第十八屆旺宏科學獎

創意說明書

參賽編號：SA18-426

作品名稱：三分搜優化機器學習 – 以點餐系統為例

姓名：吳邦寧

關鍵字：機器學習、訓練優化、點餐預測

摘要

訓練一個神經網路曠日廢時，要如何解決這個問題呢？於是，我發明了一個基於三分搜尋法搭配梯度上升法的優化器，這個優化器的效能十分優秀，效能比 *Adagrad*、*Momentum* 優化器還快，也能夠避免超參數*(hyper parameter)*設定不當造成的損失函數震盪，更能夠保證損失函數永遠會越來越接近完成，具有種種優點。

我也成功推導出讓演算法能夠順利運行的必備條件，「使損失函數權重代入任意直線，其二階導函數恆負」。而 *Linear Regression*、*Logistic Regression* 等等的回歸分析模型都滿足這個條件；特定神經網路使用 *AutoEncoder* 進行逐層貪婪訓練，損失函數也滿足這個條件；或甚至只是為了求得一個函數的局部最小值，只要函數滿足條件，也能使用演算法在極短時間內求得局部最小值，這使得演算法的應用層面極廣。

為了發揮演算法的應用價值，於是把演算法應用於我為學校建置的午餐點餐系統的預測模型上，藉此預測餐點的需求量，以減少廚餘廢料的浪費，發揮該演算法的社會價值。

經過網路查證後，發現沒有人提出過相同的演算法；在常見的機器學習套件中，也沒有見到相同的演算法，因此，演算法的發明為本研究中最為創新的部分，也是本研究探討的主軸。

壹、研究動機

機器學習已在全球成為一股熱潮，而機器學習非常倚賴大量的運算來訓練模型。我認為一定有更好的演算法能夠加快機器學習的訓練，經過推導及試驗，發現可以用三分搜尋法加快訓練過程。

正好，我在學校建立了一個供全校師生點餐的午餐系統，為了減少過度備料造成浪費，我發展一套午餐需求預測模型供廠商參考。但因初建置時，常需進行大量的訓練，耗費相當多的時間，所以我發展出這套三分搜優化演算法，來大幅降低訓練時間。若能將優化演算法應用於預測餐點的需求量，那就能夠讓廠商準備適當份數的餐點，不致浪費，符合經濟、環保、社會正義的永續理念，發揮演算法的社會價值。

貳、研究目的

由於點餐系統與預測點餐已經有大量文獻在研究了，因此本研究將著重於使用三分搜尋法優化機器學習。

一、製作三分搜優化的演算法，此為本研究的第一目標。

二、應用三分搜優化演算法於點餐系統的預測模型上，此為本研究的第二目標。

參、文獻回顧

一、關於三分搜優化機器學習的文獻

優化機器學習大多以數學方法進行優化，例如：

(一)、梯度上升法

(二)、*Momentum* 法

(三)、*Adagrad* 法【*Duchi*，2011】

(四)、*Adam* 法【*Kingma*，2015】

卻沒有人使用三分搜尋法搭配梯度上升法來優化機器學習，因此，本文將著重於探討三分搜尋法搭配梯度上升法優化機器學習。

二、關於機器學習與點餐系統的文獻

點餐系統的相關文獻簇繁不及載，在此不予贅述；機器學習的相關文獻也相當繁多；預測點餐的相關研究亦有人做過，例如：

(一) 點餐系統：互動式電子點餐系統【許誠，2016】

(二) 點餐預測：資料探勘應用於點餐系統之研究-以員工餐廳為例【李淑玲，2010】

(三) 機器學習：利用機器學習作法之中文意見分析【李孟潔，2009】

但是，卻沒有人將三者混雜起來，以機器學習的方式來進行預測點餐並搭載到點餐系統上，因此，本研究將探討以機器學習的方式進行預測點餐。

肆、研究器材與名詞解釋

<table>
<tr><th></th><th>名詞</th><th>解釋</th></tr>
<tr><td rowspan="8">名詞解釋</td><td>偏微分</td><td>對函數 f 的 x 偏微分 $\frac{\partial}{\partial x}f(x,y)$</td></tr>
<tr><td>梯度 ∇</td><td>$\nabla f(x_1,\dots,x_n)=\begin{bmatrix}\frac{\partial}{\partial x_1}f\\ \frac{\partial}{\partial x_2}f\\ \vdots\\ \frac{\partial}{\partial x_n}f\end{bmatrix}$</td></tr>
<tr><td>梯度上升法</td><td>又名爬山法，Tensorflow 中的實作為
Gradient Descent Optimizer</td></tr>
<tr><td>Logistic Regression</td><td>邏輯特回歸，等同於一個以 Sigmoid 為激勵函數的神經元</td></tr>
<tr><td>基礎神經元</td><td>有多個輸入、單一輸出的神經結構</td></tr>
<tr><td>本演算法</td><td>搭配三分搜尋法的梯度上升演算法</td></tr>
<tr><td>逐層貪婪訓練</td><td>先訓練第一個隱藏層，再訓練第二個隱藏層…最後以這些訓練好的參數為初始值，對整個網路進行訓練。</td></tr>
<tr><td>AutoEncoder</td><td>以「輸出的神經元數量與輸入一樣」為核心概念的訓練用演算法</td></tr>
</table>

附：梯度上升法與梯度下降法是相同的演算法，唯一差別只有損失函數的正負號

<table>
<tr><th></th><th>工具</th><th>用途</th></tr>
<tr><td rowspan="3">數學工具</td><td>Desmos</td><td>繪製 2D 的幾何圖形</td></tr>
<tr><td>GeoGebra</td><td>繪製 3D 或是 2D 的幾何圖形</td></tr>
<tr><td>Symbolab</td><td>代為計算繁瑣的微分方程</td></tr>
</table>

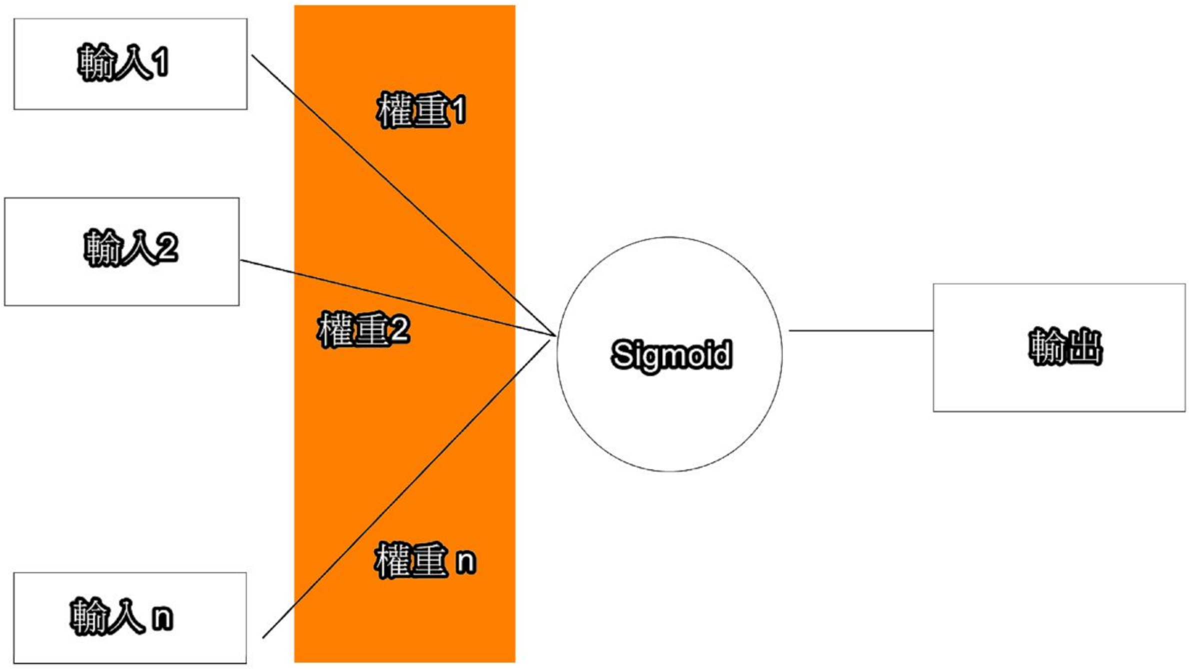
	器材	用途
開發點餐系統所需器材	*Google Firebase*	*App* 分析用的第三方插件
	Google Analytics	網頁分析用的第三方插件
	Ubuntu 18.04 LTS	伺服器作業系統
	Nginx	網頁伺服器
	No-IP Dynamic DNS	伺服器網域
	Mac	*iOS App* 的開發環境
	iPhone	*iOS App* 的測試環境
	Xcode	*iOS App* 的開發工具
	Android 手機	*Android App* 的測試環境
	Android Studio	*Android App* 的開發工具
	Chrome	網頁前端測試環境
	Visual Studio	開發廠商前端的開發工具
	Visual Studio Code	開發後端的開發工具
	HTML + CSS + Bootstrap	網頁前端開發框架
	JavaScript + jQuery + Ajax	網頁前端開發語言
	PHP	後端開發語言
	MySQL	資料庫
	Kotlin	*Android App* 開發語言
	Swift	*iOS App* 開發語言
	Python	數據分析測試語言
	C#	廠商前端開發語言
	Linux terminal bash + Crontab	資料庫自動備份

伍、研究方法

一、三分搜尋法優化之推導

設計一個沒有隱藏層，只有一個輸出，而且激勵函數*(Activation Funciton)*為*Sigmoid* 的神經網路模型，我們稱呼他為基礎神經元。

該模型可以用下圖表達：



橘色的部分叫做權重，如何找出一個適合的權重使得基礎神經元有良好的預測能力，是本研究探討的重點。

模型也可以用下列數學式表達：

$$F(X) = Sigmoid(X \cdot w + b) = \frac{1}{1 + e^{-(X \cdot w + b)}}$$

下表為數學式中的名詞解釋。

名詞	意思	資料型態
F	代表模型的數學函數	函數
w	模型的權重	向量
b	模型的偏移量	實數
X	模型的輸入	向量
$F(X)$	模型的輸出	實數

採用交叉熵(*Cross Entropy*)作為損失函數，於是得到下式：

$$Cost(X,Y,w,b)=\sum_i Y_i\, ln\left(\frac{1}{1+e^{-(X\cdot w+b)}}\right)+(1-Y_i)\, ln\left(1-\frac{1}{1+e^{-(X\cdot w+b)}}\right)$$

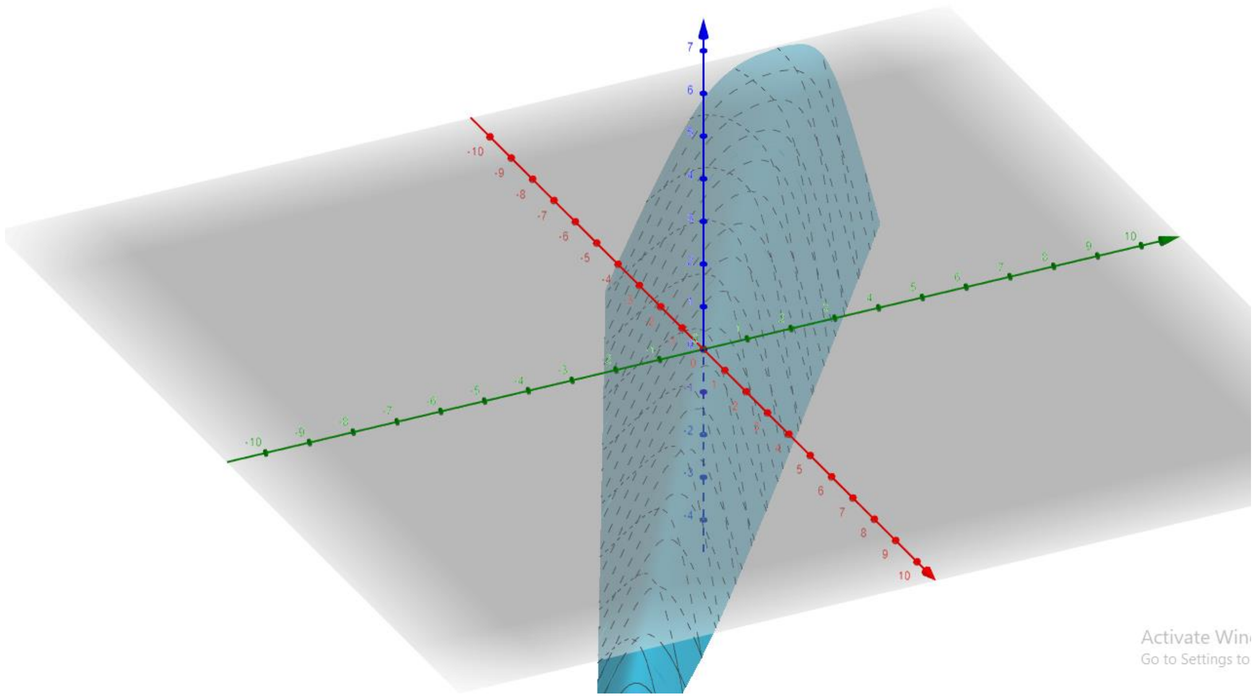
若 X_{n+1} 永遠設為 1，則 $w_{n+1}=b$ ，因此，可化簡為下式：

$$Cost(X,Y,w)=\sum_i Y_i\, ln\left(\frac{1}{1+e^{-X\cdot w}}\right)+(1-Y_i)\, ln\left(1-\frac{1}{1+e^{-X\cdot w}}\right)$$

下表為數學式中的名詞解釋。

名詞	意思	資料型態
$Cost$	模型的損失函數	函數
$Cost(X,Y,w)$	模型的損失函數值	實數
X	模型的訓練用輸入資料集合	矩陣
X_i	模型的訓練用輸入資料	向量
Y	模型的訓練輸出資料集合	向量
Y_i	模型的訓練輸出資料	$Y_i\ \in\ \{0,1\}$
w	模型的權重	向量
b	偏移量，可被併入 w 中	實數

若 $X=[[1,2],[3,4]]$ 且 $Y=[1,0]$ ，則損失函數圖形如下：



下表為圖形與損失函數的關係。

圖形中	損失函數中
X 軸、紅軸	w_1
Y 軸、綠軸	w_2
Z 軸、藍軸	$Cost(X,Y,w)$

將 XY 平面上的任意直線代入圖形，如下式：

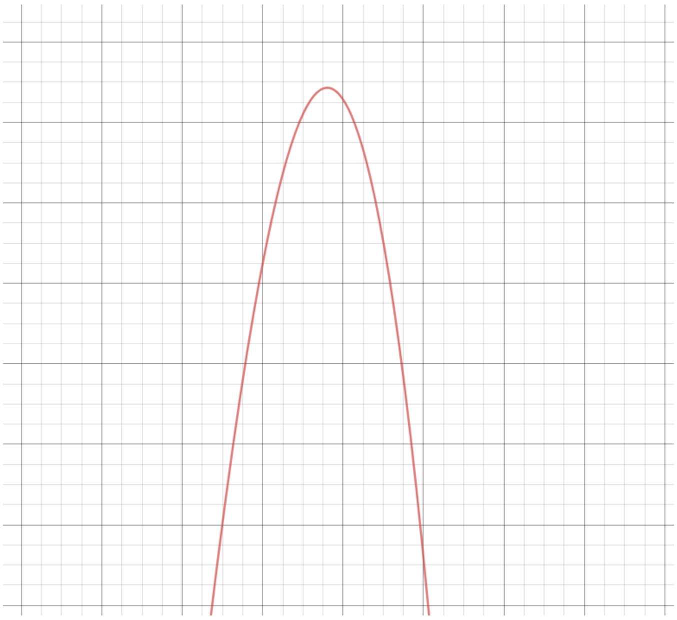
$$Cost(X,Y,w)=\sum_i Y_i\, ln\left(\frac{1}{1+e^{-X\cdot w}}\right)+(1-Y_i)\, ln\left(1-\frac{1}{1+e^{-X\cdot w}}\right)$$

$$Line=\begin{cases} w_1=a_1t+b_1 \\ w_2=a_2t+b_2 \end{cases}, t\in R$$

$$A=\sum_i X_i(a_it+b_i)=X\cdot w$$

$$Cost(X,Y,t)=\sum_i Y_i\, ln\left(\frac{1}{1+e^{-A}}\right)+(1-Y_i)\, ln\left(1-\frac{1}{1+e^{-A}}\right)$$

繪製圖形 $Cost(X,Y,t)$ ，可得到下圖。



下表為繪製出來的圖形與數學式的關係。

圖形中	數學式
X 軸	t
Y 軸	$Cost(X,Y,t)$
a_1,a_2 之值	5,6
b_1,b_2 之值	7,8

$$g(t)=\frac{\partial^2}{\partial^2 t}Cost(X,Y,t)=\sum_i -\frac{X_{ii}^2e^{-X_i\cdot w}}{(e^{-X_i\cdot w}+1)^2}$$

計算 $Cost(X,Y,t)$ 對 t 的二階導函數 $g(t)$，發現 $g(t)$ 恆負，因此，我們得到一個重要性質，如下。

基礎神經元的損失函數權重代入任意直線，其二階導函數恆負。

性質 1

二、三分搜尋法優化之演算法

對於任意函數 $f(x)$，若其二階導函數在區間 $[L, R]$ 恆負或是恆正，則可以使用三分搜尋法求得 $f(x)$ 在區間 $[L, R]$ 內的極值，下面為三分搜尋法的虛擬碼：

Algorithm: Ternary Search

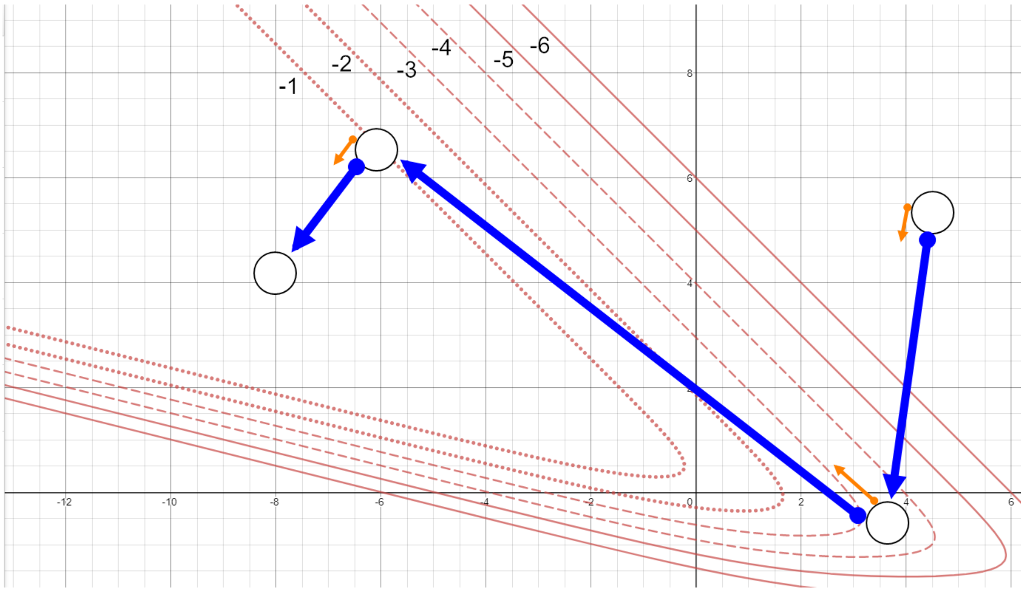
```
float l = -10 ,r = 10 ,ll ,rr ,precision = 1e-3;
while(abs(l - r) > precision)
    ll = (l + l + r) / 3 ,rr = (l + r + r) / 3;
    if(f(ll) > f(rr)) l = ll;
    if(f(ll) < f(rr)) r = rr;
    if(f(ll) == f(rr)) l = ll ,r = rr;
```

由於 $g(t)$ 恆負，則可以對 $Cost(X, Y, t)$ 進行三分搜，即可獲得在區間 $[L, R]$ 使得 $Cost(X, Y, t)$ 最大化的 t 。將這種方法搭配梯度上升演算法，即為本研究主要在探討的演算法，下面為本演算法的虛擬碼：

Algorithm: Ternary + Gradient

```
int count = 10 ,ternary = 20;
vector<float> w = 0, grad = gradient(w);
float l ,r ,lmid ,rmid ,alpha = 8;
while(count--) {
    for(int i = 0 ,l = 0 ,r = alpha;i != ternary;i++) {
        lmid = length(gradient((l + l + r) / 3));
        rmid = length(gradient((l + r + r) / 3));
        if(lmid == rmid) break;   //reaches the maximum precision
        else if(lmid < rmid) l = (l + l + r) / 3;
        else if(lmid > rmid) r = (l + r + r) / 3;
    }
    tmp = grad * (l + r) / 2; w = w + tmp;
}
```

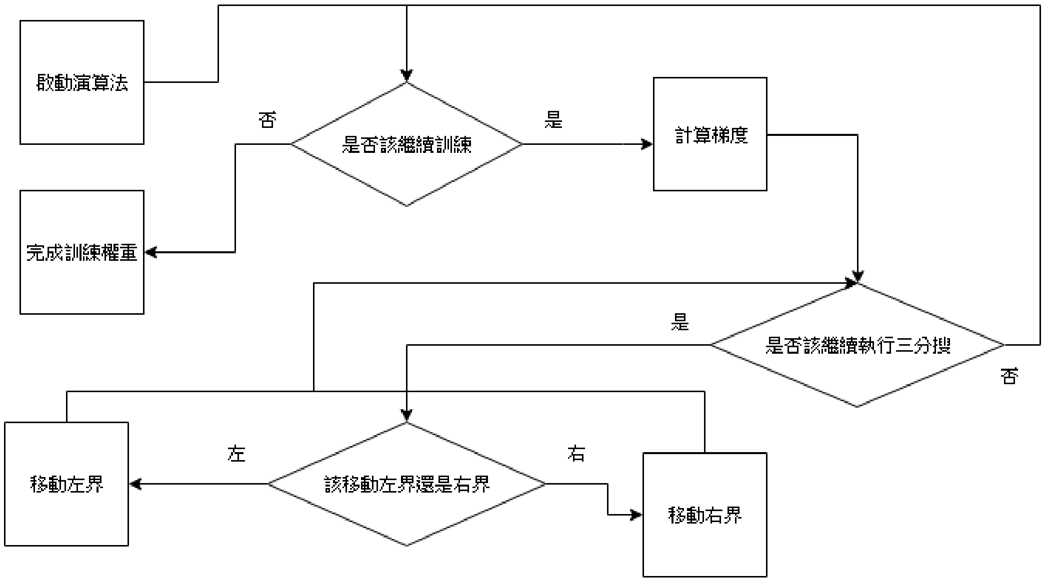
演算法相當於「根據梯度做直線，使用三分搜尋法求得在直線上使得損失函數最大的點，並將所在位置轉移到該點」，下圖為演算法的運行時期的樣子。



下表為符號的解釋。

符號	意思
橘色箭頭	∇w
白色圓形	圖形的座標代表 w
藍色箭頭	移動軌跡
紅色線條	等高線，旁有數字標註高度

下圖展現本演算法的流程圖。



由此可見，搭配三分搜尋法後的梯度上升法與普通的梯度上升法會有顯著的效能優化，普通的梯度上升法如同登山客一步一步朝向山頂前進，而搭配三分搜尋法的梯度上升法如同登山客看到一個最高點之後直接瞬間移動過去。

三、與其他演算法的比較

有多種可以訓練神經網路的演算法，如：

(一)、*Momentum* 法，其權重迭代公式如下：

$$w \coloneqq w + \alpha\nabla Cost(w) + \sum_{i=1}^{n} \alpha\nabla Cost(w_i)\beta^{n-i+1}$$

(二)、*Adagrad* 法，其權重迭代公式如下：

$$w \coloneqq w + \frac{\alpha\nabla Cost(w)}{\sqrt{\varepsilon + \sum_{i=1}^{n}(\nabla Cost(w_i))^2}}$$

(三)、三分搜尋法，其權重迭代公式如下：

$$w \coloneqq Ternary(w - \alpha\nabla Cost(w), w + \alpha\nabla Cost(w))$$

下表為公式中的名詞解釋。

名詞	意思
w	當前權重
w_i	第 i 次更新前的權重
n	權重共更新了幾次
α	迭代步長參數
β	動量衰退參數
ε	平滑值，避免分母為零，通常取 $1e-8$
$Ternary$	三分搜尋法函數 第一個參數為直線的左界 第二個參數為直線的右界 輸出為「在直線上損失函數發生最大值的位置」
$\nabla Cost(w)$	$Cost$ 對 w 的梯度

附：預設進行三分搜尋法 20 次，演算法中的虛擬碼曾提及

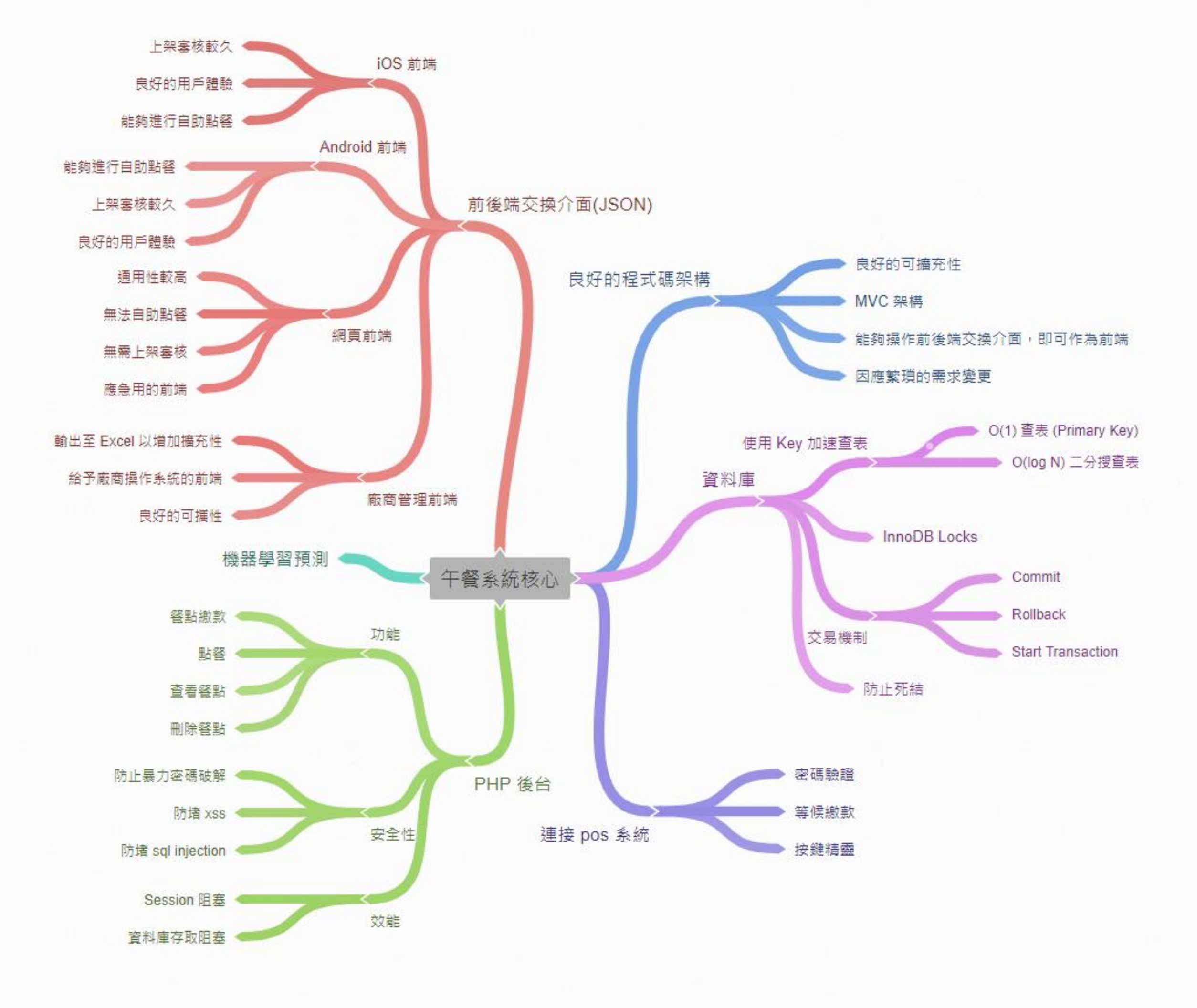
由於本演算法並非典型的演算法，難以使用 *Big-O* 函數進行複雜度分析，下面列出比較演算法效能的方式。

	優點	缺點
根據所耗時間	方便，容易計算	多執行緒系統上易失準
根據 *global epoch*	多執行緒系統上不失準	三分搜尋法運行一個 epoch 所需的時間為其他演算法的數十倍
根據計算梯度次數	多執行緒系統上不失準	在本實驗中並無缺點

計算梯度是整個演算法中最耗時的操作，根據上表的比較，採用計算梯度的次數作為計時單位。

四、應用於點餐系統的預測模型

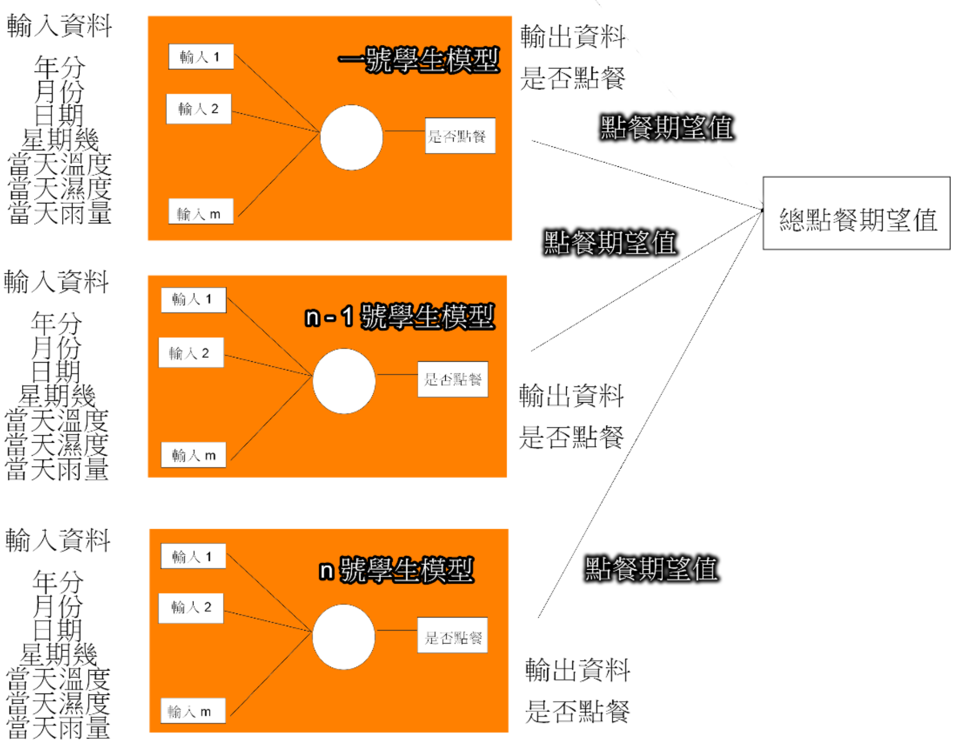
我們打造了一個專門拿來在學校訂購午餐的系統，稱作午餐系統，下圖為午餐系統的架構圖。



午餐系統的後台是由 *PHP* 做出來的，資料庫是 *MySQL*，後台只會輸出 *json* 資料，使得各種前端能夠與後台連接；午餐系統也經過安全性測試、壓力測試，系統核心於 2018/04 開發完成，系統於 2018/09/18 正式於學校上線運作。

每一個人點餐的資料全部都會被系統紀錄下來，日積月累下來，資料量非常可觀，將資料交給預測模型預測後，希望能夠藉此減少學校的廚餘廢料。

下圖為預測模型的架構圖，橘色方塊為一個基礎神經元，根據每個人的點餐資料，訓練完基礎神經元後預測點餐期望值，點餐期望值加總即為模型輸出。



根據先前發現的性質 1，「基礎神經元的損失函數權重代入任意直線，其二階導函數恆負」，採用本研究探討的主要演算法「搭配三分搜優化的梯度上升法」來進行訓練。

如此淺層的神經網路難以過度擬合，而為了驗證模型是否準確，採用卡方檢定判斷該模型的可靠程度，首先要定義虛無假說以及對立假說。

虛無假說 H_0	本模型有效，測試樣本與與模型預測吻合
對立假說 H_1	本模型無效，測試樣本與模型預測不吻合。

根據下列公式計算卡方統計值。

$$\chi^2 = \sum_{i=1}^{n} \frac{(S_i - T_i)^2}{T_i}$$

顯著水準 $p = 0.01$，自由度為 $df = n - 1$，計算卡方臨界值後與卡方統計值後比較，若卡方統計值大於卡方臨界值則拒絕虛無假說，也代表模型失準。

下表為名詞解釋。

名詞	含意
T_i	第 i 天總點餐期望值
S_i	第 i 天總點餐量
χ^2	卡方統計值
n	測試天數

陸、現階段研究成果

一、使得三分搜優化可行的數學性質

要使得三分搜優化可行，就必須要滿足「任意直線代入損失函數權重，滿足其二階導函數恆正或是恆負」，如下：

$$L = \begin{cases} w_1 = a_1 t + b_1 \\ \quad\quad \dots \\ w_n = a_n t + b_n \end{cases}, t \in R$$

$$Cost(X,Y,w) = Cost(X,Y,[a_1 t + b_1, \dots, a_n t + b_n]) = Cost(X,Y,t)$$

若下列二式擇一恆成立，則可以使用本演算法訓練。	
$\frac{\partial^2}{\partial^2 t} Cost(X,Y,t) < 0$	$\frac{\partial^2}{\partial^2 t} Cost(X,Y,t) > 0$

凡是滿足這個數學性質的模型，皆可以用三分搜優化以加快效能。

二、可被三分搜優化的模型

有不少模型滿足這個數學性質，例如：

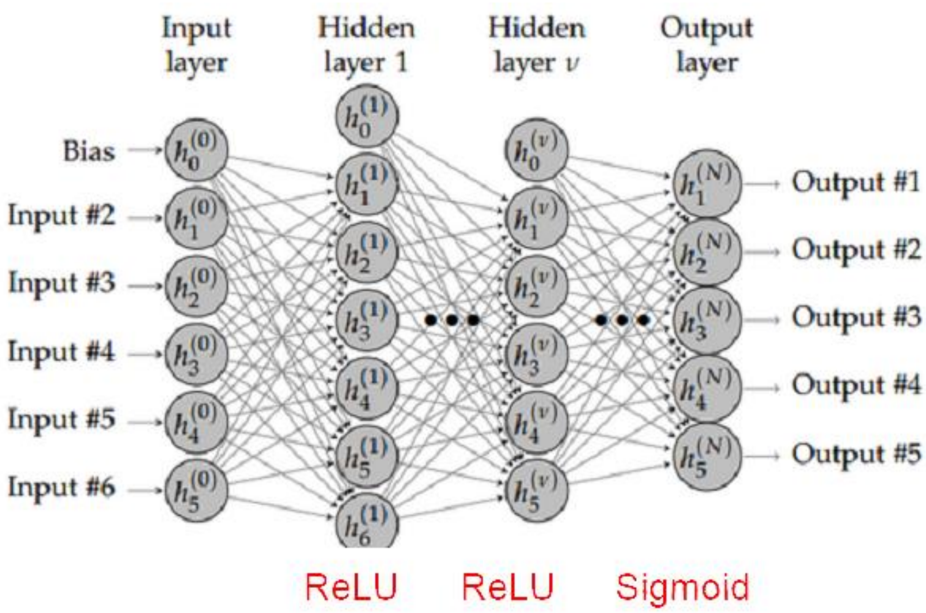
(一)、*Logistic Regression* 的損失函數。

(二)、*Linear Regression* 的損失函數。

(三)、使用 *AutoEncoder* 對 *ReLU* 網路逐層貪婪訓練的損失函數。

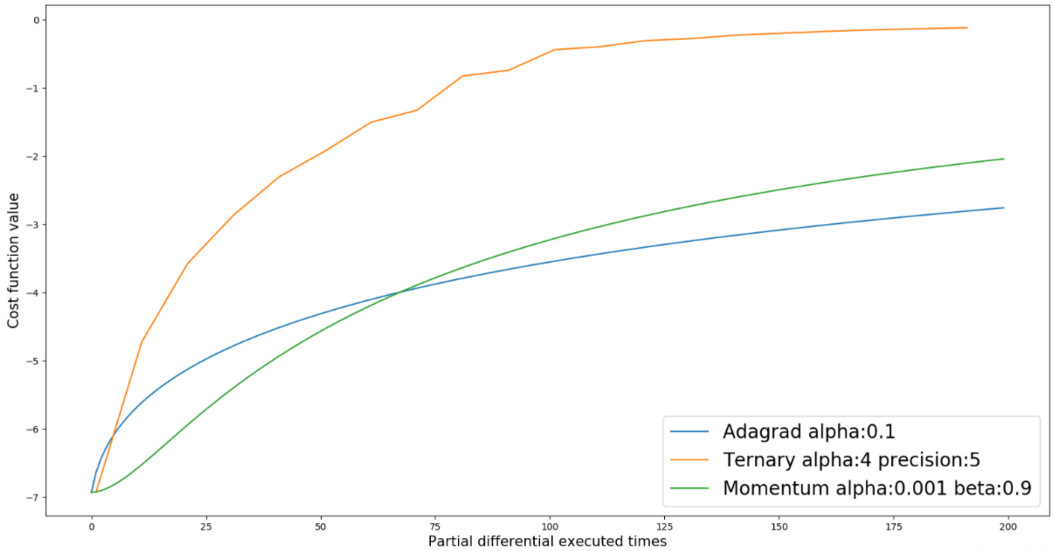
這些數學模型的損失函數都滿足數學條件，因此可以將本演算法應用於這些模型上。

ReLU 網路的架構圖如下，隱藏層與輸入層皆為 *ReLU*，輸出層為 *Sigmoid*。

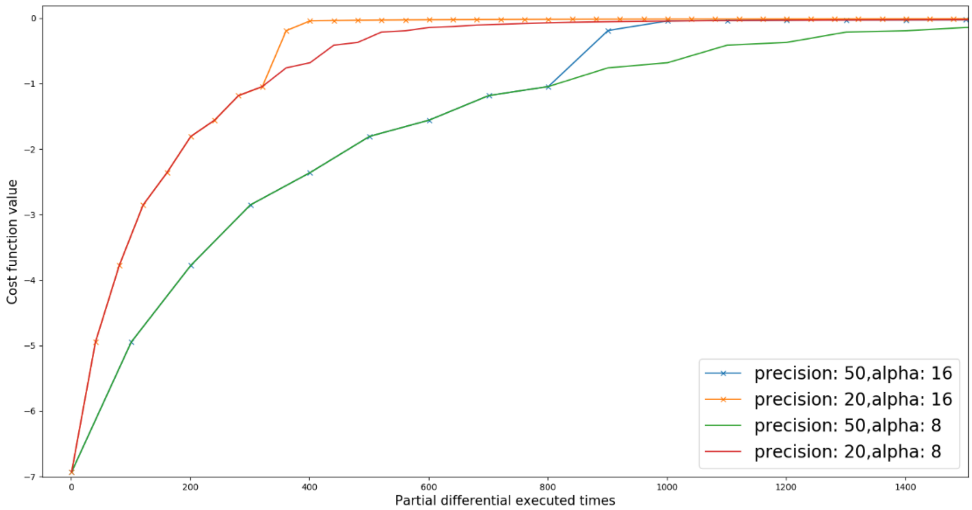


三、三分搜優化的特性

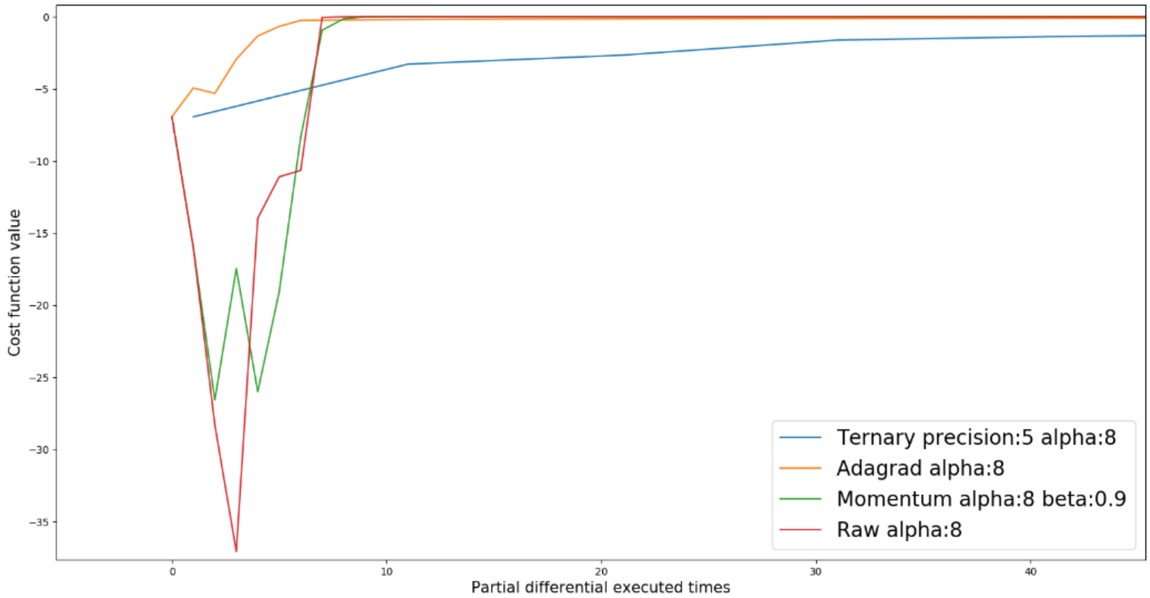
下面三張圖中展現了本演算法的特性，其中橫軸為計算梯度的次數，縱軸為損失函數值，損失函數越大越接近訓練完成。



上圖比較了三種演算法的效能優劣，我們可以看到 *Adagrad* 一開始有優秀的效能，超越了其他兩種演算法，不過很快的 *Adagrad* 就被本演算法追過了，再來也被動量法追過了，本演算法在這次實驗中有最優秀的效能。



上圖中，每一條線都是以本演算法作為訓練演算法，用意是比較各種超參數對效能的影響。如果將 α 設定大一點能夠跳出局部最大值，但是也有可能造成浮點數精度問題，而且 α 對於演算法效能幾乎不影響；反而是運行三分搜的次數與效能較為相關，這是因為過高的三分搜精度對於增加損失函數沒有太大的幫助。



上圖模擬了設置不良的超參數，可以見到各種每一種演算法都有走倒退路，除了本演算法沒有走倒退路，這是本演算法的特點之一。

下表總結了三分搜優化與其他演算法之間的優劣。

	優勢	劣勢
Adagrad	訓練初期效能優秀	訓練後期影響極小
		容易受到超參數的影響
Raw Gradient Ascent	實作簡單	容易震盪
		訓練速度慢
Momentum	能夠跳脫局部最小值	容易受到超參數的影響
	能夠減緩震盪	
三分搜優化	不易受到超參數的影響	必須滿足數學性質
	運行效能十分優秀	
	損失函數只會越來越接近完成	
	不會損失函數發生震盪	

由此表可知，若是能夠滿足數學性質，都應該使用本演算法，下表列舉為何應該使用本演算法。

(一)、不易受到超參數的影響，這是因為每次執行三分搜尋法，會使得 α 以指數速率縮小，這大幅減少了調整超參數的麻煩。

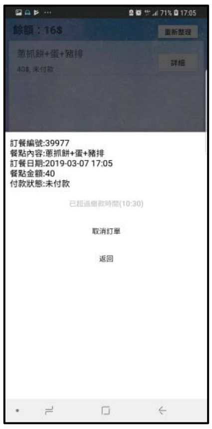
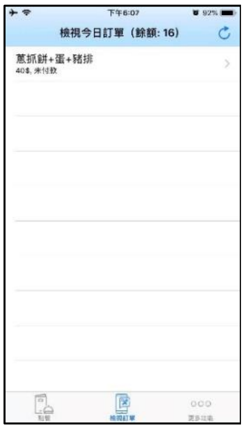
(二)、運行效能十分優秀，這是因為三分搜尋法能夠跳過大量不必要的梯度計算，使得三分搜尋法能飛快的完成訓練。

(三)、損失函數只會越來越接近完成，這是因為三分搜尋法的算法特性。

(四)、不會發生損失函數震盪，發生震盪多數是因為超參數設置不佳，但是本演算法對於超參數幾乎不敏感。

四、點餐系統與預測模型

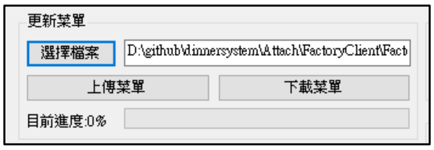
下表展示了點餐系統的使用簡圖，我們有三種前端供使用者操作，網頁前端的功能較少，但是網頁前端的通用性較好，只要有瀏覽器的作業系統都可以使用。

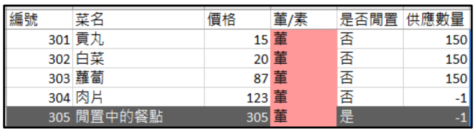
	登入	點餐	查看點單	繳款
安卓前端				
蘋果前端				
網頁前端				

下表為實際使用廠商管理插件的截圖，以下簡介三種最常被用到的功能。

廠商管理插件　　匯出表格

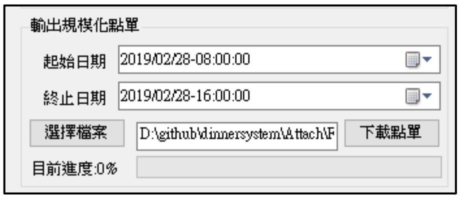
更新菜單

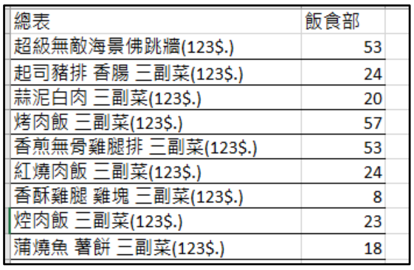




編號	菜名	價格	葷/素	是否閒置	供應數量
301	貢丸	15	葷	否	150
302	白菜	20	葷	否	150
303	蘿蔔	87	葷	否	150
304	肉片	123	葷	否	-1
305	閒置中的餐點	305	葷	是	-1

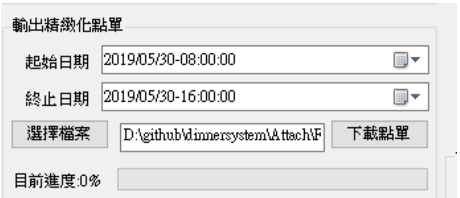
查看點單





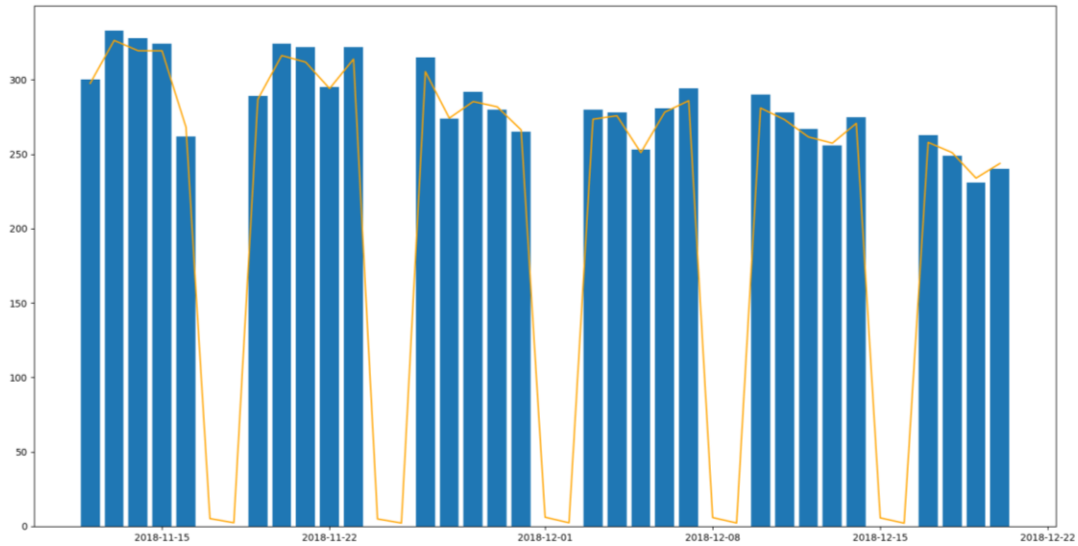
總表	飯食部
超級無敵海景佛跳牆(123$.)	53
起司豬排 香腸 三副菜(123$.)	24
蒜泥白肉 三副菜(123$.)	20
烤肉飯 三副菜(123$.)	57
香煎無骨雞腿排 三副菜(123$.)	53
紅燒肉飯 三副菜(123$.)	24
香酥雞腿 雞塊 三副菜(123$.)	8
焢肉飯 三副菜(123$.)	23
蒲燒魚 薯餅 三副菜(123$.)	18

查看自助點單



點單號	點餐者	時間	總價	馬鈴薯	寬麵	雞絲麵	貢丸
12345	小明	11:00	87	1			1

使用卡方檢定進行篩選，篩出具有預測能力的模型，下表展示該模型的預測能力，長條為實際點餐人數，折線為模型預測人數，橫軸為時間，縱軸為點餐人數。



經過計算後得知，誤差值不超過 5%，意即模型說今天會需要製作 300 份餐點，實際上大概需要準備 285 份到 315 份，這份資料能夠給廠商進行參考，發揮演算法的社會價值。

柒、結論

一、本研究的貢獻

本研究作了以下的貢獻：

(一)、發明出一個新的訓練演算法，可縮短訓練時間。

(二)、給出一個條件，凡滿足這個條件的損失函數都能使用本演算法。

(三)、將演算法應用在現實生活上。

二、本研究的特點

一個好的演算法，如二分搜尋法，應該要能夠廣泛的運用在各種層面，使得二分搜尋法能夠順利運行的條件是搜尋目標具有單調性，而二分搜尋法已被應用在各種層面上，例如：資料庫的搜尋、二分搜尋樹、B-Tree 等等…本研究主要探討的演算法也有類似的潛能，要先滿足特定條件才能運作，而且能夠在應用在各種問題上，這是本演算法的潛能。

一個理論，在沒有付諸實作之前，都只是一堆想法、一堆紙張，俗話說「紙上談兵」，本研究將演算法應用於點餐預測上，減少廚餘，造福社會，這是本演算法的社會價值。

三、結論

雖製作點餐系統並非本研究的主旨，但是我們也耗費了相當的精力在點餐系統上，能夠在點餐系統上搭載模型是原先的初衷，不過意外發現了這個演算法，於是將演算法搭載到模型上，點餐系統有如虎添翼，文中對點餐系統著墨較淺，有興趣的話在網路上 Google 午餐系統就找的到了。

本演算法有各種潛能可應用在其他層面上，或是搭配其他演算法都是可行的；本演算法也有相當的社會價值，搭載在點餐系統上對於減少廚餘有顯著的成效。

捌、參考資料

逐層貪婪訓練：https://blog.csdn.net/JNingWei/article/details/78836823

各種訓練演算法詳解：https://bit.ly/2JLfFCP

深度學習教學： https://bit.ly/2Wwrd2E

Python 與各種數學套件：https://www.books.com.tw/products/0010778000

三分搜尋法：https://www.geeksforgeeks.org/ternary-search/

午餐系統：https://dinnersystem.com；測試用帳號：dinnersystem；測試用密碼：2rjurrru

午餐系統核心原始碼：https://github.com/lawrence910426/dinnersystem

午餐系統 Android 原始碼：https://github.com/seanpai96/dinnersys-android

午餐系統 iOS 原始碼：https://github.com/seanpai96/dinnersys-ios

午餐系統分析器原始碼（由於是研究用途，程式碼稍嫌雜亂）：https://github.com/lawrence910426/dinnersys_analysis

神經網路架構圖來源：https://itw01.com/M9QEJFI.html

AutoEncoder：https://bit.ly/2KipAzc