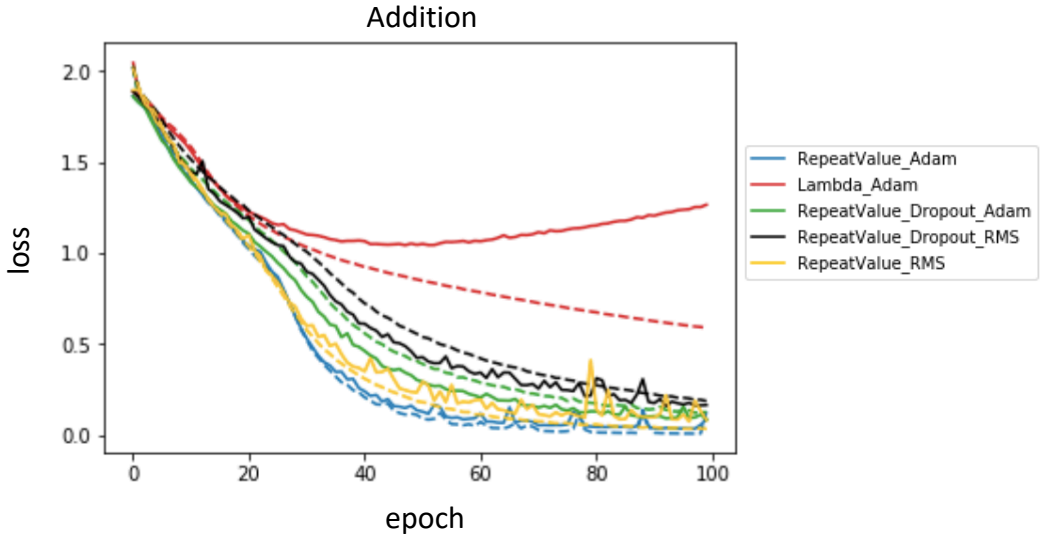
DSAI Assignment 2

一、Adder



1. RepeatVector vs Lambda layer (紅色 vs.藍色)

(1) 說明：

我的 model 主要有三大層，依序為 LSTM、LSTM、Dense。其中紅色表「第一層 LSTM 的 return_sequences=True，接著是一層 Lambda(lambda x: x[:, -(DIGITS + 1):, :])，即只取第一層 LSTM 的最後 DIGITS + 1 個 output vector」。藍色則表「第一層 LSTM 的 return_sequences=False，接著是一層 layers.RepeatVector(DIGITS + 1)」。由上圖可見，在兩層 LSTM 間插入 RepeatVector 遠好於前者。

(2) 原因猜想：

就物理意義而言，第一層 LSTM 是在 encode（總結、理解）input 的資訊，而第二層 LSTM 是在做運算、decode。而藍色 model 就相當於還沒看完整串 input 的算式，就開始總結資訊給下一層。至少在加減法問題，沒看完就開始總結資訊可能較不合理。比如說 input 算式是 123+456，但 encoder 在看完 123+45 時，就告訴 decoder 說自己看到了 123+45，然而 123+45 和 123+456 之間有頗大的差異（如：4 從十位變百位），這樣 123+45 就成了雜訊。

2. Dropout（綠色 vs 藍色 或 黑色 vs 黃色）

無 Dropout 的 model training loss 下降得比較有 Dropout 的 model 快，很合理。同時，無 Dropout 的 model 的 validation loss 也同步下降，且在後期穩定的維持在很低的值，可能是因為加法問題不太容易 overfit。

另外，有 Dropout 的 model 在訓練過程中，validation loss 比 training loss 更低。應是 model.fit()在 training 時有 dropout，而在 validation 時沒有 dropout 所致。

3. Optimizer: RMSprop vs Adam（黃色 vs 藍色 或 黑色 vs 綠色）

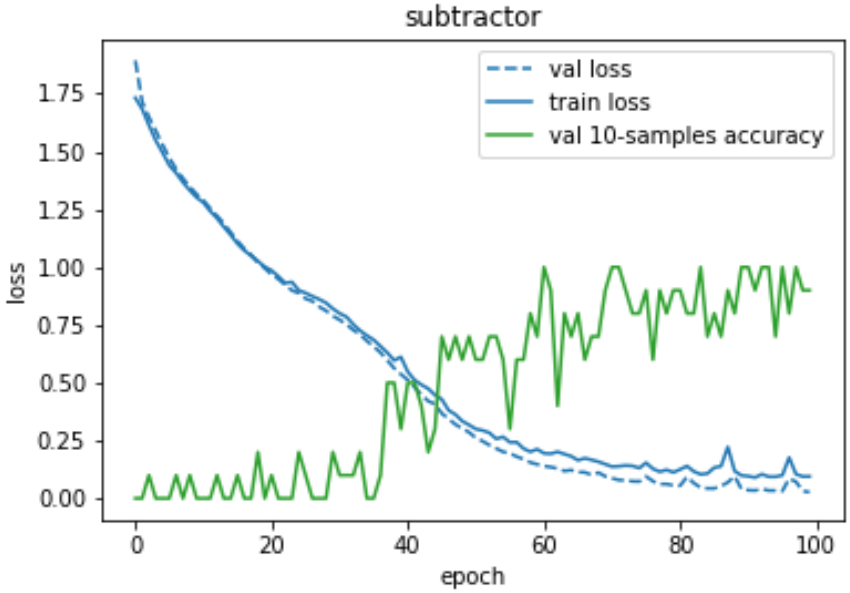
不論無 dropout 或有 dropout，Adam 都能訓練得比 RMSprop 快，可見 Adam 發揮 momentum 的性質，較不易在 gradient descent 時走彎路。

不論無 dropout 或有 dropout，Adam 的 training 曲線都能比 RMSprop 的 training 曲線隆起處來得少。

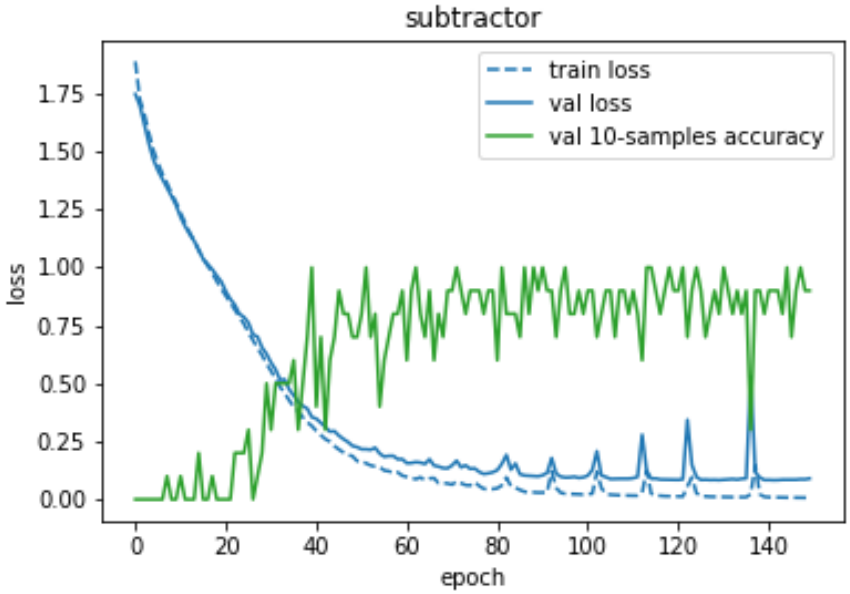
4. 根據以上，adder 在 testing 時決定採用 RepeatValue, 無 dropout, Adam 這些超參數訓練出的 model。另外，亦決定以此設定開始 subtractor 和 Mix 的測試。

5. 將 test_x 對應的 prediction 和 test_y 進行比較（程式碼與在訓練時的每個 epoch 下，測試 10 個 validation sample 的程式碼類似），得到加法運算正確率為 96.4%。

二、Subtractor



此 model 的 hyperparameter 和「一、adder」中藍色線對應的 model 相同，但在 adder，於 epoch 為 100 時，validation loss 早已趨向穩定，而此 subtractor 在此時仍有一些下降趨勢（減法訓練得比加法來得慢，猜想是因為減法的借位等運算較為複雜），因此決定再以更多 epoch (150)進行 training 和 validate，並畫出下圖。

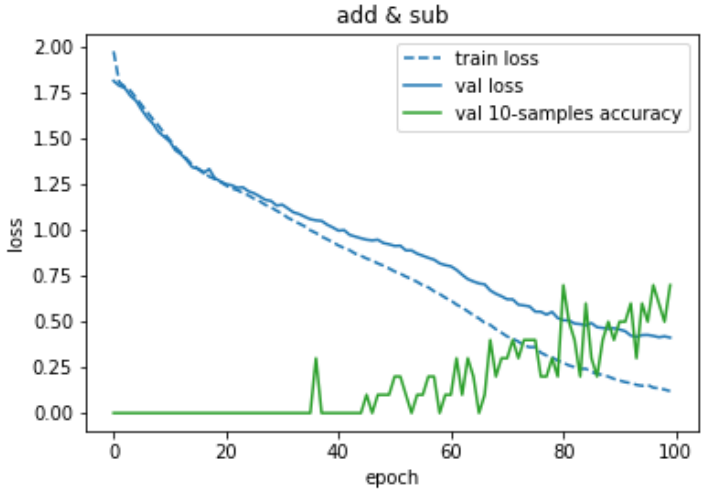


觀察到在 loss 趨穩後會有明顯的隆起。但之後又會回到原本的低 loss rate，且 training loss 也是一起增加，所以應非所謂的 overfit。目前不確定原因，只能猜想是 loss space 的特性。

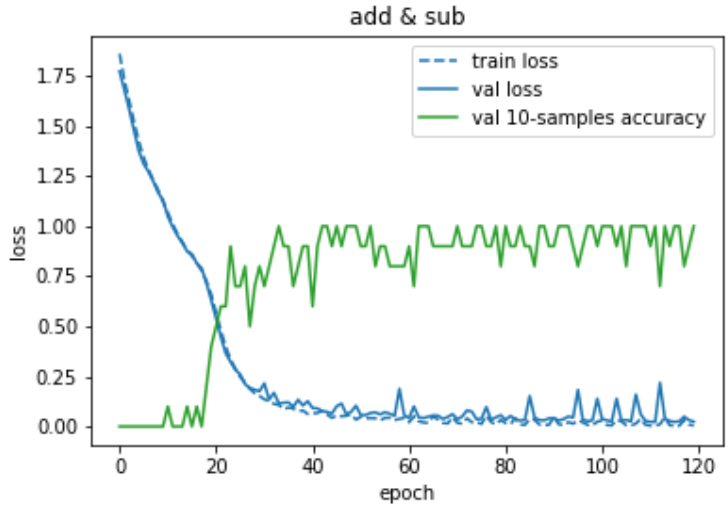
Loss 再過 epoch 100 後確實有繼續降低，又因上圖的 model 最終落回了低 loss，因此決定以上圖的 model 進行 test。

將 test_x 對應的 prediction 和 test_y 進行比較（程式碼與在訓練時的每個 epoch 下，測試 10 個 validation sample 的程式碼類似），得到減法運算正確率為 90.8%。

三、Mix Task (add & subtract)



上圖的為 training set = 18000（同前述的 adder、subtractor）。而我們可見在後段有 overfitting 之情況。因此以下以 training set = 54000 進行 training 和 validate，並畫出下圖。



Overfit 的情況消失了，可見該 training set 能夠提供足夠的有效資訊。

以此 model 進行 test。將 test_x 對應的 prediction 和 test_y 進行比較（程式碼與在訓練時的每個 epoch 下，測試 10 個 validation sample 的程式碼類似），得到加/減法運算正確率為 96.9%。

四、乘法

我認為以相同的模型設定去訓練做乘法，也會有不錯的效果。因為乘法運算與加法運算能相互對應如下：

- 乘法的 1*1 = 1, 1*2 = 2 等九九乘法規則，對應到 1+1 = 2, 1+2 = 3…等加法規則。
- 3*9 = 27 中，模型必須要識別出 2 是進位，對應到加法中的 7+9 = 16 模型可識別出 1 是進位。
- 乘法必須要把進位「加」到下一位的乘法運算結果上。這部分用的正是加法。

後來進行測試的結果如下（超參數與之前相同），並不如加減法來得優異，但 test loss 仍得以持續下降。而該模型在測試資料上的表現只有 8.7%的正確率。與單純加法相比，乘法會有比較差的效果是可以解釋的，因為如上述的第三點所述，進位值是「加」到高一位的「乘」的結果上，這部分較加法為複雜。