Fully-connected NN and “**Dropout**”

1. 使用 CIFAR10 準備 Training Dataset 和 Testing Dataset，shape 如下

```
Train data shape:  torch.Size([40000, 3072])
Train labels shape:  torch.Size([40000])
Validation data shape:  torch.Size([10000, 3072])
Validation labels shape:  torch.Size([10000])
Test data shape:  torch.Size([10000, 3072])
Test labels shape:  torch.Size([10000])
```

Linear Layer

測試 Linear Layer Forward 跟 Backward 的 relative error 是否小於 1e-7

Linear Layer：Forward

```
Testing Linear.forward function:
difference:  3.683042917976506e-08
```

Linear Layer：Backward

Relative error 應約略為 1e-10 附近

```
Testing Linear.backward function:
dx error:  5.221943563709987e-10
dw error:  3.498388787266994e-10
db error:  5.373171200544344e-10
```

ReLU activation

ReLU activation：Forward

```
Testing ReLU.forward function:
difference:  4.5454545613554664e-09
```

Relative error 應小於 **1e-7**，有達成。

ReLU activation：Backward

```
Testing ReLU.backward function:
dx error:  2.6317796097761553e-10
```

Relative error 應小於 **1e-8**，有達成。

“Sandwich” Layer

以線性層搭配 **ReLU** 層為一常見的方法來實作神經網路，會先經由線性層的 **forward** 函數計算後傳至 **ReLU** 層的 **forward** 函數進行激活，再由 **ReLU** 的 **backward** 函數計算後傳至線性層的 **backward** 函數 **check gradient numerically**，應小於 **1e-8**，有達成。

```
Testing Linear_ReLU.forward and Linear_ReLU.backward:
dx error:  1.2107596995452444e-09
dw error:  7.462948482161807e-10
db error:  8.915028842081707e-10
```

Loss Layer：Softmax and SVM

藉由二 **loss function** 來進行 **numerically gradient checking**，驗證實作，**SVM** 的 **error** 應小於 **1e-7**，**softmax** 則是 **1e-6**，皆有達成。

```
Testing svm_loss:
loss:  9.000430792478463
dx error:  2.0074935157654598e-08

Testing softmax_loss:
loss:  2.3026286102347924
dx error:  1.0417990899757076e-07
```

Two-Layer Network

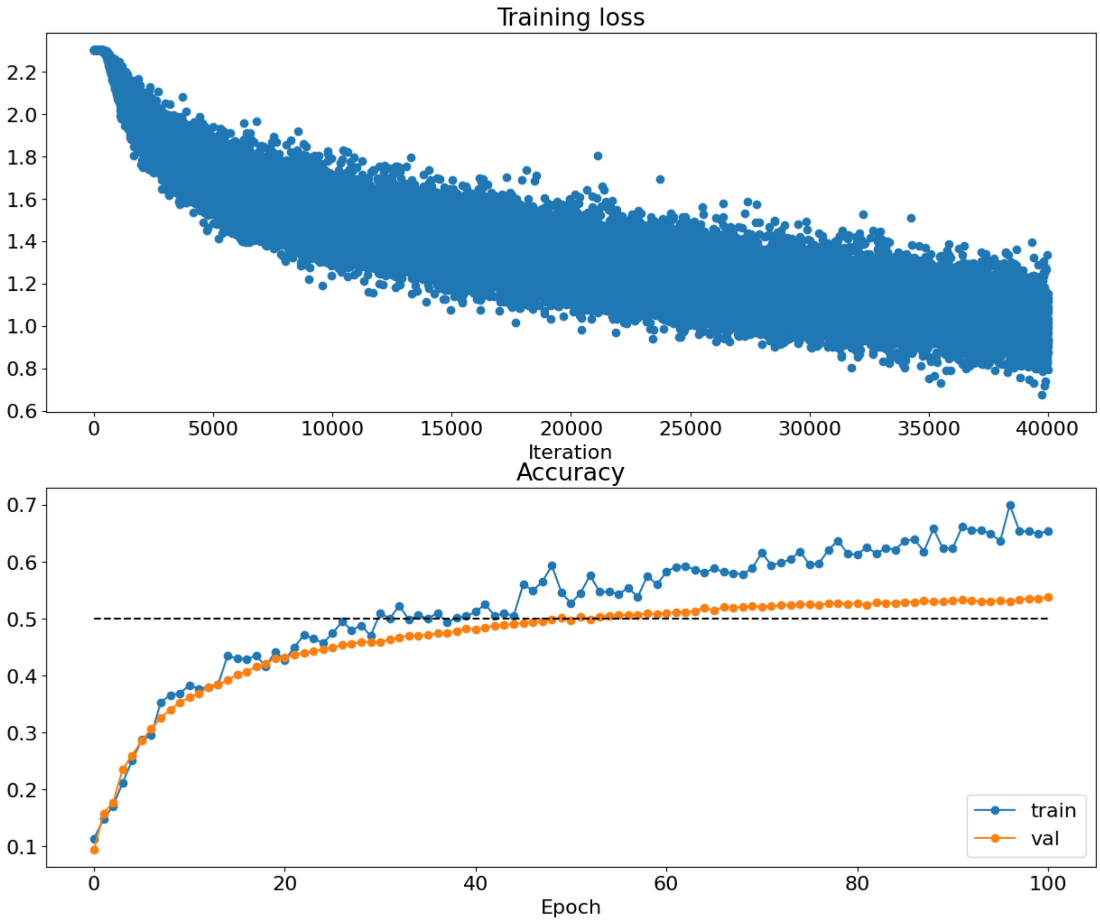
依據不同的正規化參數實作兩層的神經網路，各參數誤差應低於 **1e-6**，皆有達成。

```
Testing initialization ...
Testing test-time forward pass ...
Testing training loss (no regularization)
Running numeric gradient check with reg =  0.0
/content/drive/My Drive/Colab Notebooks/fully_connected_networks.py:120:
  dx = dout*torch.tensor(x > 0, dtype=dout.dtype, device=dout.device)
W1 relative error: 2.94e-07
W2 relative error: 1.65e-09
b1 relative error: 1.01e-06
b2 relative error: 4.63e-09
Running numeric gradient check with reg =  0.7
W1 relative error: 2.70e-08
W2 relative error: 9.86e-09
b1 relative error: 2.28e-06
b2 relative error: 2.90e-08
```

Solver

利用 **Solver** 實體來實作兩層的神經網路，自行設定 **num_epochs = 100**，並希望在驗證集能達到 **50%**的準確率，有達成**(53.78%)**。

```
(Time 197.72 sec; Iteration 39881 / 40000) loss: 0.998507
(Time 197.77 sec; Iteration 39891 / 40000) loss: 1.028019
(Time 197.82 sec; Iteration 39901 / 40000) loss: 1.132077
(Time 197.87 sec; Iteration 39911 / 40000) loss: 0.971259
(Time 197.92 sec; Iteration 39921 / 40000) loss: 1.031292
(Time 197.97 sec; Iteration 39931 / 40000) loss: 0.913660
(Time 198.03 sec; Iteration 39941 / 40000) loss: 1.089694
(Time 198.08 sec; Iteration 39951 / 40000) loss: 1.059240
(Time 198.14 sec; Iteration 39961 / 40000) loss: 0.831488
(Time 198.19 sec; Iteration 39971 / 40000) loss: 1.115215
(Time 198.25 sec; Iteration 39981 / 40000) loss: 0.953088
(Time 198.30 sec; Iteration 39991 / 40000) loss: 0.935250
(Epoch 100 / 100) train acc: 0.654000; val_acc: 0.537800
```



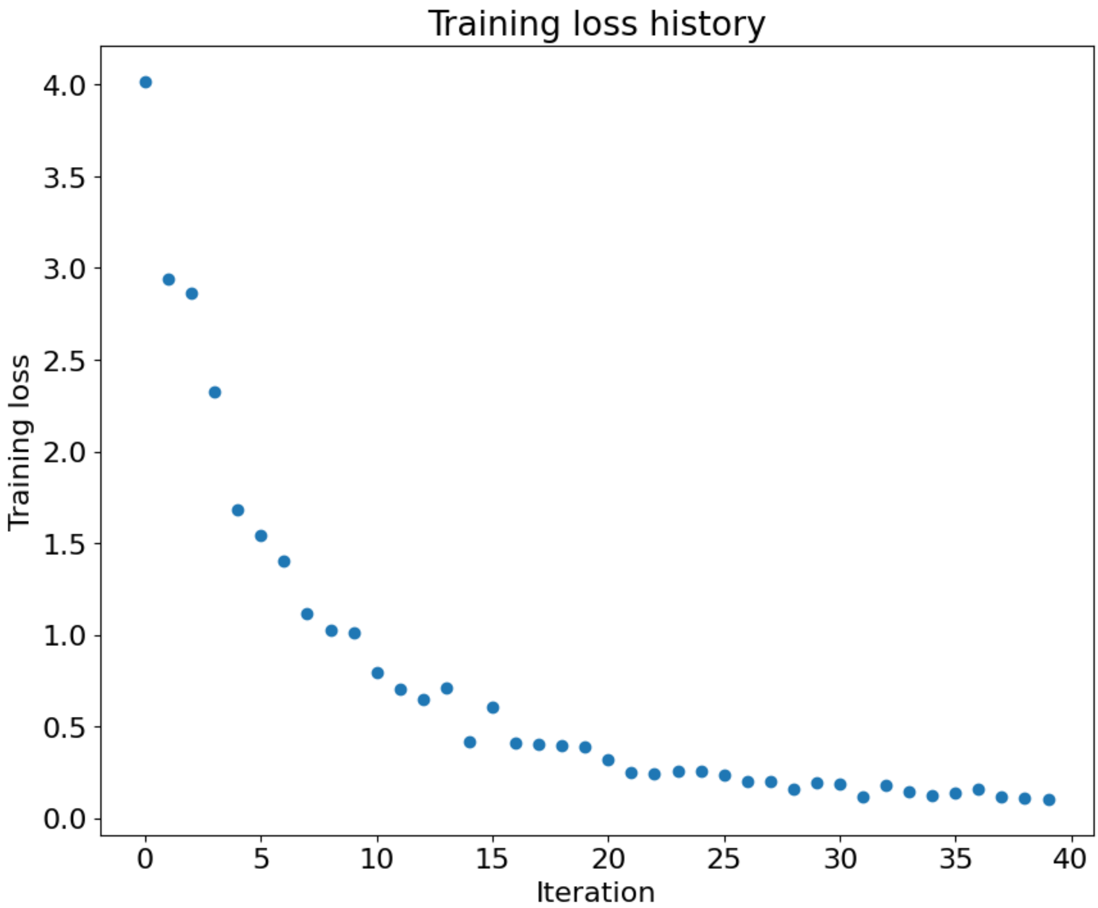
Multilayer Network

在有無正規化的情況下分別訓練 **Fully-Connected Network**，有正規化的情況下 **relative error** 應小於 **1e-6**；無正規化的情況下 **relative error** 應小於 **1e-5**，皆有達成。

```
Running check with reg =  0
Initial loss:  2.3053575717037686
W1 relative error: 6.06e-08
W2 relative error: 1.02e-07
W3 relative error: 5.89e-08
b1 relative error: 1.28e-07
b2 relative error: 2.05e-08
b3 relative error: 3.41e-09
Running check with reg =  3.14
Initial loss:  12.278358041494133
W1 relative error: 5.60e-09
W2 relative error: 8.54e-09
W3 relative error: 1.27e-08
b1 relative error: 5.76e-07
b2 relative error: 1.46e-07
b3 relative error: 1.53e-08
```

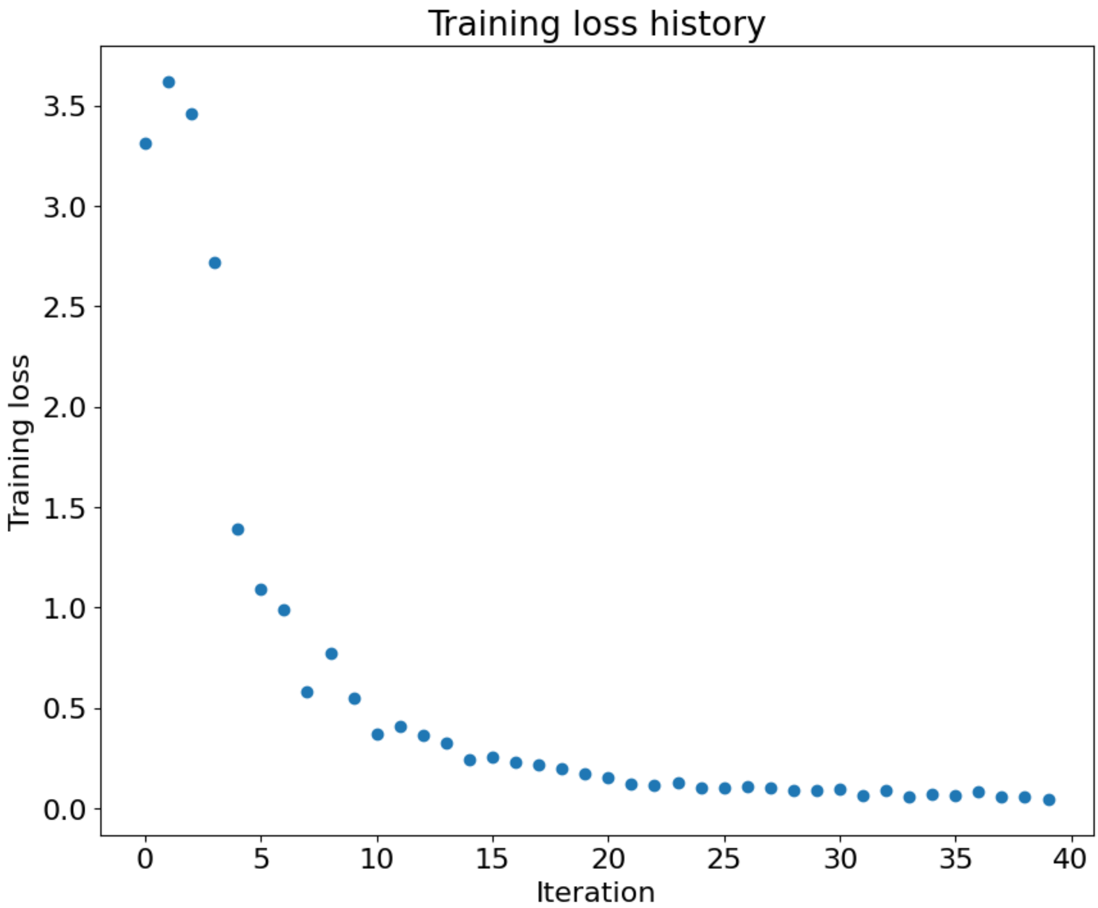
為了更詳盡的完整性檢查，我們先透過 **3** 層的 **network** 來測試若每層 **hidden layer** 皆設置 **100** 個單位，**weight scale** 設置 **0.15**，**learning rate** 給予 **0.05**，是否能在 **50** 個小樣本裡面達到 **overfitting**。

```
(Epoch 3 / 20) train acc: 0.580000; val_acc: 0.131600
(Epoch 4 / 20) train acc: 0.700000; val_acc: 0.145100
(Epoch 5 / 20) train acc: 0.800000; val_acc: 0.147800
(Time 0.31 sec; Iteration 11 / 40) loss: 0.796434
(Epoch 6 / 20) train acc: 0.900000; val_acc: 0.145700
(Epoch 7 / 20) train acc: 0.920000; val_acc: 0.151300
(Epoch 8 / 20) train acc: 0.940000; val_acc: 0.154500
(Epoch 9 / 20) train acc: 0.960000; val_acc: 0.154700
(Epoch 10 / 20) train acc: 0.960000; val_acc: 0.152900
(Time 0.45 sec; Iteration 21 / 40) loss: 0.320513
(Epoch 11 / 20) train acc: 1.000000; val_acc: 0.160800
(Epoch 12 / 20) train acc: 1.000000; val_acc: 0.156600
(Epoch 13 / 20) train acc: 0.980000; val_acc: 0.160000
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.161700
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.160300
(Time 0.59 sec; Iteration 31 / 40) loss: 0.189558
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.158400
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.159700
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.161200
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.162900
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.161700
```



接著是 5 層(weight scale 給予 0.15、learning rate 給予 0.05)：

```
(Epoch 2 / 20) train acc: 0.540000; val_acc: 0.118700
(Epoch 3 / 20) train acc: 0.780000; val_acc: 0.135300
(Epoch 4 / 20) train acc: 0.940000; val_acc: 0.141700
(Epoch 5 / 20) train acc: 0.980000; val_acc: 0.144500
(Time 0.28 sec; Iteration 11 / 40) loss: 0.372073
(Epoch 6 / 20) train acc: 0.980000; val_acc: 0.147100
(Epoch 7 / 20) train acc: 1.000000; val_acc: 0.150700
(Epoch 8 / 20) train acc: 1.000000; val_acc: 0.156400
(Epoch 9 / 20) train acc: 1.000000; val_acc: 0.154000
(Epoch 10 / 20) train acc: 1.000000; val_acc: 0.152000
(Time 0.51 sec; Iteration 21 / 40) loss: 0.150553
(Epoch 11 / 20) train acc: 1.000000; val_acc: 0.155100
(Epoch 12 / 20) train acc: 1.000000; val_acc: 0.155400
(Epoch 13 / 20) train acc: 1.000000; val_acc: 0.157200
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.156500
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.155100
(Time 0.74 sec; Iteration 31 / 40) loss: 0.096860
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.158600
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.158500
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.159000
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.159500
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.160300
```



SGD + Momentum

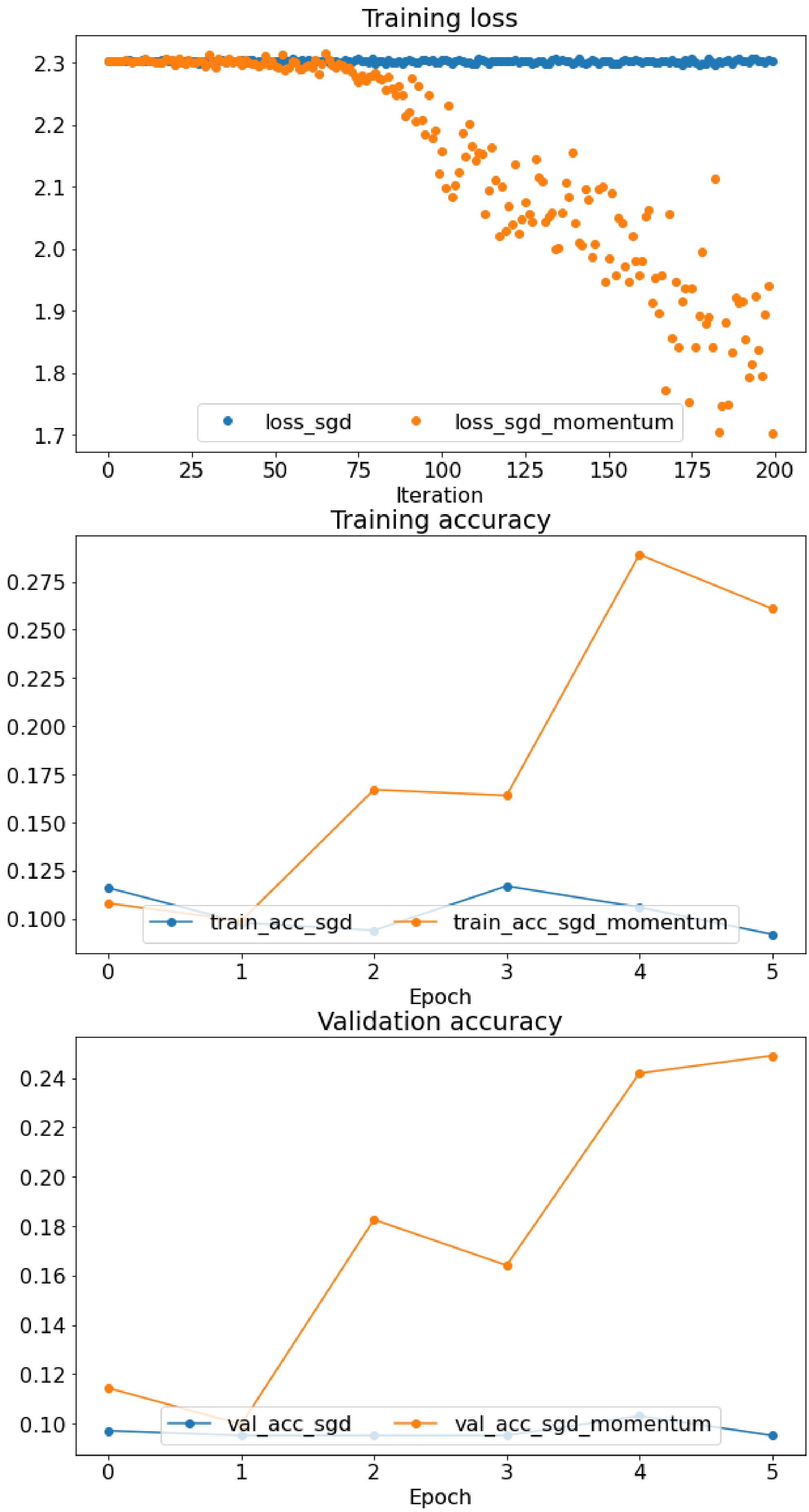
除了以 **SGD** 做為更新 **weight** 的方式以外，我們也常加入 **Momentum** 的方式更新我們的 **weight**，理想的 **relative error** 應低於 **1e-8**，有達成。

```
next_w error:  1.6802078709310813e-09
velocity error:  2.9254212825785614e-09
```

同時用 **SGD** 和 **SGD+Momentum** 的方式分別訓練 **six-layer network**，由圖形結果我們可明顯看出後者在降低 **loss** 的速率快很多

```
running with  sgd
(Time 0.00 sec; Iteration 1 / 200) loss: 2.302603
(Epoch 0 / 5) train acc: 0.116000; val_acc: 0.097100
(Epoch 1 / 5) train acc: 0.098000; val_acc: 0.095200
(Epoch 2 / 5) train acc: 0.094000; val_acc: 0.095200
(Epoch 3 / 5) train acc: 0.117000; val_acc: 0.095200
(Epoch 4 / 5) train acc: 0.106000; val_acc: 0.103100
(Epoch 5 / 5) train acc: 0.092000; val_acc: 0.095200

running with  sgd_momentum
(Time 0.00 sec; Iteration 1 / 200) loss: 2.302174
(Epoch 0 / 5) train acc: 0.108000; val_acc: 0.114300
(Epoch 1 / 5) train acc: 0.099000; val_acc: 0.099700
(Epoch 2 / 5) train acc: 0.167000; val_acc: 0.182600
(Epoch 3 / 5) train acc: 0.164000; val_acc: 0.164100
(Epoch 4 / 5) train acc: 0.289000; val_acc: 0.241900
(Epoch 5 / 5) train acc: 0.261000; val_acc: 0.249000
```


Training loss
2.3
2.2
2.1
2.0
1.9
1.8
1.7
loss_sgd
loss_sgd_momentum
0
25
50
75
100
125
150
175
200
Iteration
Training accuracy
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
train_acc_sgd
train_acc_sgd_momentum
0
1
2
3
4
5
Epoch
Validation accuracy
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
val_acc_sgd
val_acc_sgd_momentum
0
1
2
3
4
5
Epoch

RMSprop

$$\sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

理想的 **relative error** 應低於 **1e-6**，有達成。

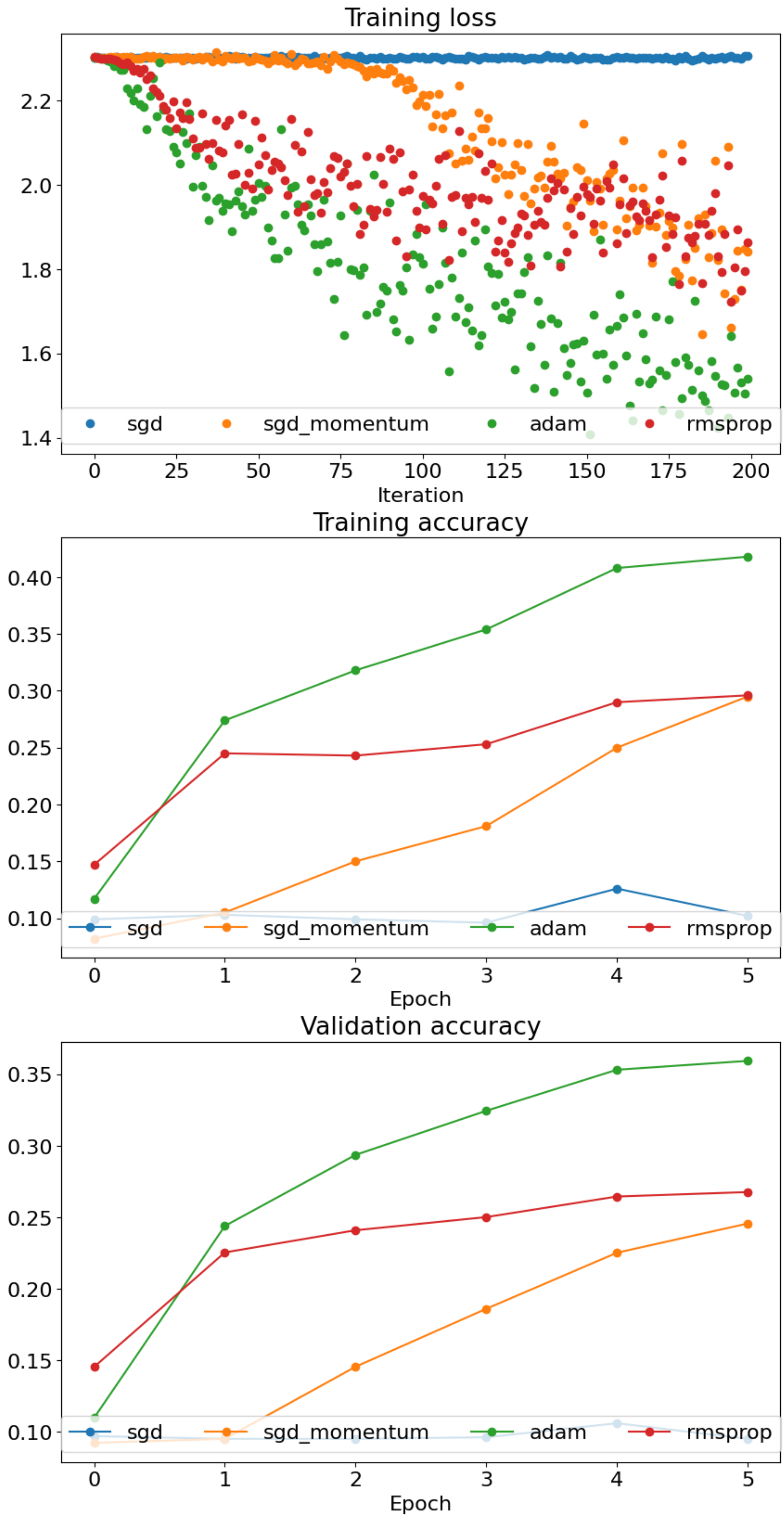
```
next_w error:  4.064797880829826e-09
cache error:  1.8620321382570356e-09
```

Adam

理想的 **next_w relative error** 應低於 **1e-6**，理想的 **v** 和 **m relative error** 應低於 **1e-8**，有達成。

```
next_w error:  3.756728297598868e-09
v error:  3.4048987160545265e-09
m error:  2.786377729853651e-09
```

比較使用 **SGD**、**SGD+Momentum**、**RMSProp**、**Adam** 的 **Training Loss**、**Training Accuracy** 以及 **Validation Accuracy**，可明顯看出 **Adam** 的表現最好。


Training loss
2.2
2.0
1.8
1.6
1.4
sgd
sgd_momentum
adam
rmsprop
0
25
50
75
100
125
150
175
200
Iteration
Training accuracy
0.40
0.35
0.30
0.25
0.20
0.15
0.10
sgd
sgd_momentum
adam
rmsprop
0
1
2
3
4
5
Epoch
Validation accuracy
0.35
0.30
0.25
0.20
0.15
0.10
sgd
sgd_momentum
adam
rmsprop
0
1
2
3
4
5
Epoch

Dropout

Regularizing neural networks by randomly setting some output activations to zero during the forward pass

Dropout：Forward

```
Running tests with p =  0.25
Mean of input:  9.997330335850453
Mean of train-time output:  9.989851559328836
Mean of test-time output:  9.997330335850453
Fraction of train-time output set to zero:  0.2505599856376648
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.4
Mean of input:  9.997330335850453
Mean of train-time output:  9.973639826710299
Mean of test-time output:  9.997330335850453
Fraction of train-time output set to zero:  0.40133199095726013
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.7
Mean of input:  9.997330335850453
Mean of train-time output:  10.007508905208196
Mean of test-time output:  9.997330335850453
Fraction of train-time output set to zero:  0.6997119784355164
Fraction of test-time output set to zero:  0.0
```

Dropout：Backward

numerically gradient-check

```
dx relative error:  3.914942325636866e-09
```

Fully-connected nets with dropout

Error 應小於 1e-5

```
Running check with dropout =  0
Initial loss:  2.3053575717037686
W1 relative error: 6.06e-08
W2 relative error: 1.02e-07
W3 relative error: 5.89e-08
b1 relative error: 1.28e-07
b2 relative error: 2.05e-08
b3 relative error: 3.41e-09

Running check with dropout =  0.25
Initial loss:  2.304579158010139
W1 relative error: 3.29e-08
W2 relative error: 5.84e-08
W3 relative error: 3.43e-08
b1 relative error: 1.16e-07
b2 relative error: 1.62e-08
b3 relative error: 3.88e-09

Running check with dropout =  0.5
Initial loss:  2.2843557967595594
W1 relative error: 9.21e-09
W2 relative error: 1.66e-08
W3 relative error: 1.40e-08
b1 relative error: 2.95e-08
b2 relative error: 1.62e-08
b3 relative error: 2.78e-09
```

Regularization Experiment

為了瞭解 **dropout** 如何正規化神經網路，我們對以下三種雙層神經網路進行訓練

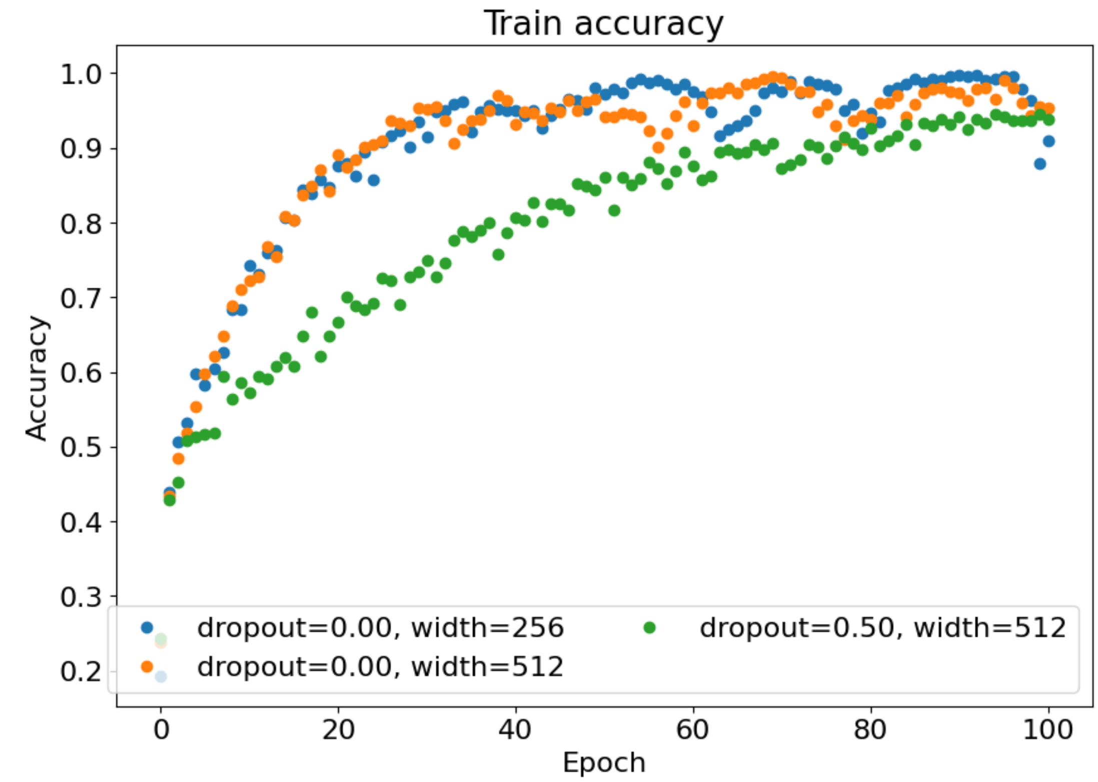
1. Hidden size 256, dropout = 0
2. Hidden size 512, dropout = 0
3. Hidden size 512, dropout = 0.5

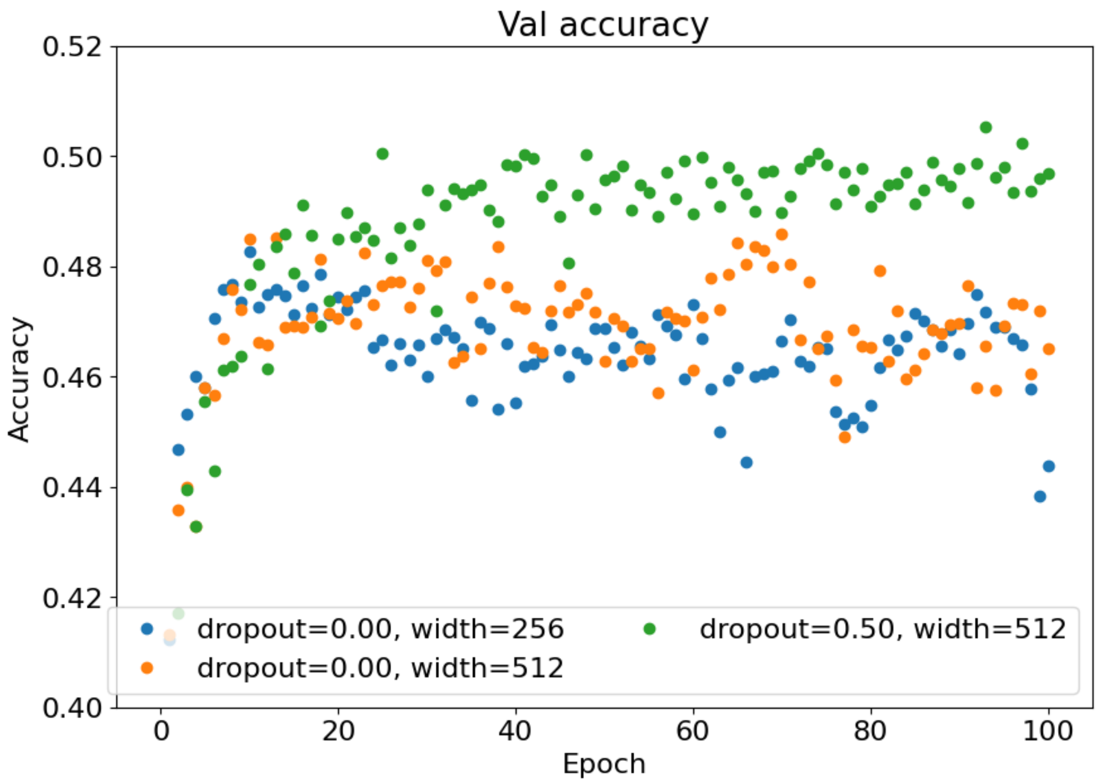
```
Training a model with dropout=0.00 and width=256
(Time 0.01 sec; Iteration 1 / 3900) loss: 2.304467
(Epoch 0 / 100) train acc: 0.193000; val_acc: 0.198200
(Epoch 10 / 100) train acc: 0.742000; val_acc: 0.482600
(Epoch 20 / 100) train acc: 0.876000; val_acc: 0.474400
(Epoch 30 / 100) train acc: 0.914000; val_acc: 0.460100
(Epoch 40 / 100) train acc: 0.950000; val_acc: 0.455200
(Epoch 50 / 100) train acc: 0.972000; val_acc: 0.468700
(Epoch 60 / 100) train acc: 0.976000; val_acc: 0.473000
(Epoch 70 / 100) train acc: 0.975000; val_acc: 0.466500
(Epoch 80 / 100) train acc: 0.947000; val_acc: 0.454900
(Epoch 90 / 100) train acc: 0.998000; val_acc: 0.464100
(Epoch 100 / 100) train acc: 0.910000; val_acc: 0.443900
```

```
Training a model with dropout=0.00 and width=512
(Time 0.00 sec; Iteration 1 / 3900) loss: 2.302387
(Epoch 0 / 100) train acc: 0.239000; val_acc: 0.220000
(Epoch 10 / 100) train acc: 0.723000; val_acc: 0.484900
(Epoch 20 / 100) train acc: 0.891000; val_acc: 0.470500
(Epoch 30 / 100) train acc: 0.951000; val_acc: 0.481100
(Epoch 40 / 100) train acc: 0.931000; val_acc: 0.472800
(Epoch 50 / 100) train acc: 0.941000; val_acc: 0.462900
(Epoch 60 / 100) train acc: 0.930000; val_acc: 0.461200
(Epoch 70 / 100) train acc: 0.994000; val_acc: 0.485900
(Epoch 80 / 100) train acc: 0.939000; val_acc: 0.465400
(Epoch 90 / 100) train acc: 0.973000; val_acc: 0.469700
(Epoch 100 / 100) train acc: 0.953000; val_acc: 0.465100
```

```
Training a model with dropout=0.50 and width=512
(Time 0.01 sec; Iteration 1 / 3900) loss: 2.304556
(Epoch 0 / 100) train acc: 0.244000; val_acc: 0.235500
(Epoch 10 / 100) train acc: 0.572000; val_acc: 0.476800
(Epoch 20 / 100) train acc: 0.666000; val_acc: 0.485000
(Epoch 30 / 100) train acc: 0.750000; val_acc: 0.493900
(Epoch 40 / 100) train acc: 0.806000; val_acc: 0.498300
(Epoch 50 / 100) train acc: 0.861000; val_acc: 0.495800
(Epoch 60 / 100) train acc: 0.876000; val_acc: 0.489600
(Epoch 70 / 100) train acc: 0.872000; val_acc: 0.489700
(Epoch 80 / 100) train acc: 0.926000; val_acc: 0.490800
(Epoch 90 / 100) train acc: 0.941000; val_acc: 0.497700
(Epoch 100 / 100) train acc: 0.938000; val_acc: 0.496900
```

由上圖可得知，在同樣 **width = 512** 的情況下，如果有 **dropout**，雖然會犧牲一些訓練的準確性，但是在驗證集的表現則是比較好的。我們以視覺化方式呈現最終分別在訓練集和驗證集的表現結果。


Train accuracy
Accuracy
Epoch
dropout=0.00, width=256
dropout=0.00, width=512
dropout=0.50, width=512


Val accuracy
Accuracy
Epoch
dropout=0.00, width=256
dropout=0.00, width=512
dropout=0.50, width=512