Implementing a Neural Network

In this exercise we will develop a neural network with fully-connected layers to perform classification, and test it out on the CIFAR-10 dataset.

In [1]:

```
# A bit of setup
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

from cs231n.classifiers.neural_net import TwoLayerNet


%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
    """ returns relative error """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
print ("setup done\n")
```

```
setup done
```

We will use the class `TwoLayerNet` in the file `cs231n/classifiers/neural_net.py` to represent instances of our network. The network parameters are stored in the instance variable `self.params` where keys are string parameter names and values are numpy arrays. Below, we initialize toy data and a toy model that we will use to develop your implementation.

In [3]:

```
# Create a small net and some toy data to check your implementations.
# Note that we set the random seed for repeatable experiments.

input_size = 4
hidden_size = 10
num_classes = 3
num_inputs = 5

def init_toy_model():
    np.random.seed(0)
    return TwoLayerNet(input_size, hidden_size, num_classes, std=1e-1)

def init_toy_data():
    np.random.seed(1)
    X = 10 * np.random.randn(num_inputs, input_size)
    y = np.array([0, 1, 2, 2, 1])
    return X, y

net = init_toy_model()
X, y = init_toy_data()

print ("done with init\n")
```

```
done with init
```

Forward pass: compute scores

Open the file `cs231n/classifiers/neural_net.py` and look at the method `TwoLayerNet.loss`. This function is very similar to the loss functions you have written for the SVM and Softmax exercises: It takes the data and weights and computes the class scores, the loss, and the gradients on the parameters.

Implement the first part of the forward pass which uses the weights and biases to compute the scores for all inputs.

In [4]:

```
scores = net.loss(X)
print('Your scores:')
print(scores)
print()
print('correct scores:')
correct_scores = np.asarray([
  [-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]])
print(correct_scores)
print()

# The difference should be very small. We get < 1e-7
print('Difference between your scores and correct scores:')
print(np.sum(np.abs(scores - correct_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:
3.6802720925453725e-08
```

Forward pass: compute loss

In the same function, implement the second part that computes the data and regularizaion loss.

In [5]:

```
loss, _ = net.loss(X, y, reg=0.1)
correct_loss = 1.30378789133
print ("computed loss : ",loss,"   correct loss :",correct_loss)

# should be very small, we get < 1e-12
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

```
computed loss :  1.3037878913298202    correct loss : 1.30378789133
Difference between your loss and correct loss:
1.7985612998927536e-13
```

Backward pass

Implement the rest of the function. This will compute the gradient of the loss with respect to the variables `W1`, `b1`, `W2`, and `b2`. Now that you (hopefully!) have a correctly implemented forward pass, you can debug your backward pass using a numeric gradient check:

In [6]:

In [6]:

```
from cs231n.gradient_check import eval_numerical_gradient

# Use numeric gradient checking to check your implementation of the backward pass.
# If your implementation is correct, the difference between the numeric and
# analytic gradients should be less than 1e-8 for each of W1, W2, b1, and b2.

loss, grads = net.loss(X, y, reg=0.05)

# these should all be less than 1e-8 or so
for param_name in grads:
    f = lambda W: net.loss(X, y, reg=0.05)[0]
    param_grad_num = eval_numerical_gradient(f, net.params[param_name], verbose=False)
    print('%s max relative error: %e' % (param_name, rel_error(param_grad_num, grads[param_name])))
```

```
W2 max relative error: 3.440708e-09
b2 max relative error: 4.447646e-11
W1 max relative error: 3.561318e-09
b1 max relative error: 2.738421e-09
```

Train the network

To train the network we will use stochastic gradient descent (SGD), similar to the SVM and Softmax classifiers. Look at the function `TwoLayerNet.train` and fill in the missing sections to implement the training procedure. This should be very similar to the training procedure you used for the SVM and Softmax classifiers. You will also have to implement `TwoLayerNet.predict`, as the training process periodically performs prediction to keep track of accuracy over time while the network trains.

Once you have implemented the method, run the code below to train a two-layer network on toy data. You should achieve a training loss less than 0.2.

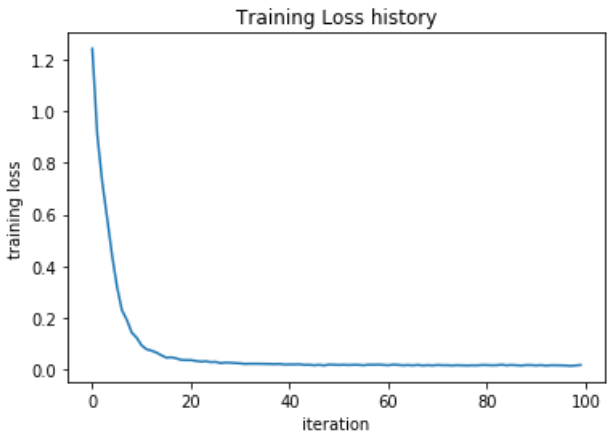
In [7]:

```
net = init_toy_model()
stats = net.train(X, y, X, y,
            learning_rate=1e-1, reg=5e-6,
            num_iters=100, verbose=False)

print('Final training loss: ', stats['loss_history'][-1])

# plot the loss history
plt.plot(stats['loss_history'])
plt.xlabel('iteration')
plt.ylabel('training loss')
plt.title('Training Loss history')
plt.show()
```

```
Final training loss:  0.01714364353292376
```



Load the data

Now that you have implemented a two-layer network that passes gradient checks and works on toy data, it's time to load up our favorite CIFAR-10 data so we can use it to train a classifier on a real dataset.

In [8]:

```
from cs231n.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the two-layer neural net classifier. These are the same steps as
    we used for the SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'

    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Reshape data to rows
    X_train = X_train.reshape(num_training, -1)
    X_val = X_val.reshape(num_validation, -1)
    X_test = X_test.reshape(num_test, -1)

    return X_train, y_train, X_val, y_val, X_test, y_test


# Cleaning up variables to prevent loading data multiple times (which may cause memory issue)
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
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

To train our network we will use SGD. In addition, we will adjust the learning rate with an exponential learning rate schedule as optimization proceeds; after each epoch, we will reduce the learning rate by multiplying it by a decay rate.

In [9]:

```
input_size = 32 * 32 * 3
```

```
hidden_size = 50
num_classes = 10
net = TwoLayerNet(input_size, hidden_size, num_classes)

# Train the network
stats = net.train(X_train, y_train, X_val, y_val,
            num_iters=1000, batch_size=200,
            learning_rate=1e-4, learning_rate_decay=0.95,
            reg=0.25, verbose=True)

# Predict on the validation set
val_acc = (net.predict(X_val) == y_val).mean()
print('Validation accuracy: ', val_acc)
```

```
iteration 0 / 1000: loss 2.302762
iteration 100 / 1000: loss 2.302358
iteration 200 / 1000: loss 2.297404
iteration 300 / 1000: loss 2.258897
iteration 400 / 1000: loss 2.202975
iteration 500 / 1000: loss 2.116816
iteration 600 / 1000: loss 2.049789
iteration 700 / 1000: loss 1.985711
iteration 800 / 1000: loss 2.003726
iteration 900 / 1000: loss 1.948076
Validation accuracy:  0.287
```

Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.29 on the validation set. This isn't very good.

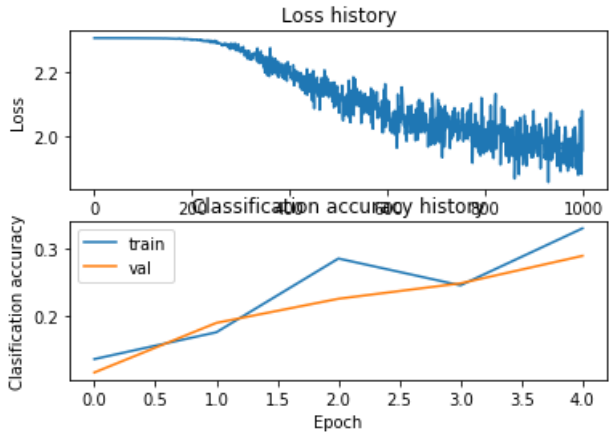
One strategy for getting insight into what's wrong is to plot the loss function and the accuracies on the training and validation sets during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

In [10]:

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(stats['loss_history'])
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(stats['train_acc_history'], label='train')
plt.plot(stats['val_acc_history'], label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
plt.legend()
plt.show()
```



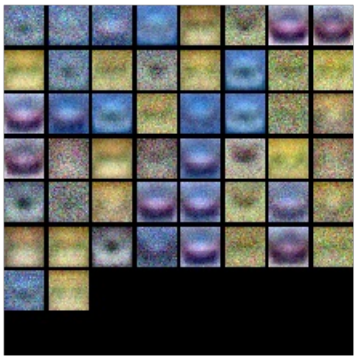
In [11]:

```
from cs231n.vis_utils import visualize_grid

# Visualize the weights of the network

def show_net_weights(net):
    W1 = net.params['W1']
    W1 = W1.reshape(32, 32, 3, -1).transpose(3, 0, 1, 2)
    plt.imshow(visualize_grid(W1, padding=3).astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(net)
```



Tune your hyperparameters

What's wrong?. Looking at the visualizations above, we see that the loss is decreasing more or less linearly, which seems to suggest that the learning rate may be too low. Moreover, there is no gap between the training and validation accuracy, suggesting that the model we used has low capacity, and that we should increase its size. On the other hand, with a very large model we would expect to see more overfitting, which would manifest itself as a very large gap between the training and validation accuracy.

Tuning. Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks, so we want you to get a lot of practice. Below, you should experiment with different values of the various hyperparameters, including hidden layer size, learning rate, numer of training epochs, and regularization strength. You might also consider tuning the learning rate decay, but you should be able to get good performance using the default value.

Approximate results. You should be aim to achieve a classification accuracy of greater than 48% on the validation set. Our best network gets over 52% on the validation set.

Experiment: You goal in this exercise is to get as good of a result on CIFAR-10 as you can, with a fully-connected Neural Network. Feel free implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.).

In [12]:

```
best_net = None # store the best model into this

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

# defaults
input_size = 32 * 32 * 3
num_classes = 10

# hyperparameters
learning_rates = [1e-2, 5e-4]
```

```
regularization_strengths = [0.1, 1]
hidden_sizes = [20,300]
batch_sizes = [100,600]

# additional variables
best_lr = None
best_reg = None
best_hidd = None
best_batch_size = None

# tune list of hyperparameters used
range_lr = np.linspace(learning_rates[0],learning_rates[1],4)
range_reg = np.linspace(regularization_strengths[0],regularization_strengths[1],3)
range_hs = [20,  60,100,150,180, 200]
range_bs = [200,300, 400]

best_val_acc = -1
print ("starting optimization, this may take a while :(")
for hs in range_hs:
    for lr in range_lr:
        for bs in range_bs:
            for rg in range_reg:
                #print ('hs:', hs, 'lr:', lr, 'bs', bs, 'reg', rg)

                net = TwoLayerNet(input_size, hs, output_size=num_classes)

                stats = net.train(X_train, y_train, X_val, y_val,num_iters=500,learning_rate=lr,reg
=rg, batch_size=bs)
                # check validation accuracy
                val_acc = (net.predict(X_val) == y_val).mean()

                print ('hs:', hs, 'lr:', lr, 'bs', bs, 'reg', rg)
                print ('Validation accuracy: ', val_acc)

                # store optimal hyperparameters
                if val_acc > best_val_acc:
                    best_val_acc = val_acc
                    best_net = net
                    best_lr = lr
                    best_reg = rg
                    best_hidd = hs
                    best_batch_size = bs

print ("\n** best validation achieved during cross validation: " ,best_val_acc)
```

```
starting optimization, this may take a while :(
hs: 20 lr: 0.01 bs 200 reg 0.1
```

```
/Users/colby/Documents/CS291 Independent Study/assignment1/cs231n/classifiers/neural_net.py:119: R
untimeWarning: divide by zero encountered in log
  total_loss = np.sum(-np.log(s[range(N), y]))
/Users/colby/Documents/CS291 Independent Study/assignment1/cs231n/classifiers/neural_net.py:115: R
untimeWarning: overflow encountered in subtract
  shift_scores = scores - np.max(scores, axis=1).reshape(-1, 1)
/Users/colby/Documents/CS291 Independent Study/assignment1/cs231n/classifiers/neural_net.py:115: R
untimeWarning: invalid value encountered in subtract
  shift_scores = scores - np.max(scores, axis=1).reshape(-1, 1)
/Users/colby/Documents/CS291 Independent Study/assignment1/cs231n/classifiers/neural_net.py:94: Ru
ntimeWarning: invalid value encountered in maximum
  s1= np.maximum(layer_1, 0) # shape (N,H)
/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:83: RuntimeWarning: invalid value
encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/Users/colby/Documents/CS291 Independent Study/assignment1/cs231n/classifiers/neural_net.py:136: R
untimeWarning: invalid value encountered in greater
  d_hidden = (s1 > 0) * d_hidden
/Users/colby/Documents/CS291 Independent Study/assignment1/cs231n/classifiers/neural_net.py:252: R
untimeWarning: invalid value encountered in maximum
  s1= np.maximum(layer_1, 0) # shape (N,H)
```

```
hs: 20 lr: 0.01 bs 200 reg 0.1
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 200 reg 0.55
hs: 20 lr: 0.01 bs 200 reg 0.55
```

```
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 200 reg 1.0
hs: 20 lr: 0.01 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 300 reg 0.1
hs: 20 lr: 0.01 bs 300 reg 0.1
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 300 reg 0.55
hs: 20 lr: 0.01 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 300 reg 1.0
hs: 20 lr: 0.01 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 400 reg 0.1
hs: 20 lr: 0.01 bs 400 reg 0.1
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 400 reg 0.55
hs: 20 lr: 0.01 bs 400 reg 0.55
Validation accuracy:  0.087
hs: 20 lr: 0.01 bs 400 reg 1.0
hs: 20 lr: 0.01 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 20 lr: 0.006833333333333334 bs 200 reg 0.1
hs: 20 lr: 0.006833333333333334 bs 200 reg 0.1
Validation accuracy:  0.132
hs: 20 lr: 0.006833333333333334 bs 200 reg 0.55
hs: 20 lr: 0.006833333333333334 bs 200 reg 0.55
Validation accuracy:  0.074
hs: 20 lr: 0.006833333333333334 bs 200 reg 1.0
hs: 20 lr: 0.006833333333333334 bs 200 reg 1.0
Validation accuracy:  0.063
hs: 20 lr: 0.006833333333333334 bs 300 reg 0.1
hs: 20 lr: 0.006833333333333334 bs 300 reg 0.1
Validation accuracy:  0.072
hs: 20 lr: 0.006833333333333334 bs 300 reg 0.55
hs: 20 lr: 0.006833333333333334 bs 300 reg 0.55
Validation accuracy:  0.046
hs: 20 lr: 0.006833333333333334 bs 300 reg 1.0
hs: 20 lr: 0.006833333333333334 bs 300 reg 1.0
Validation accuracy:  0.102
hs: 20 lr: 0.006833333333333334 bs 400 reg 0.1
hs: 20 lr: 0.006833333333333334 bs 400 reg 0.1
Validation accuracy:  0.045
hs: 20 lr: 0.006833333333333334 bs 400 reg 0.55
hs: 20 lr: 0.006833333333333334 bs 400 reg 0.55
Validation accuracy:  0.089
hs: 20 lr: 0.006833333333333334 bs 400 reg 1.0
hs: 20 lr: 0.006833333333333334 bs 400 reg 1.0
Validation accuracy:  0.083
hs: 20 lr: 0.003666666666666667 bs 200 reg 0.1
hs: 20 lr: 0.003666666666666667 bs 200 reg 0.1
Validation accuracy:  0.357
hs: 20 lr: 0.003666666666666667 bs 200 reg 0.55
hs: 20 lr: 0.003666666666666667 bs 200 reg 0.55
Validation accuracy:  0.394
hs: 20 lr: 0.003666666666666667 bs 200 reg 1.0
hs: 20 lr: 0.003666666666666667 bs 200 reg 1.0
Validation accuracy:  0.341
hs: 20 lr: 0.003666666666666667 bs 300 reg 0.1
hs: 20 lr: 0.003666666666666667 bs 300 reg 0.1
Validation accuracy:  0.369
hs: 20 lr: 0.003666666666666667 bs 300 reg 0.55
hs: 20 lr: 0.003666666666666667 bs 300 reg 0.55
Validation accuracy:  0.415
hs: 20 lr: 0.003666666666666667 bs 300 reg 1.0
hs: 20 lr: 0.003666666666666667 bs 300 reg 1.0
Validation accuracy:  0.367
hs: 20 lr: 0.003666666666666667 bs 400 reg 0.1
hs: 20 lr: 0.003666666666666667 bs 400 reg 0.1
Validation accuracy:  0.419
hs: 20 lr: 0.003666666666666667 bs 400 reg 0.55
hs: 20 lr: 0.003666666666666667 bs 400 reg 0.55
Validation accuracy:  0.425
hs: 20 lr: 0.003666666666666667 bs 400 reg 1.0
hs: 20 lr: 0.003666666666666667 bs 400 reg 1.0
Validation accuracy:  0.415
hs: 20 lr: 0.0005 bs 200 reg 0.1
```

```
hs: 20 lr: 0.0005 bs 200 reg 0.1
Validation accuracy:  0.38
hs: 20 lr: 0.0005 bs 200 reg 0.55
hs: 20 lr: 0.0005 bs 200 reg 0.55
Validation accuracy:  0.377
hs: 20 lr: 0.0005 bs 200 reg 1.0
hs: 20 lr: 0.0005 bs 200 reg 1.0
Validation accuracy:  0.387
hs: 20 lr: 0.0005 bs 300 reg 0.1
hs: 20 lr: 0.0005 bs 300 reg 0.1
Validation accuracy:  0.387
hs: 20 lr: 0.0005 bs 300 reg 0.55
hs: 20 lr: 0.0005 bs 300 reg 0.55
Validation accuracy:  0.381
hs: 20 lr: 0.0005 bs 300 reg 1.0
hs: 20 lr: 0.0005 bs 300 reg 1.0
Validation accuracy:  0.389
hs: 20 lr: 0.0005 bs 400 reg 0.1
hs: 20 lr: 0.0005 bs 400 reg 0.1
Validation accuracy:  0.386
hs: 20 lr: 0.0005 bs 400 reg 0.55
hs: 20 lr: 0.0005 bs 400 reg 0.55
Validation accuracy:  0.385
hs: 20 lr: 0.0005 bs 400 reg 1.0
hs: 20 lr: 0.0005 bs 400 reg 1.0
Validation accuracy:  0.383
hs: 60 lr: 0.01 bs 200 reg 0.1
hs: 60 lr: 0.01 bs 200 reg 0.1
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 200 reg 0.55
hs: 60 lr: 0.01 bs 200 reg 0.55
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 200 reg 1.0
hs: 60 lr: 0.01 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 300 reg 0.1
hs: 60 lr: 0.01 bs 300 reg 0.1
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 300 reg 0.55
hs: 60 lr: 0.01 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 300 reg 1.0
hs: 60 lr: 0.01 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 400 reg 0.1
hs: 60 lr: 0.01 bs 400 reg 0.1
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 400 reg 0.55
hs: 60 lr: 0.01 bs 400 reg 0.55
Validation accuracy:  0.087
hs: 60 lr: 0.01 bs 400 reg 1.0
hs: 60 lr: 0.01 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 60 lr: 0.006833333333333334 bs 200 reg 0.1
hs: 60 lr: 0.006833333333333334 bs 200 reg 0.1
Validation accuracy:  0.122
hs: 60 lr: 0.006833333333333334 bs 200 reg 0.55
hs: 60 lr: 0.006833333333333334 bs 200 reg 0.55
Validation accuracy:  0.098
hs: 60 lr: 0.006833333333333334 bs 200 reg 1.0
hs: 60 lr: 0.006833333333333334 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 60 lr: 0.006833333333333334 bs 300 reg 0.1
hs: 60 lr: 0.006833333333333334 bs 300 reg 0.1
Validation accuracy:  0.053
hs: 60 lr: 0.006833333333333334 bs 300 reg 0.55
hs: 60 lr: 0.006833333333333334 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 60 lr: 0.006833333333333334 bs 300 reg 1.0
hs: 60 lr: 0.006833333333333334 bs 300 reg 1.0
Validation accuracy:  0.06
hs: 60 lr: 0.006833333333333334 bs 400 reg 0.1
hs: 60 lr: 0.006833333333333334 bs 400 reg 0.1
Validation accuracy:  0.118
hs: 60 lr: 0.006833333333333334 bs 400 reg 0.55
hs: 60 lr: 0.006833333333333334 bs 400 reg 0.55
Validation accuracy:  0.154
```

```
hs: 60 lr: 0.006833333333333334 bs 400 reg 1.0
hs: 60 lr: 0.006833333333333334 bs 400 reg 1.0
Validation accuracy:  0.104
hs: 60 lr: 0.003666666666666667 bs 200 reg 0.1
hs: 60 lr: 0.003666666666666667 bs 200 reg 0.1
Validation accuracy:  0.395
hs: 60 lr: 0.003666666666666667 bs 200 reg 0.55
hs: 60 lr: 0.003666666666666667 bs 200 reg 0.55
Validation accuracy:  0.372
hs: 60 lr: 0.003666666666666667 bs 200 reg 1.0
hs: 60 lr: 0.003666666666666667 bs 200 reg 1.0
Validation accuracy:  0.41
hs: 60 lr: 0.003666666666666667 bs 300 reg 0.1
hs: 60 lr: 0.003666666666666667 bs 300 reg 0.1
Validation accuracy:  0.38
hs: 60 lr: 0.003666666666666667 bs 300 reg 0.55
hs: 60 lr: 0.003666666666666667 bs 300 reg 0.55
Validation accuracy:  0.405
hs: 60 lr: 0.003666666666666667 bs 300 reg 1.0
hs: 60 lr: 0.003666666666666667 bs 300 reg 1.0
Validation accuracy:  0.346
hs: 60 lr: 0.003666666666666667 bs 400 reg 0.1
hs: 60 lr: 0.003666666666666667 bs 400 reg 0.1
Validation accuracy:  0.391
hs: 60 lr: 0.003666666666666667 bs 400 reg 0.55
hs: 60 lr: 0.003666666666666667 bs 400 reg 0.55
Validation accuracy:  0.463
hs: 60 lr: 0.003666666666666667 bs 400 reg 1.0
hs: 60 lr: 0.003666666666666667 bs 400 reg 1.0
Validation accuracy:  0.446
hs: 60 lr: 0.0005 bs 200 reg 0.1
hs: 60 lr: 0.0005 bs 200 reg 0.1
Validation accuracy:  0.388
hs: 60 lr: 0.0005 bs 200 reg 0.55
hs: 60 lr: 0.0005 bs 200 reg 0.55
Validation accuracy:  0.394
hs: 60 lr: 0.0005 bs 200 reg 1.0
hs: 60 lr: 0.0005 bs 200 reg 1.0
Validation accuracy:  0.403
hs: 60 lr: 0.0005 bs 300 reg 0.1
hs: 60 lr: 0.0005 bs 300 reg 0.1
Validation accuracy:  0.401
hs: 60 lr: 0.0005 bs 300 reg 0.55
hs: 60 lr: 0.0005 bs 300 reg 0.55
Validation accuracy:  0.394
hs: 60 lr: 0.0005 bs 300 reg 1.0
hs: 60 lr: 0.0005 bs 300 reg 1.0
Validation accuracy:  0.39
hs: 60 lr: 0.0005 bs 400 reg 0.1
hs: 60 lr: 0.0005 bs 400 reg 0.1
Validation accuracy:  0.413
hs: 60 lr: 0.0005 bs 400 reg 0.55
hs: 60 lr: 0.0005 bs 400 reg 0.55
Validation accuracy:  0.402
hs: 60 lr: 0.0005 bs 400 reg 1.0
hs: 60 lr: 0.0005 bs 400 reg 1.0
Validation accuracy:  0.4
hs: 100 lr: 0.01 bs 200 reg 0.1
hs: 100 lr: 0.01 bs 200 reg 0.1
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 200 reg 0.55
hs: 100 lr: 0.01 bs 200 reg 0.55
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 200 reg 1.0
hs: 100 lr: 0.01 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 300 reg 0.1
hs: 100 lr: 0.01 bs 300 reg 0.1
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 300 reg 0.55
hs: 100 lr: 0.01 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 300 reg 1.0
hs: 100 lr: 0.01 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 400 reg 0.1
hs: 100 lr: 0.01 bs 400 reg 0.1
```

```
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 400 reg 0.55
hs: 100 lr: 0.01 bs 400 reg 0.55
Validation accuracy:  0.087
hs: 100 lr: 0.01 bs 400 reg 1.0
hs: 100 lr: 0.01 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 100 lr: 0.006833333333333334 bs 200 reg 0.1
hs: 100 lr: 0.006833333333333334 bs 200 reg 0.1
Validation accuracy:  0.04
hs: 100 lr: 0.006833333333333334 bs 200 reg 0.55
hs: 100 lr: 0.006833333333333334 bs 200 reg 0.55
Validation accuracy:  0.063
hs: 100 lr: 0.006833333333333334 bs 200 reg 1.0
hs: 100 lr: 0.006833333333333334 bs 200 reg 1.0
Validation accuracy:  0.105
hs: 100 lr: 0.006833333333333334 bs 300 reg 0.1
hs: 100 lr: 0.006833333333333334 bs 300 reg 0.1
Validation accuracy:  0.11
hs: 100 lr: 0.006833333333333334 bs 300 reg 0.55
hs: 100 lr: 0.006833333333333334 bs 300 reg 0.55
Validation accuracy:  0.121
hs: 100 lr: 0.006833333333333334 bs 300 reg 1.0
hs: 100 lr: 0.006833333333333334 bs 300 reg 1.0
Validation accuracy:  0.093
hs: 100 lr: 0.006833333333333334 bs 400 reg 0.1
hs: 100 lr: 0.006833333333333334 bs 400 reg 0.1
Validation accuracy:  0.079
hs: 100 lr: 0.006833333333333334 bs 400 reg 0.55
hs: 100 lr: 0.006833333333333334 bs 400 reg 0.55
Validation accuracy:  0.089
hs: 100 lr: 0.006833333333333334 bs 400 reg 1.0
hs: 100 lr: 0.006833333333333334 bs 400 reg 1.0
Validation accuracy:  0.08
hs: 100 lr: 0.003666666666666667 bs 200 reg 0.1
hs: 100 lr: 0.003666666666666667 bs 200 reg 0.1
Validation accuracy:  0.37
hs: 100 lr: 0.003666666666666667 bs 200 reg 0.55
hs: 100 lr: 0.003666666666666667 bs 200 reg 0.55
Validation accuracy:  0.408
hs: 100 lr: 0.003666666666666667 bs 200 reg 1.0
hs: 100 lr: 0.003666666666666667 bs 200 reg 1.0
Validation accuracy:  0.388
hs: 100 lr: 0.003666666666666667 bs 300 reg 0.1
hs: 100 lr: 0.003666666666666667 bs 300 reg 0.1
Validation accuracy:  0.422
hs: 100 lr: 0.003666666666666667 bs 300 reg 0.55
hs: 100 lr: 0.003666666666666667 bs 300 reg 0.55
Validation accuracy:  0.397
hs: 100 lr: 0.003666666666666667 bs 300 reg 1.0
hs: 100 lr: 0.003666666666666667 bs 300 reg 1.0
Validation accuracy:  0.43
hs: 100 lr: 0.003666666666666667 bs 400 reg 0.1
hs: 100 lr: 0.003666666666666667 bs 400 reg 0.1
Validation accuracy:  0.49
hs: 100 lr: 0.003666666666666667 bs 400 reg 0.55
hs: 100 lr: 0.003666666666666667 bs 400 reg 0.55
Validation accuracy:  0.406
hs: 100 lr: 0.003666666666666667 bs 400 reg 1.0
hs: 100 lr: 0.003666666666666667 bs 400 reg 1.0
Validation accuracy:  0.371
hs: 100 lr: 0.0005 bs 200 reg 0.1
hs: 100 lr: 0.0005 bs 200 reg 0.1
Validation accuracy:  0.387
hs: 100 lr: 0.0005 bs 200 reg 0.55
hs: 100 lr: 0.0005 bs 200 reg 0.55
Validation accuracy:  0.406
hs: 100 lr: 0.0005 bs 200 reg 1.0
hs: 100 lr: 0.0005 bs 200 reg 1.0
Validation accuracy:  0.404
hs: 100 lr: 0.0005 bs 300 reg 0.1
hs: 100 lr: 0.0005 bs 300 reg 0.1
Validation accuracy:  0.405
hs: 100 lr: 0.0005 bs 300 reg 0.55
hs: 100 lr: 0.0005 bs 300 reg 0.55
Validation accuracy:  0.399
hs: 100 lr: 0.0005 bs 300 reg 1.0
```

```
hs: 100 lr: 0.0005 bs 300 reg 1.0
Validation accuracy:  0.398
hs: 100 lr: 0.0005 bs 400 reg 0.1
hs: 100 lr: 0.0005 bs 400 reg 0.1
Validation accuracy:  0.393
hs: 100 lr: 0.0005 bs 400 reg 0.55
hs: 100 lr: 0.0005 bs 400 reg 0.55
Validation accuracy:  0.398
hs: 100 lr: 0.0005 bs 400 reg 1.0
hs: 100 lr: 0.0005 bs 400 reg 1.0
Validation accuracy:  0.41
hs: 150 lr: 0.01 bs 200 reg 0.1
hs: 150 lr: 0.01 bs 200 reg 0.1
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 200 reg 0.55
hs: 150 lr: 0.01 bs 200 reg 0.55
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 200 reg 1.0
hs: 150 lr: 0.01 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 300 reg 0.1
hs: 150 lr: 0.01 bs 300 reg 0.1
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 300 reg 0.55
hs: 150 lr: 0.01 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 300 reg 1.0
hs: 150 lr: 0.01 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 400 reg 0.1
hs: 150 lr: 0.01 bs 400 reg 0.1
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 400 reg 0.55
hs: 150 lr: 0.01 bs 400 reg 0.55
Validation accuracy:  0.087
hs: 150 lr: 0.01 bs 400 reg 1.0
hs: 150 lr: 0.01 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 150 lr: 0.006833333333333334 bs 200 reg 0.1
hs: 150 lr: 0.006833333333333334 bs 200 reg 0.1
Validation accuracy:  0.128
hs: 150 lr: 0.006833333333333334 bs 200 reg 0.55
hs: 150 lr: 0.006833333333333334 bs 200 reg 0.55
Validation accuracy:  0.116
hs: 150 lr: 0.006833333333333334 bs 200 reg 1.0
hs: 150 lr: 0.006833333333333334 bs 200 reg 1.0
Validation accuracy:  0.086
hs: 150 lr: 0.006833333333333334 bs 300 reg 0.1
hs: 150 lr: 0.006833333333333334 bs 300 reg 0.1
Validation accuracy:  0.054
hs: 150 lr: 0.006833333333333334 bs 300 reg 0.55
hs: 150 lr: 0.006833333333333334 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 150 lr: 0.006833333333333334 bs 300 reg 1.0
hs: 150 lr: 0.006833333333333334 bs 300 reg 1.0
Validation accuracy:  0.094
hs: 150 lr: 0.006833333333333334 bs 400 reg 0.1
hs: 150 lr: 0.006833333333333334 bs 400 reg 0.1
Validation accuracy:  0.104
hs: 150 lr: 0.006833333333333334 bs 400 reg 0.55
hs: 150 lr: 0.006833333333333334 bs 400 reg 0.55
Validation accuracy:  0.047
hs: 150 lr: 0.006833333333333334 bs 400 reg 1.0
hs: 150 lr: 0.006833333333333334 bs 400 reg 1.0
Validation accuracy:  0.083
hs: 150 lr: 0.003666666666666667 bs 200 reg 0.1
hs: 150 lr: 0.003666666666666667 bs 200 reg 0.1
Validation accuracy:  0.341
hs: 150 lr: 0.003666666666666667 bs 200 reg 0.55
hs: 150 lr: 0.003666666666666667 bs 200 reg 0.55
Validation accuracy:  0.397
hs: 150 lr: 0.003666666666666667 bs 200 reg 1.0
hs: 150 lr: 0.003666666666666667 bs 200 reg 1.0
Validation accuracy:  0.11
hs: 150 lr: 0.003666666666666667 bs 300 reg 0.1
hs: 150 lr: 0.003666666666666667 bs 300 reg 0.1
Validation accuracy:  0.442
```

```
hs: 150 lr: 0.003666666666666667 bs 300 reg 0.55
hs: 150 lr: 0.003666666666666667 bs 300 reg 0.55
Validation accuracy:  0.344
hs: 150 lr: 0.003666666666666667 bs 300 reg 1.0
hs: 150 lr: 0.003666666666666667 bs 300 reg 1.0
Validation accuracy:  0.355
hs: 150 lr: 0.003666666666666667 bs 400 reg 0.1
hs: 150 lr: 0.003666666666666667 bs 400 reg 0.1
Validation accuracy:  0.445
hs: 150 lr: 0.003666666666666667 bs 400 reg 0.55
hs: 150 lr: 0.003666666666666667 bs 400 reg 0.55
Validation accuracy:  0.423
hs: 150 lr: 0.003666666666666667 bs 400 reg 1.0
hs: 150 lr: 0.003666666666666667 bs 400 reg 1.0
Validation accuracy:  0.432
hs: 150 lr: 0.0005 bs 200 reg 0.1
hs: 150 lr: 0.0005 bs 200 reg 0.1
Validation accuracy:  0.412
hs: 150 lr: 0.0005 bs 200 reg 0.55
hs: 150 lr: 0.0005 bs 200 reg 0.55
Validation accuracy:  0.393
hs: 150 lr: 0.0005 bs 200 reg 1.0
hs: 150 lr: 0.0005 bs 200 reg 1.0
Validation accuracy:  0.405
hs: 150 lr: 0.0005 bs 300 reg 0.1
hs: 150 lr: 0.0005 bs 300 reg 0.1
Validation accuracy:  0.417
hs: 150 lr: 0.0005 bs 300 reg 0.55
hs: 150 lr: 0.0005 bs 300 reg 0.55
Validation accuracy:  0.388
hs: 150 lr: 0.0005 bs 300 reg 1.0
hs: 150 lr: 0.0005 bs 300 reg 1.0
Validation accuracy:  0.407
hs: 150 lr: 0.0005 bs 400 reg 0.1
hs: 150 lr: 0.0005 bs 400 reg 0.1
Validation accuracy:  0.41
hs: 150 lr: 0.0005 bs 400 reg 0.55
hs: 150 lr: 0.0005 bs 400 reg 0.55
Validation accuracy:  0.412
hs: 150 lr: 0.0005 bs 400 reg 1.0
hs: 150 lr: 0.0005 bs 400 reg 1.0
Validation accuracy:  0.395
hs: 180 lr: 0.01 bs 200 reg 0.1
hs: 180 lr: 0.01 bs 200 reg 0.1
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 200 reg 0.55
hs: 180 lr: 0.01 bs 200 reg 0.55
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 200 reg 1.0
hs: 180 lr: 0.01 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 300 reg 0.1
hs: 180 lr: 0.01 bs 300 reg 0.1
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 300 reg 0.55
hs: 180 lr: 0.01 bs 300 reg 0.55
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 300 reg 1.0
hs: 180 lr: 0.01 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 400 reg 0.1
hs: 180 lr: 0.01 bs 400 reg 0.1
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 400 reg 0.55
hs: 180 lr: 0.01 bs 400 reg 0.55
Validation accuracy:  0.087
hs: 180 lr: 0.01 bs 400 reg 1.0
hs: 180 lr: 0.01 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 180 lr: 0.006833333333333334 bs 200 reg 0.1
hs: 180 lr: 0.006833333333333334 bs 200 reg 0.1
Validation accuracy:  0.15
hs: 180 lr: 0.006833333333333334 bs 200 reg 0.55
hs: 180 lr: 0.006833333333333334 bs 200 reg 0.55
Validation accuracy:  0.091
hs: 180 lr: 0.006833333333333334 bs 200 reg 1.0
hs: 180 lr: 0.006833333333333334 bs 200 reg 1.0
```

```
Validation accuracy:  0.094
hs: 180 lr: 0.006833333333333334 bs 300 reg 0.1
hs: 180 lr: 0.006833333333333334 bs 300 reg 0.1
Validation accuracy:  0.078
hs: 180 lr: 0.006833333333333334 bs 300 reg 0.55
hs: 180 lr: 0.006833333333333334 bs 300 reg 0.55
Validation accuracy:  0.061
hs: 180 lr: 0.006833333333333334 bs 300 reg 1.0
hs: 180 lr: 0.006833333333333334 bs 300 reg 1.0
Validation accuracy:  0.141
hs: 180 lr: 0.006833333333333334 bs 400 reg 0.1
hs: 180 lr: 0.006833333333333334 bs 400 reg 0.1
Validation accuracy:  0.061
hs: 180 lr: 0.006833333333333334 bs 400 reg 0.55
hs: 180 lr: 0.006833333333333334 bs 400 reg 0.55
Validation accuracy:  0.143
hs: 180 lr: 0.006833333333333334 bs 400 reg 1.0
hs: 180 lr: 0.006833333333333334 bs 400 reg 1.0
Validation accuracy:  0.135
hs: 180 lr: 0.003666666666666667 bs 200 reg 0.1
hs: 180 lr: 0.003666666666666667 bs 200 reg 0.1
Validation accuracy:  0.411
hs: 180 lr: 0.003666666666666667 bs 200 reg 0.55
hs: 180 lr: 0.003666666666666667 bs 200 reg 0.55
Validation accuracy:  0.343
hs: 180 lr: 0.003666666666666667 bs 200 reg 1.0
hs: 180 lr: 0.003666666666666667 bs 200 reg 1.0
Validation accuracy:  0.374
hs: 180 lr: 0.003666666666666667 bs 300 reg 0.1
hs: 180 lr: 0.003666666666666667 bs 300 reg 0.1
Validation accuracy:  0.387
hs: 180 lr: 0.003666666666666667 bs 300 reg 0.55
hs: 180 lr: 0.003666666666666667 bs 300 reg 0.55
Validation accuracy:  0.392
hs: 180 lr: 0.003666666666666667 bs 300 reg 1.0
hs: 180 lr: 0.003666666666666667 bs 300 reg 1.0
Validation accuracy:  0.389
hs: 180 lr: 0.003666666666666667 bs 400 reg 0.1
hs: 180 lr: 0.003666666666666667 bs 400 reg 0.1
Validation accuracy:  0.448
hs: 180 lr: 0.003666666666666667 bs 400 reg 0.55
hs: 180 lr: 0.003666666666666667 bs 400 reg 0.55
Validation accuracy:  0.44
hs: 180 lr: 0.003666666666666667 bs 400 reg 1.0
hs: 180 lr: 0.003666666666666667 bs 400 reg 1.0
Validation accuracy:  0.409
hs: 180 lr: 0.0005 bs 200 reg 0.1
hs: 180 lr: 0.0005 bs 200 reg 0.1
Validation accuracy:  0.414
hs: 180 lr: 0.0005 bs 200 reg 0.55
hs: 180 lr: 0.0005 bs 200 reg 0.55
Validation accuracy:  0.401
hs: 180 lr: 0.0005 bs 200 reg 1.0
hs: 180 lr: 0.0005 bs 200 reg 1.0
Validation accuracy:  0.389
hs: 180 lr: 0.0005 bs 300 reg 0.1
hs: 180 lr: 0.0005 bs 300 reg 0.1
Validation accuracy:  0.404
hs: 180 lr: 0.0005 bs 300 reg 0.55
hs: 180 lr: 0.0005 bs 300 reg 0.55
Validation accuracy:  0.396
hs: 180 lr: 0.0005 bs 300 reg 1.0
hs: 180 lr: 0.0005 bs 300 reg 1.0
Validation accuracy:  0.404
hs: 180 lr: 0.0005 bs 400 reg 0.1
hs: 180 lr: 0.0005 bs 400 reg 0.1
Validation accuracy:  0.404
hs: 180 lr: 0.0005 bs 400 reg 0.55
hs: 180 lr: 0.0005 bs 400 reg 0.55
Validation accuracy:  0.402
hs: 180 lr: 0.0005 bs 400 reg 1.0
hs: 180 lr: 0.0005 bs 400 reg 1.0
Validation accuracy:  0.4
hs: 200 lr: 0.01 bs 200 reg 0.1
hs: 200 lr: 0.01 bs 200 reg 0.1
Validation accuracy:  0.087
hs: 200 lr: 0.01 bs 200 reg 0.55
```

hs: 200 lr: 0.01 bs 200 reg 0.55
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 200 reg 1.0
hs: 200 lr: 0.01 bs 200 reg 1.0
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 300 reg 0.1
hs: 200 lr: 0.01 bs 300 reg 0.1
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 300 reg 0.55
hs: 200 lr: 0.01 bs 300 reg 0.55
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 300 reg 1.0
hs: 200 lr: 0.01 bs 300 reg 1.0
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 400 reg 0.1
hs: 200 lr: 0.01 bs 400 reg 0.1
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 400 reg 0.55
hs: 200 lr: 0.01 bs 400 reg 0.55
Validation accuracy: 0.087
hs: 200 lr: 0.01 bs 400 reg 1.0
hs: 200 lr: 0.01 bs 400 reg 1.0
Validation accuracy: 0.087
hs: 200 lr: 0.006833333333333334 bs 200 reg 0.1
hs: 200 lr: 0.006833333333333334 bs 200 reg 0.1
Validation accuracy: 0.135
hs: 200 lr: 0.006833333333333334 bs 200 reg 0.55
hs: 200 lr: 0.006833333333333334 bs 200 reg 0.55
Validation accuracy: 0.052
hs: 200 lr: 0.006833333333333334 bs 200 reg 1.0
hs: 200 lr: 0.006833333333333334 bs 200 reg 1.0
Validation accuracy: 0.129
hs: 200 lr: 0.006833333333333334 bs 300 reg 0.1
hs: 200 lr: 0.006833333333333334 bs 300 reg 0.1
Validation accuracy: 0.087
hs: 200 lr: 0.006833333333333334 bs 300 reg 0.55
hs: 200 lr: 0.006833333333333334 bs 300 reg 0.55
Validation accuracy: 0.131
hs: 200 lr: 0.006833333333333334 bs 300 reg 1.0
hs: 200 lr: 0.006833333333333334 bs 300 reg 1.0
Validation accuracy: 0.058
hs: 200 lr: 0.006833333333333334 bs 400 reg 0.1
hs: 200 lr: 0.006833333333333334 bs 400 reg 0.1
Validation accuracy: 0.063
hs: 200 lr: 0.006833333333333334 bs 400 reg 0.55
hs: 200 lr: 0.006833333333333334 bs 400 reg 0.55
Validation accuracy: 0.116
hs: 200 lr: 0.006833333333333334 bs 400 reg 1.0
hs: 200 lr: 0.006833333333333334 bs 400 reg 1.0
Validation accuracy: 0.07
hs: 200 lr: 0.003666666666666667 bs 200 reg 0.1
hs: 200 lr: 0.003666666666666667 bs 200 reg 0.1
Validation accuracy: 0.432
hs: 200 lr: 0.003666666666666667 bs 200 reg 0.55
hs: 200 lr: 0.003666666666666667 bs 200 reg 0.55
Validation accuracy: 0.399
hs: 200 lr: 0.003666666666666667 bs 200 reg 1.0
hs: 200 lr: 0.003666666666666667 bs 200 reg 1.0
Validation accuracy: 0.372
hs: 200 lr: 0.003666666666666667 bs 300 reg 0.1
hs: 200 lr: 0.003666666666666667 bs 300 reg 0.1
Validation accuracy: 0.465
hs: 200 lr: 0.003666666666666667 bs 300 reg 0.55
hs: 200 lr: 0.003666666666666667 bs 300 reg 0.55
Validation accuracy: 0.389
hs: 200 lr: 0.003666666666666667 bs 300 reg 1.0
hs: 200 lr: 0.003666666666666667 bs 300 reg 1.0
Validation accuracy: 0.383
hs: 200 lr: 0.003666666666666667 bs 400 reg 0.1
hs: 200 lr: 0.003666666666666667 bs 400 reg 0.1
Validation accuracy: 0.425
hs: 200 lr: 0.003666666666666667 bs 400 reg 0.55
hs: 200 lr: 0.003666666666666667 bs 400 reg 0.55
Validation accuracy: 0.405
hs: 200 lr: 0.003666666666666667 bs 400 reg 1.0
hs: 200 lr: 0.003666666666666667 bs 400 reg 1.0
Validation accuracy: 0.425

```
hs: 200 lr: 0.0005 bs 200 reg 0.1
hs: 200 lr: 0.0005 bs 200 reg 0.1
Validation accuracy:  0.399
hs: 200 lr: 0.0005 bs 200 reg 0.55
hs: 200 lr: 0.0005 bs 200 reg 0.55
Validation accuracy:  0.402
hs: 200 lr: 0.0005 bs 200 reg 1.0
hs: 200 lr: 0.0005 bs 200 reg 1.0
Validation accuracy:  0.409
hs: 200 lr: 0.0005 bs 300 reg 0.1
hs: 200 lr: 0.0005 bs 300 reg 0.1
Validation accuracy:  0.423
hs: 200 lr: 0.0005 bs 300 reg 0.55
hs: 200 lr: 0.0005 bs 300 reg 0.55
Validation accuracy:  0.402
hs: 200 lr: 0.0005 bs 300 reg 1.0
hs: 200 lr: 0.0005 bs 300 reg 1.0
Validation accuracy:  0.407
hs: 200 lr: 0.0005 bs 400 reg 0.1
hs: 200 lr: 0.0005 bs 400 reg 0.1
Validation accuracy:  0.412
hs: 200 lr: 0.0005 bs 400 reg 0.55
hs: 200 lr: 0.0005 bs 400 reg 0.55
Validation accuracy:  0.407
hs: 200 lr: 0.0005 bs 400 reg 1.0
hs: 200 lr: 0.0005 bs 400 reg 1.0
Validation accuracy:  0.393

** best validation achieved during cross validation:  0.49
```

In [13]:

```
# visualize the weights of the best network
show_net_weights(best_net)
```

Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set; you should get above 48%.

In [17]:

```
# winning hyperparameters
print ("best hypeparameters ->  hidden ", best_hidd, ' lr ', best_lr, ' bs', best_batch_size, ' reg
', best_reg)
# add validation accuracy just for clarity for below inline question
val_acc = (best_net.predict(X_val) == y_val).mean()
print ("Best net Validation accuracy :", val_acc)

test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
best hypeparameters ->  hidden  100  lr  0.003666666666666667  bs 400  reg  0.1
Best net Validation accuracy : 0.49
Test accuracy:  0.476
```

Inline Question

Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

Your answer: Increase the regulization strength

Your explanation: The low regulization strength leads to highly likely overfitting of the model on the validation set, so decreasing it makes the model more general. With the trained network, tweaking the model based on the validation set leads to overfitting, as it is not a validation set anymore, as it effectively becomes part of the training set, albeit in a somewhat obfuscated way.

In [ ]: