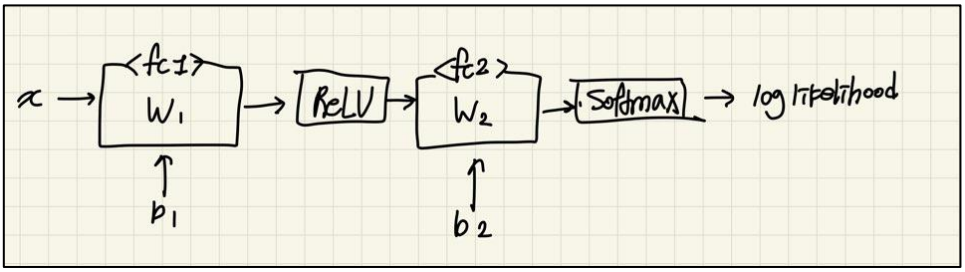
The explanation of the code

[neural_net.py]

주어진 코드는 아래의 그림과 같은 two layer neural net이다.

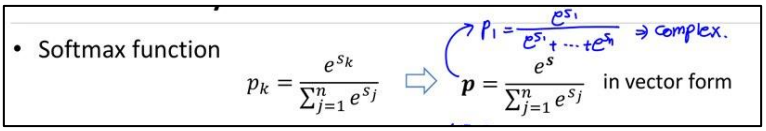



```
#############################################################################
# TODO: Perform the forward pass, computing the class scores for the input. #
# Store the result in the scores variable, which should be an array of      #
# shape (N, C).                                                             #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

fc1=X.dot(W1) + b1
X2=np.maximum(0, fc1)
scores=X2.dot(W2)+b2

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#############################################################################
#                               END OF YOUR CODE                            #
#############################################################################
```

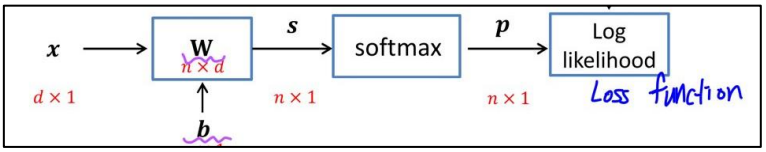
Score를 계산하는 식은 S=Wx + b 이므로 input을 fc1의 weight와 dot product 계산 후, fc1에서의 b값인 b1을 더한다. ReLU의 식은 max(0, x)이므로 앞에서 계산한 fc1 값을 ReLU 함수에 넣은 후, fc2의 weight와 dot product 계산을 한다. 여기에 b 값인 b2를 더한 결과가 score값이 된다. 위의 neural net 구조 그림에서 fc2까지의 값을 계산한 것이다.

```
#############################################################################
# TODO: Finish the forward pass, and compute the loss. This should include  #
# both the data loss and L2 regularization for W1 and W2. Store the result  #
# in the variable loss, which should be a scalar. Use the Softmax           #
# classifier loss.                                                          #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

scores_exp=np.exp(scores)
softmax_matrix=scores_exp/np.sum(scores_exp, axis=1, keepdims=True)
loss=np.sum(-np.log(softmax_matrix[np.arange(N), y]))

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#############################################################################
#                               END OF YOUR CODE                            #
#############################################################################
```

Fc2까지 계산된 score값을 softmax 함수에 넣어 forward pass를 마무리한다. Softmax 함수의 식은 아래의 식과 같다.



따라서 e^s를 계산하기 위하여 numpy의 exp( )함수를 사용했다. score값에 exponential을 취한 값들의 합을 분모로, 각각의 score값에 exponential을 취한 값이 분자로 들어가서 softmax_loss값 계산을 한다.

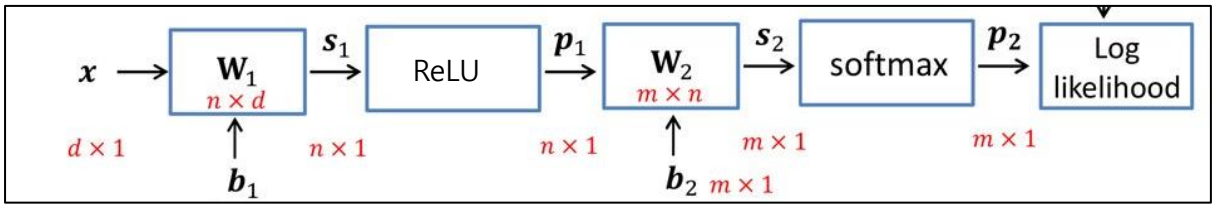


$$L = -\log p_y \text{ where } y \text{ satisfies } z_y = 1$$

Softmax 값 계산 후에는 log likelihood loss를 함께 계산해주기 때문에 위의 식에서 p_y의 자리에 softmax_matrix를 넣어주어 loss 계산을 마친다.

```
#############################################################################
# TODO: Compute the backward pass, computing the derivatives of the weights #
# and biases. Store the results in the grads dictionary. For example,       #
# grads['W1'] should store the gradient on W1, and be a matrix of same size #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# Let z be the result after W * X, z after activation function becomes y.

softmax_matrix[np.arange(N) ,y] -= 1
softmax_matrix /= N

dW2 = X2.T.dot(softmax_matrix)
db2 = softmax_matrix.sum(axis=0)

dW1 = softmax_matrix.dot(W2.T)
dfc1 = dW1 * (fc1>0)
dW1 = X.T.dot(dfc1)
db1 = dfc1.sum(axis=0)

dW1 += reg * W1
dW2 += reg * W2

grads = {'W1':dW1, 'b1':db1, 'W2':dW2, 'b2':db2}

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
```

Loss를 확인하여 loss를 줄이기 위한 optimization으로 back propagation을 한다. 이는 1계 도함수를 이용하여 loss를 생성하는 과정에 영향을 미친 parameter들을 영향을 미친 만큼 조절해주는 것으로 합성함수의 미분을 위한 chain rule을 사용하였다. 이 코드의 neural net이 아래와 같이 구성되어 있다고 가정하였을 때 back propagation은 가장 끝에 있는 p2에 대하여 Loss를 편미분하는 것부터 시작된다.



$$\frac{\partial L}{\partial \boldsymbol{p}_2} = \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix} \quad y^{th}\ \text{row}$$

이것이 loss를 p2에 대하여 편미분하였을 때의 결과이며 이에 따라 코드를 작성하였다.

$$\frac{\partial L}{\partial \boldsymbol{s}_2} = \frac{\partial \boldsymbol{p}_2}{\partial \boldsymbol{s}_2}\frac{\partial L}{\partial \boldsymbol{p}_2} = \mathbf{D}\frac{\partial L}{\partial \boldsymbol{p}_2} \qquad D_{ab} = p_a(\delta_{ab} - p_b) \qquad \delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

다음으로 s2에 대한 loss의 편미분 값을 구해야 한다. 하지만 dL/ds2 값을 바로 구할 수 없기 때문에 chain rule을 이용하여 앞에서 구한 dL/dp2 를 이용한다. dL/dp2의 값은 softmax_matrix에 저장되어 있으며 위 식의 D값이 코드 상의 X2이다.

$$\frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}$$

이 식에 따라 b2 gradient를 계산하였다.

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}\boldsymbol{p}_1^{\mathrm{T}}$$

이 식에 따라 W2 gradient 또한 계산하였다.

$$\frac{\partial L}{\partial \boldsymbol{p}_1} = \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{p}_1}\frac{\partial L}{\partial \boldsymbol{s}_2} = \mathbf{W}_2^{\mathrm{T}}\frac{\partial L}{\partial \boldsymbol{s}_2}$$

이 식에 따라 p1 gradient를 계산하고,

$$\frac{\partial L}{\partial \boldsymbol{s}_1} = \frac{\partial \boldsymbol{p}_1}{\partial \boldsymbol{s}_1}\frac{\partial L}{\partial \boldsymbol{p}_1} = diag\big((1-\sigma(s_{1,j}))\sigma(s_{1,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_1}$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}\boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}$$

이 식에 따라 s1 gradient를 계산하면 fc1의 weight와 biases gradient 계산이 가능해지며 이것이 back propagation의 과정이다.

이렇게 계산한 dw1, db1, dw2, db2 값을 마크 다운의 지시사항대로 grads에 딕셔너리 형태로 저장해주어서 다음 코드에서 사용할 수 있도록 하였다.

```
################################################################################
# TODO: Create a random minibatch of training data and labels, storing  #
# them in X_batch and y_batch respectively.                             #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

batch_indices = np.random.choice(num_train, batch_size)
X_batch = X[batch_indices]
y_batch = y[batch_indices]

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                             END OF YOUR CODE                                 #
################################################################################
```

random하게 값을 생성하기 위하여 numpy의 random함수를 사용하여 X_batch와 y_batch 값을 지정해주었다.

```
#############################################################################
# TODO: Use the gradients in the grads dictionary to update the             #
# parameters of the network (stored in the dictionary self.params)          #
# using stochastic gradient descent. You'll need to use the gradients       #
# stored in the grads dictionary defined above.                             #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for key in self.params:
    self.params[key] -= learning_rate * grads[key]

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
```

이 코드는 optimization을 위한 gradient descent계산으로

$$\mathbf{W}^{\mathrm{T}+1} = \mathbf{W}^{\mathrm{T}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{\mathrm{T}}}$$ 이 식과 같이 W^{T+1}값을 계산하는 과정이다.

```
#############################################################################
# TODO: Implement this function; it should be VERY simple!                  #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

y_pred = np.argmax( self.loss(X), axis=1)

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
```

Train된 weight를 이용하여 input data인 X를 받아서 각각의 class를 지정해주는 predict 코드이다. Predict를 하는 방법은 각 N개의 data에 대하여 가장 높은 score를 가지는 class로 지정해주는 방식이다. 따라서 numpy의 argmax( )함수를 사용하여 predict함수를 완성하였다.

[Two_layer_net.ipynb]

```
#############################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained              #
# model in best_net.                                                                        #
#                                                                                           #
# To help debug your network, it may help to use visualizations similar to the              #
# ones we used above; these visualizations will have significant qualitative                #
# differences from the ones we saw above for the poorly tuned network.                      #
#                                                                                           #
# Tweaking hyperparameters by hand can be fun, but you might find it useful to              #
# write code to sweep through possible combinations of hyperparameters                      #
# automatically like we did on the previous exercises.                                      #
#############################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

```
best_val = -1
best_stats = []

def generate_random_hyperparams(lr_min, lr_max, reg_min, reg_max, h_min, h_max):
    lr = 10**np.random.uniform(lr_min,lr_max)
    reg = 10**np.random.uniform(reg_min,reg_max)
    hidden = np.random.randint(h_min, h_max)
    return lr, reg, hidden

def random_search_hyperparams(lr_values, reg_values, h_values):
    lr = lr_values[np.random.randint(0,len(lr_values))]
    reg = reg_values[np.random.randint(0,len(reg_values))]
    hidden = h_values[np.random.randint(0,len(h_values))]
    return lr, reg, hidden
```

코드 위의 지시사항에 따라 tuning을 하는 대상으로 hidden layer size, learning rate, regularization strength로 정하였다. 이 값들은 random함수를 이용하여 생성하였고 그 중 가장 적합한 hyperparameter를 찾을 수 있도록 여러 번 코드를 돌려보았다. Generate_random_hyperparams 함수로 random한 값을 일정 개수만큼 생성한 후에 random_search_hyperparams 함수에 넣어서 일정 개수의 값 중 하나를 random하게 고를 수 있도록 코드를 작성하였다.

```
input_size = 32 * 32 * 3
num_classes = 10

np.random.seed(0)

for i in range(20):
    lr, reg, hidden_size = random_search_hyperparams([0.001], [0.05, 0.1, 0.15], [50, 80, 100, 120, 150, 180, 200])
    net = TwoLayerNet(input_size, hidden_size, num_classes)
    stats = net.train(X_train, y_train, X_val, y_val,
                num_iters=2000, batch_size=200,
                learning_rate=lr, learning_rate_decay=0.95,
                reg=reg, verbose=False)

    train_accuracy = (net.predict(X_train) == y_train).mean()

    val_accuracy = (net.predict(X_val) == y_val).mean()

    if val_accuracy > best_val:
        best_val = val_accuracy
        best_net = net
        best_stats = stats

    print('lr %e reg %e hid %d  train accuracy: %f val accuracy: %f' % (
                lr, reg, hidden_size, train_accuracy, val_accuracy))
print('best validation accuracy achieved: %f' % best_val)

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

20번의 과정을 거쳐 그 중 가장 적합한 hyperparameter를 print할 수 있도록 코드를 작성하였다. net변수에 neural net.py에서 작성한 TwoLayerNet class를 저장하여 train 전까지의 준비를 하였다. 그 후, parameter들을 알맞게 넣어서 net을 train하고, predict와 validation을 진행하였다. 이 과정을 통해서 가장 높은 정확도를 보이는 parameter를 best_ 변수에 저장하여서 48퍼센트 이상의 정확도가 나올 수 있도록 코드를 작성하였다.

The results of the code

- Forward pass: compute scores

```
Your scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

Difference between your scores and correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]
3.680272093239262e-08
```

- Forward pass: compute loss

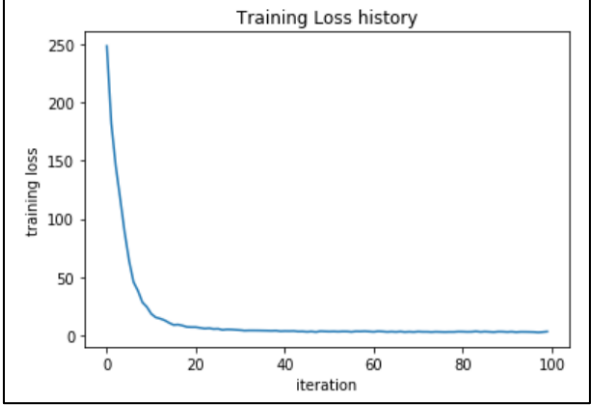
```
Difference between your loss and correct loss:
5.0254973692602665
```

- Backward pass

```
W1 max relative error: 1.000000e+00
b1 max relative error: 6.666667e-01
W2 max relative error: 1.000000e+00
b2 max relative error: 6.666667e-01
```

- Train the network

```
Final training loss:  3.4277446900346638
```



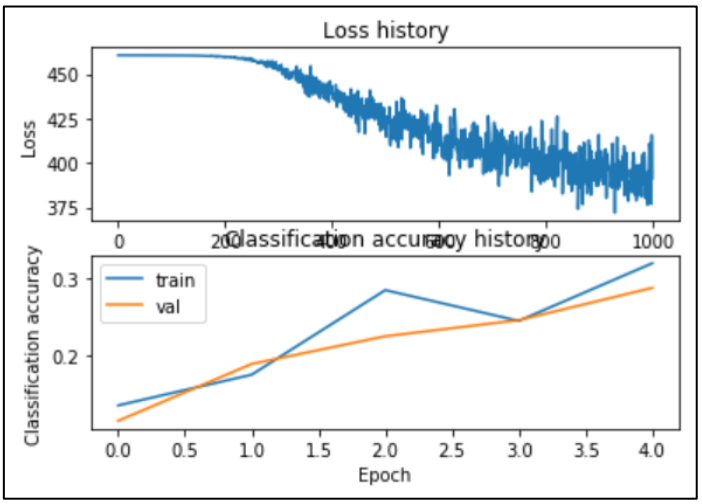
- Load the data

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

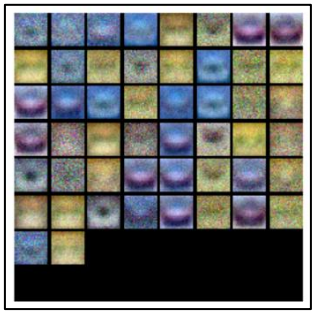
- Train a network

```
iteration 0 / 1000: loss 460.513923
iteration 100 / 1000: loss 460.433603
iteration 200 / 1000: loss 459.450971
iteration 300 / 1000: loss 451.816091
iteration 400 / 1000: loss 440.639413
iteration 500 / 1000: loss 423.395387
iteration 600 / 1000: loss 409.860521
iteration 700 / 1000: loss 397.122895
iteration 800 / 1000: loss 400.633510
iteration 900 / 1000: loss 389.492913
Validation accuracy:  0.287
```

- Debug the training



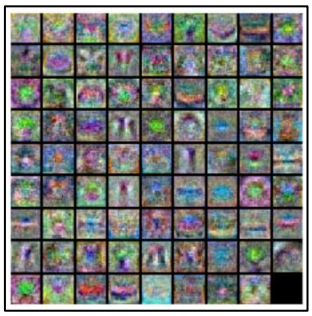
- The result of show_net_weights( ) function



- Tune your hyperparameters

```
lr 1.000000e-03 reg 5.000000e-02 hid 180  train accuracy: 0.566061 val accuracy: 0.491000
lr 1.000000e-03 reg 1.500000e-01 hid 200  train accuracy: 0.571510 val accuracy: 0.521000
lr 1.000000e-03 reg 1.500000e-01 hid 80  train accuracy: 0.539163 val accuracy: 0.500000
lr 1.000000e-03 reg 1.000000e-01 hid 80  train accuracy: 0.555327 val accuracy: 0.524000
lr 1.000000e-03 reg 1.500000e-01 hid 100  train accuracy: 0.554510 val accuracy: 0.517000
lr 1.000000e-03 reg 5.000000e-02 hid 150  train accuracy: 0.575531 val accuracy: 0.515000
lr 1.000000e-03 reg 1.000000e-01 hid 150  train accuracy: 0.566755 val accuracy: 0.509000
lr 1.000000e-03 reg 1.500000e-01 hid 180  train accuracy: 0.567857 val accuracy: 0.522000
lr 1.000000e-03 reg 5.000000e-02 hid 120  train accuracy: 0.566000 val accuracy: 0.518000
lr 1.000000e-03 reg 1.500000e-01 hid 200  train accuracy: 0.568816 val accuracy: 0.514000
lr 1.000000e-03 reg 1.500000e-01 hid 150  train accuracy: 0.562633 val accuracy: 0.516000
lr 1.000000e-03 reg 5.000000e-02 hid 80  train accuracy: 0.555367 val accuracy: 0.492000
lr 1.000000e-03 reg 5.000000e-02 hid 100  train accuracy: 0.555184 val accuracy: 0.510000
lr 1.000000e-03 reg 5.000000e-02 hid 80  train accuracy: 0.547367 val accuracy: 0.511000
lr 1.000000e-03 reg 1.000000e-01 hid 120  train accuracy: 0.565959 val accuracy: 0.514000
lr 1.000000e-03 reg 5.000000e-02 hid 80  train accuracy: 0.552898 val accuracy: 0.492000
lr 1.000000e-03 reg 1.500000e-01 hid 150  train accuracy: 0.567796 val accuracy: 0.521000
lr 1.000000e-03 reg 1.000000e-01 hid 50  train accuracy: 0.531408 val accuracy: 0.495000
lr 1.000000e-03 reg 1.500000e-01 hid 200  train accuracy: 0.569898 val accuracy: 0.524000
lr 1.000000e-03 reg 1.000000e-01 hid 80  train accuracy: 0.545367 val accuracy: 0.511000
best validation accuracy achieved: 0.524000
```

- The result of show_net_weights( ) function



- Run on the test set

```
Test accuracy:  0.52
```

Analysis

train과정에서 loss가 줄어드는 것을 print문과 그래프를 통해 확인할 수 있어 올바르게 optimization이 진행되고 있음을 확인할 수 있다. Tune your parameter cell에서 48퍼센트 이상의 확률이 나오도록 parameter tuning을 진행하라는 지시사항에 따라 test accuracy가 52퍼센트가 나나온 것 확인하였고 이를 통해 random하게 parameter값을 생성하여 그중 가장 적합한 parameter를 사용하는 것이 크게 잘못된 방법이 아님을 알 수 있다.