# ADL 2022 Fall - HW1 Report

Yang, Yu Chun (D11922022)

tags: 課程報告

## Q1: Data processing (2%)

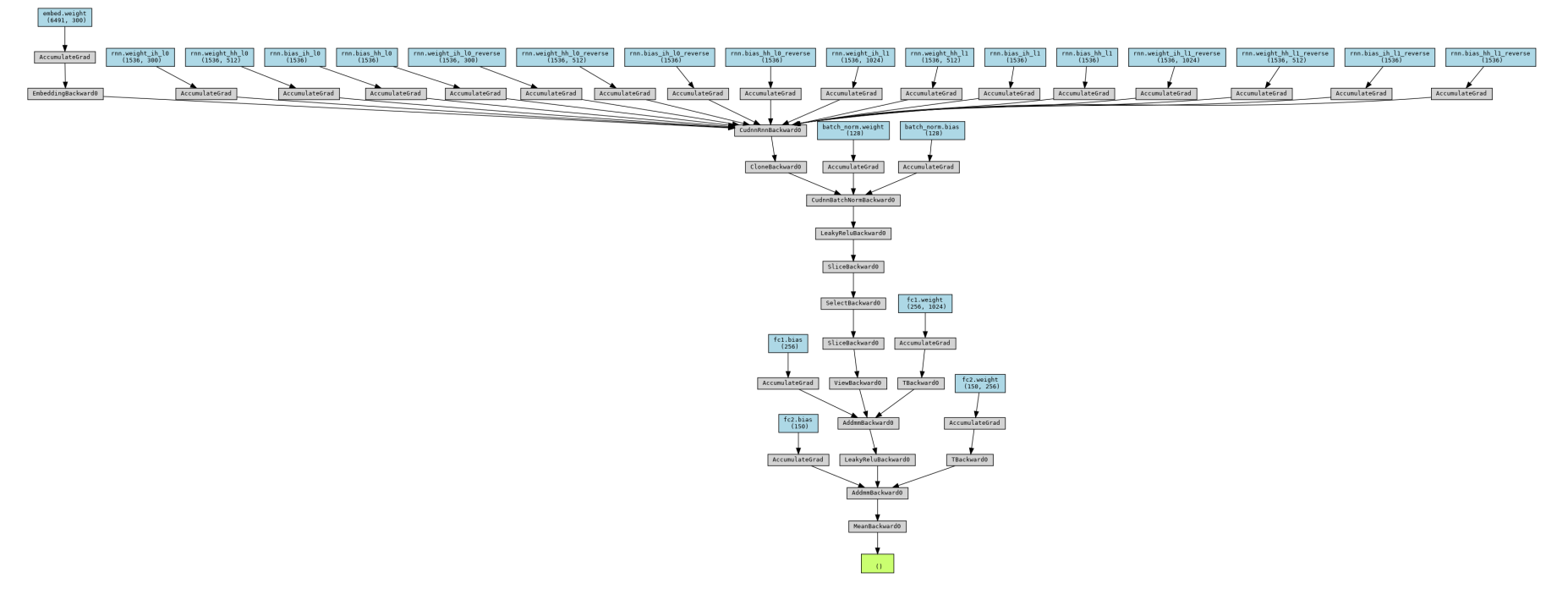
Describe how do you use the data for `intent_cls.sh` , `slot_tag.sh` :

1. How do you tokenize the data.
   對 intent classification、 slot tagging 兩個 task，都先對每個句子以 python 內建的 `str.split()` 針對空白字串 `' '` 將句子內的單字切成 token。
2. The pre-trained embedding you used.
   接著以 [GloVe 的 Common Crawl glove.840B.300d] 這個 pre-trained 的 300 維 word embedding 將上一步切好的 token 向量化。然而某些不被 GloVe 所涵蓋的 token，其 embedding 則用 300 維的隨機值作為其向量。

## Q2: Describe your intent classification model. (2%)

Describe

1. your model



輸入為 300 維的 word embedding，接著以 bidirectional GRU 作為特徵抽取層，其中 $h_t$ 作為 bidirectional GRU 在 $t$ 時間的輸出。$\rightarrow$ 代表 forward layer 的方向；$\leftarrow$ 代表 backward layer 的方向。
$\overrightarrow{h_t} = \text{GRU}(\overrightarrow{h_t}, \overrightarrow{h_{t-1}})$
$\overleftarrow{h_t} = \text{GRU}(\overleftarrow{h_t}, \overleftarrow{h_{t+1}})$

bidirectional GRU 的輸出會將 $\overrightarrow{h_t}$、$\overleftarrow{h_t}$ concat 在一起，再經過 batch normalization 成為 $\text{BatchNorm}([\overrightarrow{h_t}, \overleftarrow{h_t}])$

後面則為 2 層的 MLP (multi-layer perceptron)，每層的 activation function 為 $\text{LeakyReLU}$，並且接著 $\text{Dropout}$。

故整體會變成 $\text{MLP}(\text{BatchNorm}([\overrightarrow{h_t}, \overleftarrow{h_t}]))$ 或是 $\text{MLP}(\text{BatchNorm}([\text{GRU}(\overrightarrow{h_t}, \overrightarrow{h_{t-1}}), \text{GRU}(\overleftarrow{h_t}, \overleftarrow{h_{t+1}})]))$

2. performance of your model.

| 項目 | 數值 |
|---|---|
| Train Acc | 0.99773 |
| Train Loss | 0.00770 |
| Valid Acc | 0.90067 |
| Valid Loss | 0.89981 |
| public score on kaggle | 0.90177 |

3. the loss function you used:

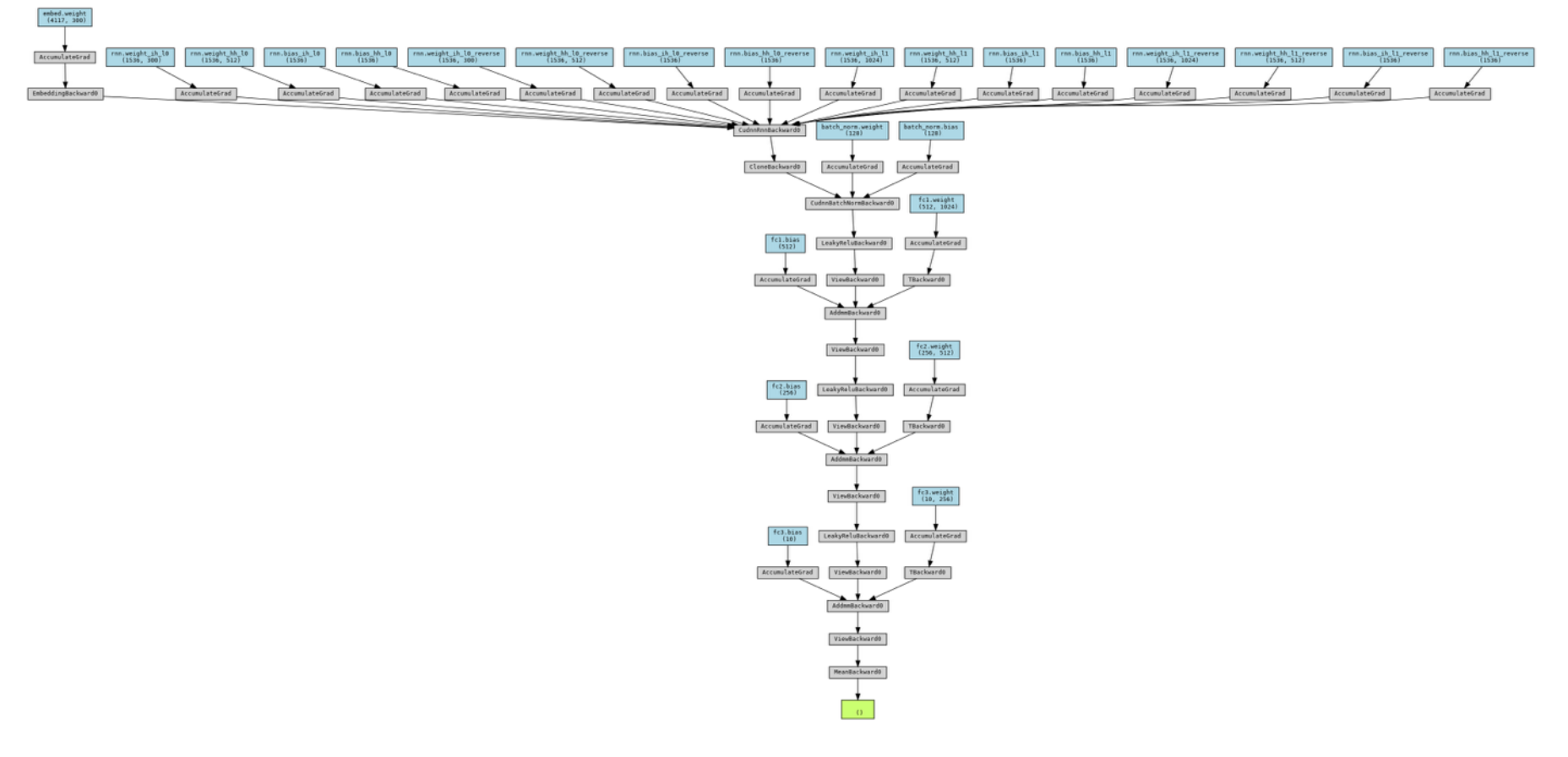
| 表達法 | 表達式 | 註 |
|---|---|---|
| python | `torch.nn.CrossEntropyLoss()` | |
| latex | $\text{CrossEntropyLoss}(\hat{y}, y)$ | $\hat{y}$：預測值<br>$y$：真實值 |

4. The optimization algorithm (e.g. Adam), learning rate and batch size.
   - optimization algorithm: `Adam`
   - learning rate: `1e-3`
   - batch size: `128`

## Q3: Describe your slot tagging model. (2%)

Describe

1. your model



輸入為 300 維的 word embedding，接著以 bidirectional GRU 作為特徵抽取層，其中 $h_t$ 作為 bidirectional GRU 在 $t$ 時間的輸出。$\rightarrow$ 代表 forward layer 的方向；$\leftarrow$ 代表 backward layer 的方向。
$\overrightarrow{h_t} = \text{GRU}(\overrightarrow{h_t}, \overrightarrow{h_{t-1}})$
$\overleftarrow{h_t} = \text{GRU}(\overleftarrow{h_t}, \overleftarrow{h_{t+1}})$

bidirectional GRU 的輸出會將 $\overrightarrow{h_t}$、$\overleftarrow{h_t}$ concat 在一起，再經過 batch normalization 成為 $\text{BatchNorm}([\overrightarrow{h_t}, \overleftarrow{h_t}])$

後面則為 3 層的 MLP (multi-layer perceptron)，每層的 activation function 為 $\text{LeakyReLU}$，並且接著 $\text{Dropout}$。

故整體會變成 $\text{MLP}(\text{BatchNorm}([\overrightarrow{h_t}, \overleftarrow{h_t}]))$ 或是 $\text{MLP}(\text{BatchNorm}([\text{GRU}(\overrightarrow{h_t}, \overrightarrow{h_{t-1}}), \text{GRU}(\overleftarrow{h_t}, \overleftarrow{h_{t+1}})]))$

2. performance of your model.

| 項目 | 數值 |
|---|---|
| Train Acc | 0.99199 |
| Train Loss | 0.00022 |
| Valid Acc | 0.77600 |
| Valid Loss | 0.01995 |
| public score on kaggle | 0.72278 |

3. the loss function you used.

| 表達法 | 表達式 | 註 |
|---|---|---|
| python | `torch.nn.CrossEntropyLoss()` | |
| latex | $\text{CrossEntropyLoss}(\hat{y}, y)$ | $\hat{y}$：預測值<br>$y$：真實值 |

4. The optimization algorithm (e.g. Adam), learning rate and batch size.
   - optimization algorithm: `Adam`
   - learning rate: `1e-3`
   - batch size: `128`

## Q4: Sequence Tagging Evaluation (2%)

1. Please use seqeval to evaluate your model in Q3 on validation set and report `classification_report(scheme=IOB2, mode='strict')`.

```
              precision    recall  f1-score   support

        date       0.75      0.77      0.76       206
  first_name       0.87      0.84      0.86       102
   last_name       0.73      0.67      0.70        78
      people       0.74      0.76      0.75       238
        time       0.86      0.85      0.86       218

   micro avg       0.79      0.79      0.79       842
   macro avg       0.79      0.78      0.78       842
weighted avg       0.79      0.79      0.79       842
```

2. Explain the differences between the evaluation method in seqeval, token accuracy, and joint accuracy.
   - `seqeval`:
     - \(\text{Recall} = \dfrac{TP}{TP + FN}\)
       若將分類任務以打靶類比，Recall 是指所有的靶，你最後命中多少靶，看中的是該打下來的目標被你打下來多少，並不在意你用多少子彈去打。
     - \(\text{Precession} = \dfrac{TP}{TP + FP}\)
       若將分類任務以打靶類比，Precession 是指你擊發出去的子彈，有幾發是有打中把的，比較在意的是你是否有浪費子彈，對於標靶被擊落多少倒沒那麼在意。
     - \(\text{F1-score} = \dfrac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precession}}}\)
       是 Recall 跟 Precessiion 的調和平均數。可以說是同時考慮擊落的靶數 (其實是命中率)，又考慮你的子彈使用效率 (其實就是精確度) 的**綜合準度**。
   - `token accuracy`: 即使是同一個句子內的 token 也將 token 視為相互獨立，以所有 token 的 \(TP\) 為分子，是以一個 token 為單位來計算 accuracy 的。而 accuracy 是指 Recall、Precession、F1-score 或其他定義。
   - `joint accuracy`: 一個句子內所有的 token (或 slot) 全對才算是命中，以整個句子的 \(TP\) 為分子，所以是以一個 sentence 為單位來計算 accuracy。而 accuracy 是指 Recall、Precession、F1-score 或其他定義。

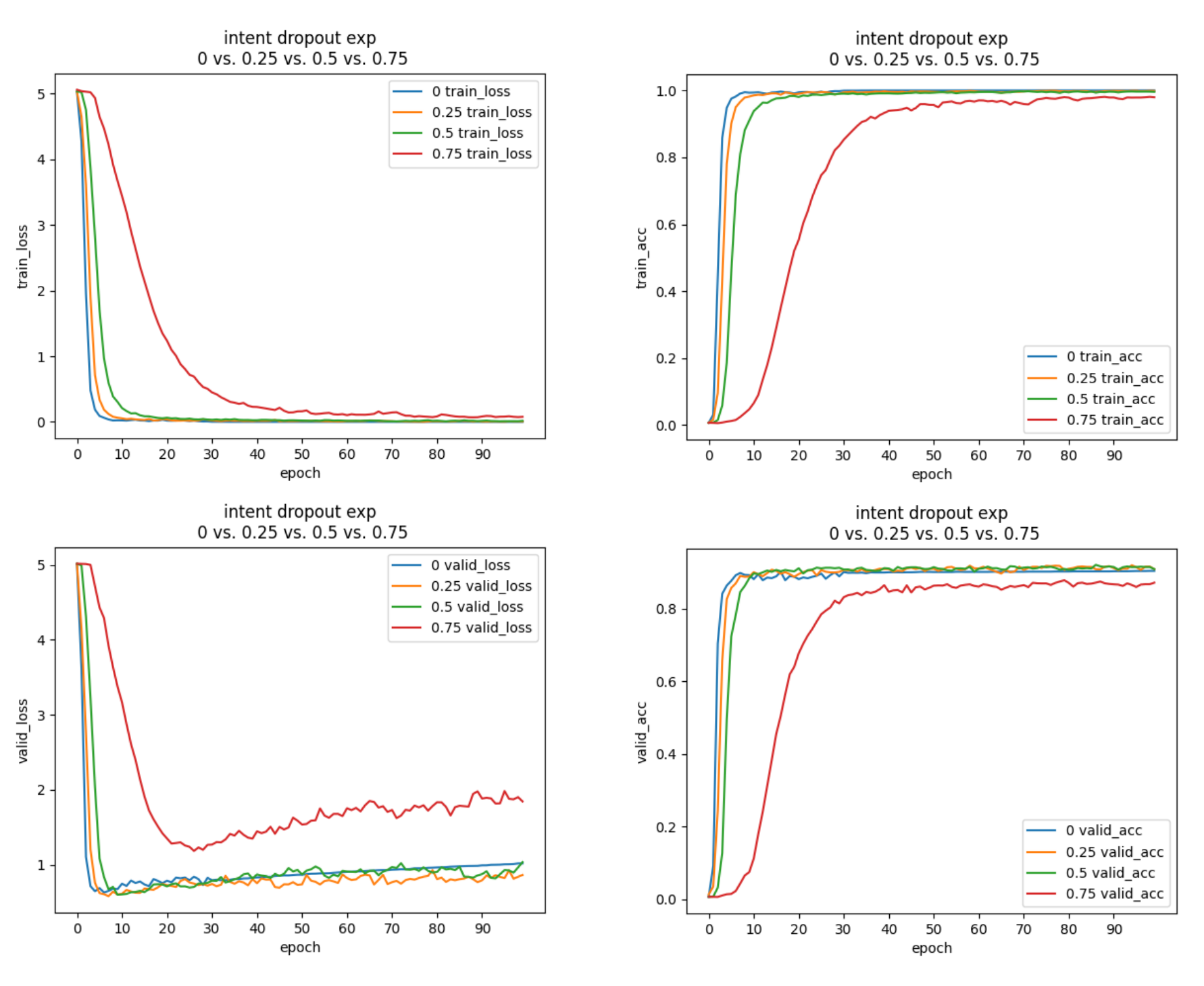
## Q5: Compare with different configurations (1% + Bonus 1%)

- Please try to improve your baseline method (in Q2 or Q3) with different configuration (includes but not limited to different number of layers, hidden dimension, GRU/LSTM/RNN) and EXPLAIN how does this affects your performance / speed of convergence / ...
- Some possible BONUS tricks that you can try: multi-tasking, few-shot learning, zero-shot learning, CRF, CNN-BiLSTM
- This question will be grade by the completeness of your experiments and your findings.

**實驗一： dropout 比例**

##### intent classification

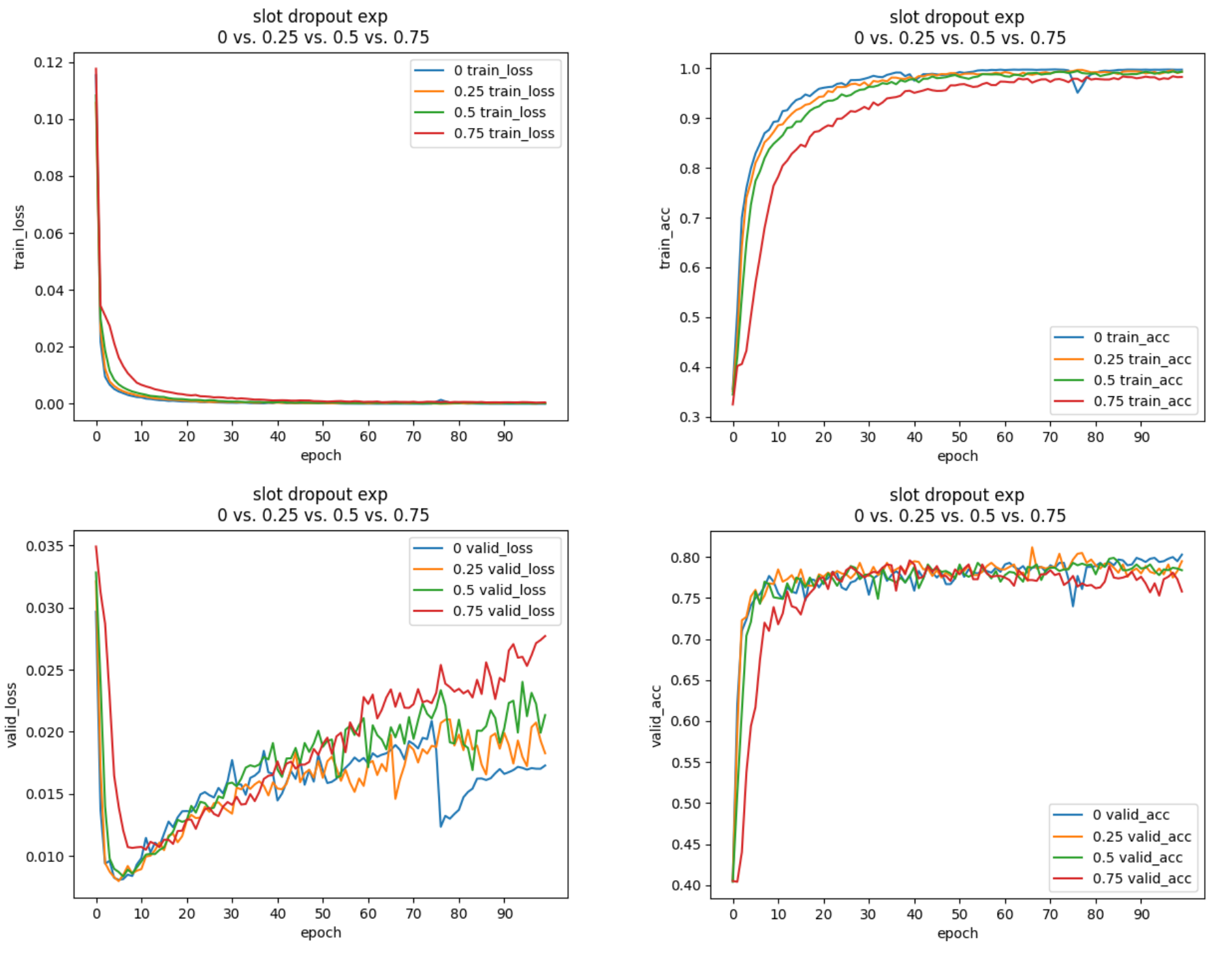
以前述之模型，分別設定不同的 dropout 比例 $0.0$、$0.25$、$0.5$、$0.75$，分別觀察 intent classification 這個 task，在 100 個 epoch 之下的 train loss (左上)、train accuracy (右上)、valid loss (左下)、valid accuracy (右下)。



觀察發現，$0.75$ 的 droupout rate，無論是在 loss 或 accuracy 的表現都較 $0.0$、$0.25$、$0.5$ 差。而 $0.0$、$0.25$、$0.5$ 這三種 droupout rate 雖然收斂速度不同，但最終在 loss 跟 accuracy 的表現都差不多。比較 train/valid loss 往下掉的速度，以及 train/valid accuracy 往上升的速度，可以發現以收斂性由好 (快) 到壞 (慢)，依序是 $0.0$、$0.25$、$0.5$、$0.75$。可以看到 intent classification 以這個模型架構來說，模型複雜度算是剛好，因為在 dropout rate $0.0$、$0.25$、$0.5$ 的設定下，並沒有發生 train loss 小、valid loss 大的 overfitting 狀況，也沒有 train loss 大、valid loss 也大的 underfitting 狀況。然而 $0.75$ 的設定下就稍微有 overfitting 的狀況了

##### slot tagging

以前述之模型，分別設定不同的 dropout 比例 $0.0$、$0.25$、$0.5$、$0.75$，分別觀察 slot tagging 在 100 個 epoch 之下的 train loss (左上)、train accuracy (右上)、valid loss (左下)、valid accuracy (右下)。



觀察發現，$0.75$ 的 droupout rate，無論是在 loss 或 accuracy 的表現都較 $0.0$、$0.25$、$0.5$ 差。而 $0.0$、$0.25$、$0.5$ 這三種 droupout rate 雖然收斂速度不同，但最終在 loss 跟 accuracy 的表現都差不多。比較 train/valid loss 往下掉的速度，以及 train/valid accuracy 往上升的速度，可以發現以收斂性由好 (快) 到壞 (慢)，依序是 $0.0$、$0.25$、$0.5$、$0.75$。然而可以看到四種設定下的 valid loss 都大於 train loss，這雖然看起來像是典型的 overfitting 現象，但個人認為可能是因為 loss function 的算法是 token accuracy，而 accuracy 我是採用 joint accuracy 所致 。

**實驗二： optimizer 的選擇**

##### 各種 optimizer 簡介

傳統的 vanilla gradient descent，每次都是以 learning rate 乘上梯度來修正參數。然而固定的 learning rage 對於不同的 loss function surface 區域來說不見得合適，畢竟在靠近 optimal 的區域，如果固定的 learning rate ，恐怕會造成參數不斷在 optimal 附近擺盪。為了能調整 learning rate 於是便有了 AdaGrad，它的公式如下：

$$W_{t+1} \leftarrow W_t - \frac{\eta}{\sigma_t}\frac{\partial L_t}{\partial W_t}$$
$$\sigma_t = \sqrt{\sum_{t=1}^{T}\left(\frac{\partial L_t}{\partial W_t}\right)^2 + \epsilon}$$
$$W_{t+1} \leftarrow W_t - \frac{\eta}{\sqrt{\sum_{t=1}^{T}\left(\frac{\partial L_t}{\partial W_t}\right)^2 + \epsilon}}\frac{\partial L}{\partial W}$$

$n$ 為前面所有梯度值的平方和，利用前面學習的梯度值平方和來調整learning rate ，$\epsilon$ 為平滑值，加上 $\epsilon$ 的原因是為了不讓分母為 $0$，$\epsilon$ 一般值為 $1e-8$。前期梯度較小的時候，$n$ 較小，能夠放大學習率。後期梯度較大的時候，$n$ 較大，能夠約束學習率，但分母上梯度平方的累加會越來越大，會使梯度趨近於 $0$，訓練便會結束 (但其實還沒走到 optimal)，為了防止這個情況，後面有開發出 RMSprop。

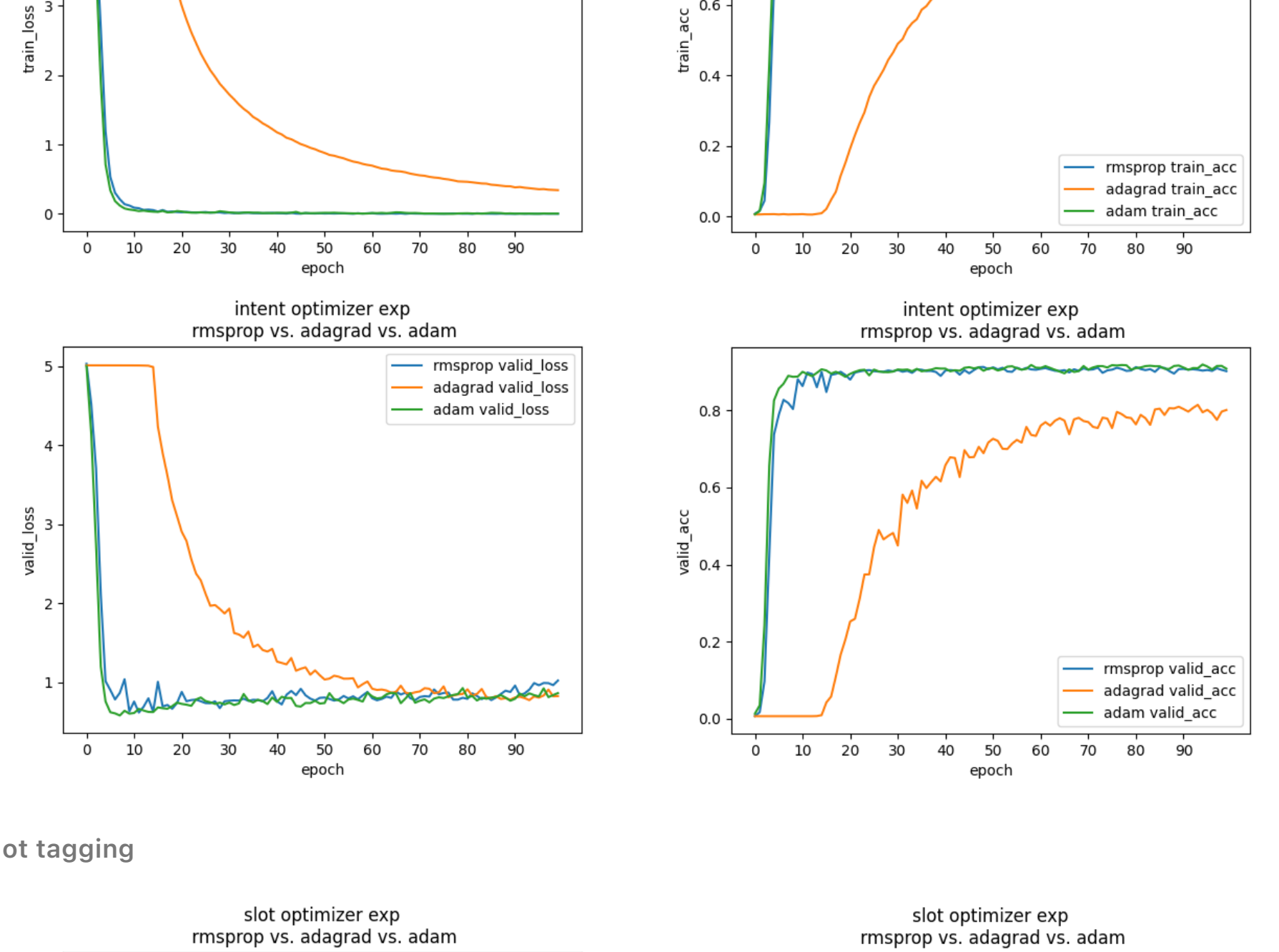
RMSprop 能夠有效改善 AdaGrad 提前結束訓練的問題。並且適合處理複雜的、non-convex 的 error surface。它在 learning rate 調整上面多了一個參數 $\alpha$，可以在新舊梯度上面做調節：

\(W_{t+1} \leftarrow W_t - \dfrac{\eta}{\sigma_t}\dfrac{\partial L_t}{\partial W_t}\)
\(\sigma_t=\sqrt{\alpha(\sigma_{t-1}) +(1-\alpha)(\frac{\partial L_t}{\partial W_t})^2 \epsilon}\)

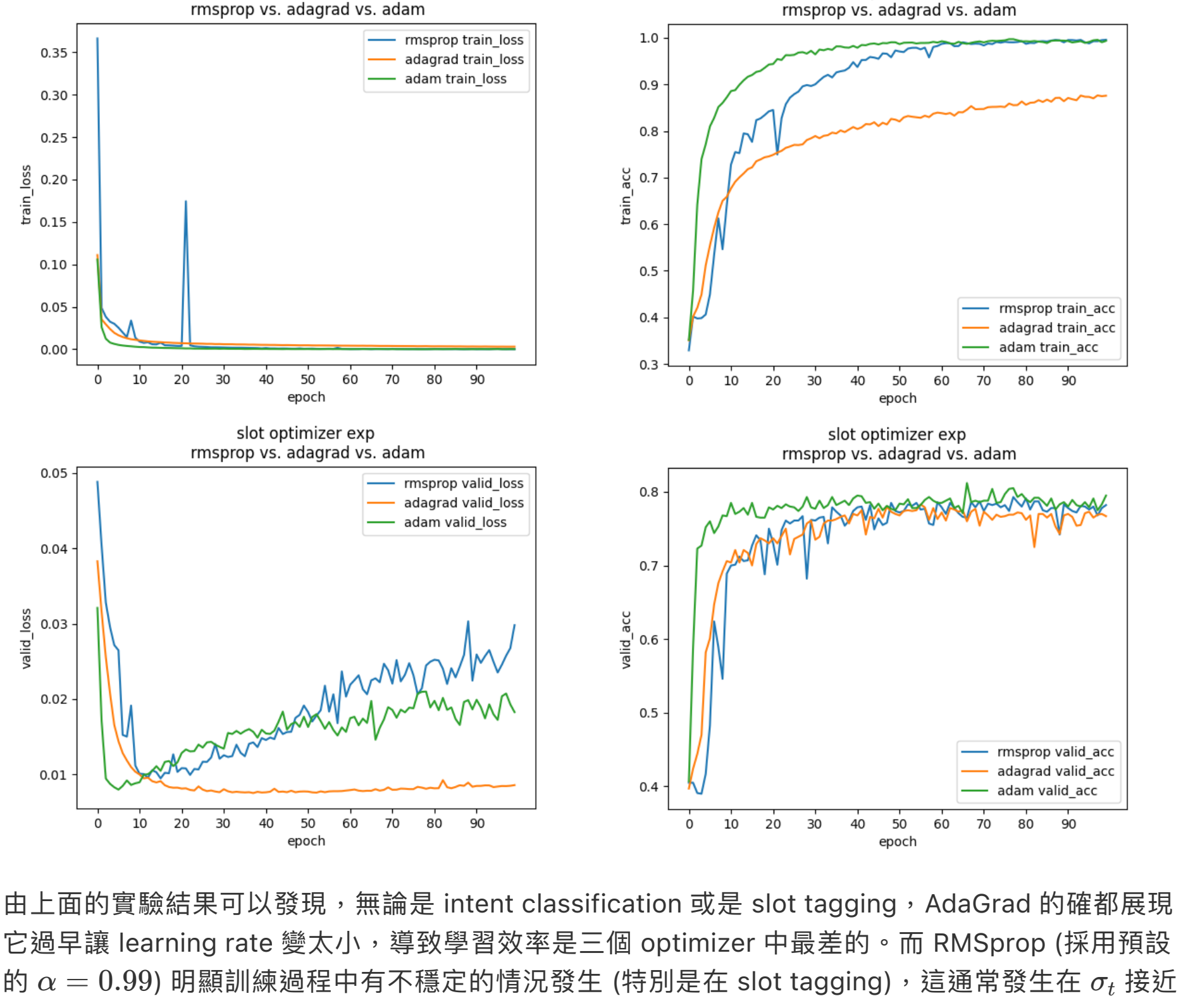
若 \(\alpha\) 較大，表示在過程中更倚重舊梯度的資訊。

Adam 則其實就是加入了動量概念的 RMSprop，且在更新 gradient 過程中考慮了 bias-correction，由於公式相當繁複，就不在此贅述。總之 Adam 結合了 AdaGrad、RMSprop 及 Momentum 的優點，對所有不同的參數都有新舊之間的權衡調節，各種狀況均適用，是目前較為主流的 optimizer。

##### intent classification



##### slot tagging



由上面的實驗結果可以發現，無論是 intent classification 或是 slot tagging，AdaGrad 的確都展現它過早讓 learning rate 變太小，導致學習效率是三個 optimizer 中最差的。而 RMSprop (採用預設的 $\alpha = 0.99$) 明顯訓練過程中有不穩定的情況發生 (特別是在 slot tagging)，這通常發生在 $\sigma_t$ 接近 $0$ 時使得 learngin rate 修正後爆炸。這樣看起來三個 optimzier 中，Adam 的確展現出穩定、學習速度快的優點，可說是最佳選擇。AdaGrad 則是學習效率過慢；RMSprop 則是不夠穩定。