再更仔细想一下，这个 EWC 这些方法，这些 Regularization Based 的方法，它们有，它们需要占用额外的空间，来储存旧的模型跟储存 bi ，所以刚才讲的那些 Regularization Based 的方法，它也需要占用到额外的空间，这些额外的空间，包括一个旧的模型，还有 bi 这个所谓的数值。所以如果 GEM，它今天虽然存了一些旧的资料，只要它存的旧的资料所占用的记忆体的量，没有比多存 bi 还有旧的模型多的话，也许也是可以接受的。所以如果 GEM 没有存太多资料的话，其实也是一个可以被接受的做法。

Gradient Episodic Memory (GEM)

<https://arxiv.org/abs/1706.08840>



---> : negative gradient of current task

—> : negative gradient of previous task

—> : update direction

$$g' \cdot g^b \geq 0$$

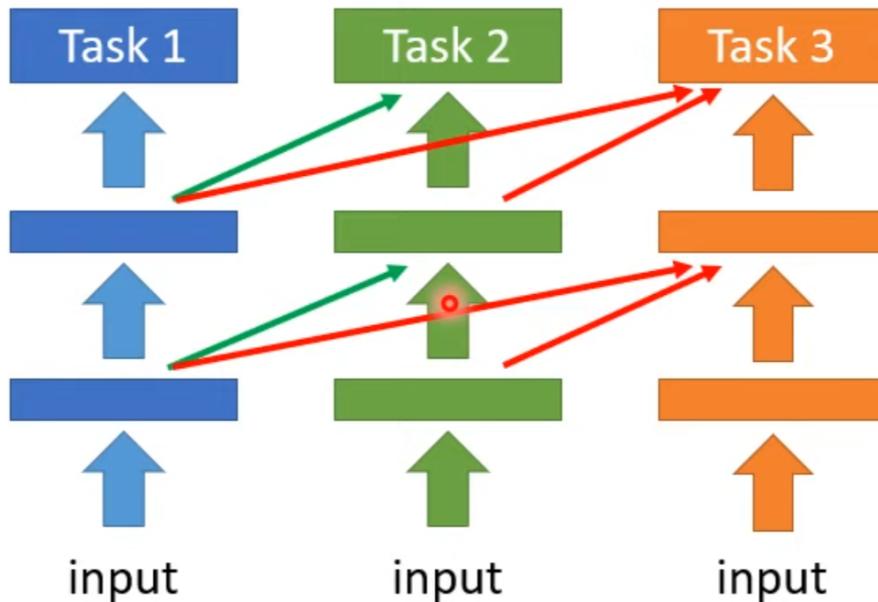
Additional Neural Resource Allocation

好 那接下来呢，另外两个做法，我们就是非常快地带过去。第一个做法呢，是 Additional Neural Resource Allocation。也就是我们改变一下，使用在每一个任务里面的 Neural 的 Resource。

Progressive Neural Network

什么意思呢，一个最早的做法，叫做 Progressive Neural Networks，它的想法是这个样子的，我们训练任务一的时候，有一个模型，那训练任务二的时候，你就不要再去动任务一学到的那个模型了，你另外再多开一个 Network，这个 Network 呢，它会吃任务一的，Hidden Layer 的 Output 作为输入，所以如果任务一有学到什么有用的资讯，任务二也是可以利用它的，但是任务一这边的参数，任务一学出来的参数，都不要再去动它了，我们只多新增一些额外的参数，我们只 Train 额外的参数，那任务三也是一样，我们有一组专门给任务三的参数，当训练任务三的时候，任务一、任务二训练出来的参数，就不要再动它了。

那对于解决 Catastrophic Forgetting 而言，这当然是一个有效的方法，你完全不会有 Catastrophic Forgetting 的问题，因为旧的参数，你根本完全没有动到它嘛。但是 Progressive Neural Networks 它会，它会造成的问题是，你每一次训练一个新的任务的时候，你会需要额外的空间，去产生额外的 Neural，你每一次加一个新的任务，你的模型就会长大一点，如果今天，你的模型长大的速率，跟新增任务是成正比的话，那你最终，你当你的任务不断地新增下去的时候，最终你的 Memory 还是会耗尽的，你的模型终究会太大，大到你没有办法把它存下来，所以 Progressive Networks，看起来并没有完全解决，Catastrophic Forgetting 的问题。但是在任务量没有很多的时候，Progressive Networks，仍然是可以派得上用场的。



<https://arxiv.org/abs/1606.04671>

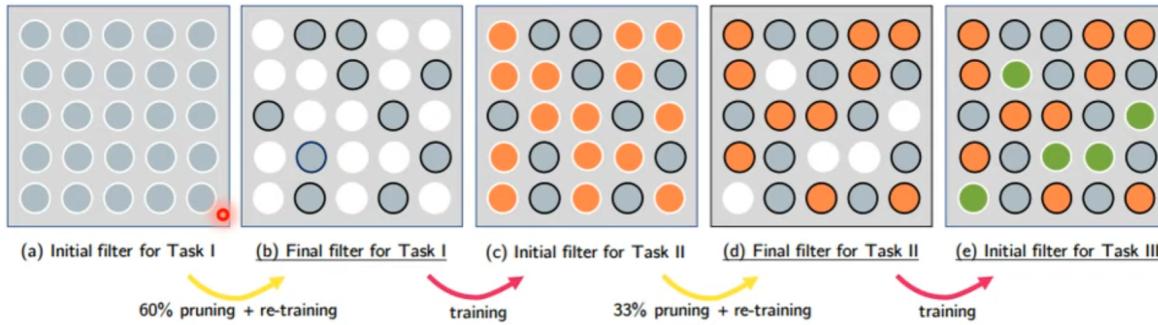
PackNet

然后有另外一个方法叫做 PackNet，它是 Progressive Networks 的反过来，Progressive Networks 是说，每一次有新的任务进来，我们就多加一些 Neural，那 PackNet 呢，正好是用另外一个想法，它说我们先开一个比较大的 Network，然后接下来，每一次有新的任务进来的时候，我们只用这个大 Network 的其中一部分，就任务一的资料进来，我们就这边在这个图示里面，我们就把每一个圈圈想成是，Network 里面的一个参数，然后任务一的资料进来，只准使用这边，有这个，黑色框框的这些圈圈的参数，然后任务二的资料再进来，只准用这边，橙色的参数，任务三的资料再进来，只准用这边，绿色的参数，那这样的好处就是，你的参数数量，不会随着任务增多，而不断增加，但是如果相较于 Progressive Networks 的方法，想这个方法其实也只是朝三暮四而已，我们只是一开始，开一个比较大的 Network，然后说，每一个任务不要把所有的参数都用尽，只用部分的参数，然后这样子，你就不会

有 Catastrophic Forgetting 的问题，但是相较于不断增加新的参数，你只是提早把更多的记忆体用完而已，那这个有点朝三暮四的感觉。

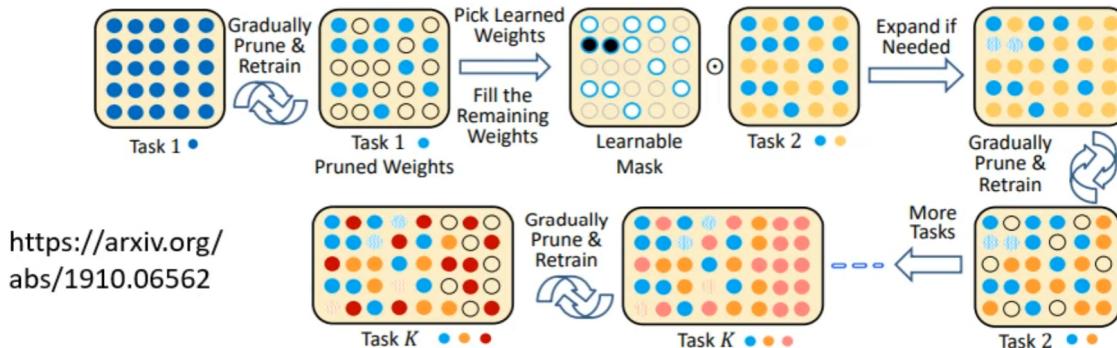
PackNet

<https://arxiv.org/abs/1711.05769>



然后 PackNet，跟 Progressive Networks，是可以结合在一起的，那这个结合的方法，也是一个很知名的做法，叫做 CPG: Compactig, Picking, and Growing。CPG，它就是我们的 Model，既可以增加新的参数，那每一次呢，又都只保留部分的参数，可以拿来做训练，那至于这些方法的细节我们就不细讲，就留给大家慢慢研究。

Compacting, Picking, and Growing (CPG)



Memory Reply

第三个做法，叫做 Memory Replay，第三个做法非常直觉，我们之前有讲说，只要把所有的资料通通倒在一起，就不会有 Catastrophic Forgetting 的问题，但我们又说，不能够存过去的资料，那我们干脆就训练一个 Generative 的 Model，这个 Generative 的 Model 就是，会产生 Pseudo-data，会产生，就我们不能够存过去的资料，但是我们训练一个 Generative Model，把过去的资料在训练的时候，即时地产生出来。

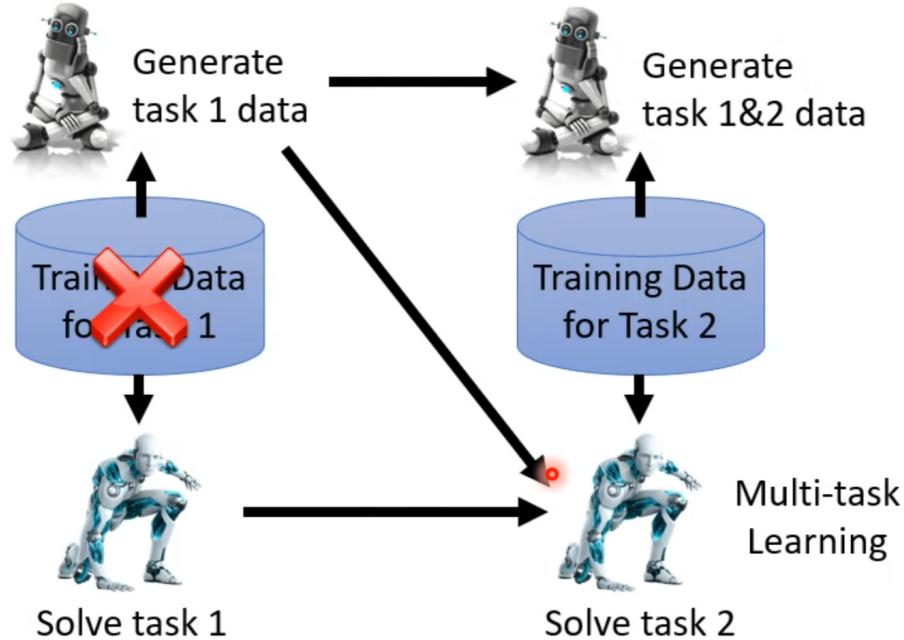
也就是说，我们现在有第一个任务的训练资料，我们不止训练一个 Classifier 来解任务一，我们同时训练一个 Generate，它会产生任务一的资料。接下来，你在训练任务二的时候，如果你只把任务二的资料倒给 Machine，那它可能会有 Catastrophic Forgetting 的问题，但是你又不能把任务一的资料拿出来，那怎么办，用 Generate 产生任务一的资料，你用这个 Generate，产生任务一的资料，给第二个任务的 Classifier 做训练。所以这个 Classifier，它在训练的时候，不是只看到任务二的资料，它还看到 Generate 产生的任务一的资料，所以用这个方法，就可以避免 Catastrophic Forgetting 的问题，然后接下来，你又有任务二的资料，那也许你就会把任务二的资料，跟任务一产生出来的。这个 Pseudo 的资料再倒在一起，再训练一个 Generate，这个 Generate 可以同时产生任务一，跟任务二的资料，然后这个过程，就反覆继续下去，好那这个方法，到底合不合理呢，就是见仁见智。因为你需要另外产生一个 Generate 嘛，那这个 Generate，当然也是会占用一些空间，但是如果这个 Generate，占用的空间，比你储存资料来讲，还要更小的话，那也许这就是一个有效的方法，那事实上呢，我们实

验室，也有做过一些 Lifelong Learning 的 Study，在我们的经验上，这一种 Generate Data 的方法呢，其实是非常有效的，用这种 Generate Data 的方法，往往你都可以逼近，Lifelong Learning 的 Upper Bound，往往你都可以做到跟，Multi-Task Learning 差不多的结果。

Generating Data

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

- Generating pseudo-data using generative model for previous tasks



Adding new classes

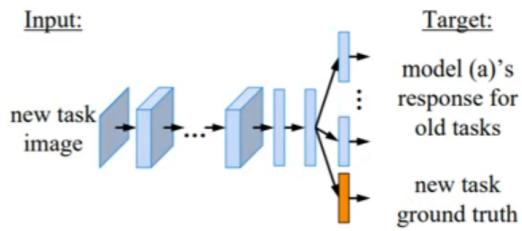
好那接下来，如果你想看，我们刚才讲的 Lifelong Learning 的 Scenarios，我们都假设说，每一个任务，需要的模型就是一样的，我们甚至强迫限制说，每一个任务，我们要训练的 Classifier，它们需要的 Class 量，都是一样的。

那假设不同的任务，它们的 Class 的数目不一样，有没有办法解呢？第一个任务，有 10 个 Class，第二个任务，有 20 个 Class，第三个任务有 100 个 Class，你训练新的任务的时候，你同时要增加新的 Class，有没有办法解呢？是有办法解的，那这边就列一些文献，比如说 Learning without forgetting，还有 LwF，还有 iCaRL，Incremental Classifier and Representation Learning，给大家参考，那助教呢，在这个，在我们的这个作业，Lifelong Learning 的作业的选择题里面呢，我们也问大家一些，有关这些做法的问题，如果你有兴趣，再自己去读一下这些文献。

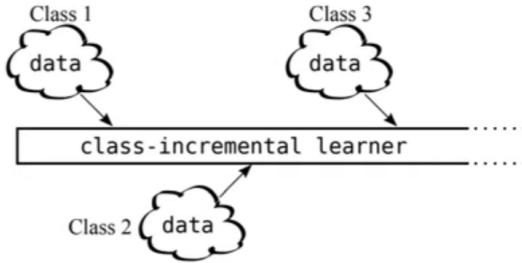
那其实，我们今天讲的 Lifelong Learning，也就是 Continual Learning，只是整个 Lifelong Learning 领域研究里面的，其中一小块，其中某一个情境而已。其实 Lifelong Learning，也就是 Continual Learning，还有很多不同的情境，你可以阅读一下下面这边统整的文献，它会告诉你说，Lifelong Learning 有三个情境，我们今天讲的，只是那三个情境里面，最简单的一种而已，最容易的一种，剩下两个更有挑战性的情境要怎么解，我们留，这个剩下另外两种更有挑战性的情境是什么，我们留在选择题里面，让大家自己去看看，另外两种情境是什么样子，好那这个呢，就是有关 Lifelong Learning 的三个研究方向。

Adding new classes

Learning without forgetting (LwF)
<https://arxiv.org/abs/1606.09282>



iCaRL: Incremental Classifier and Representation Learning
<https://arxiv.org/abs/1611.07725>



Three scenarios for continual learning

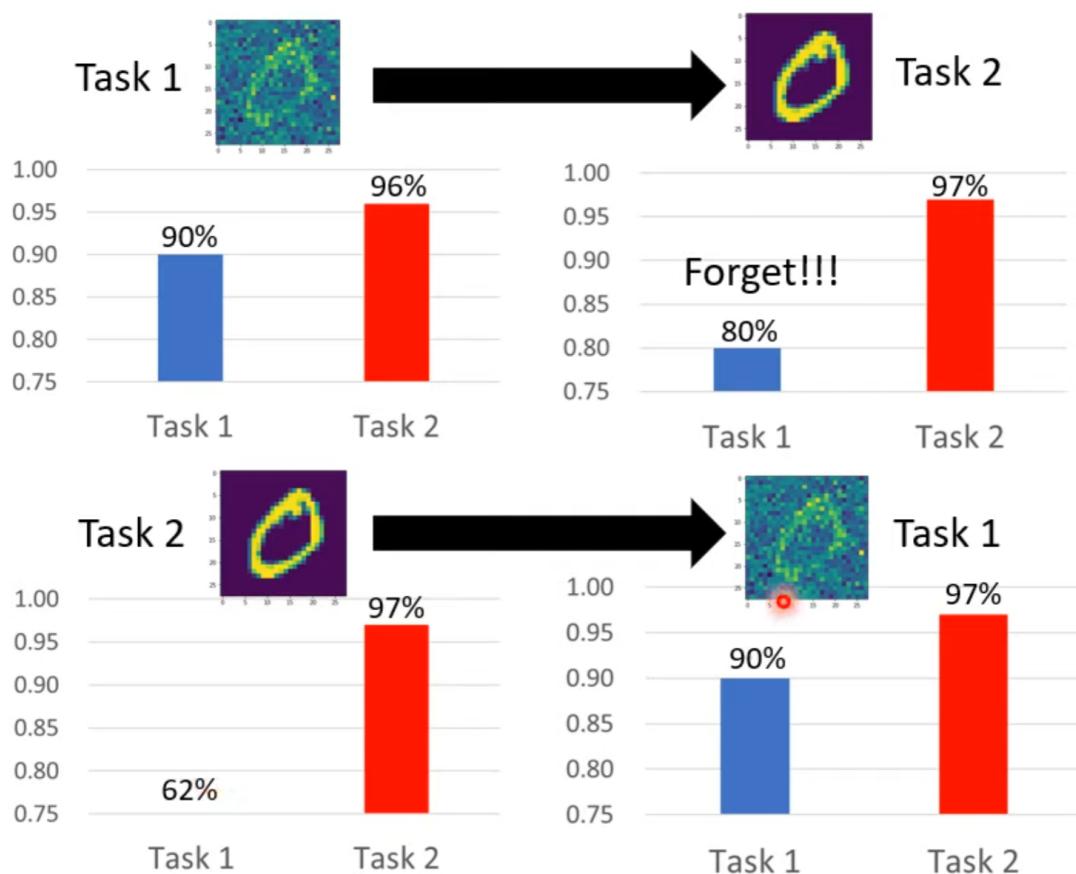
<https://arxiv.org/abs/1904.07734>

Curriculum Learning

那刚才有同学问到说，如果我们调换学任务学习的顺序，会不会有非常不一样的结果呢，确实是会有，那这边就举一个具体的例子来跟大家说明。

在刚才我们一开头讲的，Lifelong Learning 的例子里面，我们说，先让机器，先学这一种有杂讯的图片，接下来再学没有杂讯的图片，但是反过来，如果先学没有杂讯的图片，再学有杂讯的图片，会有什么样的状况呢，如果让机器先学没有杂讯的图片的话，在任务二上，正确率 97%，在任务一上，正确率 62%，看起来能够解没有杂讯图片的分类，看到有杂讯的图片，还是 Handle 不了的，但是如果说，我们更进一步让机器学任务一的话，这个时候你发现，它任务一 任务二，都可以做好，这个时候，没有 Catastrophic Forgetting 的问题，所以看起来任务的顺序是重要的，有一些顺序，会有 Forgetting 的问题，有一些顺序，其实也没有 Forgetting 的问题，而研究什么样的顺序才是好的，什么样的顺序，才对学习是有效的这个问题，叫做 Curriculum Learning，

Curriculum Learning



Curriculum Learning : what is the proper learning order?

