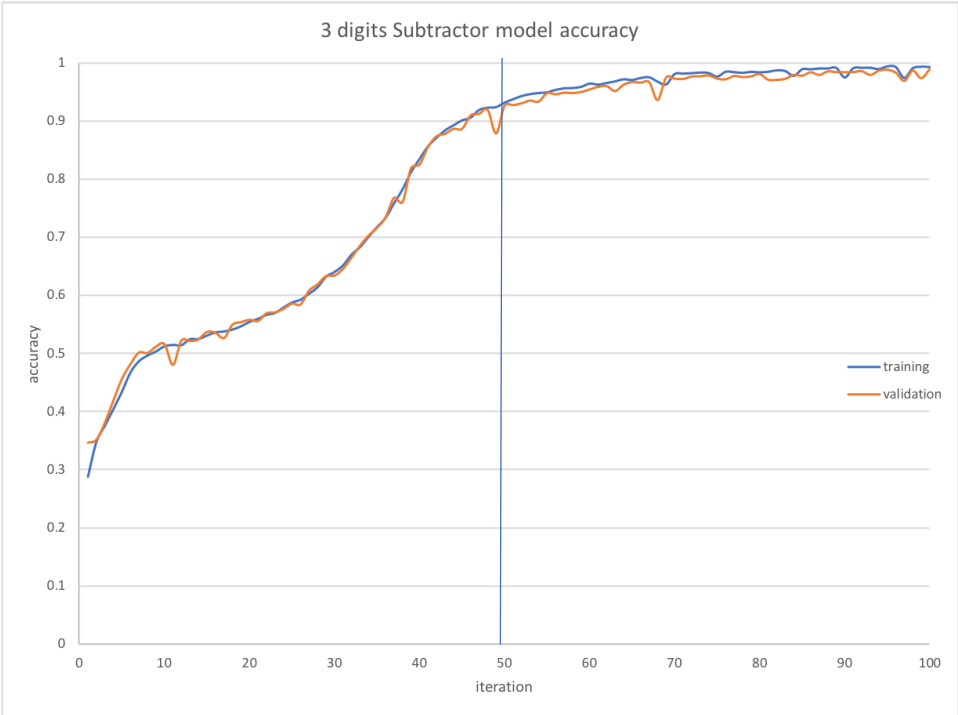
Experiment 1:

Iteration=100, training size=18000, validation size=2000, testing size=60000

1. If the number of digits is less, it is faster when training. That is 3 digits is faster than 4 digits, since the accuracy grows more rapidly in 3 digits (at about 50th iteration, the accuracy becomes higher than 0.9 and converge early. But in 4 digits, it need 80 iterations to get 0.9 accuracy in training stage). In contrast, loss function value decreases more rapidly in 3 digits than 4 digits.
2. Training dataset v.s. validation dataset: because model is fitted by the training data, so the training lines are usually performing slightly better than validation lines. We can see this result by both accuracy and loss. (training data's accuracy is higher and loss function value is lower.)
3. Training time in each epoch: in 3 digits, it need 6s 334us and 8s 418us in 4 digits for each epoch when training. The method apply to the subtractor is supervised learning with one-hot code as the label. The combination of 4 digits – 4 digits is more than 3 digits – 3 digits, so it takes more time to propagate and update parameter in the model.
4. Accuracy for the testing data: after training 100 iteration, the accuracy of 3 digits=0.84 and 4 digits=0.81. For the same reason above, the combination of 3 digits is less than 4 digits. As result, for the same testing size, 3 digits performs better.

Fig1. accuracy for 3/4 digits subtractor:



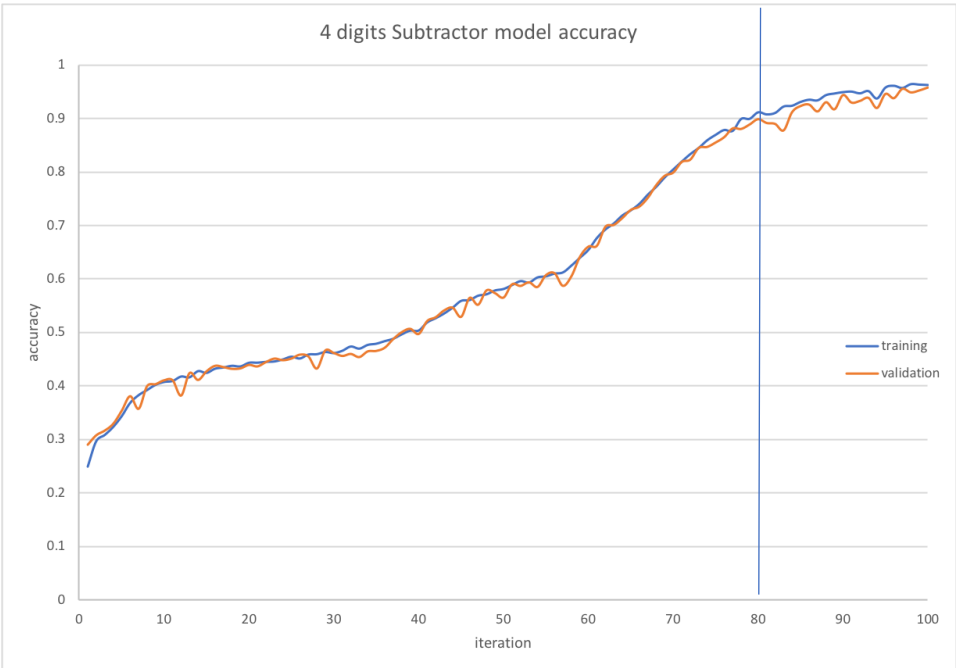
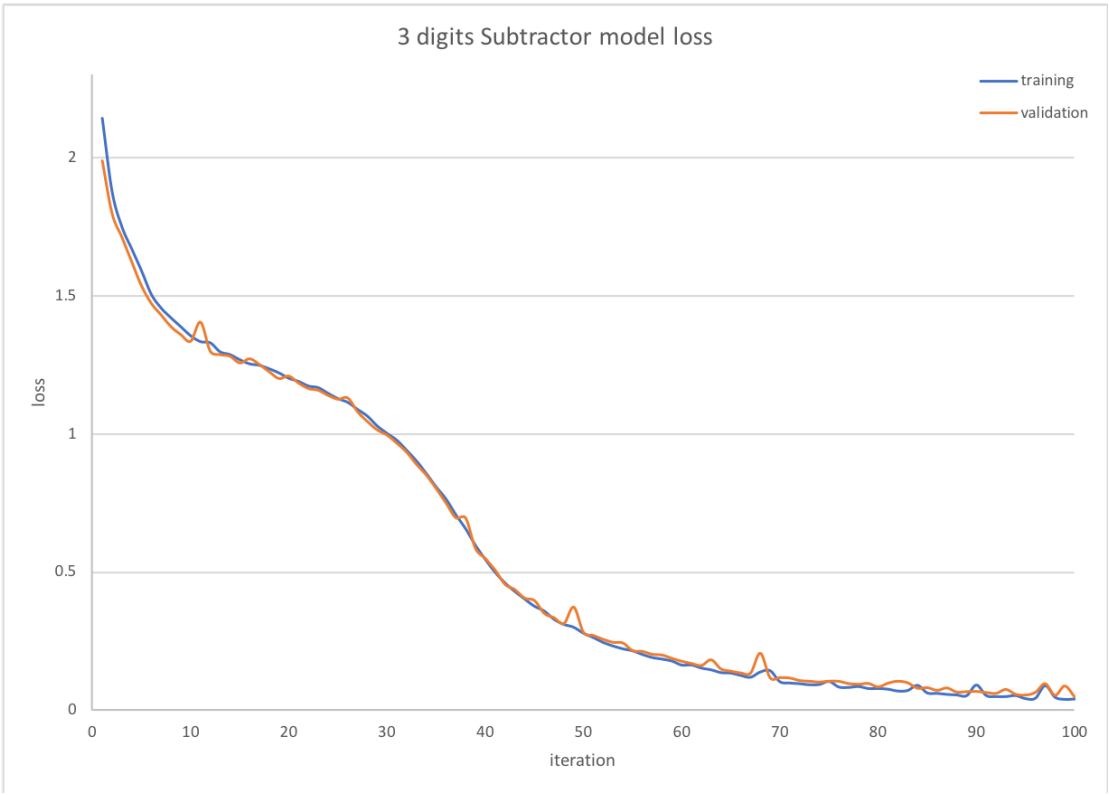
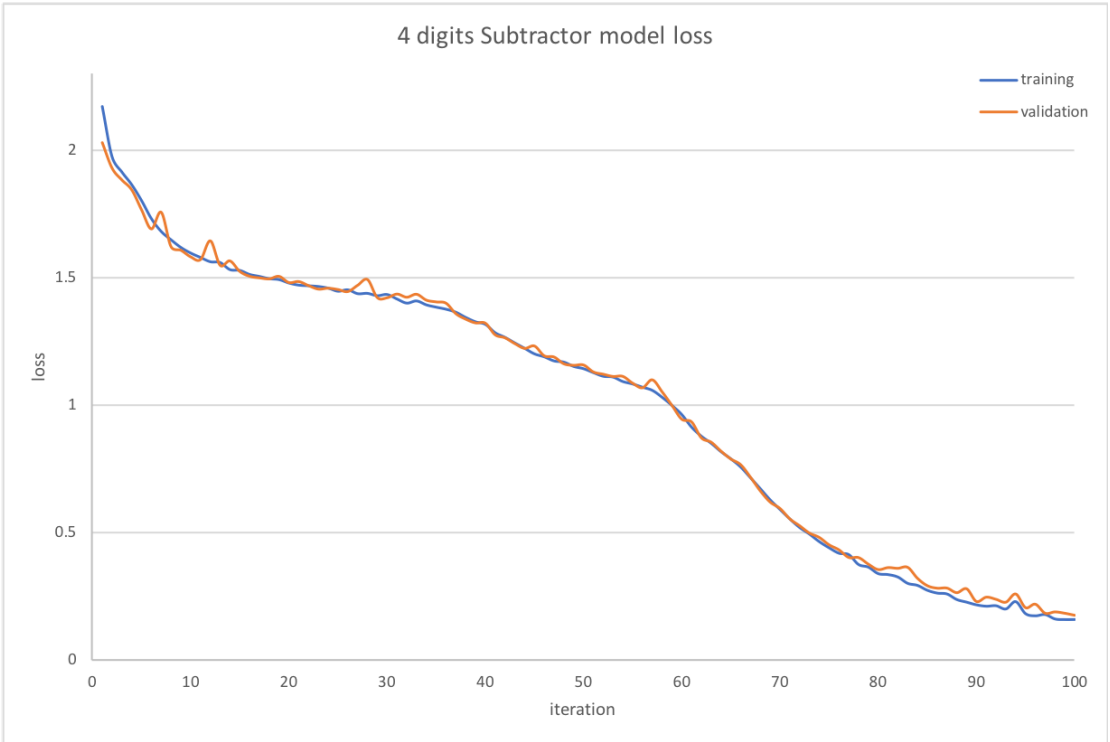


Fig2. loss for 3/4 digits subtractor





Experiment 2: 3 digits testing data in different iteration in training process

The plot shows that testing data grows slowly in the early period, then increase the accuracy after 50th iteration. Over fitting doesn't appear in the model, as training and validation line converge to 1.0 in accuracy, but testing line still growing.

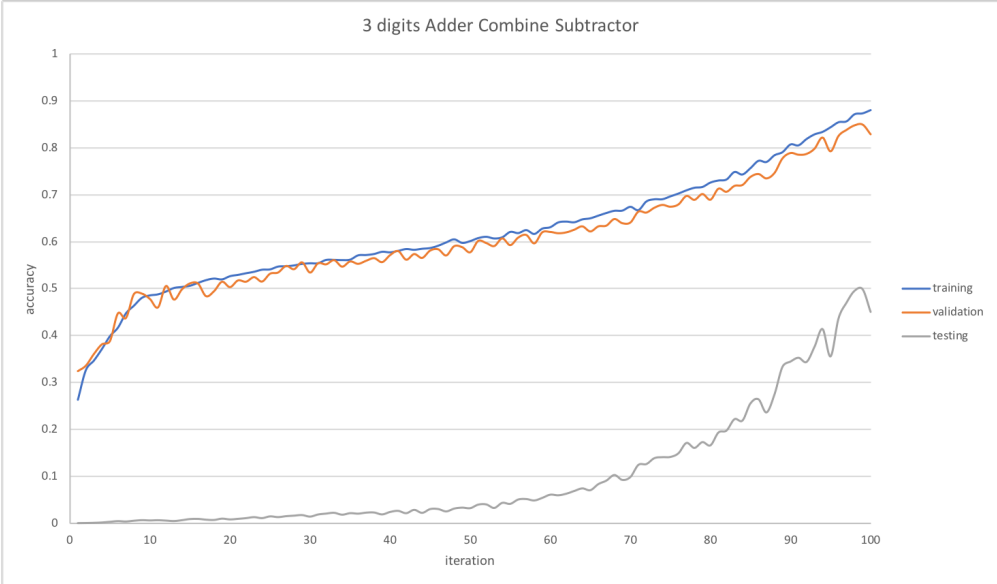
After the 100th iteration, all of the results are almost converging to accuracy=1.0(upper bound), so the model can be trained within 100 iterations and generate pretty good result.

Experiment 3: Combine + and -:

Screenshot

```
Iteration 91
Train on 18000 samples, validate on 2000 samples
Epoch 1/1
18000/18000 [==============================] - 6s 326us/step - loss: 0.5106 - acc: 0.8186
 - val_loss: 0.5582 - val_acc: 0.7865
logs
{'val_loss': [0.5581964101791382], 'val_acc': [0.7864999990463257], 'loss': [0.5105643385
251363], 'acc': [0.818597222328186]}
Q 387-820 T -433 ☒ -432
Q 750-511 T 239  ☒ 229
Q 974+303 T 1277 ☑ 1277
Q 289-585 T -296 ☒ -297
Q 519+922 T 1441 ☑ 1441
Q 558+359 T 917  ☒ 996
Q 678+225 T 903  ☒ 893
Q 938-315 T 623  ☑ 623
Q 224-958 T -734 ☒ -735
Q 488-506 T -18  ☒ -28
MSG : Accuracy is 0.3438333333333333
```

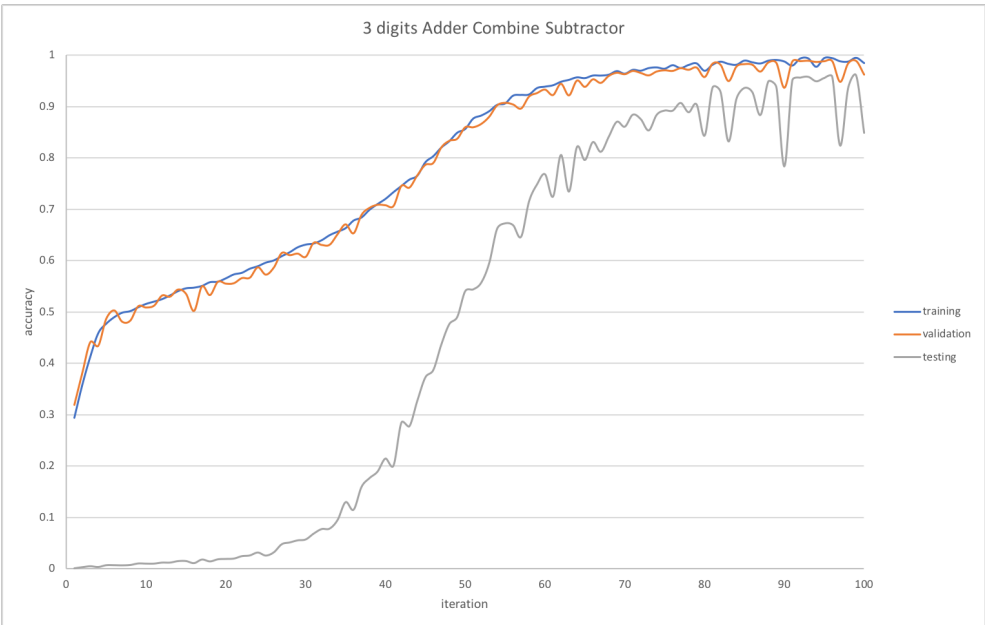
Iteration=100, training size=18000, validation size=2000, testing size=60000



As the screenshot shows, we train + and – together in the same model and use the same data size as we do in the only + model and – model.

Until 100 iteration, the best performance of training and validation line still cannot reach 0.9, worse than the +model and -model. For testing line, it grows slowly before 50th and then becomes faster. Although +model and -model can almost reach 0.98 in finally, +/-model can just get accuracy=0.5 in the best. The reason is that training size may be too small for + and -, so try to double training size:

Iteration=100, training size=36000, validation size=4000, testing size=40000



When training size doubled, it has twice better performance in the same training iteration. Also, there is more uncertainty in +/- model, so the line obviously oscillates

in training process.

Can we apply the same training approach for multiplication?

It is possible to train multiplier using this method, but it is more challenging. Because there are 5 or 6 digits for the answer, respectively for the minimum and maximum case (100*100, 999*999), and it has wider range for the answer, so the model need more iteration or training data size to get better performance on the testing size. Because the model uses NN with one-hot code, the model learns the relationship within the digits with correspond operator in fact and just mapping the combination and update the parameter in the model by supervised learning, rather than knows the true meaning of +, -, *. As result, if the training size is large enough to enumerate all possible expression, we can apply the same approach.