More About Lifelong Learning

楊舒涵

Lifelong Learning Programme

Just joking!

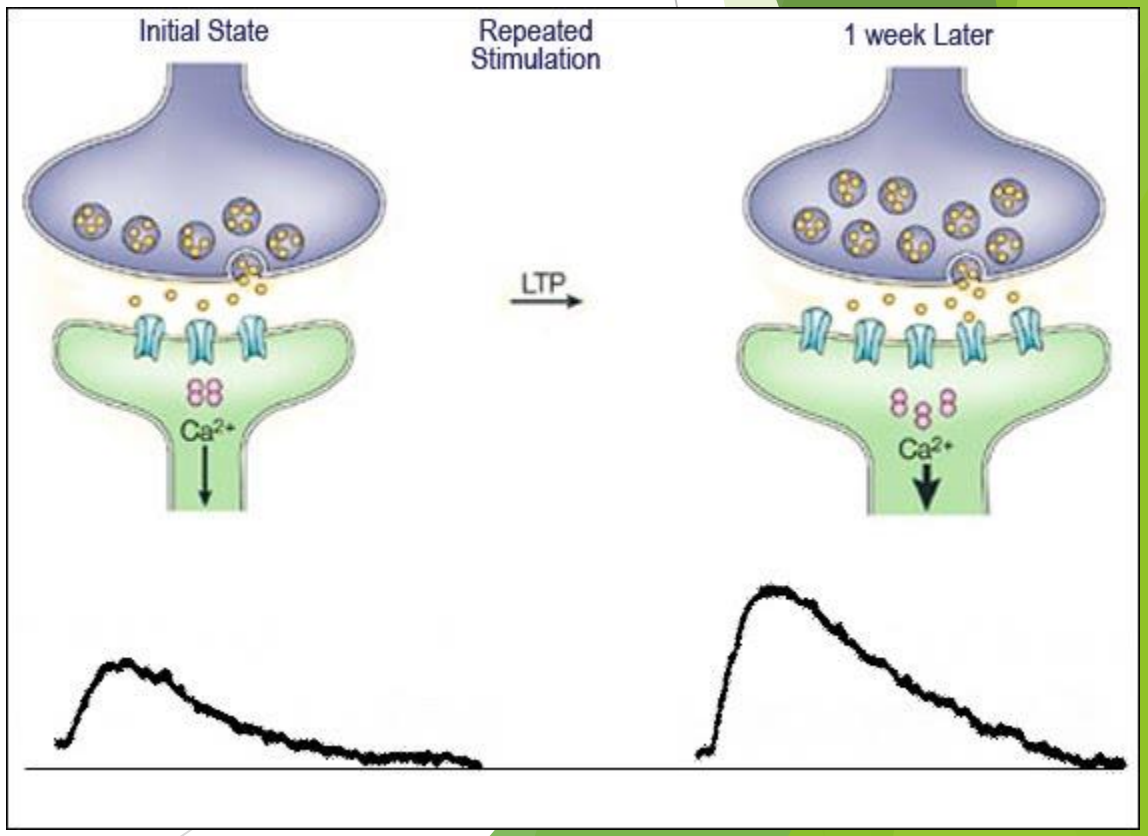
Outline

- **Hebbian Theory**
  - Long-term Potentiation(LTP)
  - Hebbian Theory
  - Competitive Hebbian Learning
- **Knowledge Distillation**
- **Memory Aware Synapses : Learning what (not) to forget (MAS)**
- **Learning without Forgetting (LwF)**
- **Large Scale Incremental Learning (BiC)**
  - Few-Shot Class-Incremental Learning (FSCIL)
- **LLL Nowadays & Future(?)**

Hebbian Theory
(Humans & Machine Learning)

長期增強效應（LTP)

- 學習與記憶一般被認為是透過過改變突觸強度進行的
- LTP最早被發現於兔子的海馬迴，後來在生物的小腦、大腦皮質、杏仁核都發現到此現象
- LTP和構成、強化學習與記憶相關
- 突觸可塑性說明突觸強度是會改變的
- LTP是突觸可塑性的例子之一：透過長期高頻率的刺激讓突觸強度增強的現象(1966 實驗於兔子海馬迴)



赫布理論 (Hebbian Theory)

- 解釋了在學習過程中腦神經所發生的變化的神經科學理論
- 赫布理論描述了突觸可塑性 ： 突觸前神經元向突觸後神經元的持續重複刺激，可以導致突觸傳遞效能的增加
- 在NN中，可將神經突觸間傳遞作用的變化看作是weight的變化

$w_{ij} = x_i x_j$, w_{ij} : weight between node j and node i

x_i : input of node I

Or

$$w_{ij} = \frac{1}{p} \sum_{k=1}^{p} x_i^k x_j^k$$, p : number of training patterns

Competitive Hebbian Learning

- 非監督式學習，input傳入後，神經元群體會競爭對外界刺激的響應能力
- 競爭取勝的神經元的weight變化往對這個刺激模式競爭更為有力的方向進行
- 任何時候output layer的神經元中有「恰好一個」是active (output最大者)
- 只有一個active : 適合用於classification和clustering

Knowledge Distillation

Problem Statement

訓練model和使用model的需求不同、model compression

Butterfly

Caterpillar

攝食、成長
飛行、繁衍

Problem Statement

訓練model和使用model的需求不同、model compression

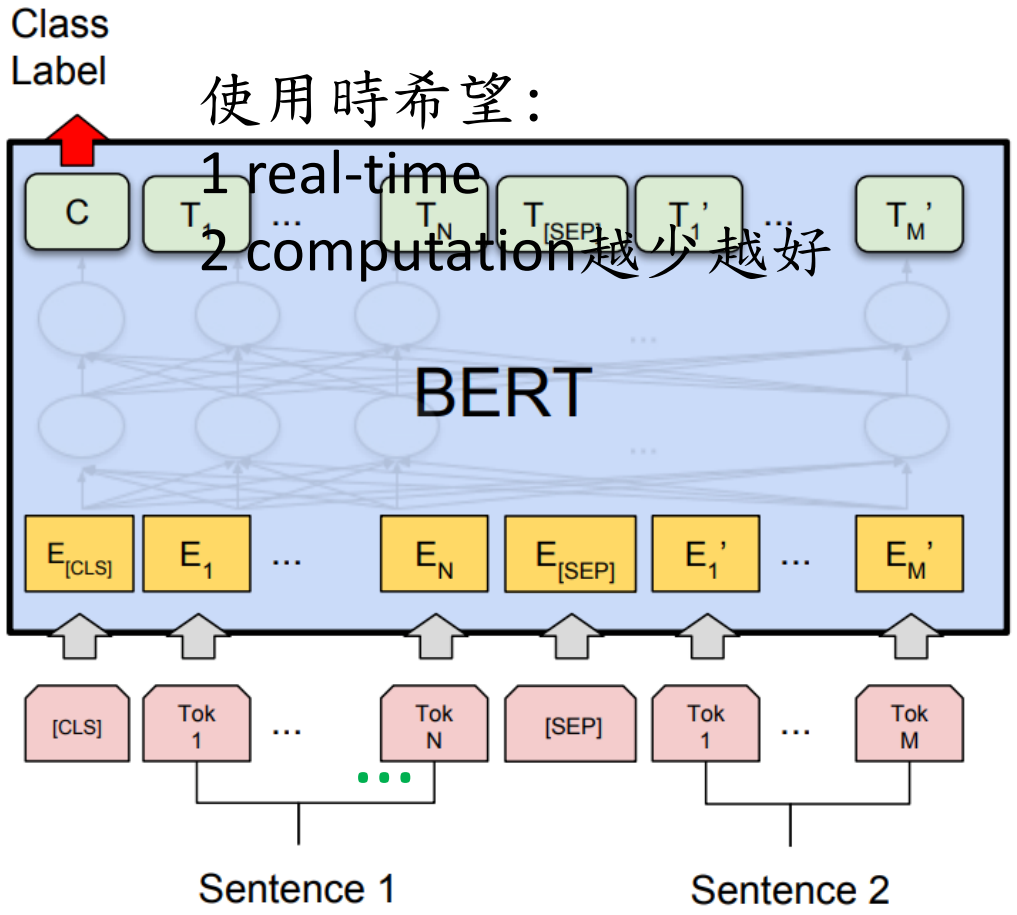
訓練時想達到好成效的方法:

1. 利用多種不同方法，再平均各方法的結果
2. 單一模型使用大量的dataset

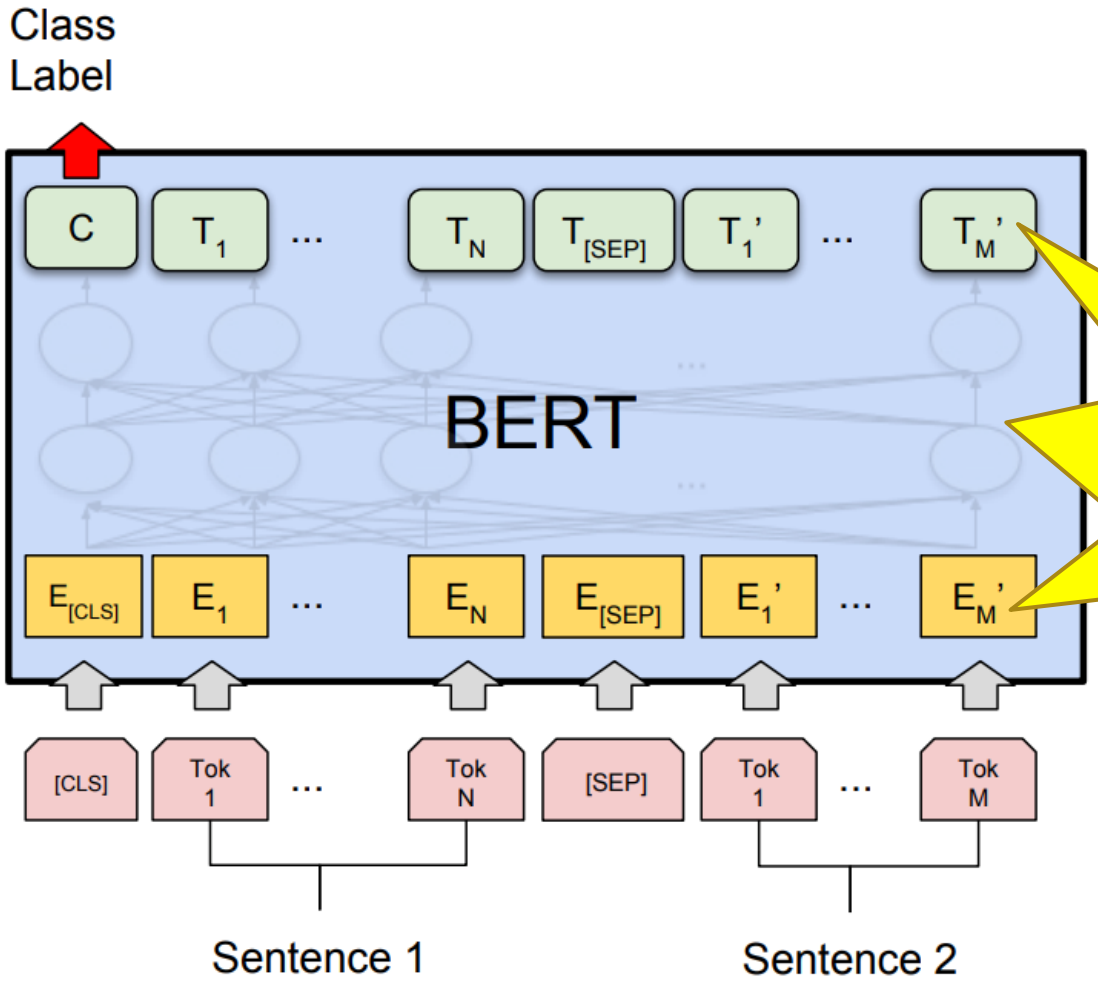
使用時希望:

1 real-time

2 computation越少越好



Is it possible?



差不多效果?!

小模型

訓練結果 → 測試用

What is "Knowledge"?

Knowledge : 平時可能會覺得是整個模型的參數、或機器學習算法，但這裡會將其認為是**學習把input vector映射到output vector**

儘管正常model在學習後大多能夠在正確的類別上有最大的機率(判斷正確)，但還是會有判定錯誤的時候，而不同model判定錯誤的機率也會有差

預測錯誤的模型的預測機率可以反映model對新鮮樣本的適應能力

Ferrari Car模型 (99%) ←

Bus模型 (0.95%) ←

19倍

Carrot模型 (0.05%) ←

Knowledge Distillation

一般model的class probabilities通常都使用softmax output layer將算式的z_i經考慮所有類別的情況後轉換成 機率q_i，以此和其他z component作比較

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

T又被稱為temperature(蒸餾的溫度)，當 T= 1時，即為普通的softmax function

What is "Temperature" for?

可以在此對 $$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$ 做觀察：

若現在z = [3, 10]，則在不同蒸餾溫度下呈現的類別機率為：

T = 1時， 類別機率陣列 : [0.091%, 99.91%]

T = 5時，類別機率陣列 : [19.78%, 80.22%]

T = 10時，類別機率陣列 : [33.18%, 66.82%]

由上面的觀察中，可以看出蒸餾溫度T越大，機率會越接近。

換言之，T越大，function會越soft

Knowledge Distillation

由前一頁的結論，可定義出soft targets (由蒸餾法算出的class probability)和hard targets (softmax生成的class probability)

這裡可以想到一個方法達成knowledge distillation : 可由大model產生soft targets 當成小model的ground truth 訓練小model，而soft targets含有更高的資訊量(entropy)，故能使小model用更少的樣本、更大的learning rate去學到接近大model的表現

最後，把小model在 soft targets 上訓練後的cross entropy loss，加上在hard targets上訓練後的cross entropy loss乘上 1/T^2 相加得到overall loss (Hard targets 上面乘1/T^2是因为 soft targets 生成過程中蒸餾法求導函數會產生1/T^2 ，而為了保持兩個loss 的影響接近才這麼做的)

Why knowledge distillation works?

可以想成T越大，class probabilties就越相近，使訓練上更嚴格，
而切換回一般softmax就回歸容易模式，故效果較佳。

平時訓練綁鉛塊的小李(辛苦)

比試時拿掉鉛塊的小李(輕盈)

Memory Aware Synapses : Learning what(not) to forget (MAS)

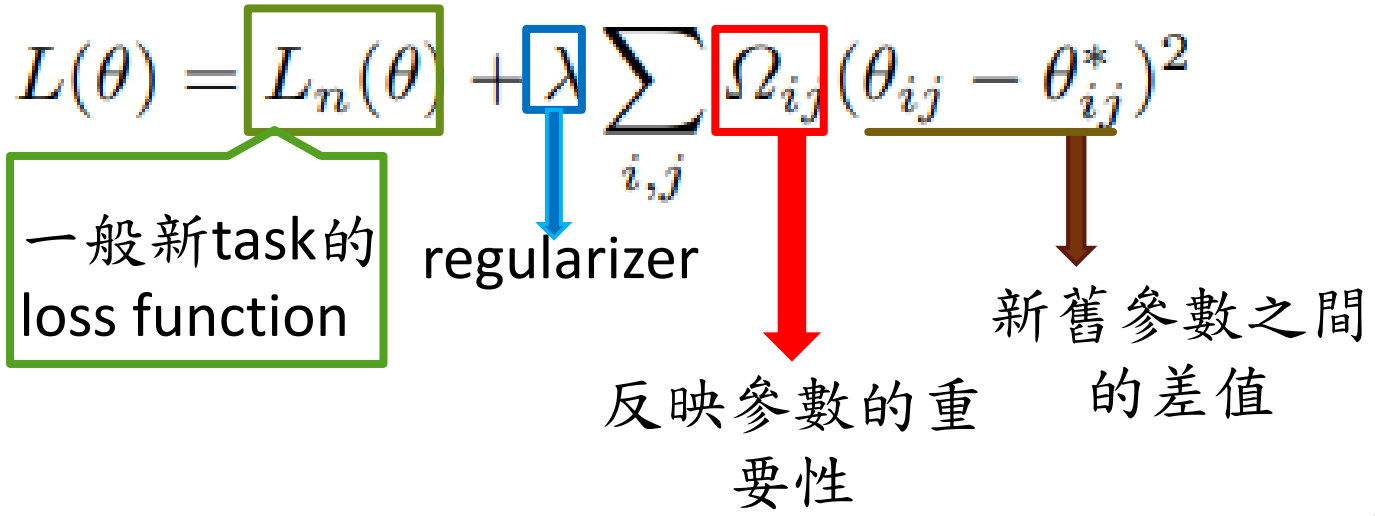
Problem Statement

本篇要解決的課題為: 隨著學習新tasks，不斷累積新知

改善災難性遺忘

Model-Based : 在訓練過程中不會額外新增參數，也不會使用到過去的資料。Loss function部分，除了原有的loss，還會新增一項代表各參數的重要性的正則項。

Ex:



達成項目

1. Constant Memory：Model佔用的內存應為constant，避免內存的使用隨著任務的數量進行增加
2. Problem Agnostic：Model通常來說，不該被限制在某種特定類型的task (Ex: 只做分類)
3. On Pretrained：可以在已有的優良pretrained model上，根據一己需求使用並並調整些微參數
4. Unlabelled Data：能夠用於unlabeled data和unsupervised learning (第一篇達成者)
5. Adaptive：對於固定capacity的NN来说，能夠適當調整必須保留的參數量即可為之後更多新的tasks保留更多學習的capacity

Concepts

過去包含EWC等論文，importance的定義通常和參數改變對loss的敏感程度相關，而MAS則有所不同:

$$F(x_k;\theta+\delta)-F(x_k;\theta)\approx\sum_{i,j}g_{ij}(x_k)\delta_{ij}$$

上式中的F為此model訓練出的學習函數；x_k為input的點；

θ為此model F的各個參數；δ為微小的參數改變

根據像是泰勒展開的公式，而在 $g_{ij}(x_k)=\frac{\partial(F(x_k;\theta))}{\partial\theta_{ij}}$ 下，可完成上面的簡化。

意思為: 更動參數對「model自己學得的 learning function」影響是否敏感

Concepts

根據計算上的方便和後續推展(左右都取L2-square)，可將 $g_{ij}(x_k)$ 重新定義為：

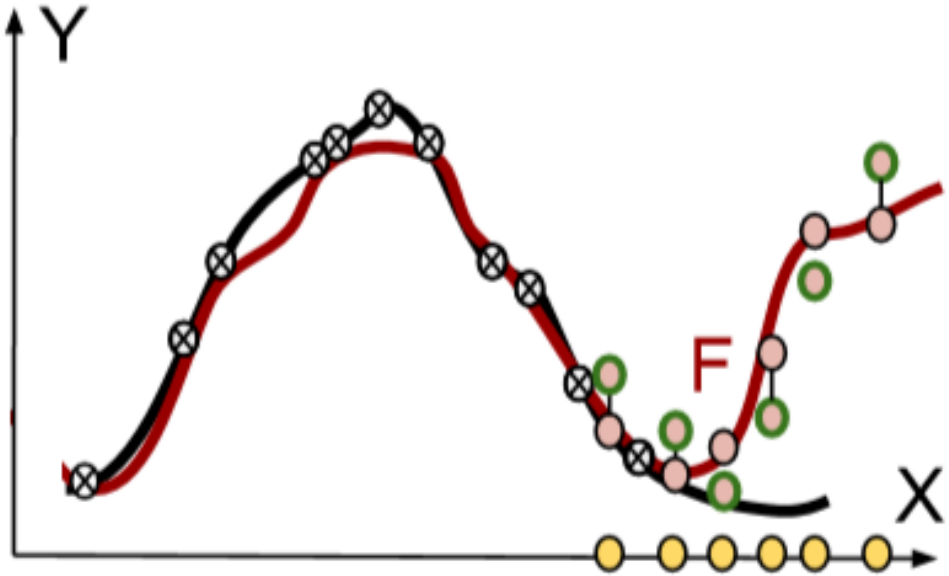
$$g_{ij}(x_k) = \frac{\partial[\ell_2^2(F(x_k;\theta))]}{\partial\theta_{ij}}$$

對Scalar加總比對vector加總容易處理，而其重要性權重為:

$$\Omega_{ij} = \frac{1}{N}\sum_{k=1}^{N} \| g_{ij}(x_k) \| \text{，} N \text{為所有點的數量}$$

最終可得到：$L(\theta) = L_n(\theta) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta_{ij}^*)^2$

Learned function vs. Loss



Importance和learned function相關：

Importance和loss相關：

新增的unlabelled data是無法的通常會看自身預測值和ground truth的差別

Importance和loss相關：

因為沒有那些點的ground truth

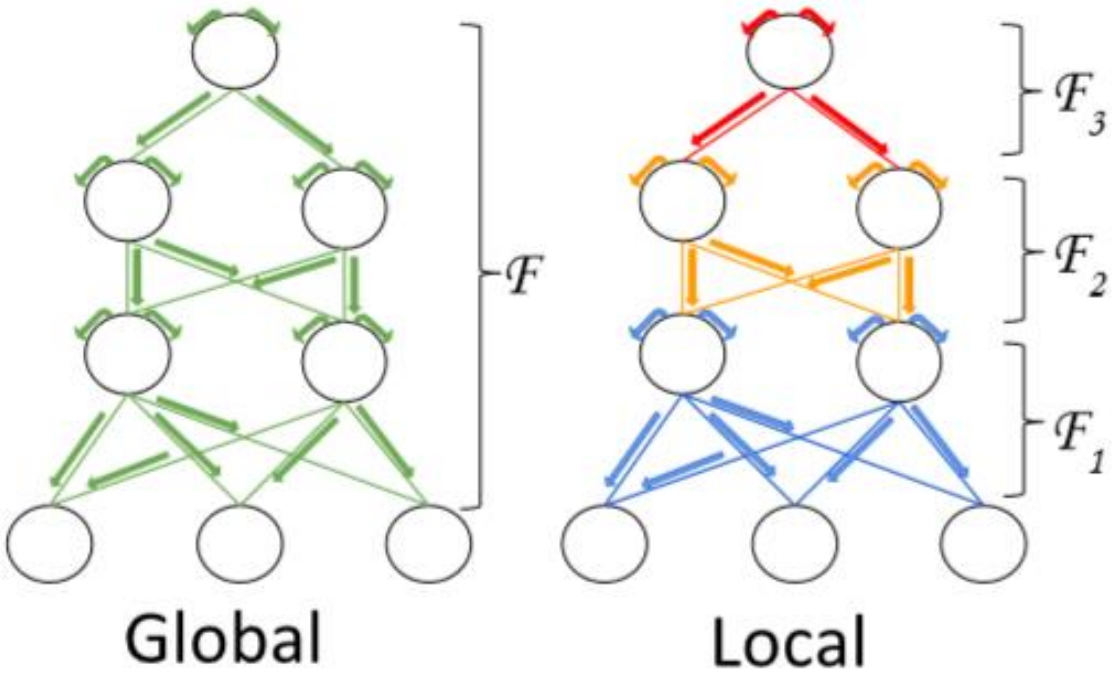
learned function為mapping function，故不一定要ground truth (可添加unlabelled data一起訓練)

Connection to Hebbian Learning

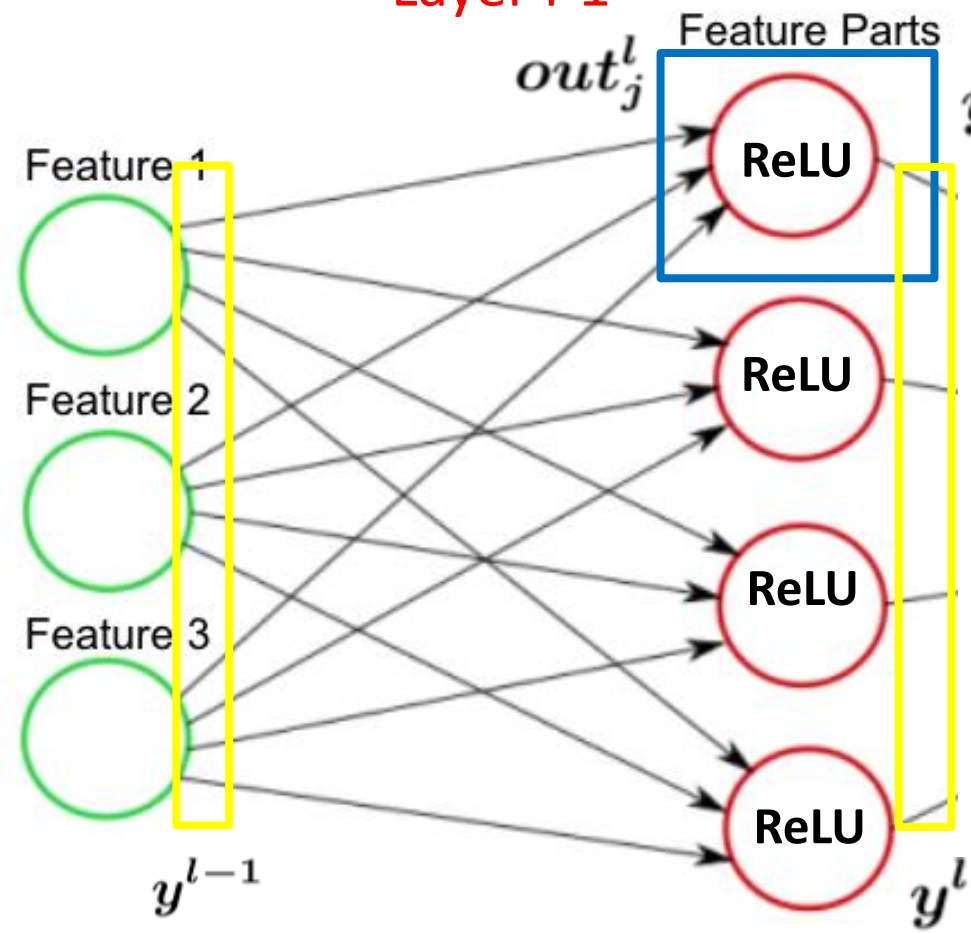
一個learned function F 可以以每一層為單位分開拆解成

$$F(x) = F_L(F_{L-1}(...(F_1(x))))$$

下圖為示意圖



Connection to Hebbian Learning



$$y^{l-1} = \{y_i^{l-1}\} \quad y^l = \{y_j^l\}$$

$$y^{l-1} = F_{l-1}(...(F_1(x)))$$

Local function : F_l

$$\ell_2^2(F_l(y^{l-1};\theta_l+\delta_l)) - \ell_2^2(F_l(y^{l-1};\theta_l)) \approx \sum_{i,j} g_{ij}(x)\delta_{ij}$$

A fully connected layer with I ∗ J parameters :

$$y_j^l = ReLU(out_j^l) \; out_j^l = \sum_{h=1}^{I} \theta_{hj} * y_h^{l-1}$$

$$g_{ij}(x) = \frac{\partial[\ell_2^2(F_l(y^{l-1};\theta_l))]}{\partial\theta_{ij}}$$

$$= \frac{\partial[\sum_{o=1}^{J}(y_o^l)^2]}{\partial\theta_{ij}} = \sum_{o=1}^{J} \frac{\partial((y_o^l)^2)}{\partial\theta_{ij}}$$

Connection to Hebbian Learning

Since $\frac{\partial((y_o^l)^2)}{\partial\theta_{ij}} = 0$ when $o \neq j$, we get

$$g_{ij}(x) = \frac{\partial((y_j^l)^2)}{\partial\theta_{ij}} = 2 * y_j^l * \frac{\partial(y_j^l)}{\partial\theta_{ij}} = 2 * y_j^l * \frac{\partial(ReLU(out_j^l))}{\partial\theta_{ij}}$$

1. if $out_j^l > 0$, $ReLU(out_j^l) = out_j^l$ and $(ReLU)' = 1$:

$$\frac{\partial(ReLU(out_j^l))}{\partial\theta_{ij}} = \frac{\partial(ReLU(out_j^l))}{\partial out_j^l}\frac{\partial(out_j^l)}{\partial\theta_{ij}} = \frac{\partial(out_j^l)}{\partial\theta_{ij}}$$

$$= \frac{\partial(\sum_{h=1}^{I}\theta_{hj} * y_h^{l-1})}{\partial\theta_{ij}} = \frac{\partial(\theta_{ij} * y_i^{l-1})}{\partial\theta_{ij}} = y_i^{l-1}$$

$$\Rightarrow g_{ij}(x) = 2 * y_j^l * y_i^{l-1}$$

Connection to Hebbian Learning

2. in the other case, $ReLU(out_j^l) = 0$ and $(ReLU)' = 0$, so

$$g_{ij}(x) = 0$$

At the same time,

$$ReLU(out_j^l) = 0 \Rightarrow y_j^l = 0 \text{ and } 2 * y_j^l * y_i^{l-1} = 0$$

Hence

$$g_{ij}(x) = 2 * y_j^l * y_i^{l-1} = 0$$

綜合1、2，可得到 $g_{ij}(x) = 2 * y_i^{l-1} * y_j^l$

和前述介紹的Hebbian Theory $w_{ij} = x_i x_j$ 相像

Contribution

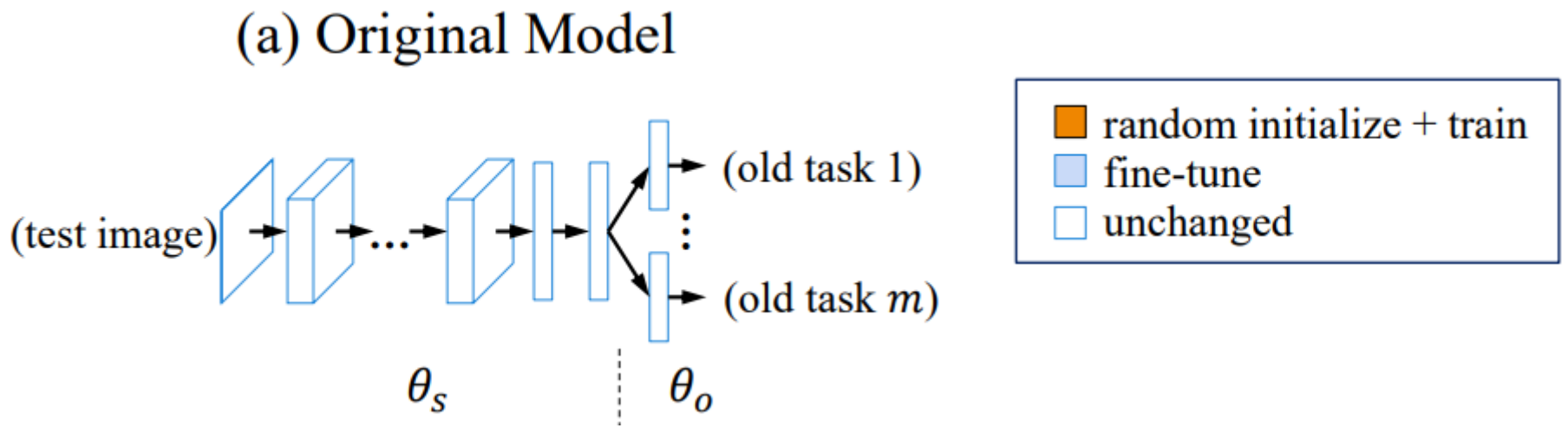
MAS estimates importance weights for all the network parameters in an unsupervised and online manner, allowing adaptation to unlabeled data.

They show how a local variant of MAS is linked to the Hebbian learning scheme.

在當時，超越state-of-the-art

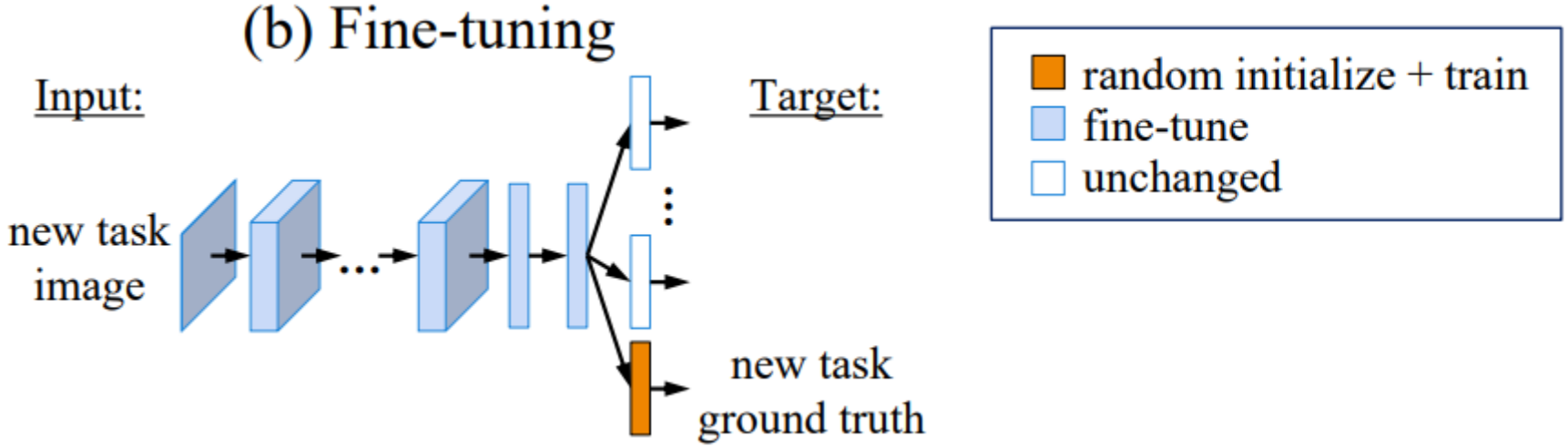
Learning without Forgetting

Original Model



上圖是 AlexNet，θ_s 為前 5 層的convolutional layers + 後2層full connected layers；最後一層是和class相關的output layer，以 θ_o 表示。若加入新的分類任務，則把 new task 的參數先隨機初始化，並令其為 θ_n。

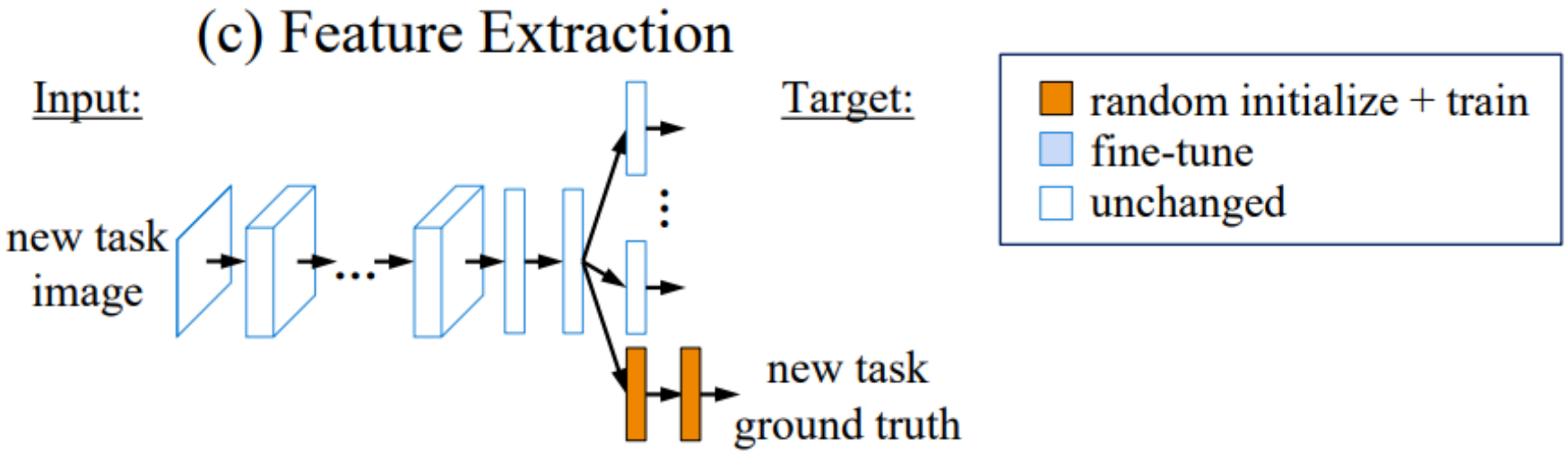
Fine-tunung



θ_s和θ_o都進行優化，其中將θ_o固定。使用low learning rate學習θ_n。另一種可能是把θ_s中的前5層參數固定，避免overfitting，而微調所有full connected layers的參數，本文稱此實驗為**Finetune-FC**。

微調因為沒有舊任務樣本而在舊任務上表現變差。

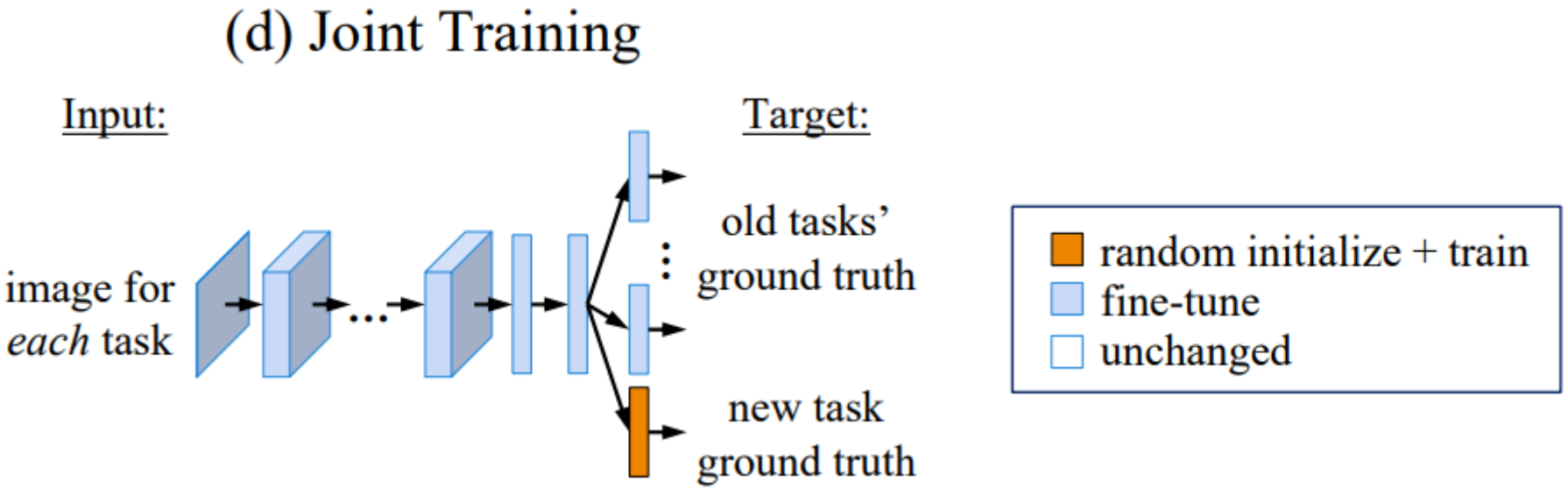
Feature Extraction



θ_s 和 θ_o 都不變，自NN中的一個或多個中間層的輸出被用來訓練新任務的參數 θ_n

Feature extraction通常在新任務上表現不佳，因为共享參數不能表示一些new tasks獨有的特徵表示

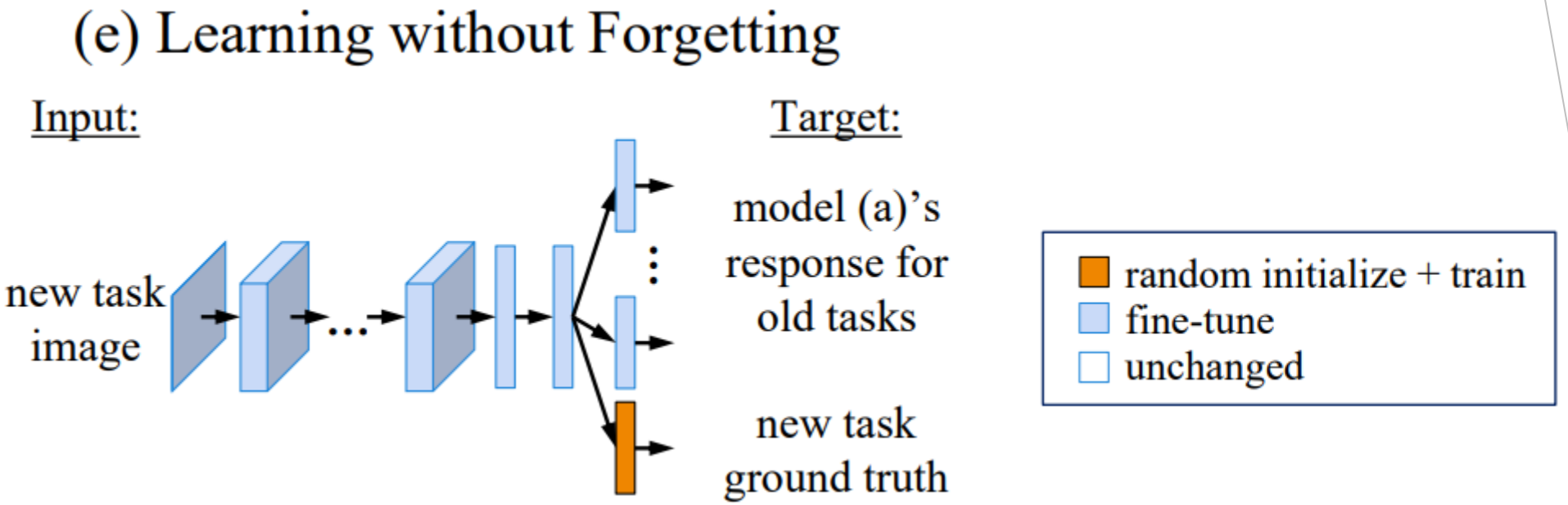
Joint Training



所有參數θ_s、θ_o、θ_n都進行學習優化，通常這個方法產生的結果是最好的，所以一般視為增量學習方法的性能上界(upper bound)

Joint Training的問題為：以前的數據可能因為隱私問題得不到，而且network的capacity也會不斷增加

Learning without Forgetting



使用帶有regularization的 SGD 訓練network

首先固定θ_s、θ_o不變，然後使用new task的dataset訓練θ_n直至收斂(warm-up step)；接著，再聯合訓練所有參數(θ_s、θ_o、θ_n)直到收斂

有點類似Joint Training，但只需要new task的dataset

Loss Function

Loss 1 : 一般針對new task的誤差估算

$$\mathcal{L}_{new}(\mathbf{y}_n, \hat{\mathbf{y}}_n) = -\mathbf{y}_n \cdot \log \hat{\mathbf{y}}_n$$

Loss 2 : 針對old task的**knowledge distillation** loss function (第一個在LLL引入)

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o)$$

$$= -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)}$$

$$y_o'^{(i)} = \frac{(y_o^{(i)})^{1/T}}{\sum_j (y_o^{(j)})^{1/T}}, \quad \hat{y}_o'^{(i)} = \frac{(\hat{y}_o^{(i)})^{1/T}}{\sum_j (\hat{y}_o^{(j)})^{1/T}}$$

Total loss: $$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

Regularization項

LwF Algorithm

LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*

Train:

Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*

$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

使用同一個dataset對new task進行監督式學習、old task進行非監督式學習從而得到可以在new/old task都表現良好的(θ_s、θ_o、θ_n)

Large Scale Incremental Learning

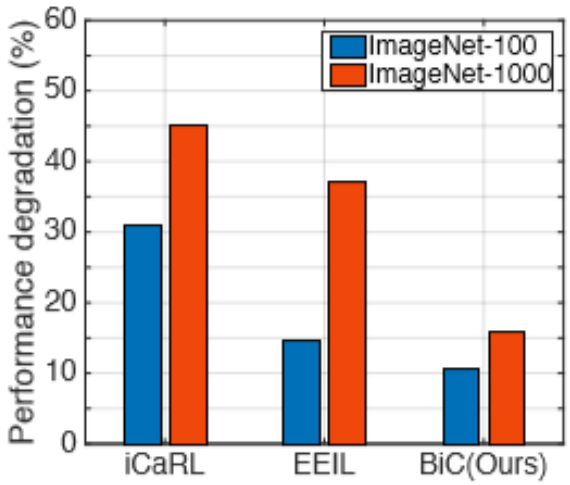
Problem

Incremental learning methods have been proposed to retain the knowledge acquired from the old classes, by using knowledge distilling and keeping a few exemplars from the old classes.

However, these methods struggle to scale up to a large number of classes.

(1) The data imbalance between the old and new classes

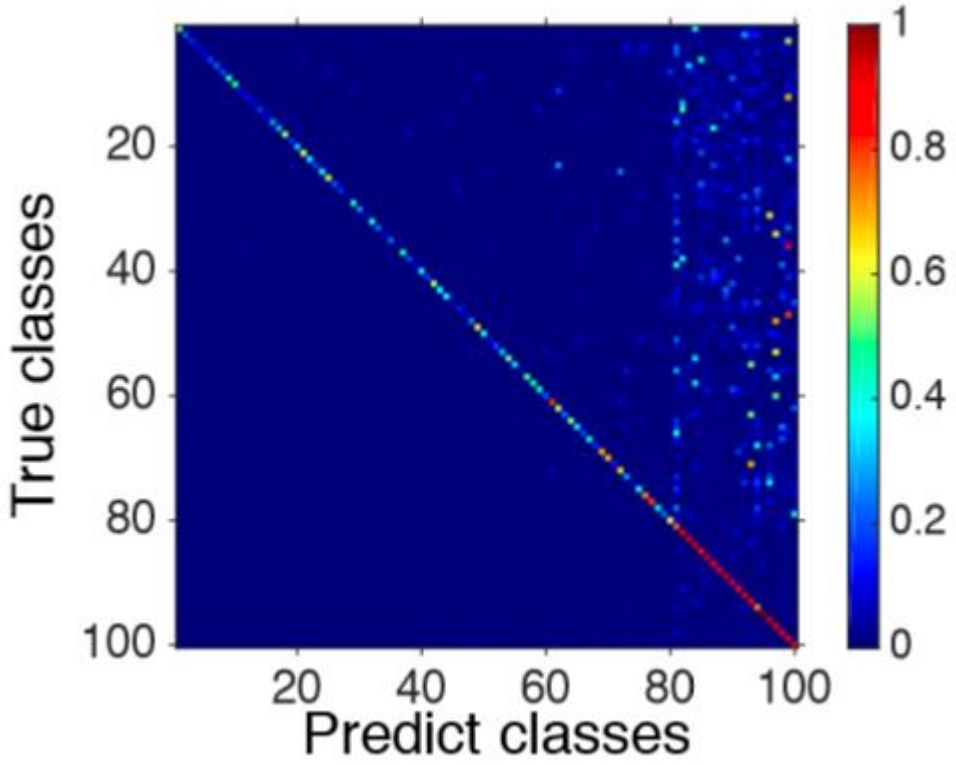
(2) The increasing number of visually similar classes



當classes數目增加時，除了本篇以外的2個incremental learning methods的performance都明顯地下降

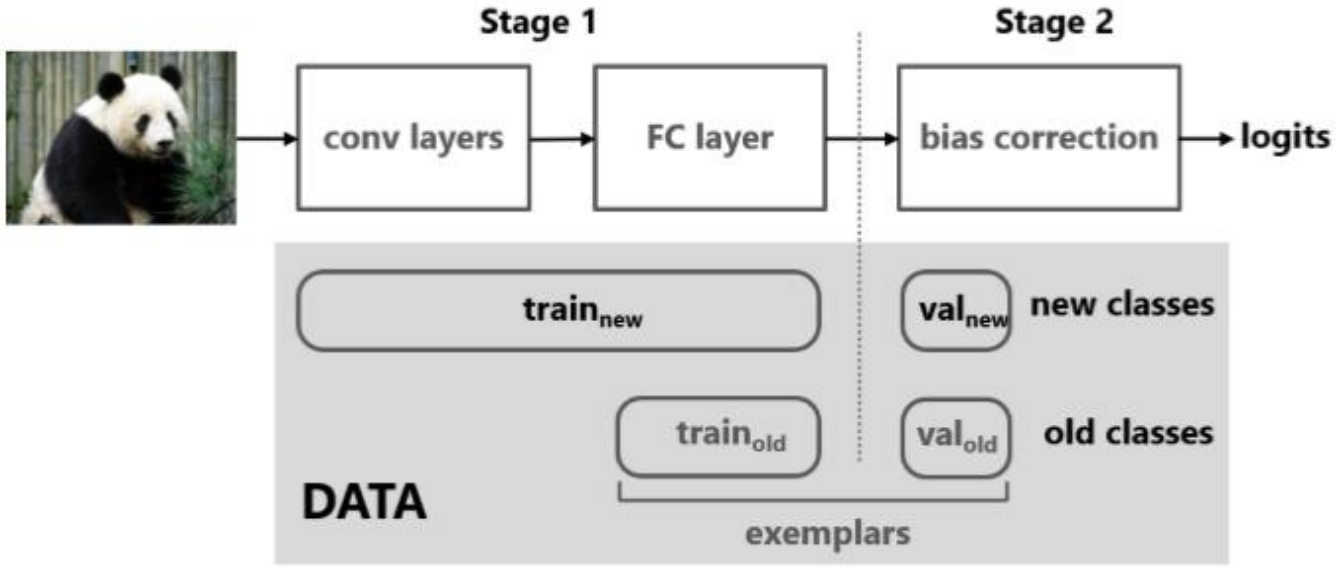
Hypothesis

The last fully connected (FC) layer is biased as the weights that are not shared across classes. (結果特別偏袒new classes)



左圖為由incremental learning methods面對由80個類別增至100個類別時，沒有做FC調整的結果

BiC (bias correction) Method

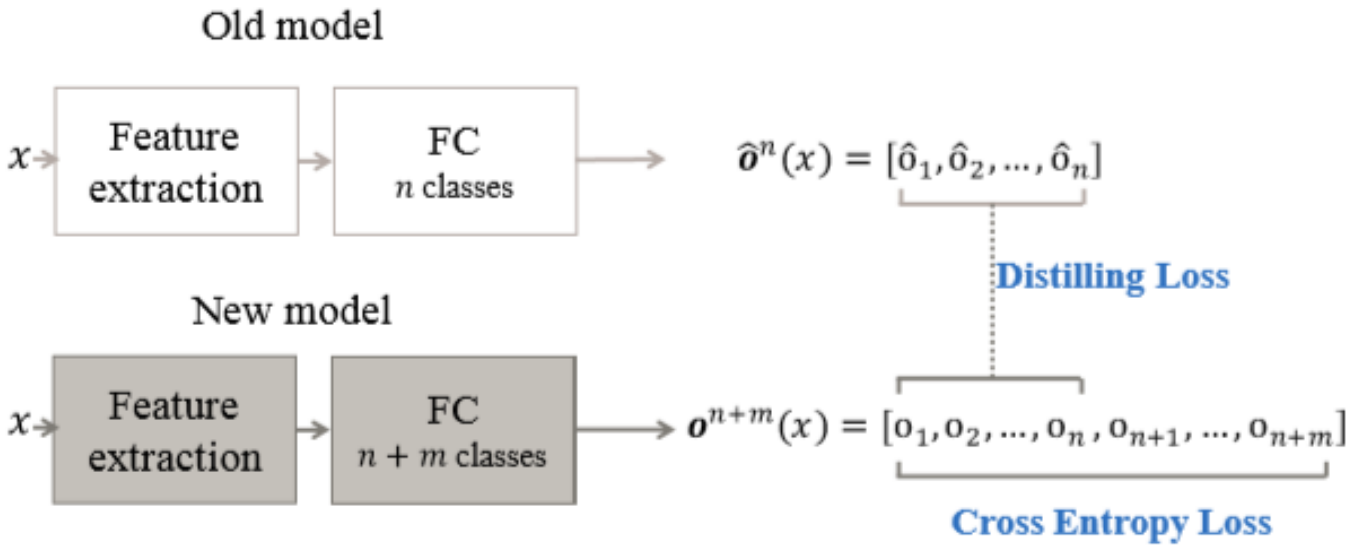


The data, including exemplars from the old classes and samples from the new classes, are split into a training set for the first stage and a validation set for the second stage.

The bias correction layer is learned at the second stage, after learning the convolution layers and FC layer at the first stage.

Note that val_{old} and val_{new} are balanced.

Stage 1



the samples of the new classes : $X^m = \{(x_i, y_i), 1 \leq i \leq M, y_i \in [n+1, .., n+m]\}$

M is the number of new samples, x_i and y_i are the image and the label, respectively.

The selected exemplars from the old n classes : $\hat{X}^n = \{(\hat{x}_j, \hat{y}_j), 1 \leq j \leq N_s, \hat{y}_j \in [1, .., n]\}$, where N_s is the number of selected old images ($N_s/n \ll M/m$).

Stage 1

the output logits of the old classifiers : $\hat{\mathbf{o}}^n(x) = [\hat{o}_1(x), ..., \hat{o}_n(x)]$

the output logits of the new classifiers :

$$\mathbf{o}^{n+m}(x) = [o_1(x), ..., o_n(x), o_{n+1}(x), ..., o_{n+m}(x)]$$

Distilling loss:

$$L_d = \sum_{x \in \hat{X}^n \cup X^m} \sum_{k=1}^{n} -\hat{\pi}_k(x) \log[\pi_k(x)],$$

T: 蒸餾溫度

$$\hat{\pi}_k(x) = \frac{e^{\hat{o}_k(x)/T}}{\sum_{j=1}^{n} e^{\hat{o}_j(x)/T}}, \quad \pi_k(x) = \frac{e^{o_k(x)/T}}{\sum_{j=1}^{n} e^{o_j(x)/T}}$$

Total loss: $L = \lambda L_d + (1 - \lambda) L_c$ Classifiacation error

The scalar λ is set to $\frac{n}{n+m}$, where n and m are the number of old and new classes.

Stage 2

Keep the output logits for the old classes (1, . . . , n) and apply a linear model to correct the bias on the output logits for the new classes (n + 1, . . . , n + m) as follows

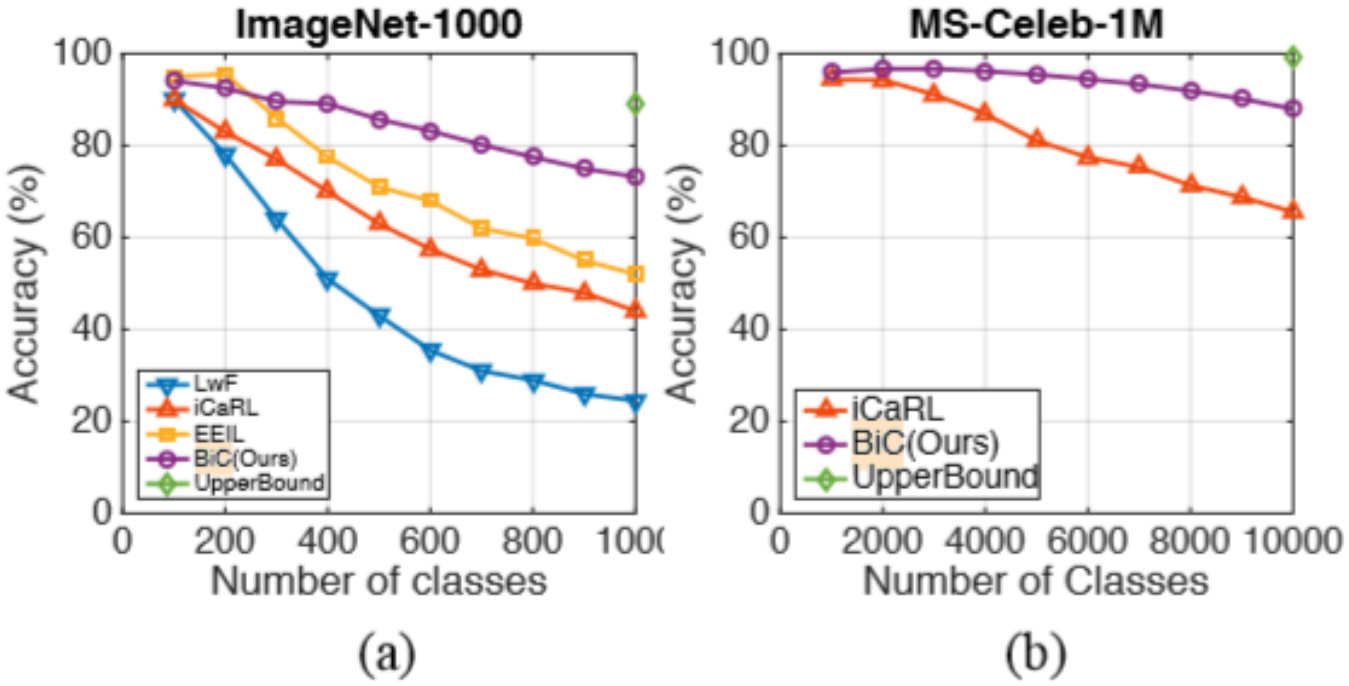
$$q_k = \begin{cases} o_k & 1 \le k \le n \\ \alpha o_k + \beta & n+1 \le k \le n+m \end{cases}$$

where α and β are the bias parameters on the new classes and o_k is the output logits for the k-th class.

When optimizing the bias parameters, the convolution and FC layers are frozen. The classification loss (softmax with cross entropy) is used to optimize the bias parameters as follows

$$L_b = -\sum_{k=1}^{n+m} \delta_{y=k} \log[softmax(q_k)]$$

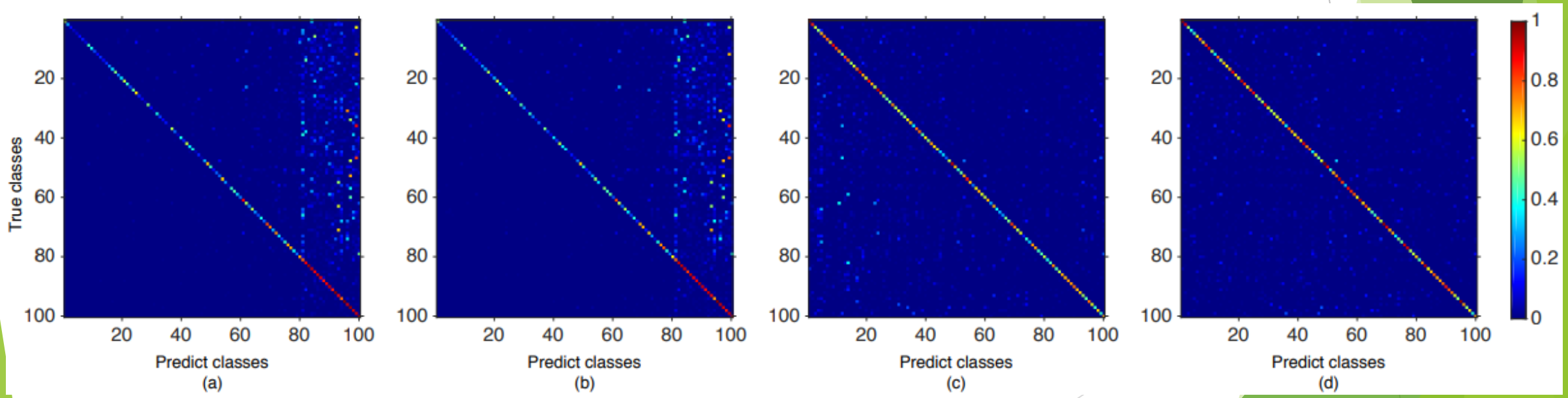
Results



隨著類別數增加，有無進行bias correction的差別便越顯著

Results

Variations	cls loss	distilling loss	bias removal	FC retrain	20	40	60	80	100
baseline-1	✓				84.40	68.30	55.10	48.52	39.83
baseline-2	✓	✓			**85.05**	72.22	59.41	50.43	40.34
BiC(Ours)	✓	✓	✓		84.00	**74.69**	**67.93**	**61.25**	**56.69**
upper bound	✓	✓		✓	84.39	76.15	69.51	64.03	60.93



baseline-1 baseline-2 BiC upper bound

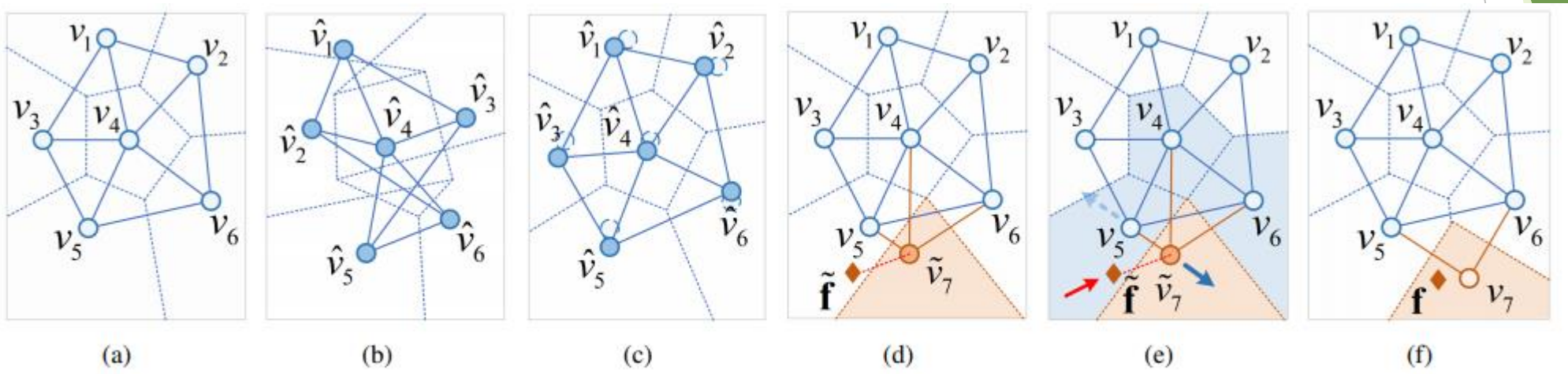
Few-Shot Class-Incremental Learning

此方法也是為了解決new/old class imbalance的問題，而其作用在小樣本上，因此不平衡的情況會很嚴重(傾向於樣本多的資料)

以Neural gas (NG)取代knowledge distillation，認為後者最主要的問題在於new/old task的不平衡，且distilling loss和classification loss會彼此影響

應用拓撲學和生理神經相關的Hebbian learning做為發想(神經元之間的連結若原本的拓撲結構被破壞，則容易遺忘)，提出TOpology-Preserving knowledge InCrementer (TOPIC)架構。

大致概念



Explanation of NG stabilization and adaptation.

每一個node會進行competitive learning，而node的增加代表class的增加。在此一演算法中，要根據data提供的ground truth對model進行調整，且必須維持model的topology structure。

LLL Nowadays & Future(?)

LLL Nowadays & Future(?)

目前LLL相關研究普遍還是以學術為主，商業方面近乎沒有：曾經問過一位Google相關的高層主管，認為當前企業在擁有大量的data下，會傾向於花更多資源將models分別訓練(相較於LLL)並得到效果更佳的model (如同對unlabelled data training的想法)

然而，也許未來會因為對適應大規模信息量的model有所需求而使這項技術被業界所關注等等也不一定(?) (傳統的深度學習演算法大多只關注於小樣本內的非類等工作，對大數據環境缺乏適應力)

現在看似沒有實際應用價值的技術未必真的沒有用，也許只是還沒想到或時機未到!

Future(?)