深度學習

HW4

學號 : B103012002

姓名 ： 林凡皓

I. SVM Classifier :

為了方便與 softmax loss classifier 做比較，SVM classifier 的部分我仍然有執行並將結果貼上。

- **svm_loss_naive :**

1. 解題思路 ：

(1) dW : 根據下圖的公式推導結果，將 dW 完成。

1° $\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0) \cdot x_i$

$$\nabla_{w_j} L_i = \frac{\partial L_i}{\partial w_j^T x_i} \frac{\partial w_j^T x_i}{\partial w_j}$$

$$\text{for } j \neq y_i,\ j = 1 \Rightarrow \frac{\partial L_1}{\partial w_1^T x_1} = \frac{\partial w_1^T x_i - w_{y_i}^T x_1 + 1}{\partial w_1^T x_i} = 1(w_1^T x_i - w_{y_i}^T x_i + 1 > 0)$$

$$j = 2 \Rightarrow \frac{\partial L_2}{\partial w_2^T x_i} = \frac{\partial w_2^T x_i - w_{y_i}^T x_i + 1}{\partial w_2^T x_i} = 1(w_2^T x_i - w_{y_i}^T x_i + 1 > 0)$$

$$\therefore \frac{\partial L_i}{\partial w_j^T x_i} = 1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0) \quad \text{and} \quad \frac{\partial w_j^T x_i}{\partial w_j} = x_i$$

$$\Rightarrow \nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0) \cdot x_i \quad ※$$

2° $\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0)\right) \cdot x_i$

$$\nabla_{w_{y_i}} L_i = \frac{\partial L_i}{\partial w_{y_i}^T x_i} \frac{\partial w_{y_i}^T x_i}{\partial w_{y_i}^T}$$

$$\frac{\partial L_i}{\partial w_{y_i}^T x_i} = \frac{\partial \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + 1)}{\partial w_{y_i}^T x_i} = \sum_{j \neq y_i} -1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0)$$

$$\frac{\partial w_{y_i}^T x_i}{\partial w_{y_i}^T} = x_i$$

$$\therefore \nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0)\right) \cdot x_i \quad ※$$

(2) loss : 先分別計算出估計出來的分數以及真實答案的分數，接著可以利用 continue 來將 j = yi 時跳過。計算出(估計分數 － 真實分數 ＋1)並判斷此數值是否大於 0，如果是就將 loss 加上計算出來的數值，否則 loss 不變。最後再將 loss 除以訓練樣本數以及加上 regularization term 即可。

2. 執行結果 ：

(1) Loss check :

```
loss: 9.000869
```

(2) gradient check :

without regularization term :

```
numerical: 0.031599 analytic: 0.031599, relative error: 3.887711e-07
numerical: 0.111444 analytic: 0.111444, relative error: 1.603834e-07
numerical: 0.011204 analytic: 0.011204, relative error: 1.003052e-06
numerical: -0.046128 analytic: -0.046128, relative error: 1.470228e-08
numerical: 0.071948 analytic: 0.071948, relative error: 1.000117e-07
numerical: 0.025688 analytic: 0.025688, relative error: 1.407617e-08
numerical: 0.185388 analytic: 0.185388, relative error: 4.086995e-08
numerical: -0.021740 analytic: -0.021740, relative error: 7.159385e-08
numerical: -0.159613 analytic: -0.159613, relative error: 9.199232e-08
numerical: 0.092690 analytic: 0.092690, relative error: 6.470382e-08
```

With regularization term :

```
numerical: 0.124849 analytic: 0.124849, relative error: 7.976456e-08
numerical: 0.168915 analytic: 0.168915, relative error: 9.920512e-08
numerical: 0.148752 analytic: 0.148752, relative error: 5.747575e-08
numerical: -0.024936 analytic: -0.024936, relative error: 6.470254e-08
numerical: -0.008570 analytic: -0.008570, relative error: 7.174549e-07
numerical: -0.103155 analytic: -0.103155, relative error: 3.462148e-08
numerical: -0.335573 analytic: -0.335573, relative error: 2.199511e-08
numerical: -0.222176 analytic: -0.222176, relative error: 1.731537e-08
numerical: 0.681163 analytic: 0.681163, relative error: 1.887528e-08
numerical: -0.004089 analytic: -0.004089, relative error: 1.101669e-06
```

- **svm_loss_vectorized :**

1. 解題思路 ：

(1) Loss： 先透過矩陣乘法取得所有類別的分數，以及對所有類別分數做 index operation 來取得正確類別的分數。接著透過 scores - correct_class_score + 1 來計算出每個類別的 loss，並將正確類別的 loss 設定成 0。最後將 loss 們相加、除以訓練樣本數並加上 regularization term。

(2) dW： 這題的整體概念與 svm_loss_navie 的 dW 計算一樣，都是將數學推導的結果寫成程式碼。

先創建一個 mask，將 margin 中大於 0 的數值設為 1，以及正確類別的分數設為該 column 的和。接著透過 torch.mm 將訓練後結果 X 和 mask 做矩陣乘法。最後再除以訓練樣本數以及加上 regularization term。

2. 執行結果：

(1) Loss :

```
Naive loss: 9.002106e+00 computed in 495.54ms
Vectorized loss: 9.002106e+00 computed in 5.00ms
Difference: -1.78e-15
Speedup: 99.17X
```

由上圖可以看到，有沒有使用 for loop 計算出來的 loss 是差不多的，但是透過 vectorization 可以讓運算速度快上 99.17 倍。

(2) Gradient :

```
Naive loss and gradient: computed in 483.28ms
Vectorized loss and gradient: computed in 5.00ms
Gradient difference: 1.82e-14
Speedup: 96.70X
```

由上圖可以看到，有沒有使用 for loop 計算出來的 gradient 是差不多的，但是透過 vectorization 可以讓運算速度快上 96.7 倍。

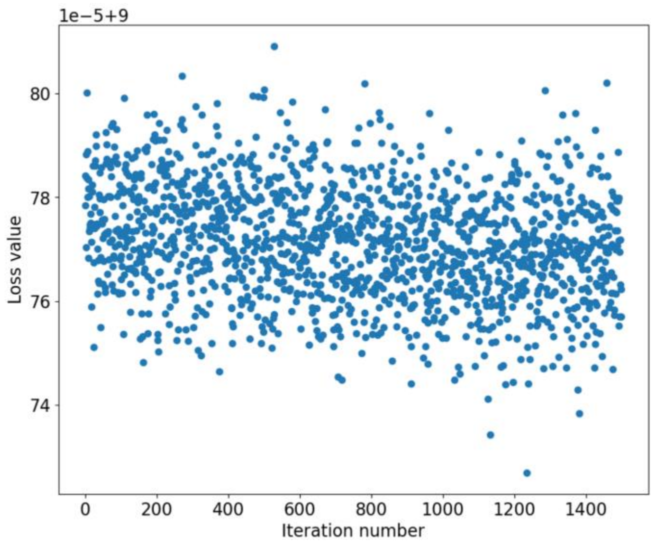
- **sample_batch :**

1. 解題思路：先利用 torch.randint 創建數值從 0 到訓練樣本數的 index(shape = (batch_size,))，然後透過 index 從 X 和 y 中抓取 batch。

- **train_linear_classifier :**

1. 解題思路：本題就是在做 training 的步驟。首先要初始化權重 W，接著在每一次迭代去計算 loss 的 gradient，並根據計算出的 gradient 去更新權重。
2. 執行結果：

```
iteration 0 / 1500: loss 9.000784
iteration 100 / 1500: loss 9.000764
iteration 200 / 1500: loss 9.000777
iteration 300 / 1500: loss 9.000768
iteration 400 / 1500: loss 9.000779
iteration 500 / 1500: loss 9.000771
iteration 600 / 1500: loss 9.000771
iteration 700 / 1500: loss 9.000769
iteration 800 / 1500: loss 9.000771
iteration 900 / 1500: loss 9.000771
iteration 1000 / 1500: loss 9.000771
iteration 1100 / 1500: loss 9.000789
iteration 1200 / 1500: loss 9.000787
iteration 1300 / 1500: loss 9.000769
iteration 1400 / 1500: loss 9.000777
That took 2.385910s
```



- **Predict_linear_classifier :**
  1. 解題思路 ： 本題要透過先前訓練好的 W 來做預測。首先要透過矩陣相乘計算 W*X，並從中選擇最大值的 index 即為預測結果。
  2. 執行結果 ：

```
Training accuracy: 9.35%
Validation accuracy: 9.11%
```

- **Get_search_params :**
  1. 解題思路 ： 本題要創建 learning_rate list 跟 regularization_strength list，分別用來存放要嘗試的 learning rate 和 regularization strength。因此只需要利用 list 的創建方法並在其中存放要嘗試的值即可。
- **Test_one_param_set :**
  1. 解題思路 ： 本題要根據 get_search_params 中的參數去計算訓練準

確度與驗證準確度。由於方才訓練的模型屬於 LinearClassifier 這個 class，因此要得到預測結果我們可以透過.predict 來得到。有了預測結果後，只要將預測正確的數量除以總數量即可得到準確度。

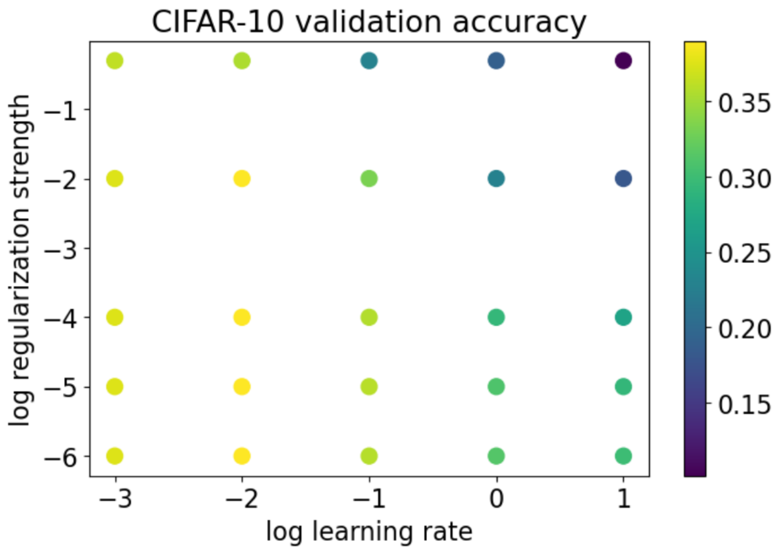
2. 執行結果：

```
Training SVM 1 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-06
Training SVM 2 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-05
Training SVM 3 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-04
Training SVM 4 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-02
Training SVM 5 / 25 with learning_rate=1.000000e-03 and reg=5.000000e-01
Training SVM 6 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-06
Training SVM 7 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-05
Training SVM 8 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-04
Training SVM 9 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-02
Training SVM 10 / 25 with learning_rate=1.000000e-02 and reg=5.000000e-01
Training SVM 11 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-06
Training SVM 12 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-05
Training SVM 13 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-04
Training SVM 14 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-02
Training SVM 15 / 25 with learning_rate=1.000000e-01 and reg=5.000000e-01
Training SVM 16 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-06
Training SVM 17 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-05
Training SVM 18 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-04
Training SVM 19 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-02
Training SVM 20 / 25 with learning_rate=1.000000e+00 and reg=5.000000e-01
Training SVM 21 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-06
Training SVM 22 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-05
Training SVM 23 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-04
Training SVM 24 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-02
Training SVM 25 / 25 with learning_rate=1.000000e+01 and reg=5.000000e-01
lr 1.000000e-03 reg 1.000000e-06 train accuracy: 0.388700 val accuracy: 0.374800
lr 1.000000e-03 reg 1.000000e-05 train accuracy: 0.388725 val accuracy: 0.374600
lr 1.000000e-03 reg 1.000000e-04 train accuracy: 0.388650 val accuracy: 0.375000
lr 1.000000e-03 reg 1.000000e-02 train accuracy: 0.388450 val accuracy: 0.374900
lr 1.000000e-03 reg 5.000000e-01 train accuracy: 0.376400 val accuracy: 0.362700
lr 1.000000e-02 reg 1.000000e-06 train accuracy: 0.414400 val accuracy: 0.390200
lr 1.000000e-02 reg 1.000000e-05 train accuracy: 0.413725 val accuracy: 0.389500
lr 1.000000e-02 reg 1.000000e-04 train accuracy: 0.413725 val accuracy: 0.389800
lr 1.000000e-02 reg 1.000000e-02 train accuracy: 0.410350 val accuracy: 0.389900
lr 1.000000e-02 reg 5.000000e-01 train accuracy: 0.362150 val accuracy: 0.355000
lr 1.000000e-01 reg 1.000000e-06 train accuracy: 0.401100 val accuracy: 0.358000
lr 1.000000e-01 reg 1.000000e-05 train accuracy: 0.401600 val accuracy: 0.358800
lr 1.000000e-01 reg 1.000000e-04 train accuracy: 0.399000 val accuracy: 0.357300
lr 1.000000e-01 reg 1.000000e-02 train accuracy: 0.355675 val accuracy: 0.331600
lr 1.000000e-01 reg 5.000000e-01 train accuracy: 0.230575 val accuracy: 0.229600
lr 1.000000e+00 reg 1.000000e-06 train accuracy: 0.346950 val accuracy: 0.313000
lr 1.000000e+00 reg 1.000000e-05 train accuracy: 0.341325 val accuracy: 0.309200
lr 1.000000e+00 reg 1.000000e-04 train accuracy: 0.325775 val accuracy: 0.293300
lr 1.000000e+00 reg 1.000000e-02 train accuracy: 0.232850 val accuracy: 0.226700
lr 1.000000e+00 reg 5.000000e-01 train accuracy: 0.183650 val accuracy: 0.188800
lr 1.000000e+01 reg 1.000000e-06 train accuracy: 0.329900 val accuracy: 0.299400
lr 1.000000e+01 reg 1.000000e-05 train accuracy: 0.324975 val accuracy: 0.292200
lr 1.000000e+01 reg 1.000000e-04 train accuracy: 0.287775 val accuracy: 0.268800
lr 1.000000e+01 reg 1.000000e-02 train accuracy: 0.178925 val accuracy: 0.179600
lr 1.000000e+01 reg 5.000000e-01 train accuracy: 0.099650 val accuracy: 0.101400
best validation accuracy achieved during cross-validation: 0.390200
```

由上圖可以看到最好的結果發生在 learning rate = 0.01，regularization strength = 0.000001。最好的結果為 39.02 %。

會有每一次的迭代並不是因為 test_one_param_set 中有 for loop，而是我們用 for loop 多次呼叫 test_one_param。

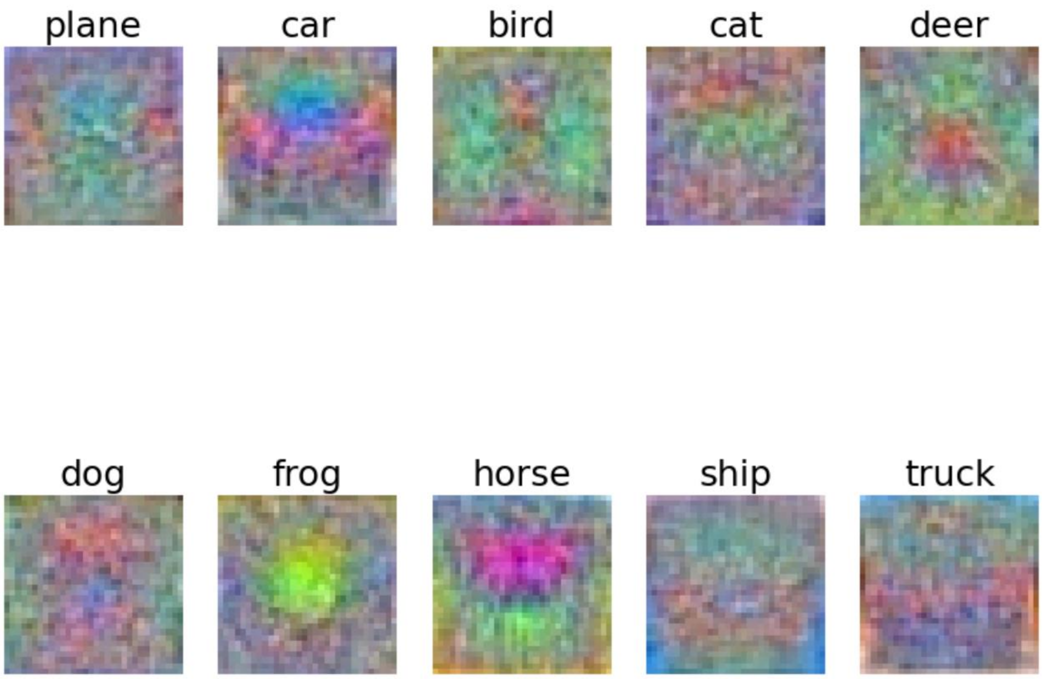
將結果視覺化：



對於測試資料來說，此模型有 38.71 %的準確度，如下圖，

```
linear SVM on raw pixels final test set accuracy: 0.387100
```

- **每個類別的模板 ：**



II. Softmax Classifier

- **Softmax_loss_naive :**

 1. 解題思路 ：

 (1) dW： 根據以下推導，將 dW 完成。

$$\begin{cases} P_{y_i} = \dfrac{e^{s_{y_i}}}{\sum_j e^{s_j}} \\ s_j = w_j^T x_j \end{cases} \qquad \frac{\partial P_{y_i}}{\partial s_j} = \begin{cases} P_{y_i}(1-P_j), & y_i = j \\ -P_{y_i}P_j, & y_i \neq j \end{cases}$$

$$\nabla_{w_j} L_i = \frac{\partial L_i}{\partial s_j} \frac{\partial s_j}{\partial w_j}$$

$$1^\circ \ \frac{\partial L_i}{\partial s_j} = \frac{\partial(-\log P_{y_i})}{\partial s_j} = -\frac{1}{P_{y_i}} \frac{\partial P_{y_i}}{\partial s_j} = \begin{cases} (P_j - 1), & y_i = j \\ P_j, & y_i \neq j \end{cases}$$

$$2^\circ \ \frac{\partial s_j}{\partial w_j} = x_j$$

$$\therefore \nabla_{w_j} L_i = \begin{cases} (P_j - 1)x_j, & y_i = j \\ P_j x_j, & y_i \neq j \end{cases}$$

(2) loss：先計算出得分，可透過 torch.mv 計算 W 與 X[i]的矩陣與向量乘法，i 為 for loop 迭代的參數，代表第幾個 training sample。為了符合矩陣乘法的規則，W 需要做轉置。接著利用 torch.exp 以及$P_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$將得分轉換為機率，並對此機率取$-\log P_i$即可得到 loss。

2. 執行結果：

(1) 檢查 loss

```
loss: 2.302797
sanity check: 2.302585
```

(2) 檢查 dW

```
numerical: 0.003046 analytic: 0.003046, relative error: 1.400934e-07
numerical: 0.006309 analytic: 0.006309, relative error: 3.253210e-07
numerical: 0.005390 analytic: 0.005390, relative error: 1.917530e-07
numerical: 0.002580 analytic: 0.002580, relative error: 4.473513e-07
numerical: 0.007512 analytic: 0.007512, relative error: 7.381182e-07
numerical: 0.006417 analytic: 0.006417, relative error: 9.004100e-08
numerical: 0.011390 analytic: 0.011390, relative error: 2.041099e-07
numerical: 0.001821 analytic: 0.001821, relative error: 1.074840e-06
numerical: -0.014710 analytic: -0.014710, relative error: 3.854999e-07
numerical: -0.005154 analytic: -0.005154, relative error: 9.740901e-07
```

(3) 檢查考慮 regularization term 之後的 dW

```
numerical: 0.004915 analytic: 0.004915, relative error: 2.079737e-07
numerical: 0.005887 analytic: 0.005887, relative error: 2.384724e-07
numerical: 0.006309 analytic: 0.006309, relative error: 2.699530e-07
numerical: 0.001580 analytic: 0.001580, relative error: 3.311530e-07
numerical: 0.005839 analytic: 0.005839, relative error: 2.359337e-07
numerical: 0.006800 analytic: 0.006800, relative error: 3.908514e-07
numerical: 0.011466 analytic: 0.011466, relative error: 2.197269e-07
numerical: 0.002314 analytic: 0.002314, relative error: 1.239312e-06
numerical: -0.016812 analytic: -0.016812, relative error: 1.622449e-07
numerical: -0.006673 analytic: -0.006673, relative error: 1.370005e-07
```

- **Softmax_loss_vectorized :**

  1. 解題思路 ：

     (1) loss : 這次不使用 for loop 去對每一個 training sample 跌代，而是透過 broadcasting 以及 torch.mm 直接對 W 和 X 做矩陣乘法。其餘步驟皆與 softmax_loss_naive 相同，差別只是 vectorized 透過 broadcasting 直接做矩陣運算。

     (2) dW : 計算方式如同 softmax_loss_naïve，差別在於 vectorized 透過 broadcasting 直接做矩陣運算。

  2. 執行結果 ：

     (1) 比較 for loop 與 vectorized 之間的表現差異 ：

     ```
     naive loss: 2.302812e+00 computed in 406.000853s
     vectorized loss: 2.302812e+00 computed in 9.532213s
     Loss difference: 4.44e-16
     Gradient difference: 3.26e-16
     Speedup: 42.59X
     ```

     由結果可以看出，計算出來的 loss 與 dW 差異很小，但是速度上卻差了 42.59 倍。

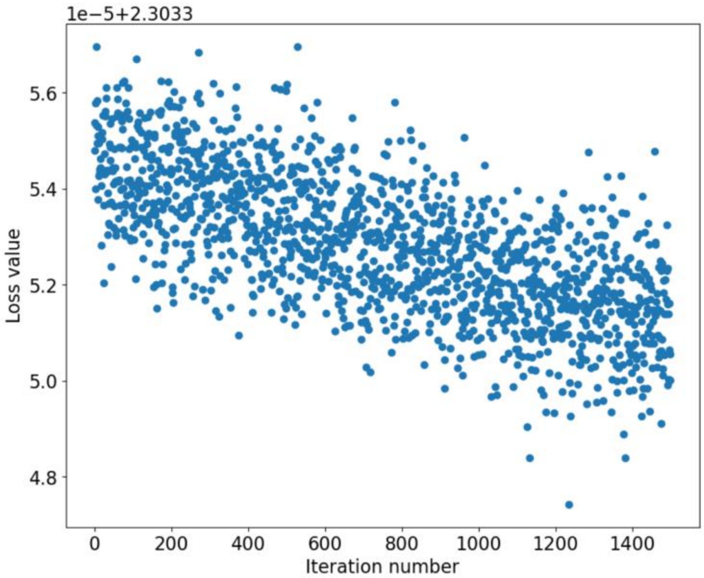
     (2) 檢查結果為 numeric stable :

     ```
     iteration 0 / 1: loss 768250002.302585
     iteration 0 / 1: loss 768250002.302585
     ```

- **Train_linear_classifier :**

  此函數與訓練 SVM loss 時相同。

```
iteration 0 / 1500: loss 2.303355
iteration 100 / 1500: loss 2.303353
iteration 200 / 1500: loss 2.303354
iteration 300 / 1500: loss 2.303353
iteration 400 / 1500: loss 2.303354
iteration 500 / 1500: loss 2.303353
iteration 600 / 1500: loss 2.303353
iteration 700 / 1500: loss 2.303353
iteration 800 / 1500: loss 2.303353
iteration 900 / 1500: loss 2.303353
iteration 1000 / 1500: loss 2.303352
iteration 1100 / 1500: loss 2.303354
iteration 1200 / 1500: loss 2.303354
iteration 1300 / 1500: loss 2.303352
iteration 1400 / 1500: loss 2.303352
That took 4.264805s
```



由此結果與 SVM loss 比較會發現到 softmax 的 loss 會比較大。對於這樣的現象，我認為並不是說 SVM loss 計算出來的值就會比 softmax 的值小，而是要考慮到更多的東西，像是資料分布。SVM loss 的值與 softmax 的值有所不同是合理的，畢竟兩個的計算公式就不相同，至於誰大誰小，我認為會根據不同的情況有所差別。

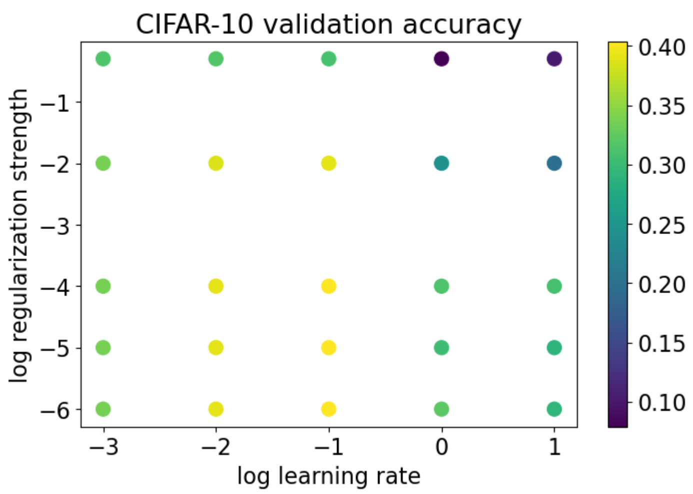
- **Predict_linear_classifier :**

```
training accuracy: 8.88%
validation accuracy: 8.42%
```

- **Training result :**

1. Training :

```
Training Softmax 1 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-06
Training Softmax 2 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-05
Training Softmax 3 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-04
Training Softmax 4 / 25 with learning_rate=1.000000e-03 and reg=1.000000e-02
Training Softmax 5 / 25 with learning_rate=1.000000e-03 and reg=5.000000e-01
Training Softmax 6 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-06
Training Softmax 7 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-05
Training Softmax 8 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-04
Training Softmax 9 / 25 with learning_rate=1.000000e-02 and reg=1.000000e-02
Training Softmax 10 / 25 with learning_rate=1.000000e-02 and reg=5.000000e-01
Training Softmax 11 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-06
Training Softmax 12 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-05
Training Softmax 13 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-04
Training Softmax 14 / 25 with learning_rate=1.000000e-01 and reg=1.000000e-02
Training Softmax 15 / 25 with learning_rate=1.000000e-01 and reg=5.000000e-01
Training Softmax 16 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-06
Training Softmax 17 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-05
Training Softmax 18 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-04
Training Softmax 19 / 25 with learning_rate=1.000000e+00 and reg=1.000000e-02
Training Softmax 20 / 25 with learning_rate=1.000000e+00 and reg=5.000000e-01
Training Softmax 21 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-06
Training Softmax 22 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-05
Training Softmax 23 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-04
Training Softmax 24 / 25 with learning_rate=1.000000e+01 and reg=1.000000e-02
Training Softmax 25 / 25 with learning_rate=1.000000e+01 and reg=5.000000e-01
lr 1.000000e-03 reg 1.000000e-06 train accuracy: 0.343225 val accuracy: 0.336500
lr 1.000000e-03 reg 1.000000e-05 train accuracy: 0.343225 val accuracy: 0.336500
lr 1.000000e-03 reg 1.000000e-04 train accuracy: 0.343225 val accuracy: 0.336500
lr 1.000000e-03 reg 1.000000e-02 train accuracy: 0.342600 val accuracy: 0.336200
lr 1.000000e-03 reg 5.000000e-01 train accuracy: 0.314500 val accuracy: 0.313900
lr 1.000000e-02 reg 1.000000e-06 train accuracy: 0.406625 val accuracy: 0.390700
lr 1.000000e-02 reg 1.000000e-05 train accuracy: 0.406625 val accuracy: 0.390700
lr 1.000000e-02 reg 1.000000e-04 train accuracy: 0.406675 val accuracy: 0.390700
lr 1.000000e-02 reg 1.000000e-02 train accuracy: 0.403650 val accuracy: 0.385500
lr 1.000000e-02 reg 5.000000e-01 train accuracy: 0.316525 val accuracy: 0.315300
lr 1.000000e-01 reg 1.000000e-06 train accuracy: 0.435125 val accuracy: 0.403800
lr 1.000000e-01 reg 1.000000e-05 train accuracy: 0.435000 val accuracy: 0.403900
lr 1.000000e-01 reg 1.000000e-04 train accuracy: 0.434725 val accuracy: 0.403400
lr 1.000000e-01 reg 1.000000e-02 train accuracy: 0.410050 val accuracy: 0.390700
lr 1.000000e-01 reg 5.000000e-01 train accuracy: 0.311975 val accuracy: 0.308800
lr 1.000000e+00 reg 1.000000e-06 train accuracy: 0.361125 val accuracy: 0.322300
lr 1.000000e+00 reg 1.000000e-05 train accuracy: 0.344675 val accuracy: 0.303400
lr 1.000000e+00 reg 1.000000e-04 train accuracy: 0.349775 val accuracy: 0.313500
lr 1.000000e+00 reg 1.000000e-02 train accuracy: 0.250225 val accuracy: 0.244500
lr 1.000000e+00 reg 5.000000e-01 train accuracy: 0.077800 val accuracy: 0.079000
lr 1.000000e+01 reg 1.000000e-06 train accuracy: 0.322275 val accuracy: 0.292100
lr 1.000000e+01 reg 1.000000e-05 train accuracy: 0.310825 val accuracy: 0.288600
lr 1.000000e+01 reg 1.000000e-04 train accuracy: 0.322425 val accuracy: 0.307700
lr 1.000000e+01 reg 1.000000e-02 train accuracy: 0.193625 val accuracy: 0.195300
lr 1.000000e+01 reg 5.000000e-01 train accuracy: 0.099650 val accuracy: 0.101400
best validation accuracy achieved during cross-validation: 0.403900
```



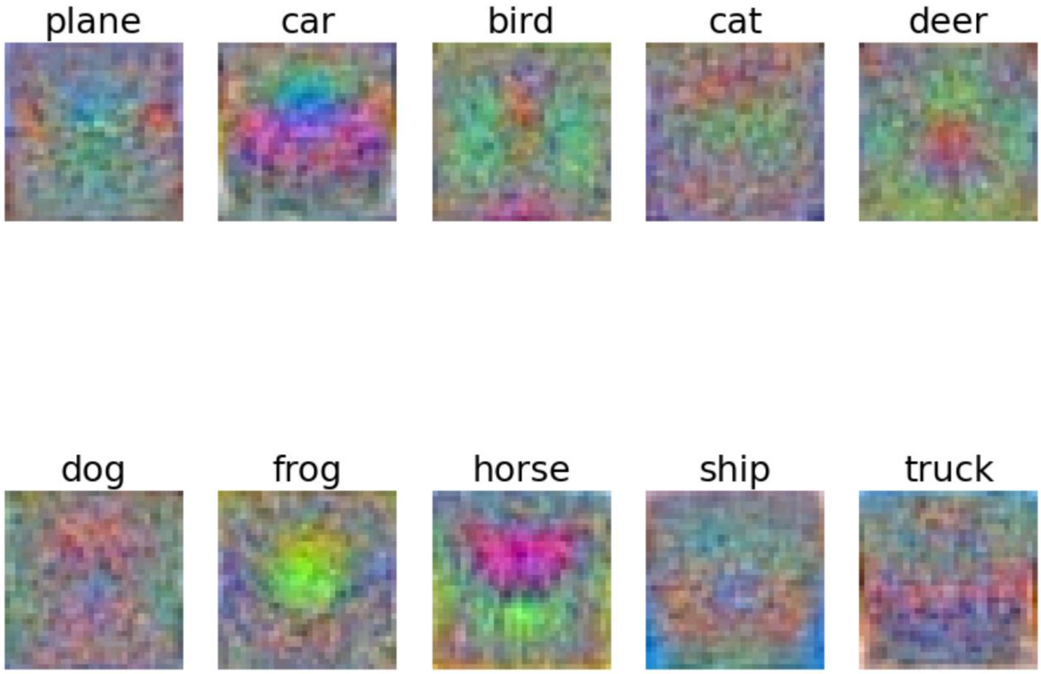
由以上結果可以看到，當 learning rate = 0.1，regularization strength = 0.00001 時，得到最佳的結果為 40.39 %。跟 SVM loss 比起來，結果稍微好一點。

2. Testing :

```
linear Softmax on raw pixels final test set accuracy: 0.403500
```

對於測試資料來說，此模型有 40.35 % 準確度。

- **每個類別的模板 ：**

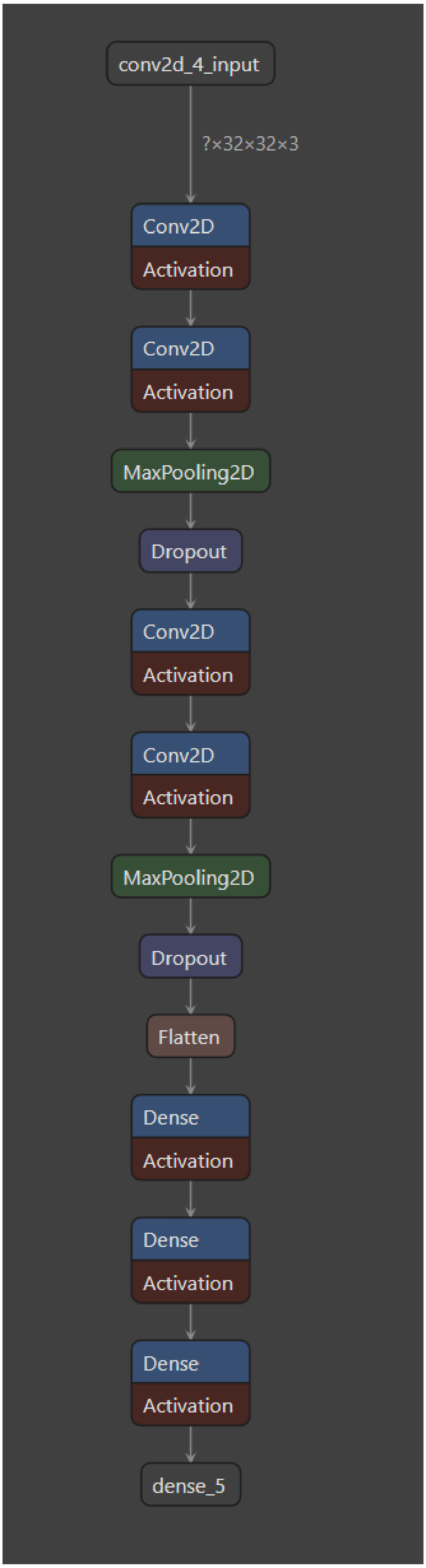


與 SVM 的模板相比，我們會發現到結果大致相同。

III.額外嘗試 ：

從第一次作業到第四次作業，我們建立的三種分類器，非別為 KNN、SVM classifier 和 softmax classifier，但是這三種模型對於 cifar-10 data set 的最佳表現只有 40 %左右的準確度，我認為還是偏低的，剛好教授上課時有提到將 linear classifier 疊加起來就會是神經網路，因此我嘗試自己建立一個 CNN(透過 tensorflow 內建的 layer 來建立)並觀察他的表現。

CNN 架構如下圖


conv2d_4_input
?×32×32×3
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Dropout
Conv2D
Activation
Conv2D
Activation
MaxPooling2D
Dropout
Flatten
Dense
Activation
Dense
Activation
Dense
Activation
dense_5

其中每一層的 activation 皆使用 relu，output layer 則是使用 softmax。在網路架構中有加上 max pooling，其目的為減少圖像尺寸以提取關鍵特徵。訓練結果如下

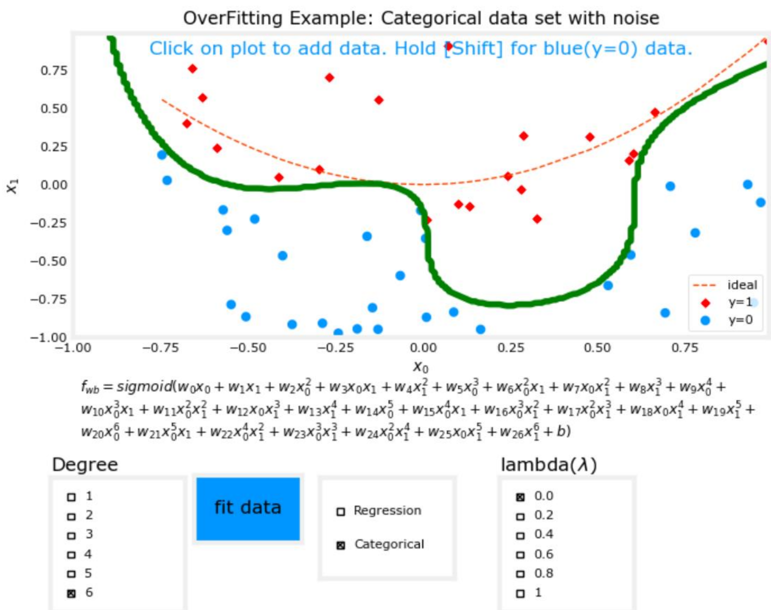
```
Epoch 36/40
1563/1563 [==============================] - 231s 148ms/step - loss: 0.6331 - a
ccuracy: 0.7808 - val_loss: 0.7554 - val_accuracy: 0.7487
Epoch 37/40
1563/1563 [==============================] - 258s 165ms/step - loss: 0.6268 - a
ccuracy: 0.7838 - val_loss: 0.7692 - val_accuracy: 0.7460
Epoch 38/40
1563/1563 [==============================] - 230s 147ms/step - loss: 0.6226 - a
ccuracy: 0.7861 - val_loss: 0.7348 - val_accuracy: 0.7548
Epoch 39/40
1563/1563 [==============================] - 239s 153ms/step - loss: 0.6108 - a
ccuracy: 0.7877 - val_loss: 0.7640 - val_accuracy: 0.7483
Epoch 40/40
1563/1563 [==============================] - 236s 151ms/step - loss: 0.6085 - a
ccuracy: 0.7885 - val_loss: 0.7949 - val_accuracy: 0.7395
```

我們可以看到 validation accuracy 最佳可以提升到 79.5 %的準確度，比起 linear classifier 的 40 %有巨大提升。

IV. 結果討論 ：

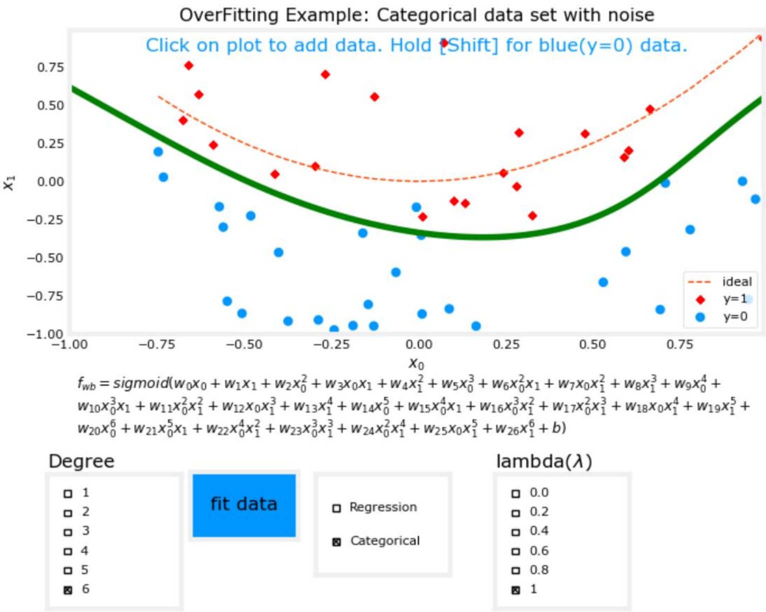
- **Regularization :**

  我到 Coursera 上找到一門由 Andrew Ng 教授所開設的 supervised machine learning 的課程中的教材，裡面有一個講述 regularization 的模型。透過這個模型能夠更加了解 regularization 的用途，以下展示模型



  上圖可以看到，現在有兩個類別，對於這兩個類別來說，最佳的分類方式為紅色虛線(有一些往上調整是為了與實際做出來的分類邊界做區隔)，綠色實現為我們做出來的結果。目前我們使用一個六次多項式以

及 lambda=0(regularization strength)來做二元分類。越高次的多項式越能夠擬和複雜的曲線，對於現在的模型來說，似乎 overfit 了。接著我們調整 lambda，結果如下圖



由此結果可以看出，我們並沒有調整分類器的多項式，而是藉由調整 lambda 就將 overfit 的問題解決掉了。藉由這個小模型的展示，可以看出 regularization 的功用為降低 overfitting 的趨勢，他藉由懲罰權重值變得太大來達成效果，我們可以透過加大 regularization strength 來讓 regularization 的效果加大。

- **SVM loss v.s. Sigmoid :**

1. 計算損失方式 ：

   (1) SVM loss : 鼓勵正確分類，懲罰錯誤分類。

   (2) Softmax : 得分代表說屬於該類別的機率，並使正確類別的機率提高，錯誤類別的機率降低。

2. 優化目標 ：

   (1) SVM loss : 最大化類別邊界的間隔。

   (2) Softmax : 最小化預測機率與正確類別之間的差異。

3. 表現差異 ：對於這次的線性分類器來說，兩個 loss function 的表現差異很小。Softmax 的表現會比 SVM 來的好一點，我覺得這是合理的，因為對於 softmax 來說，他考慮的是機率，而所有結果的機率皆會被考慮進去。但是 SVM 只會考慮滿足$W_j^T x_i - W_{y_i}^T x_i + 1 > 0$的結果，因此比起 softmax，SVM 會少考慮一些東西，造成準確

度的下降。

- **Linear classifier v.s. CNN :**
  1. 訓練時間 ： 訓練 linear classifier 非常的快速，甚至不需要一分鐘就能夠訓練完成。至於 CNN 就要訓練一段時間，這次我自己建構的 CNN 就要花兩個小時做訓練。兩者之間的訓練成本差很多。
  2. 表現 ： 在 cifar-10 資料集的表現上，CNN 表現得比 linear classifier 好許多，原因如下 ：
     (1) CNN 透過多層的 classifier 來抓取特徵，這使得 CNN 能夠學習更複雜的特徵。
     (2) 對於 CNN 來說，每個神經元只會對輸入數據的局部區域進行操作，這使得 CNN 能夠更加精確地捕捉圖像中的局部特徵。
     (3) 由 CNN 的架構圖中可以看到，CNN 中有加入 dropout layer，這會使 CNN 有更好的泛化能力。
- 心得總結 ：

  這次作業主要是實現 softmax loss 以及比較 softmax 與 SVM。大多數情況來說，softmax 的表現會比 SVM 還要好，這是因為對於 softmax 來說，他考慮的是機率，而所有結果的機率皆會被考慮進去。但是 SVM 只會考慮滿足$W_j^T x_i - W_{y_i}^T x_i + 1 > 0$的結果，這就會造成 SVM 會少考慮一些東西，造成準確度下降。

  除此之外，我還在自己嘗試建立神經網路的過程中學習到許多事物，像是實際去實現一個神經網路要如何寫程式、神經網路中的 hidden layer 有哪些參數可以調整以及調整後的效果。由這次的經驗，我也理解到教授上課所說的神經網路的 universal approximation 的含意。這次只是建立一個小小的 CNN 就對於整體的準確度有大幅提升，更何況是一個大型的神經網路。

  現實生活中做甚麼事情都是要有取捨的，訓練機器學習模型也是一樣。Linear classifier 訓練起來很容易快速，的確是一個大優點，但是做出來的準確度完全無法跟神經網路做比較，想要有好的結果就要花費大量的時間與努力才能獲得回報。

V. Reference :

[1] Standford “CS231n Convolutional Neural Networks for Visual recognition – Linear Classification”. https://cs231n.github.io/linear-classify/

[2] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/

[3] Allen Tzeng “卷積神經網路 (Convolutional Neural, CNN) https://hackmd.io/@allen108108/rkn-oVGA4

[4] Andrew Ng “C1_W3_Lab09_Regularization_Soln” https://www.coursera.org/learn/machine-learning