Lab 3: Neural Networks

```
##############################################################################
# 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). scores must contain the value extracted from the net before
# the Softmax. probs variable must contain probability extracted through the
# Softmax.
##############################################################################

z1 = np.dot(X,W1) + b1 #(N,D)*(D,H) --> (N,H)
x1 = np.maximum(0,z1) #(N,H)
scores = np.dot(x1,W2) + b2 #(N,H)*(H,C) --> (N,C)
probs = np.exp(scores) / np.exp(scores).sum(axis=1).reshape(-1,1) # (N,C)


##############################################################################
# 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. The y term in the loss must be replaced by the y_oh
# encoding.
##############################################################################

loss = np.sum(-y_oh*np.log(probs)) / N + reg*np.sum(W1**2) +
      reg*np.sum(W2**2)



##############################################################################
# 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.
# y_oh must be used as y for gradients.
##############################################################################

grads['W2'] = np.dot(x1.T, (probs-y_oh)) / N + 2*reg*W2
grads['b2'] = (probs - y_oh).sum(axis=0) / N
dz1 = np.dot((probs - y_oh), W2.T)*(1*(z1 >= 0))
grads['W1'] = np.dot(X.T, dz1) / N + 2*reg*W1
grads['b1'] = dz1.sum(axis=0) / N
```

```
#########################################################################
# TODO: Create a random minibatch of training data and labels, storing #
# them in X_batch and y_batch respectively.
#########################################################################

idx_batch = np.random.randint(0,num_train, batch_size)
X_batch, y_batch = X[idx_batch], y[idx_batch]



#########################################################################
# 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.
#########################################################################

self.params['W2'] = self.params['W2'] - learning_rate*grads['W2']
self.params['b2'] = self.params['b2'] - learning_rate*grads['b2']
self.params['W1'] = self.params['W1'] - learning_rate*grads['W1']
self.params['b1'] = self.params['b1'] - learning_rate*grads['b1']



#########################################################################
# TODO: Implement this function; it should be VERY simple!
#########################################################################

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



#########################################################################
# 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.
#########################################################################


input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10
best_net = TwoLayerNet(input_size, hidden_size, num_classes)
# Train the network
```

```
stats = best_net.train(X_train, y_train, X_val, y_val,
                       num_iters=5000, batch_size=128,
                       learning_rate=5e-4, learning_rate_decay=0.95,
                       reg=0.15, verbose=True)



################################################################################
# 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.
################################################################################

input_size = 32 * 32 * 3
hidden_size = 256
num_classes = 10
best_net = TwoLayerNet(input_size, hidden_size, num_classes)
# Train the network
stats = best_net.train(X_train, y_train, X_val, y_val,
                       num_iters=5000, batch_size=128,
                       learning_rate=5e-4, learning_rate_decay=0.95,
                       reg=0.15, verbose=True)
```