題目： LSTM 提高表現的方法比較

{1. **LSTM 中的 Forget gate 若能在初始化時，給與較大的 bias，使其長保開啟（也就是儘量不要遺忘），長短期類神經網路會得到較好的訓練效果。**

2. **在 train LSTM 模型的時候，我們可能會用多種不同的 optimizer，其中像是 RMSProp、Adam 更是常被使用。作業一的時候大家甚至可能使用的是最原始的 GradientDescent，究竟這些 optimizer 哪一個比較適合用來 train LSTM 呢? 有一個都市傳說似乎是 GradientDescent 是最好的？究竟是為什麼呢？ (optimizer Q30)**

3. **為何 LSTM 的 gate cell 大部分都用 sigmoid function，而不是用其他函數，如 relu？(Activation Q13)**

}

Source code: https://github.com/WarrenTseng/MLDS2017_final_report

實驗過程：http://mlds2017exppages.azurewebsites.net/

此外，根據某組同學的建議，我們加上了 Gradient Descent with exponential decay 及 Adam with beta1=0.5 的實驗。但因為改變參數，故不併入與其它 optimizer 的比較中。實驗結果顯示，Gradient Descent 加上 exponential decay 在此 task 中，與沒有加入 decay 差異不大；Adam with beta1=0.5 的結果則稍微優於默認的 beta1=0.9。下列網址為實驗程式及結果。

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffOP_MNIST_expDecay.html

1. 前言

傳統的 RNN 模型在訓練時，可能會因為 RNN cell 連接過長，而發生傳遞值發散至無限大或收斂至 0 等的情況[1]，LSTM (Long Short-Term Memory) 的模型因此被提出來解決該問題[4]。在許多實際的應用上，LSTM 的表現非常傑出，因此本研究希望能探討從三個方面提高 LSTM 表現方法的比較：forget gate 的選擇、forget bias 的調整及 optimizer 的選擇。

LSTM 的模型如圖 1.1，各 gate cell 扮演著非常重要的角色，包含控制什麼訊息該被遺忘、什麼訊息要進行更新及最後要輸出哪一部份的神經元狀態，其運算式如式 1.1[7][8]。在此需求的控制訊號為二元輸出：要(1)或不要(0)，因此在式 1.1 gate cell 中的 activation，通常會使用輸出值域被侷限於此區間的 Sigmoid，如圖 1.2。而 ReLU 及 tanh 等 activation 的輸出值域皆不限於此範圍，如圖 1.3 及圖 1.4，故不使用之。

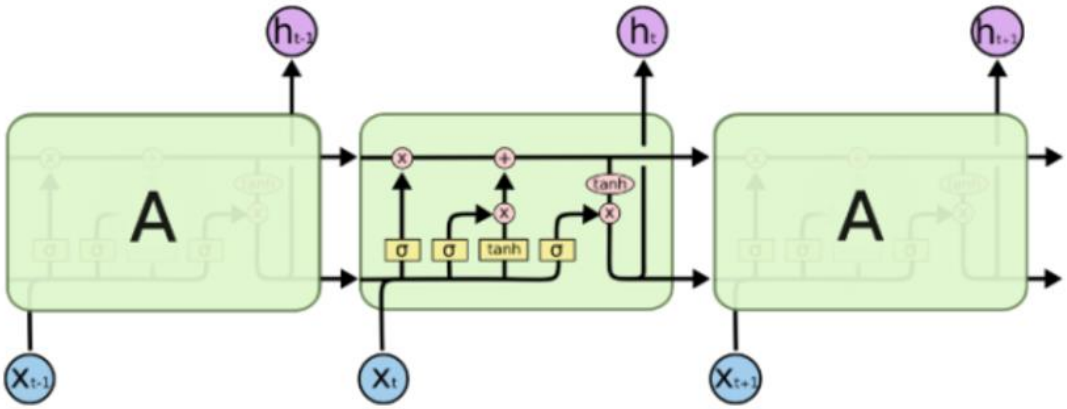


圖 1.1　LSTM 結構[8]

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \qquad \text{式 1.1}$$

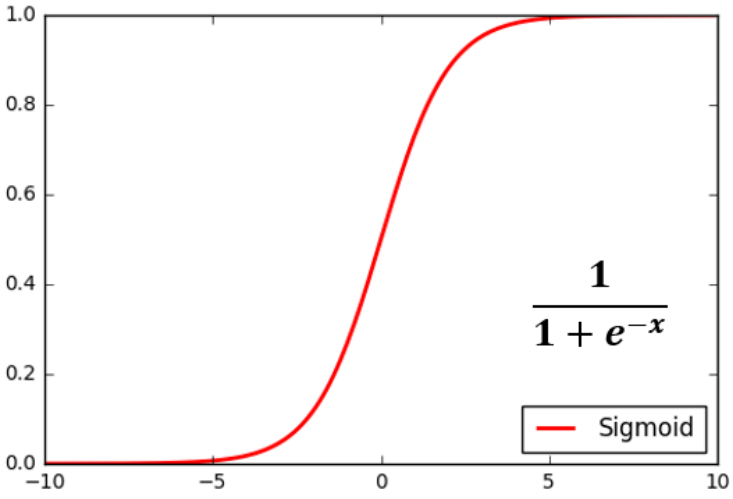


圖 1.2　Sigmoid

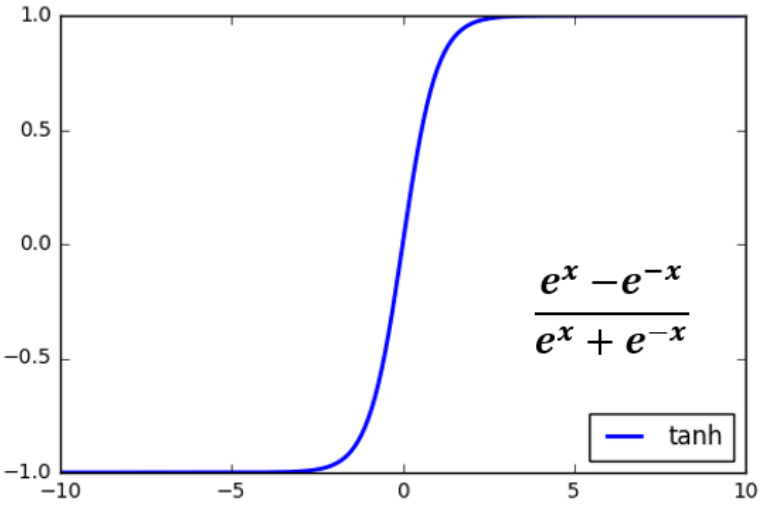


圖 1.3　tanh

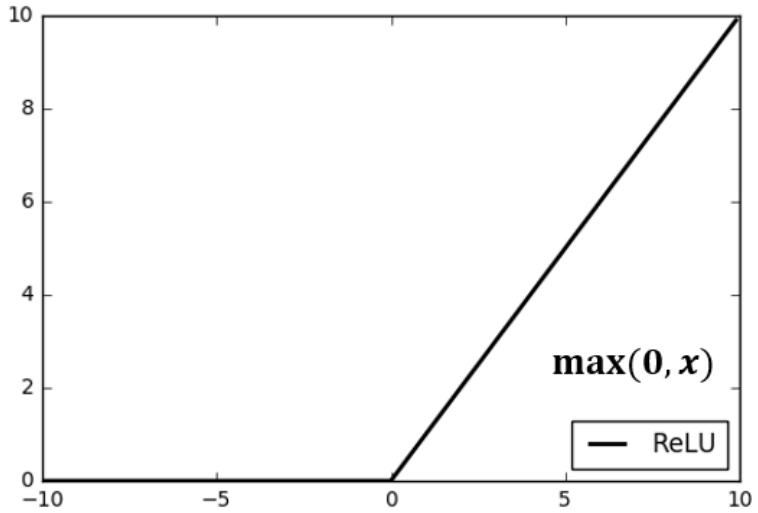


圖 1.4　ReLU

在 Forget gate 中，有否進行適當的 forget bias 初始值設定會影響 LSTM 是否能成功達成學習目標，或是整體模型的好壞[3][6]。Forget gate 決定了是否要遺忘先前保留的訊息，其運算式如式 1.2。可以看到若 bias 越大，則越容易將先前訊息記憶下來[8]。本研究將探討在剛 LSTM 剛開始訊練時，保留較多訊息是否會使其得到較好的訓練成果。

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right) \qquad \text{式 1.2}$$

Optimizer 的選擇會影響到整體訓練的結果。但曾有研究顯示，所有種類的 optimizer 的性能總的來說是相同的，只是需對症下藥，選用最適當的

optimizer [2][10]。本研究也將探討在 LSTM 中，不同的 optimizer 對於其 loss 值收斂情形的影響。

在本研究的實驗中，我認為目前沒有任何一個 task 可以當作最具代表性的範例，故考慮到資料取得的難易程度，將使用常見的 MNIST 資料庫來對 LSTM 的各項影響其成效的機制做實驗，以便大家參考或比較。本研究所使用的網路結構如圖 1.5，只有一層隱藏層先對輸入的 MNIST 圖片(28 x 28)做基本處理，接著使用 28 個 LSTM cell，將圖片分為 28 行分別以由 LSTM cell 來決定該看該圖片的哪些部份進行分類的決策。

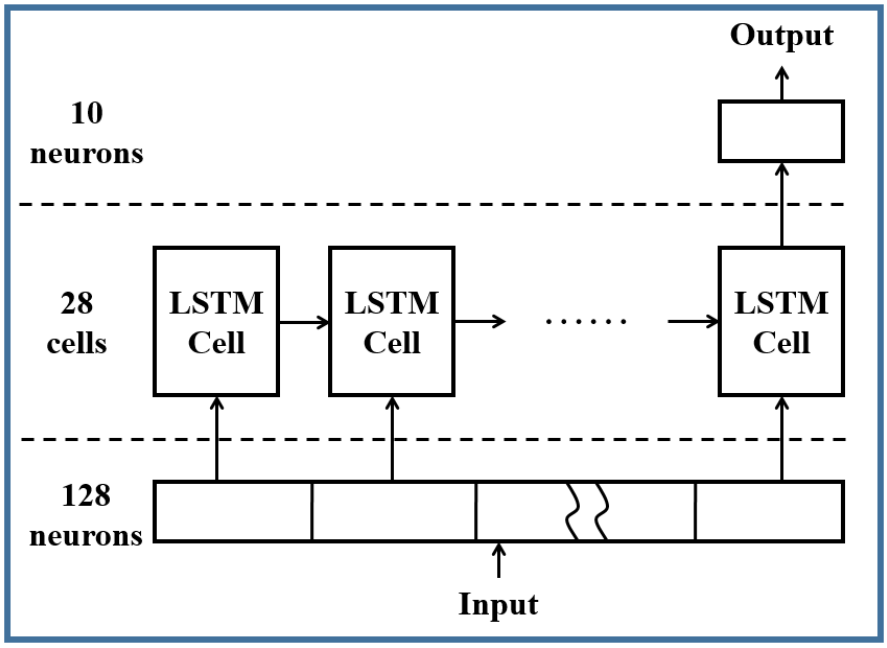


圖 1.5　網路架構

2. Gate Cell 的選擇

2.1 實驗設計

本研究將使用 3 種 LSTM gate 的 activation function (sigmoid、ReLU 及 tanh)來對 MNIST 進行訓練，並觀察不同 activation 的 loss 收斂結果，以判斷使用哪種 activation 更為合適，如表 2.1。

每個實驗 10 重覆，每次重覆皆重新初始化 weight 及 bias，並使各類 gate cell 的 model 初始 weight 及 bias 相同，以增強其鑑別力；待 loss value 皆穩定收斂後結束訓練(2000 個 epoch)，並取最後 100 次 epoch 的 loss 平均值做為該重覆的結果；此處以 Gradient Descent 做為 optimizer，learning rate 設定為 0.01。

表 2.1　不同 activation 對 loss 值的影響實驗規劃

Gate cell	
Sigmoid	實驗 1
ReLU	實驗 2
tanh	實驗 3

2.2 實驗結果

實驗程式與過程：

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffGate_MNIST.html

2.2.1　實驗數據

實驗結果的 training loss 及 testing loss 如表 2.2 與表 2.3。可以發現，ReLU 在使用於 LSTM 的 gate cell 是不可行的；就 loss 平均值的觀察可以發現，tanh 明顯較 sigmoid 的表現好，亦可由訓練過程的 loss 值趨勢比較圖觀察出此結論，如圖 2.1。

表 2.2　Training loss

Forget gate	loss
Sigmoid	$\mu = 0.7879$ $s = 0.0399$
ReLU	nan
tanh	$\mu = 0.3065$

	$s = 0.0088$

表 2.3　Testing loss

Forget gate	loss
Sigmoid	$\mu = 0.7963$ $s = 0.0401$
ReLU	nan
tanh	$\mu = 0.2925$ $s = 0.0128$

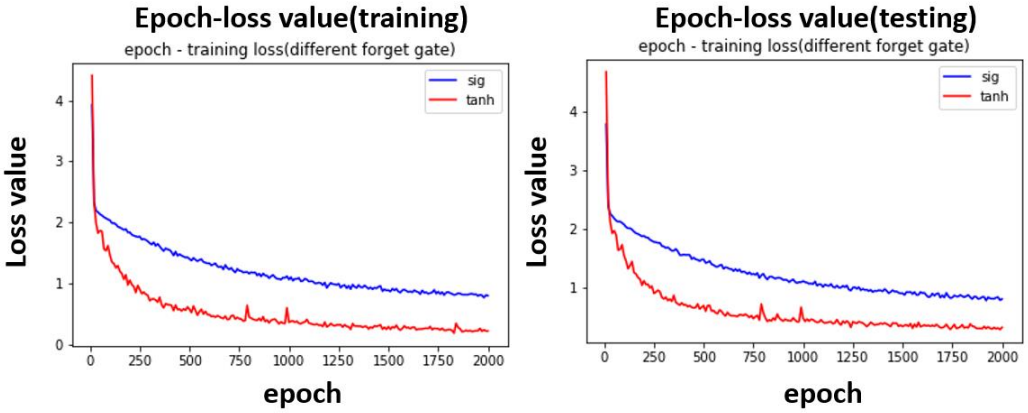


圖 2.2　loss 值隨訓練次數下降的趨勢比較

(感謝某組同學的指正，此處原本放的圖分別是 training 及 testing 的結果，但在寫 title 的時候標記錯了)

2.2.2　統計驗證

接下來以 t-test 對該觀察結果進行驗證，分析結果如表 2.4 及表 2.5。在顯著水準 α=0.05 下，由於 training loss 及 testing loss 的 P-value 值皆遠小於 0.05，故有足夠證據拒絕虛無假設，即 sigmoid 的 loss 值的確較 tanh 大，tanh 確實表現較佳。

$$\text{驗證：}\begin{cases} H_0 : \mu_{sigmoid} = \mu_{tanh} \\ H_a : \mu_{sigmoid} > \mu_{tanh} \end{cases}$$

表 2.4　Training loss t-test

	t-value	P-value
Sigmoid v.s. tanh	20.3751	0.00003
Result	$\mu_{tanh} > \mu_{sigmoid}$	

表 2.5　Testing loss t-test

	t-value	P-value
Sigmoid v.s. tanh	18.0084	0.00007
Result	$\mu_{tanh} > \mu_{sigmoid}$	

根據上述實驗結果，在選擇 LSTM 的 gate cell 時，建議還是使用 tanh 較佳。

3. Forget Bias 初始值大小的選擇

一般而言，forget bias 的初始值建議設定為 1.0 或 2.0，使其一開始盡量不要遺忘，訓練結果較佳；但也有研究指出，有時初始值較大可能會產生過擬合的現像，使其在 Validation 時的結果較差[5][6]，如圖 3.1。以下將以 MNIST 分類為例進行實驗，驗證 bias 初始值對於 LSTM 模型 loss 值的影響。

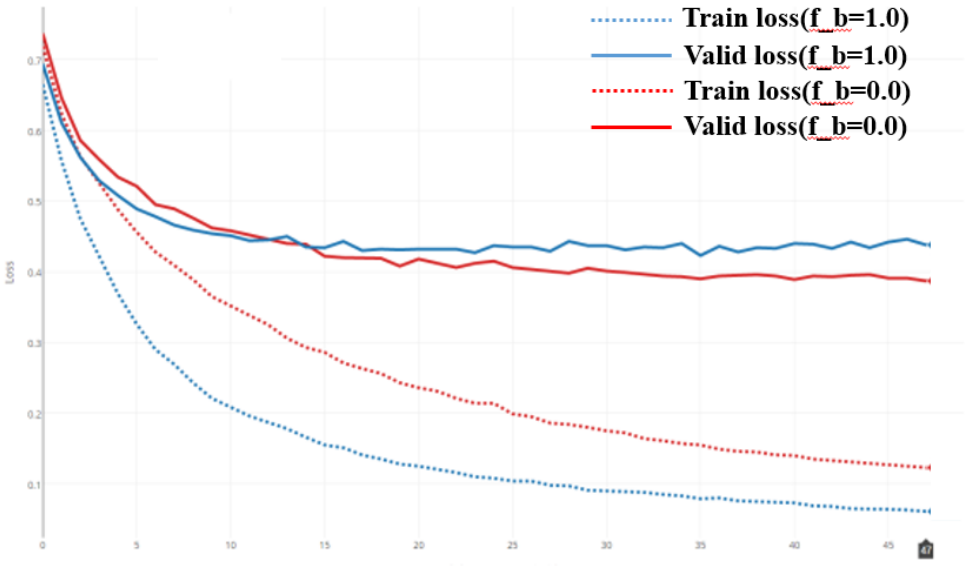


圖 3.1　Forget bias 初始值為 0 及 1 的 loss 比較[5]

3.1 實驗設計

本研究將 forget bias 初始值對訓練結果的影響實驗分為 2 部份：bias 由 1.0 到 10.0，盡量將 bias 調大，觀察其結果；bias 由 0.1 到 1.0，觀察 bias 調低的結果，如表 3.1。

每個實驗 10 重覆，每次重覆皆重新初始化 weight 及 bias，並使左右兩組各組的初始 weight 及 bias 相同，以增強其鑑別力；待 loss value 皆穩定收斂後結束訓練(2000 個 epoch)，並取最後 100 次 epoch 的 loss 平均值做為該重覆的結果；此處以 Gradient Descent 做為 optimizer，learning rate 設定為 0.01。

表 3.1　Forget bias 初始值對 loss 值的影響實驗規劃

Bias 初始值		Bias 初始值	

1.0	實驗 1	0.1	實驗 11
2.0	實驗 2	0.2	實驗 12
3.0	實驗 3	0.3	實驗 13
4.0	實驗 4	0.4	實驗 14
5.0	實驗 5	0.5	實驗 15
6.0	實驗 6	0.6	實驗 16
7.0	實驗 7	0.7	實驗 17
8.0	實驗 8	0.8	實驗 18
9.0	實驗 9	0.9	實驗 19
10.0	實驗 10	1.0	實驗 20

3.2 實驗結果

實驗程式與過程：

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffBias_.1to1_MNIST.html

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffBias_1to10_MNIST.html

3.2.1 實驗數據

實驗結果的 training loss 及 testing loss 如表 3.2 與表 3.3。可以發現於 bias 由 1.0 至 10.0 的實驗中，bias 越大，其 loss value 越大，如圖 3.2；bias 由 0.1 至 1.0 的實驗中，bias 的變化對於 loss value 的影響較少，推論其在訓練過程中，已經自動擬合至最適合的 bias 大小，如圖 3.3。

表 3.2 Training loss

Bias	loss	Bias	loss
1.0	μ = 0.1917 s = 0.0163	0.1	μ = 0.2371 s = 0.0153
2.0	μ = 0.1809 s = 0.0102	0.2	μ = 0.2115 s = 0.0130
3.0	μ = 0.2056 s = 0.111	0.3	μ = 0.2149 s = 0.0198
4.0	μ = 0.2409 s = 0.0248	0.4	μ = 0.2007 s = 0.0112
5.0	μ = 0.2879 s = 0.0178	0.5	μ = 0.2095 s = 0.0115

6.0	$\mu = 0.2987$ $s = 0.0342$	0.6	$\mu = 0.1999$ $s = 0.0148$
7.0	$\mu = 0.3069$ $s = 0.0341$	0.7	$\mu = 0.2056$ $s = 0.0130$
8.0	$\mu = 0.3415$ $s = 0.0284$	0.8	$\mu = 0.2037$ $s = 0.0159$
9.0	$\mu = 0.3475$ $s = 0.0352$	0.9	$\mu = 0.1963$ $s = 0.0143$
10.0	$\mu = 0.3415$ $s = 0.0284$	1.0	$\mu = 0.2019$ $s = 0.0123$

表 3.3　Testing loss

Bias	loss	Bias	loss
1.0	$\mu = 0.2799$ $s = 0.0118$	0.1	$\mu = 0.3206$ $s = 0.0171$
2.0	$\mu = 0.2799$ $s = 0.0223$	0.2	$\mu = 0.2944$ $s = 0.0131$
3.0	$\mu = 0.2948$ $s = 0.0170$	0.3	$\mu = 0.2985$ $s = 0.0235$
4.0	$\mu = 0.3285$ $s = 0.0238$	0.4	$\mu = 0.2829$ $s = 0.0123$
5.0	$\mu = 0.3767$ $s = 0.0199$	0.5	$\mu = 0.2930$ $s = 0.0143$
6.0	$\mu = 0.3876$ $s = 0.0388$	0.6	$\mu = 0.2879$ $s = 0.0156$
7.0	$\mu = 0.4011$ $s = 0.0305$	0.7	$\mu = 0.1910$ $s = 0.0104$
8.0	$\mu = 0.4366$ $s = 0.0344$	0.8	$\mu = 0.2887$ $s = 0.0159$
9.0	$\mu = 0.4437$ $s = 0.0392$	0.9	$\mu = 0.2850$ $s = 0.0143$
10.0	$\mu = 0.4290$ $s = 0.0354$	1.0	$\mu = 0.2866$ $s = 0.0143$

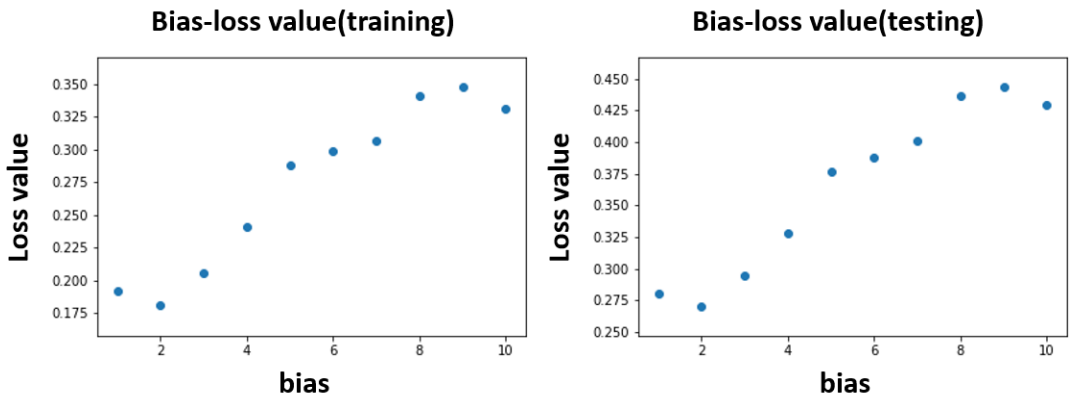


圖 3.2　Bias 由 1.0 到 10.0 的 loss 平均值變化

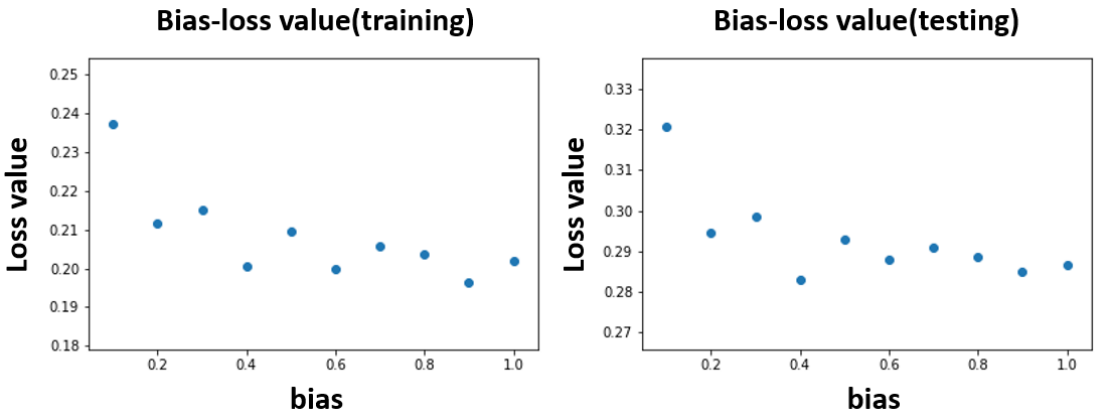


圖 3.3　Bias 由 0.1 到 1.0 的 loss 平均值變化

3.2.2　統計驗證

在圖 3.2 中，可以發現 bias 在 1.0 至 3.0 間，loss value 會有較好表現，以下將以 t-test 驗證 bias=1.0、bias=2.0 及 bias=3.0 loss value 的關係。分析結果如表 3.4 及表 3.5 所示；於驗證一中，在顯著水準 α=0.05 下，由於 training loss 及 testing loss 的 P-value 值皆大於 0.05，故沒有足夠證據拒絕虛無假設，即 bias=1.0 與 bias=2.0 的 loss value 相等；於驗證二中，在顯著水準 α=0.05 下，由於 training loss 的 P-value 值小於 0.05，故有足夠證據拒絕虛無假設，即 bias=3.0 的 loss value 較 bias=2.0 的大；testing loss 的 P-value 值大於 0.05，故沒有足夠證據拒絕虛無假設，即 bias=3.0 與 bias=2.0 的 loss value 相等。

$$\text{驗證一：}\begin{cases} H_0：\mu_{b=1.0} = \mu_{b=2.0} \\ H_a：\mu_{b=1.0} > \mu_{b=2.0} \end{cases}$$

$$\text{驗證二：}\begin{cases} H_0：\mu_{b=3.0} = \mu_{b=2.0} \\ H_a：\mu_{b=3.0} > \mu_{b=2.0} \end{cases}$$

表 3.4　Training loss t-test 分析結果

	t-value	P-value
b=1.0 v.s. b=2.0	1.6739	0.1147
b=2.0 v.s. b=3.0	2.1224	0.0499
Result	$\mu_{b=1.0} = \mu_{b=2.0} < \mu_{b=3.0}$	

表 3.5　Testing loss t-test 分析結果

	t-value	P-value
b=1.0 v.s. b=2.0	1.1058	0.2879
b=2.0 v.s. b=3.0	1.6002	0.1282
Result	$\mu_{b=1.0} = \mu_{b=2.0} = \mu_{b=3.0}$	

由上述實驗結果可以推論，forget bias 的初始值大小並非越大越好，若希望以較大的 bias 開始訓練，建議初始值設定於 1.0 至 3.0 區間即可。

4. Optimizer 的選擇

在選用 optimizer 時，learning rate 的選擇是非常困難的，若 learning rate 太小，可能造成收斂過慢；若 learning rate 太大，又可能使其收斂困難，且會有較大波動，甚至波動至一定程度後就發散掉[9]。因此在選擇 optimizer 時，還需調整 learning rate 至適合的大小，使其順利收斂。常見的 optimizer 有 Adam、RMSprop 及 Gradient Descent 等演算法，以下將針對這幾類演算法以及不同的 learning rate 進行 loss 值收斂情形的實驗，探討 optimizer 及 learning rate 對於收斂的影響。

4.1 實驗設計

本研究將探討選用不同的 optimizer 及 learning rate 對於 training loss 與 testing loss 的影響，並歸納選用 optimizer 時 learning rate 該如何調整，如表 4.1。

每個實驗 10 重覆，每次重覆皆重新初始化 weight 及 bias，並使左右兩組各組的初始 weight 及 bias 相同，以增強其鑑別力；待 loss value 皆穩定收斂後結束訓練(2000 個 epoch)，並取最後 100 次 epoch 的 loss 平均值做為該重覆的結果。

最後，在訓練時的每個 epoch 皆記錄其花費時間，並以統計驗證各 optimizer 在花費時間上的效益。

表 4.1　Optimizer 及 learning rate 對 loss 的影響實驗規劃

	Adam	RMS	AdaGrad	Grad
0.1	實驗 1	實驗 2	實驗 3	實驗 4
0.01	實驗 5	實驗 6	實驗 7	實驗 8
0.001	實驗 9	實驗 10	實驗 11	實驗 12
0.0001	實驗 13	實驗 14	實驗 15	實驗 16

4.2 實驗結果

實驗程式與過程：

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffOP_MNIST_lr0.1.html

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffOP_MNIST_lr0.01.html

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffOP_MNIST_lr0.001.html

http://mlds2017exppages.azurewebsites.net/exppages/LSTM_diffOP_MNIST_lr0.0001.html

4.2.1 實驗數據

實驗結果的 training loss、accuracy 及 testing loss、accuracy 如表 4.2 至表 4.5。可以發現對於同一個 optimizer，調整不同的 learning rate 會影響其最後訓練的結果；同一個 learning rate，選用不同的 optimizer 亦對最後的 loss 值有顯著影響。

表 4.2　Training loss

	Adam	RMS	AdaGrad	Grad	mean
0.1	$\mu = 2.0329$ $s = 0.3873$	$\mu = 2.0218$ $s = 0.4200$	$\mu = 0.2585$ $s = 0.0557$	$\mu = 0.2305$ $s = 0.0316$	1.1359
0.01	$\mu = 0.1831$ $s = 0.0307$	$\mu = 0.0327$ $s = 0.0044$	$\mu = 0.0985$ $s = 0.0123$	$\mu = 0.1885$ $s = 0.0176$	0.1257
0.001	$\mu = 0.0527$ $s = 0.0064$	$\mu = 0.0105$ $s = 0.0011$	$\mu = 0.3611$ $s = 0.0142$	$\mu = 0.4725$ $s = 0.0226$	0.2242
0.0001	$\mu = 0.1318$ $s = 0.0076$	$\mu = 0.0844$ $s = 0.0066$	$\mu = 1.3889$ $s = 0.0682$	$\mu = 1.3101$ $s = 0.0910$	0.7288
mean	0.6001	0.5374	0.5268	0.5504	0.5537

表 4.3　Training accuracy

	Adam	RMS	AdaGrad	Grad
0.1	$\mu = 0.2496$ $s = 0.1025$	$\mu = 0.3423$ $s = 0.1430$	$\mu = 0.9136$ $s = 0.0288$	$\mu = 0.9381$ $s = 0.0185$
0.01	$\mu = 0.9496$ $s = 0.0089$	$\mu = 0.9925$ $s = 0.0013$	$\mu = 0.9726$ $s = 0.0034$	$\mu = 0.9387$ $s = 0.0052$
0.001	$\mu = 0.9857$ $s = 0.0015$	$\mu = 0.9983$ $s = 0.0005$	$\mu = 0.8869$ $s = 0.0047$	$\mu = 0.8598$ $s = 0.0060$
0.0001	$\mu = 0.9590$ $s = 0.0029$	$\mu = 0.9773$ $s = 0.0012$	$\mu = 0.5406$ $s = 0.0187$	$\mu = 0.5574$ $s = 0.0296$

表 4.4　Testing loss

	Adam	RMS	AdaGrad	Grad	mean
0.1	$\mu = 2.0658$ $s = 0.3991$	$\mu = 2.2015$ $s = 0.3945$	$\mu = 0.3550$ $s = 0.0566$	$\mu = 0.3354$ $s = 0.0326$	1.2394
0.01	$\mu = 0.2289$ $s = 0.0331$	$\mu = 0.1472$ $s = 0.0073$	$\mu = 0.1697$ $s = 0.0145$	$\mu = 0.2802$ $s = 0.0198$	0.2065

0.001	$\mu = 0.0863$ $s = 0.0082$	$\mu = 0.0803$ $s = 0.0049$	$\mu = 0.3761$ $s = 0.0167$	$\mu = 0.5060$ $s = 0.0209$	0.2622
0.0001	$\mu = 0.1475$ $s = 0.0080$	$\mu = 0.1600$ $s = 0.0094$	$\mu = 1.3682$ $s = 0.0745$	$\mu = 1.2972$ $s = 0.0962$	0.7432
mean	0.6321	0.6473	0.5673	0.6074	0.6129

表 4.5　Testing accuracy

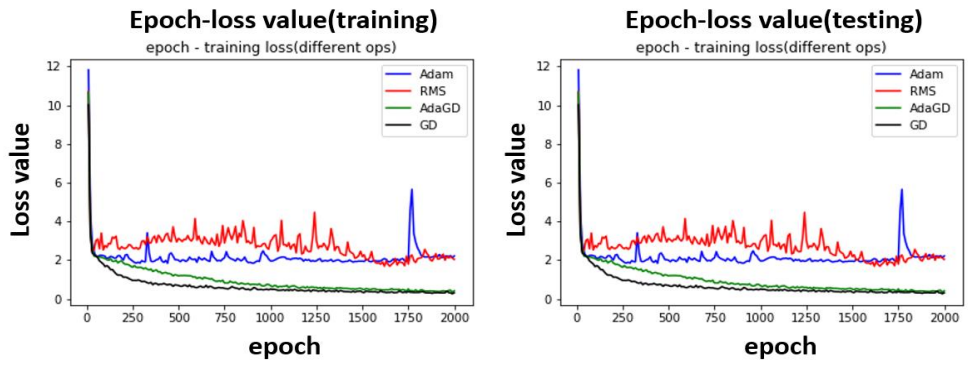
	Adam	RMS	AdaGrad	Grad
0.1	$\mu = 0.2324$ $s = 0.1040$	$\mu = 0.3017$ $s = 0.1384$	$\mu = 0.8753$ $s = 0.0277$	$\mu = 0.8987$ $s = 0.0167$
0.01	$\mu = 0.9349$ $s = 0.0093$	$\mu = 0.9567$ $s = 0.0022$	$\mu = 0.9473$ $s = 0.0038$	$\mu = 0.9077$ $s = 0.0034$
0.001	$\mu = 0.9742$ $s = 0.0026$	$\mu = 0.9766$ $s = 0.0011$	$\mu = 0.8828$ $s = 0.0044$	$\mu = 0.8483$ $s = 0.0092$
0.0001	$\mu = 0.9514$ $s = 0.0036$	$\mu = 0.9476$ $s = 0.0027$	$\mu = 0.5429$ $s = 0.0206$	$\mu = 0.5622$ $s = 0.0296$

不同 optimizer 與不同 learning rate 的影響可以在圖 4.1 中更清楚地觀察到。在圖 4.1(a)中，可以看到在 learning rate=0.1 時，Gradient Descent / AdaGrad 的收斂情形較佳，且 Gradient Descent 的收斂速度較 AdaGrad 稍快，但 Adam / RMSprop 的收斂情形不佳，且非常不穩定，推論是該 learning rate 對於 Adam / RMSprop 的方式來說過大，無法完成收斂。

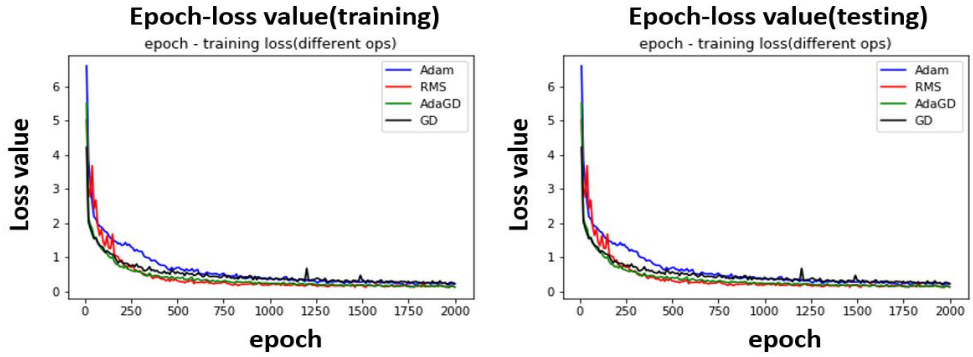
在圖 4.1(b)中，可以看到在 learning rate=0.01 時，各 optimizer 皆有不錯的表現，但可能仍較不適合 Adam，其收斂速度稍慢。

在圖 4.1(c)中，可以看到在 learning rate=0.001 時，開始體現 Adam 及 RMSprop 的優點，Adam 及 RMSprop 的收斂速度在此時會比先前快，且收斂的 loss 值表現優異；而 Gradient Descent 及 AdaGrad 則隨 learning rate 的調低而收斂速度開始下降。

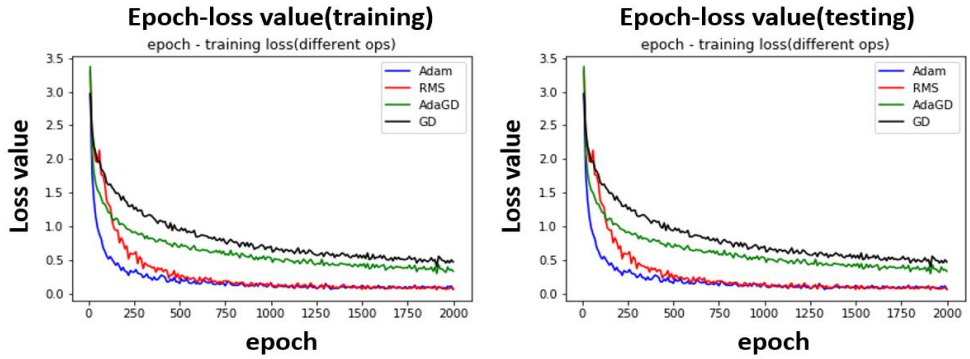
在圖 4.1(d)中，可以看到在 learning rate=0.0001 時，Adam / RMSprop 及 Gradient Descent / AdaGrad 的區別更加明顯，顯然 Adam / RMSprop 適合使用較小的 learning rate，而 Gradient Descent / AdaGrad 則適合使用較大的 learning rate。



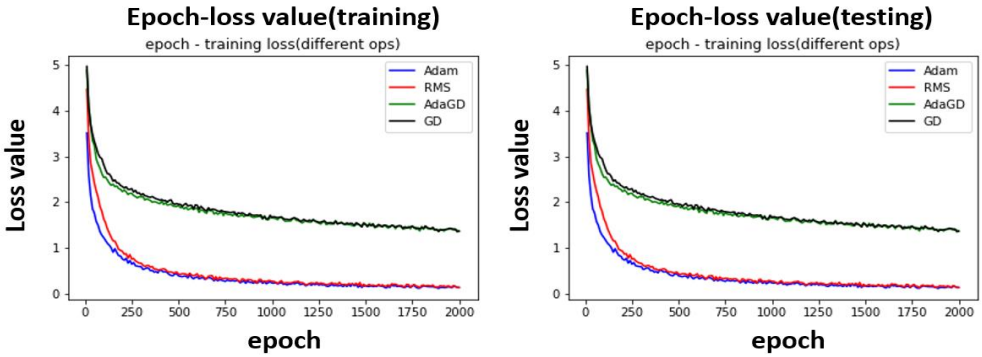
(a) Learning rate = 0.1



(b) Learning rate = 0.01



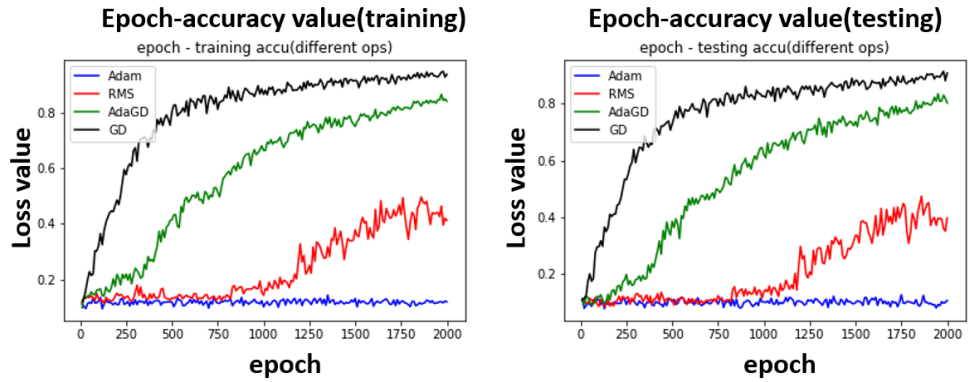
(c) Learning rate = 0.001



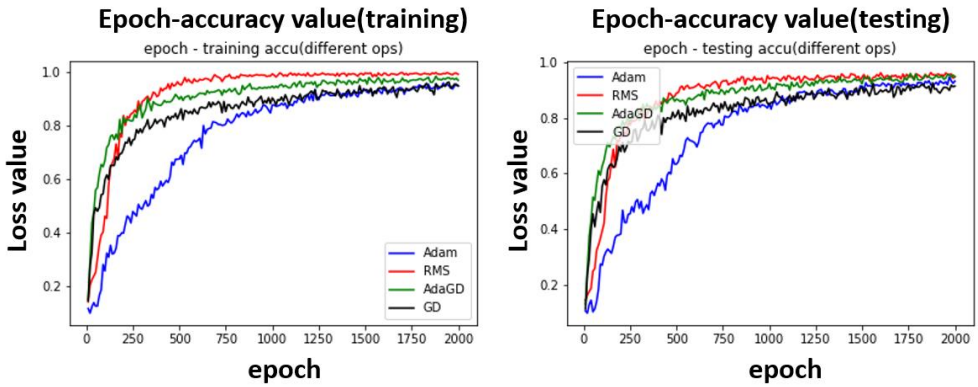
(d) Learning rate = 0.0001

圖 4.1　不同 learning rate 下各 optimizer 的 loss 趨勢比較

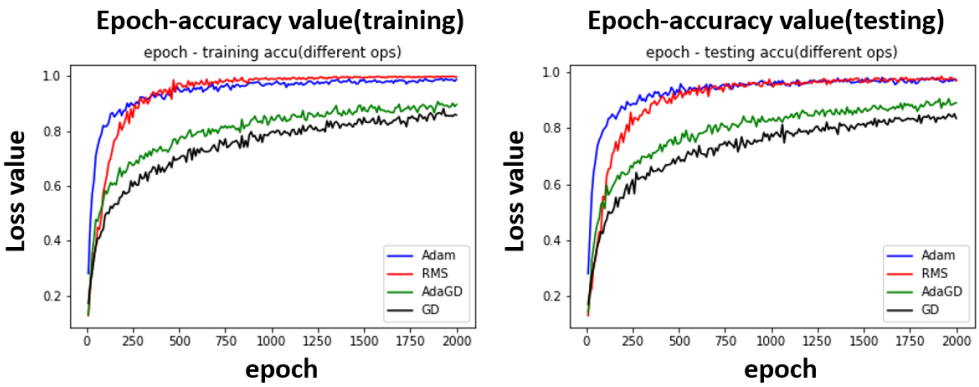
接著觀察不同 optimizer 及不同 learning rate 下的 accuracy 變化，如圖 4.2。由圖 4.1 及圖 4.2 對照比較後，可以發現 loss 的趨勢基本上可以代表 accuracy，loss 越低，accuracy 越高，並沒有出現 overfitting 的情況。值得一提的是，在 learning rate = 0.1 時，Adam 的 accuracy 並沒有隨著 loss 的下降而提高



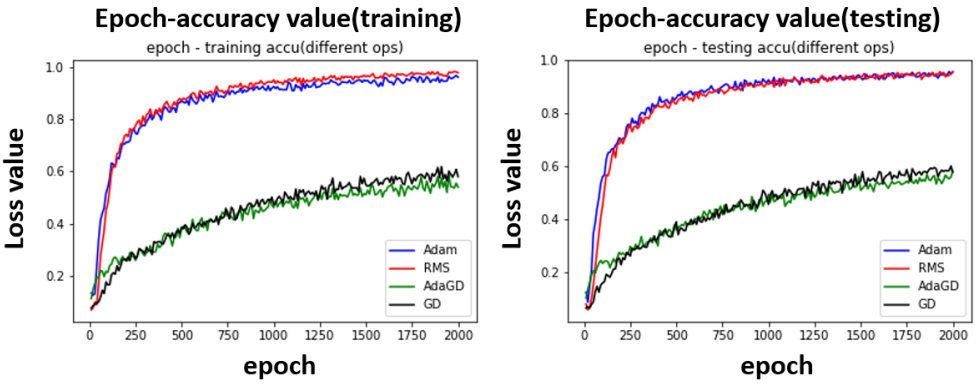
(e) Learning rate = 0.1



(f) Learning rate = 0.01



(g) Learning rate = 0.001



(h) Learning rate = 0.0001

圖 4.2　不同 learning rate 下各 optimizer 的 accuracy 趨勢比較

4.2.2　統計驗證

接下來以 two-way ANOVA 的方法驗證不同的 optimizer、learning rate 及其交互作用對於 loss 值的影響情況，分析結果如表 4.6 及表 4.7。

在顯著水準 α=0.05 的情況下，因為 interaction 的 P value 小於 0.05，故有足夠證據拒絕虛無假設，即 optimizer 及 learning rate 的交互作用對於 loss 值的結果具顯著影響。

在顯著水準 α=0.05 的情況下，因為 optimizer 的 P value 大於 0.05，故無足證據拒絕虛無假設，即在不考慮 learning rate 的情況下，optimizer 的選擇並不會造成顯著影響。但在固定 learning rate 的情況下，如表 4.8，以 learning rate=0.01 為例進行 one-way ANOVA 的分析，optimizer 的選擇就會對 loss 值結果產生顯著影響了。

在顯著水準 α=0.05 的情況下，因為 learning rate 的 P value 小於 0.05，故有足夠證據拒絕虛無假設，即在不考慮 optimizer 的情況下，調整 learning rate 仍會對 loss 值造成顯著影響。

$$\text{驗證一：}\begin{cases} H_0：Optimizer\text{及 learning rate 互不影響} \\ H_a：\text{Optimizer 及 learing rate 交互影響} \end{cases}$$

$$\text{驗證二：}\begin{cases} H_0：Optimizer\text{對 loss 沒有影響} \\ H_a：Optimizer\text{對 loss 有顯著影響} \end{cases}$$

$$\text{驗證三：}\begin{cases} H_0：\text{Learning rate 對 loss 沒有影響} \\ H_a：\text{Learning rate 對 loss 有顯著影響} \end{cases}$$

表 4.6　Training loss two-way ANOVA

	df	SS	MS	F	P
optimizer	3	0.1263	0.0421	1.7538	0.1586
learning rate	3	26.4557	8.8186	367.3552	0.0000
interaction	9	48.8381	5.4265	226.0500	0.0000
residual	144	3.4568	0.0240		

表 4.7　Testing loss two-way ANOVA

	df	SS	MS	F	P
optimizer	3	0.1480	0.0493	2.1100	0.1015
learning rate	3	27.9070	9.3023	397.8662	0.0000
interaction	9	47.3290	5.2588	224.9210	0.0000
residual	144	3.3668	0.0234		

表 4.8　Learning rate = 0.01 one-way ANOVA (training loss)

	F value	P value
Different optimizer	140.1897	0.0000
Result	不同 optimizer 會對 loss 值造成顯著影響	

接下來，將以 t-test 對不同 learning rate 及不同 optimizer 的組別一一進行分析，以比較各情況下的 loss 值結果何者較優異，分析結果如表 4.9 及表 4.10 所示。考慮到表格的可讀性，因此以顏色來進行排名的表示，顏色愈深，表現愈優異，在此處加入 accuracy 的排名與 loss 進行對照比較，loss 的部份如表 4.11 及表 4.13，accuracy 的部份則如表 4.12 及表 4.14。可以發現基本上 loss 及 accuracy 表現優異程度是非常一致的，符合我們對於圖 4.2 的判斷；且普遍來說，Adam 及 RMSprop 的表現會較 Gradient Descent / AdaGrad 佳，惟 learning rate 太大時才會無法完成收斂，而導致 loss 值的表現極差。Gradient Descent / AdaGrad 的表現在 learning rate = 0.01 時最佳，learning rate = 0.001 及 0.0001 時的收斂速度過慢，在此表現不佳。

表 4.9　各實驗的 training loss 表現優異程度排名

	Adam	RMS	AdaGrad	Grad
0.1	12	12	8	8

0.01	7	2	5	7
0.001	3	1	9	10
0.0001	6	4	11	11

表 4.10　各實驗的 testing loss 表現優異程度排名

	Adam	RMS	AdaGrad	Grad
0.1	9	9	6	6
0.01	4	2	3	5
0.001	1	1	6	7
0.0001	2	3	8	8

表 4.11　各實驗的 training loss 表現優異程度

	Adam	RMS	AdaGrad	Grad
0.1				
0.01				
0.001				
0.0001				

表 4.12　各實驗的 training accuracy 表現優異程度

	Adam	RMS	AdaGrad	Grad
0.1				
0.01				
0.001				
0.0001				

表 4.13　各實驗的 testing loss 表現優異程度

	Adam	RMS	AdaGrad	Grad
0.1				
0.01				
0.001				
0.0001				

表 4.14　各實驗的 testing accuracy 表現優異程度

	Adam	RMS	AdaGrad	Grad
0.1				

0.01				
0.001				
0.0001				

固定 optimizer 來觀察各 optimizer 在使用不同 learning rate 的表現情況，分析結果如表 4.15 及表 4.16。由結果可以發現，較適合 Adam / RMSprop 的 learning rate 為 0.001；而如果考慮收斂速度及收斂結果，較適合 Gradient Descent / AdaGrad 的 learning rate 為 0.01。

表 4.15　各 optimizer 在不同 learning rate 的表現情況 (training loss)

	Adam	RMS	AdaGrad	Grad
0.1	4	4	2	2
0.01	3	2	1	1
0.001	1	1	3	3
0.0001	2	3	4	4

表 4.16　各 optimizer 在不同 learning rate 的表現情況 (testing loss)

	Adam	RMS	AdaGrad	Grad
0.1	4	4	2	2
0.01	3	2	1	1
0.001	1	1	2	3
0.0001	2	3	3	4

最後，本研究對不同 optimizer 每一個 epoch 的訓練時間做統計，本研究使用的 GPU 型號為 *NVIDIA GTX-1060 6G*，batch size 為 128，結果如表 4.17。由表中可以發現，基本上 Adam 所花費的時間會稍微較其他 optimizer 長，但長的有限，因此基本上在考慮各 optimizer 的效能時，可以忽略花費時間的影響。

表 4.17　Training time / epoch (ms)

	Adam	RMS	AdaGrad	Grad
0.1	$\mu = 11.041$ $s = 1.943$	$\mu = 10.990$ $s = 1.504$	$\mu = 10.921$ $s = 1.476$	$\mu = 10.941$ $s = 1.473$

0.01	$\mu = 10.932$ $s = 2.128$	$\mu = 10.874$ $s = 1.538$	$\mu = 10.807$ $s = 1.497$	$\mu = 10.845$ $s = 1.528$
0.001	$\mu = 10.933$ $s = 1.878$	$\mu = 10.911$ $s = 1.570$	$\mu = 10.839$ $s = 1.495$	$\mu = 10.854$ $s = 1.492$
0.0001	$\mu = 10.937$ $s = 1.936$	$\mu = 10.895$ $s = 1.558$	$\mu = 10.830$ $s = 1.539$	$\mu = 10.855$ $s = 1.546$

5. 結論

由以上實驗，我們可以歸納出以下幾點結論：

(1) LSTM 的 gate cell 需使用適合其需求輸出值域的 activation function。

(2) Forget gate bias 的初始值並非越大越好，而需要對症下藥，case by case。

(3) 在調整 forget gate bias 的初始值時，若希望其儘量保留先前訊息，要給予較大的初始值時，建議設定的初始值為 1.0 至 3.0 間即可。

(4) 在 optimizer 的實驗中，驗證不同 optimizer 的影響力時，不考慮其他因素的話，選擇不同 optimizer 並不會對整體 loss 值的表現造成顯著影響，間接驗證了[2][10]的觀點，需要對症下藥。

(5) 若希望使用 Adam / RMSprop 等 optimizer，建議將 learning rate 數值設定小一點(0.001)，會有較優異的表現。

(6) 若希望使用 Gradient Descent / AdaGrad 等 optimizer，建議 learning rate 不要設定過小，會影響到其收斂速度。

(7) 在使用 LSTM 來進行 MNIST 分類的 case 中，各類 optimizer 的表現排名依序如下：RMSprop、Adam、AdaGrad、Gradient Descent，因此 Gradient Descent 未必是最適合 LSTM 的 optimizer。

LSTM 及其變形(GRU 等)的使用非常廣泛，在各需要使用到 RNN 的領域表現十分優異。期望未來 LSTM 的相關研究不論是在結構上進行調整，亦或是有更佳優化的演算法，使其表現能百尺竿頭，更進一步。

Reference

[1] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[2] Cetin, Y. D. Learning to Learn Gradient Descent by Gradient Descent. http://www.cs.bilkent.edu.tr/~gcinbis/courses/Spring17/CS559/presentations/W5b_YarkinDenizCetin.pdf . 6/5/2017.

[3] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471.

[4] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[5] Jango Y. 知乎：你在训练 RNN 的时候有哪些特殊的 trick？ https://www.zhihu.com/question/57828011/answer/155275958. 6/4/2017.

[6] Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 2342-2350).

[7] Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. arXiv preprint arXiv:1506.02078.

[8] Colah, Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/. 6/6/2017.

[9] Sebastian Ruder. An Overview of Gradient Descent Optimization Algorithms. http://sebastianruder.com/optimizing-gradient-descent/. 6/6/2017.

[10] Thrun, S., & Pratt, L. (Eds.). (2012). Learning to learn. Springer Science & Business Media.