HW3 成果分析

本次作業實作回歸模型。

資料集狀況

CIFAR10 dataset 的資料是數筆 32*32 pixels 的彩色影像，並且有貓和狗兩類。

```
Data type of images: uint8 (value range: [0, 255])
Shape of images: (10000, 32, 32, 3)
Shape of labels: (10000,)
Number of classes: 2
Classes: ['cat', 'dog']
```

Linear Classification 實現

先進行影像前處理，將3維資料平坦化，並且使用 Dataloader 切分成數個 Batches

```
class Dataloader(object):
  def __init__(self, dataset, batch_size=1):
    self.dataset = dataset
    self.indice = np.array(range(len(self.dataset)))
    self.batch_size = batch_size
  def __len__(self):
    num_batch = None
    num_batch = np.ceil(len(self.dataset)/self.batch_size).astype(int)
    return num_batch
  def __getitem__(self, idx):
    batch_data = self.dataset[self.indice[idx: idx+self.batch_size]]
    return batch_data
  def shuffle(self):
    np.random.shuffle(self.indice)
```

接著建構一個 Linear Classfication，裏面定義了正向傳播、反向傳播和活化函數。因為是二元分類問題，所以可以採用 Sigmoid function 作為活化函數。

```
class LinearClassifier(object):
 def __init__(self, device='cpu'):
   self.W = None # classifier weights
   self.dv = device
 def save_weights(self, path):
   torch.save(self.W, path)
 def load_weights(self, path):
   self.W = torch.load(path)
 def sigmoid(self, score):
   prob = torch.zeros_like(score)
   prob = 1/(1+np.exp(-1*score))
   return prob

 def forward(self, x):
   num_data, data_dim = x.shape
   if self.W is None:
     np.random.seed(0)
     self.W = torch.from_numpy(np.random.randn(data_dim)*1e-4).to(self.dv)
   x = x.to(self.dv)
   pred_y = torch.zeros(num_data).to(self.dv)
   pred_y = np.matmul(x, self.W)
   pred_y = self.sigmoid(pred_y)
   self.cache = (x, pred_y)
   return pred_y

 def backward(self, dL):
   dL = dL.to(self.dv)
   dW = torch.zeros(1).to(self.dv)
   x, pred_y = self.cache
   num_data, data_dim = x.shape
   dpred_y = pred_y * (1 - pred_y) # derivative of sigmoid
   dW = dpred_y * dL
   dW = np.matmul(dW, x)/num_data

   return dW
```

其中，再寫反向傳播時有一些疑問出現。

```
Result of backward: Expected -1.0628412871500468 but got [-0.47932113 -0.58352016]
```

打開 _utils.py 察看 `Backward_Test()` ，發現老師給的輸入測資為 (3個訓練資料, 2個特徵)。我的直覺是，這樣輸入層會有2個神經元，並且彙整到只有1個神經元的輸出層。也就是說，**會有兩個權重值和其權重梯度**但老師給的測資僅有一個dW輸出，試過兩個dW平均，但也不符合測資答案。

```
# _utils.py

def Backward_Test(model):
    x = torch.tensor([1, 2, 3, 4, 5, 6], dtype=torch.float64).reshape(3, 2)
    y = model.forward(x)
    val = model.backward(x)
    truth = [2.25, 3.0]
    if (rel_err(torch.tensor(truth).to(model.dv), val) < 1e-6).cpu().numpy():
        return 'Correct'
    else:
        return f'Expected {truth} but got {val.cpu().numpy()}'
```

先就目前的狀況繼續做下去吧。繼續完成 Loss Function 和 Optmizer 。使用的是 BCE Loss Function ，適合使用在二元分類問題。

```
class BCEloss(object):
  def __init__(self, device='cpu'):
    self.dv = device
  def __call__(self, y, pred_y):
    y = y.to(self.dv)
    pred_y = pred_y.to(self.dv)
    L = torch.zeros_like(y).to(self.dv)
    dL = torch.zeros_like(y).to(self.dv)
    L = -y * torch.log(pred_y) - (1 - y) * torch.log(1 - pred_y)
    dL = (pred_y - y) / (pred_y * (1 - pred_y))
    return L, dL


class Optimizer(object):
  def __init__(self, model, learning_rate):
    self.model = model
    self.lr = learning_rate
  def step(self, dW):
    dW = dW.to(model.dv)
    new_weights = self.model.W.clone()
    new_weights = self.model.W - self.lr * dW
    self.model.W = new_weights
```

除了反向傳播和測資有出入以外，其他的測資都是有通過的。

執行結果與分析

使用以下超參數進行訓練，並且儲存 loss 最低的權重值：

```
LR = 1e-5
EPOCHS = 30
model = LinearClassifier(DEVICE)
loss_func = BCEloss(DEVICE)
optimizer = Optimizer(model, learning_rate=LR)
```

訓練結果如下：

```
Epoch 27:
Training accuracy: 49.47%, loss: 0.6797
Validation accuracy: 52.80%, loss: 0.6885
[WEIGHTS SAVED]
```

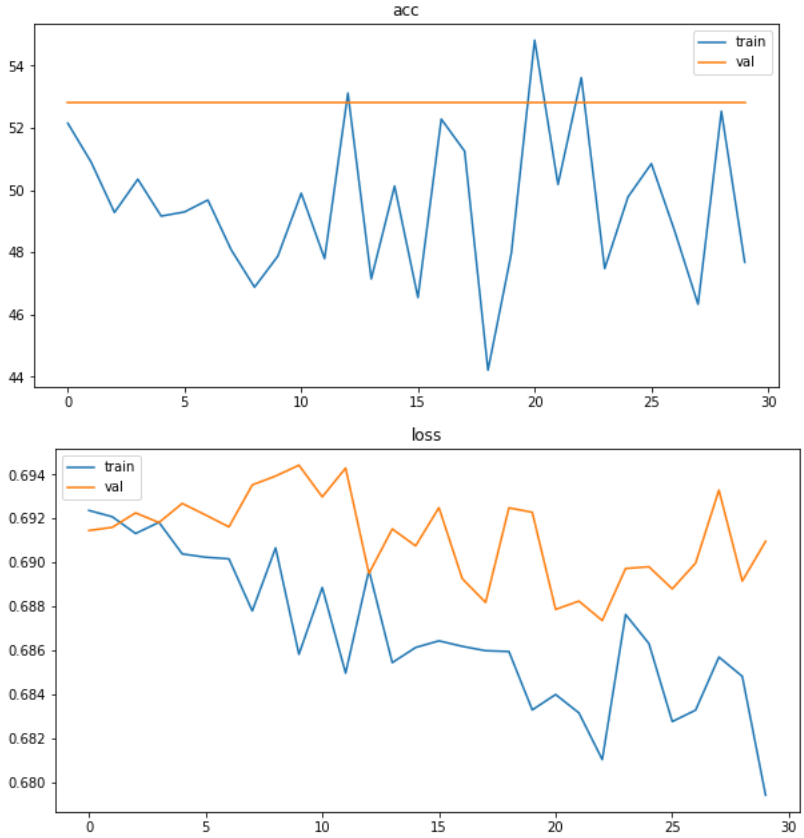
導入 `test_data` 進行預測，結果如下：

```
Got 1005 correct prediction in 2000 test data, accuracy: 50.25%
```

輸出結果和老師提供的原始檔相近，所以不太確定反向傳播的函式是否真的有錯誤。

結果分析

50.25% 幾乎是用猜的吧？
先把訓練過程畫出來：



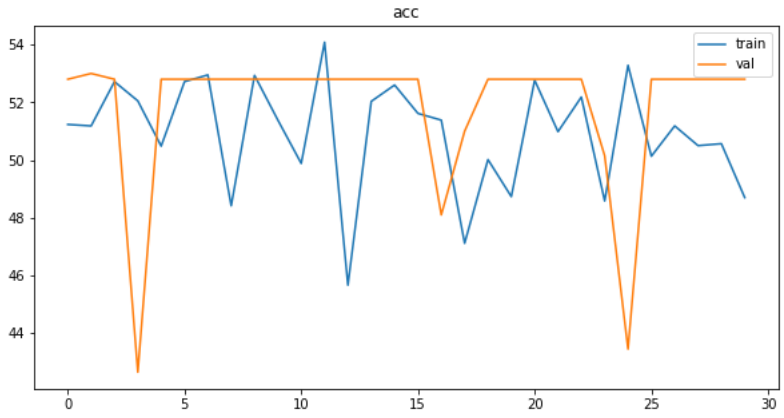
驗證資料的準確度完全沒有動靜，顯然 Backpropagation 的確出了問題。

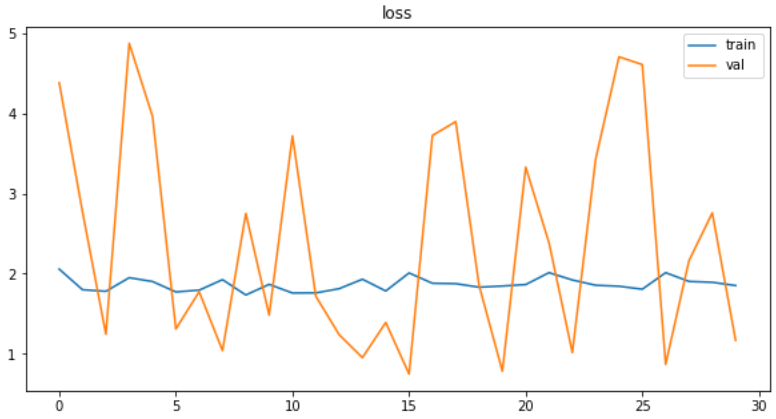
嘗試

問了助教之後，發現我的思路沒有問題，然而助教是把 dW 相加起來了，所以才會只有一個值。我們用新的模型來做訓練：

```
def backward(self, dL):
    dL = dL.to(self.dv)
    dW = torch.zeros(1).to(self.dv)
    x, pred_y = self.cache
    num_data, data_dim = x.shape
    dpred_y = pred_y * (1 - pred_y) # derivative of sigmoid
    dW = dpred_y * dL
    dW = torch.sum(torch.matmul(dW, x) / num_data)
    return dW
```

一樣印出訓練過程：





我們發現，因為dW值相同，導致梯度下降不明顯，較無法收斂 loss function。只是由於只有單顆神經元，所以面對這種高維的資料準確率實在是太低，用猜的都還比較準：

```
[97]  1 test_dataloader = Dataloader(test_dataset)
      2 total_corr = 0; total_eval = 0
      3 for batch_idx in range(len(test_dataloader)):
      4   ### Random Guess ###
      5   pred_label = torch.FloatTensor([(1 if i<0.5 else 0) for i in torch.rand(batch_img.shape[0])])
      6   ### Model Evaluation ###
      7   total_eval += 1
      8   total_corr += evaluate(batch_label, pred_label.cpu().numpy())
      9 print(f'Got {total_corr} correct prediction in {total_eval} test data, accuracy: {format((total_corr*1.0/total_eval)*100, ".2f")}%')

      Got 1034 correct prediction in 2000 test data, accuracy: 51.70%
```

之後引入深度神經模型便可以提高準確率。