two_layer_net

September 27, 2019

1 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

        import numpy as np
        import matplotlib.pyplot as plt

        %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))))
```

The neural network parameters will be stored in a dictionary (`model` below), where the keys are the parameter names and the values are numpy arrays. Below, we initialize toy data and a toy model that we will use to verify your implementations.

```
In [2]: # Create some toy data to check your implementations
        input_size = 4
        hidden_size = 10
        num_classes = 3
        num_inputs = 5

        def init_toy_model():
          model = {}
          model['W1'] = np.linspace(-0.2, 0.6, num=input_size*hidden_size).reshape(input_size, h
```

```
  model['b1'] = np.linspace(-0.3, 0.7, num=hidden_size)
  model['W2'] = np.linspace(-0.4, 0.1, num=hidden_size*num_classes).reshape(hidden_size,
  model['b2'] = np.linspace(-0.5, 0.9, num=num_classes)
  return model

def init_toy_data():
  X = np.linspace(-0.2, 0.5, num=num_inputs*input_size).reshape(num_inputs, input_size)
  y = np.array([0, 1, 2, 2, 1])
  return X, y

model = init_toy_model()
X, y = init_toy_data()
```

2 Forward pass: compute scores

Open the file cs231n/classifiers/neural_net.py and look at the function two_layer_net. This function is very similar to the loss functions you have written for the Softmax exercise in HW0: 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 [3]: from cs231n.classifiers.neural_net import two_layer_net

        scores = two_layer_net(X, model)
        print(scores)
        correct_scores = [[-0.5328368, 0.20031504, 0.93346689],
         [-0.59412164, 0.15498488, 0.9040914 ],
         [-0.67658362, 0.08978957, 0.85616275],
         [-0.77092643, 0.01339997, 0.79772637],
         [-0.89110401, -0.08754544, 0.71601312]]

        # the difference should be very small. We get 3e-8
        print('Difference between your scores and correct scores:')
        print(np.sum(np.abs(scores - correct_scores)))
```

```
[[-0.5328368   0.20031504  0.93346689]
 [-0.59412164  0.15498488  0.9040914 ]
 [-0.67658362  0.08978957  0.85616275]
 [-0.77092643  0.01339997  0.79772637]
 [-0.89110401 -0.08754544  0.71601312]]
Difference between your scores and correct scores:
3.848682278081994e-08
```

3 Forward pass: compute loss

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

```
In [4]: reg = 0.1
        loss, _ = two_layer_net(X, model, y, reg)
        correct_loss = 1.38191946092

        # should be very small, we get 5e-12
        print('Difference between your loss and correct loss:')
        print(np.sum(np.abs(loss - correct_loss)))
```

```
Difference between your loss and correct loss:
3.605849741239453e-06
```

4 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 [5]: 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 = two_layer_net(X, model, y, reg)

        # these should all be less than 1e-8 or so
        for param_name in grads:
          param_grad_num = eval_numerical_gradient(lambda W: two_layer_net(X, model, y, reg)[0],
          print('%s max relative error: %e' % (param_name, rel_error(param_grad_num, grads[param
```

```
W2 max relative error: 5.260597e-05
b2 max relative error: 6.317662e-06
W1 max relative error: 2.493902e-05
b1 max relative error: 1.449991e-03
```

5 Train the network

To train the network we will use SGD with Momentum. Last assignment you implemented vanilla SGD. You will now implement the momentum update and the RMSProp update. Open the file `classifier_trainer.py` and familiarize yourself with the `ClassifierTrainer` class. It performs optimization given an arbitrary cost function data, and model. By default it uses vanilla SGD, which we have already implemented for you. First, run the optimization below using Vanilla SGD:

```
In [6]: from cs231n.classifier_trainer import ClassifierTrainer
```

```
        model = init_toy_model()
        trainer = ClassifierTrainer()
        # call the trainer to optimize the loss
        # Notice that we're using sample_batches=False, so we're performing Gradient Descent (no
        best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                             model, two_layer_net,
                                             reg=0.001,
                                             learning_rate=1e-1, momentum=0.0, learning_
                                             update='sgd', sample_batches=False,
                                             num_epochs=100,
                                             verbose=False)
        print('Final loss with vanilla SGD: %f' % (loss_history[-1], ))
```

```
starting iteration  0
Final loss with vanilla SGD: 0.940683
```

Now fill in the **momentum update** in the first missing code block inside the `train` function, and run the same optimization as above but with the momentum update. You should see a much better result in the final obtained loss:

```
In [7]: model = init_toy_model()
        trainer = ClassifierTrainer()
        # call the trainer to optimize the loss
        # Notice that we're using sample_batches=False, so we're performing Gradient Descent (no
        best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                             model, two_layer_net,
                                             reg=0.001,
                                             learning_rate=1e-1, momentum=0.9, learning_
                                             update='momentum', sample_batches=False,
                                             num_epochs=100,
                                             verbose=False)
        correct_loss = 0.494394
        print('Final loss with momentum SGD: %f. We get: %f' % (loss_history[-1], correct_loss))
```

```
starting iteration  0
Final loss with momentum SGD: 0.494391. We get: 0.494394
```

The **RMSProp** update step is given as follows:

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-8)
```

Here, `decay_rate` is a hyperparameter and typical values are [0.9, 0.99, 0.999].
Implement the **RMSProp** update rule inside the `train` function and rerun the optimization:

```
In [8]: model = init_toy_model()
        trainer = ClassifierTrainer()
```

```
        # call the trainer to optimize the loss
        # Notice that we're using sample_batches=False, so we're performing Gradient Descent (no
        best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                                      model, two_layer_net,
                                                      reg=0.001,
                                                      learning_rate=1e-1, momentum=0.9, learning_
                                                      update='rmsprop', sample_batches=False,
                                                      num_epochs=100,
                                                      verbose=False)
        correct_loss = 0.439368
        print('Final loss with RMSProp: %f. We get: %f' % (loss_history[-1], correct_loss))
```

```
starting iteration  0
Final loss with RMSProp: 0.439363. We get: 0.439368
```

6 Load the data

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

```
In [9]: 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.
            """
            # 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 = range(num_training, num_training + num_validation)
            X_val = X_train[mask]
            y_val = y_train[mask]
            mask = range(num_training)
            X_train = X_train[mask]
            y_train = y_train[mask]
            mask = 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


# 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,)
```

7 Train a network

To train our network we will use SGD with momentum. 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 [10]: from cs231n.classifiers.neural_net import init_two_layer_model

         model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of clas
         trainer = ClassifierTrainer()
         best_model, loss_history, train_acc, val_acc = trainer.train(X_train, y_train, X_val, y
                                                      model, two_layer_net,
                                                      num_epochs=5, reg=1.0,
                                                      momentum=0.9, learning_rate_decay = 0.95,
                                                      learning_rate=1e-5, verbose=False)
```

```
starting iteration  0
starting iteration  100
starting iteration  200
starting iteration  300
starting iteration  400
starting iteration  500
```

```
starting iteration  600
starting iteration  700
starting iteration  800
starting iteration  900
starting iteration  1000
starting iteration  1100
starting iteration  1200
starting iteration  1300
starting iteration  1400
starting iteration  1500
starting iteration  1600
starting iteration  1700
starting iteration  1800
starting iteration  1900
starting iteration  2000
starting iteration  2100
starting iteration  2200
starting iteration  2300
starting iteration  2400
```

8 Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.37 on the validation set. This isn't very good.

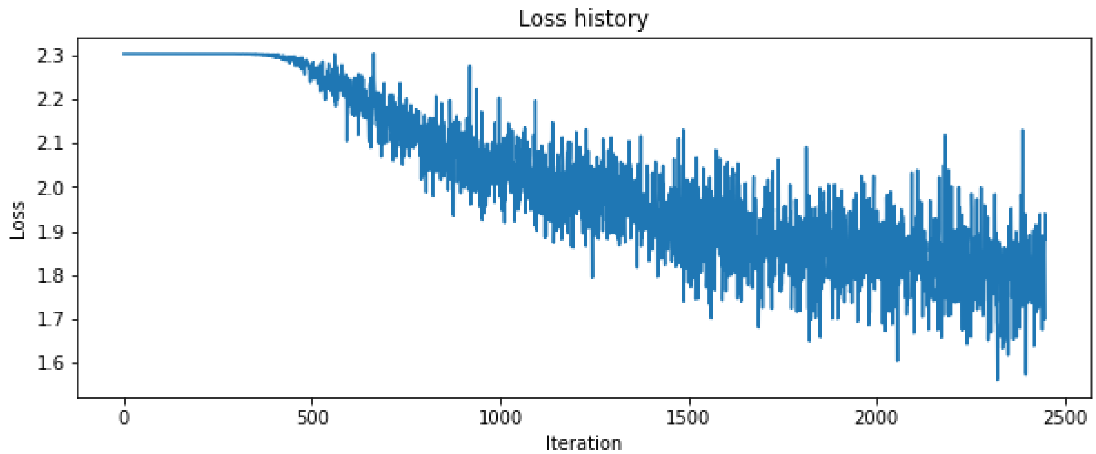
One strategy for getting insight into what's wrong is to plot the loss function and the accuracies on the training and validation sets during optimization.

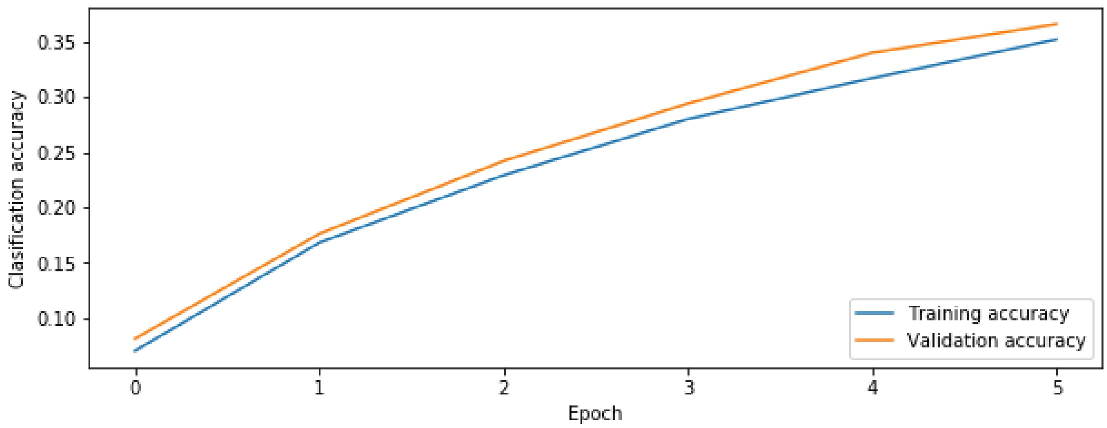
Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

```
In [11]: # Plot the loss function and train / validation accuracies
         plt.subplot(2, 1, 1)
         plt.plot(loss_history)
         plt.title('Loss history')
         plt.xlabel('Iteration')
         plt.ylabel('Loss')

         plt.subplot(2, 1, 2)
         plt.plot(train_acc)
         plt.plot(val_acc)
         plt.legend(['Training accuracy', 'Validation accuracy'], loc='lower right')
         plt.xlabel('Epoch')
         plt.ylabel('Clasification accuracy')
```

```
Out[11]: Text(0,0.5,'Clasification accuracy')
```



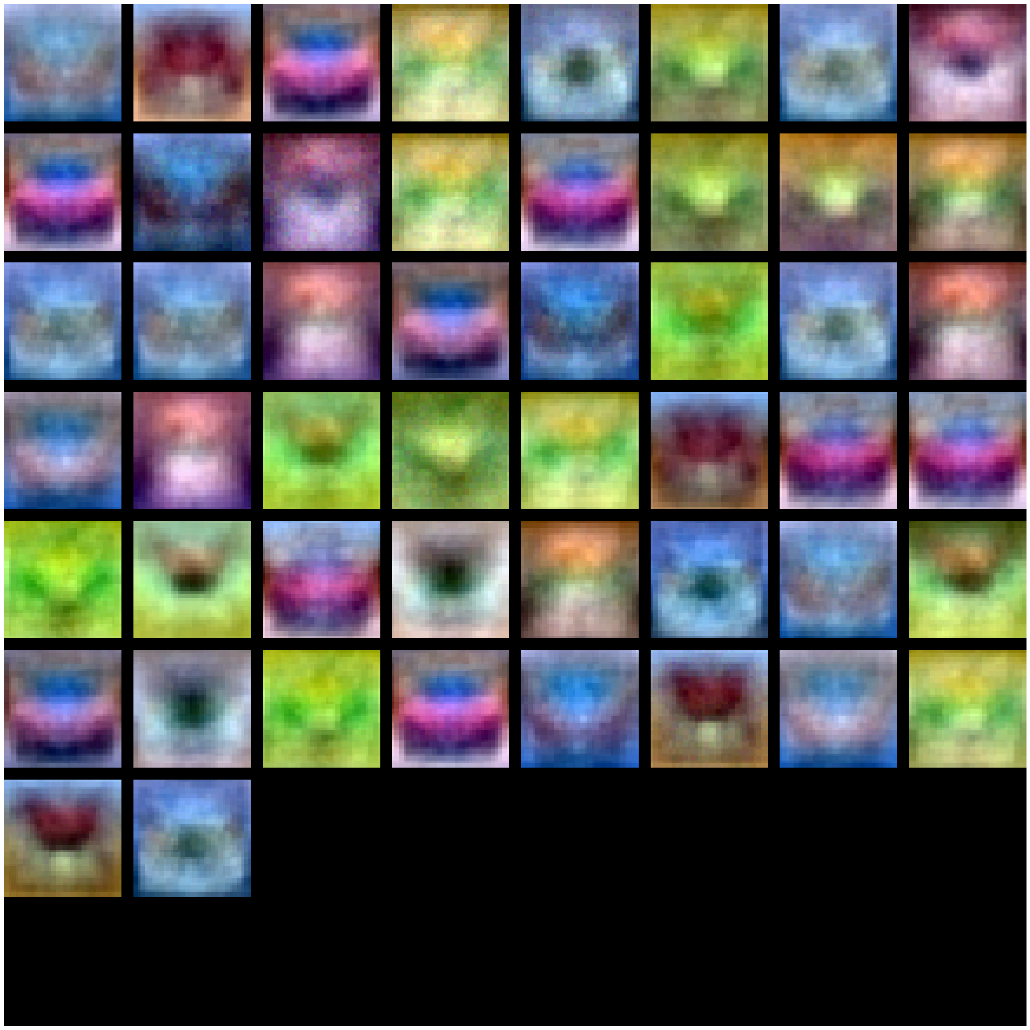



```
In [12]: from cs231n.vis_utils import visualize_grid

         # Visualize the weights of the network

         def show_net_weights(model):
             plt.imshow(visualize_grid(model['W1'].T.reshape(-1, 32, 32, 3), padding=3).astype('
             plt.gca().axis('off')
             plt.show()

         show_net_weights(model)
```



9 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 momentum and learning rate decay parameters, but you should be able to get good performance using the default values.

Approximate results. You should be aim to achieve a classification accuracy of greater than 50% on the validation set. Our best network gets over 56% 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. For every 1% above 56% on the Test set we will award you with one extra bonus point. Feel free implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.).

hyperparameters cell

1. Test = 41 num_epochs=10, reg=1, momentum=0.9, learning_rate_decay=0.95, learning_rate=1e-5, verbose=True)

2. 48.6

```
num_epochs=50, reg=1,
momentum=0.9,
learning_rate_decay=0.95,
learning_rate=1e-5, verbose=True)
```

In [19]:
```
best_model = None # store the best model into this

#################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained  #
# model in best_model.                                                          #
#                                                                               #
# 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 assignment.                         #
#################################################################################
# input size, hidden size, number of classes
model = init_two_layer_model(32*32*3, 100, 10)
trainer = ClassifierTrainer()
best_model, loss_history, train_acc, val_acc = trainer.train(X_train, y_train,
                                                             X_val, y_val,
                                                             model, two_layer_net,
                                                             num_epochs=150, reg=1,
                                                             momentum=0.9,
                                                             learning_rate_decay=0.95,
                                                             learning_rate=1e-5, verbose=True)
#################################################################################
#                                END OF YOUR CODE                               #
#################################################################################
```

```
starting iteration  0
Finished epoch 0 / 150: cost 2.302591, train: 0.106000, val 0.071000, lr 1.000000e-05
starting iteration  100
starting iteration  200
starting iteration  300
starting iteration  400
Finished epoch 1 / 150: cost 2.230374, train: 0.185000, val 0.179000, lr 9.500000e-06
starting iteration  500
starting iteration  600
starting iteration  700
starting iteration  800
starting iteration  900
Finished epoch 2 / 150: cost 2.061289, train: 0.235000, val 0.244000, lr 9.025000e-06
starting iteration  1000
starting iteration  1100
starting iteration  1200
starting iteration  1300
starting iteration  1400
Finished epoch 3 / 150: cost 1.927104, train: 0.291000, val 0.298000, lr 8.573750e-06
starting iteration  1500
starting iteration  1600
starting iteration  1700
starting iteration  1800
starting iteration  1900
Finished epoch 4 / 150: cost 1.875031, train: 0.339000, val 0.337000, lr 8.145063e-06
starting iteration  2000
starting iteration  2100
starting iteration  2200
starting iteration  2300
starting iteration  2400
Finished epoch 5 / 150: cost 1.707858, train: 0.324000, val 0.368000, lr 7.737809e-06
starting iteration  2500
starting iteration  2600
starting iteration  2700
starting iteration  2800
starting iteration  2900
Finished epoch 6 / 150: cost 1.747641, train: 0.364000, val 0.380000, lr 7.350919e-06
starting iteration  3000
starting iteration  3100
starting iteration  3200
starting iteration  3300
starting iteration  3400
Finished epoch 7 / 150: cost 1.854529, train: 0.420000, val 0.385000, lr 6.983373e-06
starting iteration  3500
starting iteration  3600
starting iteration  3700
starting iteration  3800
starting iteration  3900
```

```
Finished epoch 8 / 150: cost 1.707482, train: 0.401000, val 0.405000, lr 6.634204e-06
starting iteration  4000
starting iteration  4100
starting iteration  4200
starting iteration  4300
starting iteration  4400
Finished epoch 9 / 150: cost 1.693510, train: 0.401000, val 0.412000, lr 6.302494e-06
starting iteration  4500
starting iteration  4600
starting iteration  4700
starting iteration  4800
Finished epoch 10 / 150: cost 1.607555, train: 0.422000, val 0.433000, lr 5.987369e-06
starting iteration  4900
starting iteration  5000
starting iteration  5100
starting iteration  5200
starting iteration  5300
Finished epoch 11 / 150: cost 1.753437, train: 0.423000, val 0.437000, lr 5.688001e-06
starting iteration  5400
starting iteration  5500
starting iteration  5600
starting iteration  5700
starting iteration  5800
Finished epoch 12 / 150: cost 1.541589, train: 0.450000, val 0.446000, lr 5.403601e-06
starting iteration  5900
starting iteration  6000
starting iteration  6100
starting iteration  6200
starting iteration  6300
Finished epoch 13 / 150: cost 1.682181, train: 0.443000, val 0.445000, lr 5.133421e-06
starting iteration  6400
starting iteration  6500
starting iteration  6600
starting iteration  6700
starting iteration  6800
Finished epoch 14 / 150: cost 1.527999, train: 0.463000, val 0.457000, lr 4.876750e-06
starting iteration  6900
starting iteration  7000
starting iteration  7100
starting iteration  7200
starting iteration  7300
Finished epoch 15 / 150: cost 1.552910, train: 0.469000, val 0.455000, lr 4.632912e-06
starting iteration  7400
starting iteration  7500
starting iteration  7600
starting iteration  7700
starting iteration  7800
Finished epoch 16 / 150: cost 1.402946, train: 0.474000, val 0.463000, lr 4.401267e-06
```

```
starting iteration  7900
starting iteration  8000
starting iteration  8100
starting iteration  8200
starting iteration  8300
Finished epoch 17 / 150: cost 1.644635, train: 0.470000, val 0.455000, lr 4.181203e-06
starting iteration  8400
starting iteration  8500
starting iteration  8600
starting iteration  8700
starting iteration  8800
Finished epoch 18 / 150: cost 1.615797, train: 0.452000, val 0.461000, lr 3.972143e-06
starting iteration  8900
starting iteration  9000
starting iteration  9100
starting iteration  9200
starting iteration  9300
Finished epoch 19 / 150: cost 1.479390, train: 0.460000, val 0.473000, lr 3.773536e-06
starting iteration  9400
starting iteration  9500
starting iteration  9600
starting iteration  9700
Finished epoch 20 / 150: cost 1.526531, train: 0.497000, val 0.466000, lr 3.584859e-06
starting iteration  9800
starting iteration  9900
starting iteration  10000
starting iteration  10100
starting iteration  10200
Finished epoch 21 / 150: cost 1.515938, train: 0.464000, val 0.470000, lr 3.405616e-06
starting iteration  10300
starting iteration  10400
starting iteration  10500
starting iteration  10600
starting iteration  10700
Finished epoch 22 / 150: cost 1.551063, train: 0.471000, val 0.469000, lr 3.235335e-06
starting iteration  10800
starting iteration  10900
starting iteration  11000
starting iteration  11100
starting iteration  11200
Finished epoch 23 / 150: cost 1.622964, train: 0.494000, val 0.474000, lr 3.073569e-06
starting iteration  11300
starting iteration  11400
starting iteration  11500
starting iteration  11600
starting iteration  11700
Finished epoch 24 / 150: cost 1.496666, train: 0.475000, val 0.472000, lr 2.919890e-06
starting iteration  11800
```

```
starting iteration  11900
starting iteration  12000
starting iteration  12100
starting iteration  12200
Finished epoch 25 / 150: cost 1.400345, train: 0.484000, val 0.474000, lr 2.773896e-06
starting iteration  12300
starting iteration  12400
starting iteration  12500
starting iteration  12600
starting iteration  12700
Finished epoch 26 / 150: cost 1.635315, train: 0.495000, val 0.470000, lr 2.635201e-06
starting iteration  12800
starting iteration  12900
starting iteration  13000
starting iteration  13100
starting iteration  13200
Finished epoch 27 / 150: cost 1.764031, train: 0.495000, val 0.471000, lr 2.503441e-06
starting iteration  13300
starting iteration  13400
starting iteration  13500
starting iteration  13600
starting iteration  13700
Finished epoch 28 / 150: cost 1.553671, train: 0.498000, val 0.473000, lr 2.378269e-06
starting iteration  13800
starting iteration  13900
starting iteration  14000
starting iteration  14100
starting iteration  14200
Finished epoch 29 / 150: cost 1.512731, train: 0.481000, val 0.486000, lr 2.259355e-06
starting iteration  14300
starting iteration  14400
starting iteration  14500
starting iteration  14600
Finished epoch 30 / 150: cost 1.425067, train: 0.499000, val 0.477000, lr 2.146388e-06
starting iteration  14700
starting iteration  14800
starting iteration  14900
starting iteration  15000
starting iteration  15100
Finished epoch 31 / 150: cost 1.568940, train: 0.484000, val 0.481000, lr 2.039068e-06
starting iteration  15200
starting iteration  15300
starting iteration  15400
starting iteration  15500
starting iteration  15600
Finished epoch 32 / 150: cost 1.515593, train: 0.519000, val 0.480000, lr 1.937115e-06
starting iteration  15700
starting iteration  15800
```

```
starting iteration  15900
starting iteration  16000
starting iteration  16100
Finished epoch 33 / 150: cost 1.590110, train: 0.484000, val 0.483000, lr 1.840259e-06
starting iteration  16200
starting iteration  16300
starting iteration  16400
starting iteration  16500
starting iteration  16600
Finished epoch 34 / 150: cost 1.564076, train: 0.505000, val 0.477000, lr 1.748246e-06
starting iteration  16700
starting iteration  16800
starting iteration  16900
starting iteration  17000
starting iteration  17100
Finished epoch 35 / 150: cost 1.421460, train: 0.492000, val 0.481000, lr 1.660834e-06
starting iteration  17200
starting iteration  17300
starting iteration  17400
starting iteration  17500
starting iteration  17600
Finished epoch 36 / 150: cost 1.548828, train: 0.505000, val 0.479000, lr 1.577792e-06
starting iteration  17700
starting iteration  17800
starting iteration  17900
starting iteration  18000
starting iteration  18100
Finished epoch 37 / 150: cost 1.515039, train: 0.498000, val 0.480000, lr 1.498903e-06
starting iteration  18200
starting iteration  18300
starting iteration  18400
starting iteration  18500
starting iteration  18600
Finished epoch 38 / 150: cost 1.535161, train: 0.518000, val 0.479000, lr 1.423957e-06
starting iteration  18700
starting iteration  18800
starting iteration  18900
starting iteration  19000
starting iteration  19100
Finished epoch 39 / 150: cost 1.450373, train: 0.515000, val 0.478000, lr 1.352760e-06
starting iteration  19200
starting iteration  19300
starting iteration  19400
starting iteration  19500
Finished epoch 40 / 150: cost 1.762823, train: 0.508000, val 0.482000, lr 1.285122e-06
starting iteration  19600
starting iteration  19700
starting iteration  19800
```

```
starting iteration  19900
starting iteration  20000
Finished epoch 41 / 150: cost 1.496068, train: 0.512000, val 0.485000, lr 1.220865e-06
starting iteration  20100
starting iteration  20200
starting iteration  20300
starting iteration  20400
starting iteration  20500
Finished epoch 42 / 150: cost 1.476640, train: 0.507000, val 0.484000, lr 1.159822e-06
starting iteration  20600
starting iteration  20700
starting iteration  20800
starting iteration  20900
starting iteration  21000
Finished epoch 43 / 150: cost 1.585615, train: 0.511000, val 0.487000, lr 1.101831e-06
starting iteration  21100
starting iteration  21200
starting iteration  21300
starting iteration  21400
starting iteration  21500
Finished epoch 44 / 150: cost 1.521527, train: 0.476000, val 0.483000, lr 1.046740e-06
starting iteration  21600
starting iteration  21700
starting iteration  21800
starting iteration  21900
starting iteration  22000
Finished epoch 45 / 150: cost 1.487731, train: 0.510000, val 0.485000, lr 9.944026e-07
starting iteration  22100
starting iteration  22200
starting iteration  22300
starting iteration  22400
starting iteration  22500
Finished epoch 46 / 150: cost 1.595450, train: 0.512000, val 0.481000, lr 9.446824e-07
starting iteration  22600
starting iteration  22700
starting iteration  22800
starting iteration  22900
starting iteration  23000
Finished epoch 47 / 150: cost 1.501601, train: 0.523000, val 0.485000, lr 8.974483e-07
starting iteration  23100
starting iteration  23200
starting iteration  23300
starting iteration  23400
starting iteration  23500
Finished epoch 48 / 150: cost 1.537054, train: 0.493000, val 0.485000, lr 8.525759e-07
starting iteration  23600
starting iteration  23700
starting iteration  23800
```

```
starting iteration  23900
starting iteration  24000
Finished epoch 49 / 150: cost 1.642618, train: 0.511000, val 0.482000, lr 8.099471e-07
starting iteration  24100
starting iteration  24200
starting iteration  24300
starting iteration  24400
Finished epoch 50 / 150: cost 1.540092, train: 0.510000, val 0.484000, lr 7.694498e-07
starting iteration  24500
starting iteration  24600
starting iteration  24700
starting iteration  24800
starting iteration  24900
Finished epoch 51 / 150: cost 1.589463, train: 0.516000, val 0.482000, lr 7.309773e-07
starting iteration  25000
starting iteration  25100
starting iteration  25200
starting iteration  25300
starting iteration  25400
Finished epoch 52 / 150: cost 1.565420, train: 0.503000, val 0.475000, lr 6.944284e-07
starting iteration  25500
starting iteration  25600
starting iteration  25700
starting iteration  25800
starting iteration  25900
Finished epoch 53 / 150: cost 1.611238, train: 0.527000, val 0.479000, lr 6.597070e-07
starting iteration  26000
starting iteration  26100
starting iteration  26200
starting iteration  26300
starting iteration  26400
Finished epoch 54 / 150: cost 1.528771, train: 0.520000, val 0.484000, lr 6.267216e-07
starting iteration  26500
starting iteration  26600
starting iteration  26700
starting iteration  26800
starting iteration  26900
Finished epoch 55 / 150: cost 1.509801, train: 0.465000, val 0.481000, lr 5.953856e-07
starting iteration  27000
starting iteration  27100
starting iteration  27200
starting iteration  27300
starting iteration  27400
Finished epoch 56 / 150: cost 1.498746, train: 0.515000, val 0.481000, lr 5.656163e-07
starting iteration  27500
starting iteration  27600
starting iteration  27700
starting iteration  27800
```

```
starting iteration  27900
Finished epoch 57 / 150: cost 1.455091, train: 0.491000, val 0.484000, lr 5.373355e-07
starting iteration  28000
starting iteration  28100
starting iteration  28200
starting iteration  28300
starting iteration  28400
Finished epoch 58 / 150: cost 1.534273, train: 0.503000, val 0.490000, lr 5.104687e-07
starting iteration  28500
starting iteration  28600
starting iteration  28700
starting iteration  28800
starting iteration  28900
Finished epoch 59 / 150: cost 1.340238, train: 0.494000, val 0.489000, lr 4.849453e-07
starting iteration  29000
starting iteration  29100
starting iteration  29200
starting iteration  29300
Finished epoch 60 / 150: cost 1.375773, train: 0.502000, val 0.486000, lr 4.606980e-07
starting iteration  29400
starting iteration  29500
starting iteration  29600
starting iteration  29700
starting iteration  29800
Finished epoch 61 / 150: cost 1.585073, train: 0.527000, val 0.485000, lr 4.376631e-07
starting iteration  29900
starting iteration  30000
starting iteration  30100
starting iteration  30200
starting iteration  30300
Finished epoch 62 / 150: cost 1.567944, train: 0.516000, val 0.487000, lr 4.157799e-07
starting iteration  30400
starting iteration  30500
starting iteration  30600
starting iteration  30700
starting iteration  30800
Finished epoch 63 / 150: cost 1.570394, train: 0.514000, val 0.486000, lr 3.949909e-07
starting iteration  30900
starting iteration  31000
starting iteration  31100
starting iteration  31200
starting iteration  31300
Finished epoch 64 / 150: cost 1.522651, train: 0.536000, val 0.483000, lr 3.752414e-07
starting iteration  31400
starting iteration  31500
starting iteration  31600
starting iteration  31700
starting iteration  31800
```

```
Finished epoch 65 / 150: cost 1.529298, train: 0.493000, val 0.483000, lr 3.564793e-07
starting iteration  31900
starting iteration  32000
starting iteration  32100
starting iteration  32200
starting iteration  32300
Finished epoch 66 / 150: cost 1.537794, train: 0.488000, val 0.483000, lr 3.386554e-07
starting iteration  32400
starting iteration  32500
starting iteration  32600
starting iteration  32700
starting iteration  32800
Finished epoch 67 / 150: cost 1.518446, train: 0.526000, val 0.489000, lr 3.217226e-07
starting iteration  32900
starting iteration  33000
starting iteration  33100
starting iteration  33200
starting iteration  33300
Finished epoch 68 / 150: cost 1.491950, train: 0.533000, val 0.484000, lr 3.056365e-07
starting iteration  33400
starting iteration  33500
starting iteration  33600
starting iteration  33700
starting iteration  33800
Finished epoch 69 / 150: cost 1.419251, train: 0.529000, val 0.482000, lr 2.903546e-07
starting iteration  33900
starting iteration  34000
starting iteration  34100
starting iteration  34200
Finished epoch 70 / 150: cost 1.539193, train: 0.519000, val 0.484000, lr 2.758369e-07
starting iteration  34300
starting iteration  34400
starting iteration  34500
starting iteration  34600
starting iteration  34700
Finished epoch 71 / 150: cost 1.519022, train: 0.487000, val 0.487000, lr 2.620451e-07
starting iteration  34800
starting iteration  34900
starting iteration  35000
starting iteration  35100
starting iteration  35200
Finished epoch 72 / 150: cost 1.305177, train: 0.515000, val 0.487000, lr 2.489428e-07
starting iteration  35300
starting iteration  35400
starting iteration  35500
starting iteration  35600
starting iteration  35700
Finished epoch 73 / 150: cost 1.615088, train: 0.515000, val 0.485000, lr 2.364957e-07
```

```
starting iteration  35800
starting iteration  35900
starting iteration  36000
starting iteration  36100
starting iteration  36200
Finished epoch 74 / 150: cost 1.663850, train: 0.519000, val 0.486000, lr 2.246709e-07
starting iteration  36300
starting iteration  36400
starting iteration  36500
starting iteration  36600
starting iteration  36700
Finished epoch 75 / 150: cost 1.704994, train: 0.494000, val 0.490000, lr 2.134373e-07
starting iteration  36800
starting iteration  36900
starting iteration  37000
starting iteration  37100
starting iteration  37200
Finished epoch 76 / 150: cost 1.420432, train: 0.500000, val 0.482000, lr 2.027655e-07
starting iteration  37300
starting iteration  37400
starting iteration  37500
starting iteration  37600
starting iteration  37700
Finished epoch 77 / 150: cost 1.528244, train: 0.496000, val 0.489000, lr 1.926272e-07
starting iteration  37800
starting iteration  37900
starting iteration  38000
starting iteration  38100
starting iteration  38200
Finished epoch 78 / 150: cost 1.698223, train: 0.518000, val 0.491000, lr 1.829958e-07
starting iteration  38300
starting iteration  38400
starting iteration  38500
starting iteration  38600
starting iteration  38700
Finished epoch 79 / 150: cost 1.428640, train: 0.508000, val 0.488000, lr 1.738460e-07
starting iteration  38800
starting iteration  38900
starting iteration  39000
starting iteration  39100
Finished epoch 80 / 150: cost 1.472300, train: 0.515000, val 0.490000, lr 1.651537e-07
starting iteration  39200
starting iteration  39300
starting iteration  39400
starting iteration  39500
starting iteration  39600
Finished epoch 81 / 150: cost 1.503550, train: 0.511000, val 0.485000, lr 1.568961e-07
starting iteration  39700
```

```
starting iteration  39800
starting iteration  39900
starting iteration  40000
starting iteration  40100
Finished epoch 82 / 150: cost 1.530307, train: 0.509000, val 0.487000, lr 1.490513e-07
starting iteration  40200
starting iteration  40300
starting iteration  40400
starting iteration  40500
starting iteration  40600
Finished epoch 83 / 150: cost 1.551121, train: 0.528000, val 0.485000, lr 1.415987e-07
starting iteration  40700
starting iteration  40800
starting iteration  40900
starting iteration  41000
starting iteration  41100
Finished epoch 84 / 150: cost 1.486826, train: 0.534000, val 0.489000, lr 1.345188e-07
starting iteration  41200
starting iteration  41300
starting iteration  41400
starting iteration  41500
starting iteration  41600
Finished epoch 85 / 150: cost 1.417320, train: 0.499000, val 0.488000, lr 1.277928e-07
starting iteration  41700
starting iteration  41800
starting iteration  41900
starting iteration  42000
starting iteration  42100
Finished epoch 86 / 150: cost 1.563426, train: 0.521000, val 0.491000, lr 1.214032e-07
starting iteration  42200
starting iteration  42300
starting iteration  42400
starting iteration  42500
starting iteration  42600
Finished epoch 87 / 150: cost 1.463056, train: 0.499000, val 0.489000, lr 1.153330e-07
starting iteration  42700
starting iteration  42800
starting iteration  42900
starting iteration  43000
starting iteration  43100
Finished epoch 88 / 150: cost 1.545913, train: 0.530000, val 0.490000, lr 1.095664e-07
starting iteration  43200
starting iteration  43300
starting iteration  43400
starting iteration  43500
starting iteration  43600
Finished epoch 89 / 150: cost 1.352506, train: 0.508000, val 0.490000, lr 1.040880e-07
starting iteration  43700
```

```
starting iteration  43800
starting iteration  43900
starting iteration  44000
Finished epoch 90 / 150: cost 1.407770, train: 0.499000, val 0.490000, lr 9.888365e-08
starting iteration  44100
starting iteration  44200
starting iteration  44300
starting iteration  44400
starting iteration  44500
Finished epoch 91 / 150: cost 1.454014, train: 0.533000, val 0.490000, lr 9.393946e-08
starting iteration  44600
starting iteration  44700
starting iteration  44800
starting iteration  44900
starting iteration  45000
Finished epoch 92 / 150: cost 1.585857, train: 0.505000, val 0.486000, lr 8.924249e-08
starting iteration  45100
starting iteration  45200
starting iteration  45300
starting iteration  45400
starting iteration  45500
Finished epoch 93 / 150: cost 1.725863, train: 0.521000, val 0.489000, lr 8.478037e-08
starting iteration  45600
starting iteration  45700
starting iteration  45800
starting iteration  45900
starting iteration  46000
Finished epoch 94 / 150: cost 1.421567, train: 0.494000, val 0.489000, lr 8.054135e-08
starting iteration  46100
starting iteration  46200
starting iteration  46300
starting iteration  46400
starting iteration  46500
Finished epoch 95 / 150: cost 1.503165, train: 0.535000, val 0.490000, lr 7.651428e-08
starting iteration  46600
starting iteration  46700
starting iteration  46800
starting iteration  46900
starting iteration  47000
Finished epoch 96 / 150: cost 1.544809, train: 0.501000, val 0.490000, lr 7.268857e-08
starting iteration  47100
starting iteration  47200
starting iteration  47300
starting iteration  47400
starting iteration  47500
Finished epoch 97 / 150: cost 1.452518, train: 0.536000, val 0.490000, lr 6.905414e-08
starting iteration  47600
starting iteration  47700
```

```
starting iteration  47800
starting iteration  47900
starting iteration  48000
Finished epoch 98 / 150: cost 1.446358, train: 0.527000, val 0.489000, lr 6.560143e-08
starting iteration  48100
starting iteration  48200
starting iteration  48300
starting iteration  48400
starting iteration  48500
Finished epoch 99 / 150: cost 1.557571, train: 0.504000, val 0.490000, lr 6.232136e-08
starting iteration  48600
starting iteration  48700
starting iteration  48800
starting iteration  48900
Finished epoch 100 / 150: cost 1.393283, train: 0.522000, val 0.489000, lr 5.920529e-08
starting iteration  49000
starting iteration  49100
starting iteration  49200
starting iteration  49300
starting iteration  49400
Finished epoch 101 / 150: cost 1.591829, train: 0.526000, val 0.490000, lr 5.624503e-08
starting iteration  49500
starting iteration  49600
starting iteration  49700
starting iteration  49800
starting iteration  49900
Finished epoch 102 / 150: cost 1.595473, train: 0.522000, val 0.488000, lr 5.343278e-08
starting iteration  50000
starting iteration  50100
starting iteration  50200
starting iteration  50300
starting iteration  50400
Finished epoch 103 / 150: cost 1.347366, train: 0.538000, val 0.490000, lr 5.076114e-08
starting iteration  50500
starting iteration  50600
starting iteration  50700
starting iteration  50800
starting iteration  50900
Finished epoch 104 / 150: cost 1.470135, train: 0.509000, val 0.493000, lr 4.822308e-08
starting iteration  51000
starting iteration  51100
starting iteration  51200
starting iteration  51300
starting iteration  51400
Finished epoch 105 / 150: cost 1.557893, train: 0.516000, val 0.490000, lr 4.581193e-08
starting iteration  51500
starting iteration  51600
starting iteration  51700
```

```
starting iteration  51800
starting iteration  51900
Finished epoch 106 / 150: cost 1.570598, train: 0.524000, val 0.490000, lr 4.352133e-08
starting iteration  52000
starting iteration  52100
starting iteration  52200
starting iteration  52300
starting iteration  52400
Finished epoch 107 / 150: cost 1.507779, train: 0.527000, val 0.490000, lr 4.134526e-08
starting iteration  52500
starting iteration  52600
starting iteration  52700
starting iteration  52800
starting iteration  52900
Finished epoch 108 / 150: cost 1.537996, train: 0.498000, val 0.492000, lr 3.927800e-08
starting iteration  53000
starting iteration  53100
starting iteration  53200
starting iteration  53300
starting iteration  53400
Finished epoch 109 / 150: cost 1.547336, train: 0.544000, val 0.491000, lr 3.731410e-08
starting iteration  53500
starting iteration  53600
starting iteration  53700
starting iteration  53800
Finished epoch 110 / 150: cost 1.460860, train: 0.528000, val 0.491000, lr 3.544840e-08
starting iteration  53900
starting iteration  54000
starting iteration  54100
starting iteration  54200
starting iteration  54300
Finished epoch 111 / 150: cost 1.452390, train: 0.516000, val 0.490000, lr 3.367598e-08
starting iteration  54400
starting iteration  54500
starting iteration  54600
starting iteration  54700
starting iteration  54800
Finished epoch 112 / 150: cost 1.474742, train: 0.529000, val 0.491000, lr 3.199218e-08
starting iteration  54900
starting iteration  55000
starting iteration  55100
starting iteration  55200
starting iteration  55300
Finished epoch 113 / 150: cost 1.459167, train: 0.529000, val 0.488000, lr 3.039257e-08
starting iteration  55400
starting iteration  55500
starting iteration  55600
starting iteration  55700
```

```
starting iteration  55800
Finished epoch 114 / 150: cost 1.602999, train: 0.519000, val 0.490000, lr 2.887294e-08
starting iteration  55900
starting iteration  56000
starting iteration  56100
starting iteration  56200
starting iteration  56300
Finished epoch 115 / 150: cost 1.390391, train: 0.521000, val 0.489000, lr 2.742929e-08
starting iteration  56400
starting iteration  56500
starting iteration  56600
starting iteration  56700
starting iteration  56800
Finished epoch 116 / 150: cost 1.402484, train: 0.521000, val 0.490000, lr 2.605783e-08
starting iteration  56900
starting iteration  57000
starting iteration  57100
starting iteration  57200
starting iteration  57300
Finished epoch 117 / 150: cost 1.362262, train: 0.509000, val 0.489000, lr 2.475494e-08
starting iteration  57400
starting iteration  57500
starting iteration  57600
starting iteration  57700
starting iteration  57800
Finished epoch 118 / 150: cost 1.350107, train: 0.535000, val 0.488000, lr 2.351719e-08
starting iteration  57900
starting iteration  58000
starting iteration  58100
starting iteration  58200
starting iteration  58300
Finished epoch 119 / 150: cost 1.396687, train: 0.527000, val 0.489000, lr 2.234133e-08
starting iteration  58400
starting iteration  58500
starting iteration  58600
starting iteration  58700
Finished epoch 120 / 150: cost 1.580804, train: 0.515000, val 0.488000, lr 2.122426e-08
starting iteration  58800
starting iteration  58900
starting iteration  59000
starting iteration  59100
starting iteration  59200
Finished epoch 121 / 150: cost 1.595670, train: 0.514000, val 0.489000, lr 2.016305e-08
starting iteration  59300
starting iteration  59400
starting iteration  59500
starting iteration  59600
starting iteration  59700
```

```
Finished epoch 122 / 150: cost 1.319490, train: 0.530000, val 0.488000, lr 1.915490e-08
starting iteration  59800
starting iteration  59900
starting iteration  60000
starting iteration  60100
starting iteration  60200
Finished epoch 123 / 150: cost 1.475045, train: 0.528000, val 0.488000, lr 1.819715e-08
starting iteration  60300
starting iteration  60400
starting iteration  60500
starting iteration  60600
starting iteration  60700
Finished epoch 124 / 150: cost 1.464098, train: 0.525000, val 0.489000, lr 1.728730e-08
starting iteration  60800
starting iteration  60900
starting iteration  61000
starting iteration  61100
starting iteration  61200
Finished epoch 125 / 150: cost 1.481332, train: 0.489000, val 0.490000, lr 1.642293e-08
starting iteration  61300
starting iteration  61400
starting iteration  61500
starting iteration  61600
starting iteration  61700
Finished epoch 126 / 150: cost 1.543631, train: 0.510000, val 0.490000, lr 1.560178e-08
starting iteration  61800
starting iteration  61900
starting iteration  62000
starting iteration  62100
starting iteration  62200
Finished epoch 127 / 150: cost 1.681906, train: 0.493000, val 0.490000, lr 1.482169e-08
starting iteration  62300
starting iteration  62400
starting iteration  62500
starting iteration  62600
starting iteration  62700
Finished epoch 128 / 150: cost 1.605404, train: 0.487000, val 0.489000, lr 1.408061e-08
starting iteration  62800
starting iteration  62900
starting iteration  63000
starting iteration  63100
starting iteration  63200
Finished epoch 129 / 150: cost 1.467696, train: 0.525000, val 0.490000, lr 1.337658e-08
starting iteration  63300
starting iteration  63400
starting iteration  63500
starting iteration  63600
Finished epoch 130 / 150: cost 1.636468, train: 0.501000, val 0.490000, lr 1.270775e-08
```

```
starting iteration  63700
starting iteration  63800
starting iteration  63900
starting iteration  64000
starting iteration  64100
Finished epoch 131 / 150: cost 1.606836, train: 0.527000, val 0.490000, lr 1.207236e-08
starting iteration  64200
starting iteration  64300
starting iteration  64400
starting iteration  64500
starting iteration  64600
Finished epoch 132 / 150: cost 1.767178, train: 0.521000, val 0.490000, lr 1.146875e-08
starting iteration  64700
starting iteration  64800
starting iteration  64900
starting iteration  65000
starting iteration  65100
Finished epoch 133 / 150: cost 1.414837, train: 0.526000, val 0.490000, lr 1.089531e-08
starting iteration  65200
starting iteration  65300
starting iteration  65400
starting iteration  65500
starting iteration  65600
Finished epoch 134 / 150: cost 1.518343, train: 0.517000, val 0.490000, lr 1.035054e-08
starting iteration  65700
starting iteration  65800
starting iteration  65900
starting iteration  66000
starting iteration  66100
Finished epoch 135 / 150: cost 1.560910, train: 0.499000, val 0.490000, lr 9.833015e-09
starting iteration  66200
starting iteration  66300
starting iteration  66400
starting iteration  66500
starting iteration  66600
Finished epoch 136 / 150: cost 1.508085, train: 0.500000, val 0.490000, lr 9.341365e-09
starting iteration  66700
starting iteration  66800
starting iteration  66900
starting iteration  67000
starting iteration  67100
Finished epoch 137 / 150: cost 1.569809, train: 0.531000, val 0.490000, lr 8.874296e-09
starting iteration  67200
starting iteration  67300
starting iteration  67400
starting iteration  67500
starting iteration  67600
Finished epoch 138 / 150: cost 1.610751, train: 0.523000, val 0.490000, lr 8.430581e-09
```

```
starting iteration  67700
starting iteration  67800
starting iteration  67900
starting iteration  68000
starting iteration  68100
Finished epoch 139 / 150: cost 1.616450, train: 0.512000, val 0.489000, lr 8.009052e-09
starting iteration  68200
starting iteration  68300
starting iteration  68400
starting iteration  68500
Finished epoch 140 / 150: cost 1.455042, train: 0.516000, val 0.489000, lr 7.608600e-09
starting iteration  68600
starting iteration  68700
starting iteration  68800
starting iteration  68900
starting iteration  69000
Finished epoch 141 / 150: cost 1.546349, train: 0.512000, val 0.489000, lr 7.228170e-09
starting iteration  69100
starting iteration  69200
starting iteration  69300
starting iteration  69400
starting iteration  69500
Finished epoch 142 / 150: cost 1.482789, train: 0.510000, val 0.490000, lr 6.866761e-09
starting iteration  69600
starting iteration  69700
starting iteration  69800
starting iteration  69900
starting iteration  70000
Finished epoch 143 / 150: cost 1.531323, train: 0.515000, val 0.490000, lr 6.523423e-09
starting iteration  70100
starting iteration  70200
starting iteration  70300
starting iteration  70400
starting iteration  70500
Finished epoch 144 / 150: cost 1.538728, train: 0.510000, val 0.490000, lr 6.197252e-09
starting iteration  70600
starting iteration  70700
starting iteration  70800
starting iteration  70900
starting iteration  71000
Finished epoch 145 / 150: cost 1.484660, train: 0.510000, val 0.490000, lr 5.887389e-09
starting iteration  71100
starting iteration  71200
starting iteration  71300
starting iteration  71400
starting iteration  71500
Finished epoch 146 / 150: cost 1.434962, train: 0.543000, val 0.490000, lr 5.593020e-09
starting iteration  71600
```

```
starting iteration  71700
starting iteration  71800
starting iteration  71900
starting iteration  72000
Finished epoch 147 / 150: cost 1.476097, train: 0.498000, val 0.490000, lr 5.313369e-09
starting iteration  72100
starting iteration  72200
starting iteration  72300
starting iteration  72400
starting iteration  72500
Finished epoch 148 / 150: cost 1.551739, train: 0.536000, val 0.490000, lr 5.047701e-09
starting iteration  72600
starting iteration  72700
starting iteration  72800
starting iteration  72900
starting iteration  73000
Finished epoch 149 / 150: cost 1.474683, train: 0.500000, val 0.490000, lr 4.795316e-09
starting iteration  73100
starting iteration  73200
starting iteration  73300
starting iteration  73400
Finished epoch 150 / 150: cost 1.474601, train: 0.506000, val 0.490000, lr 4.555550e-09
finished optimization. best validation accuracy: 0.493000
```

In [20]:
```
# visualize the weights
show_net_weights(best_model)
```

10 Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set.

```
In [21]: scores_test = two_layer_net(X_test, best_model)
         print('Test accuracy: ', np.mean(np.argmax(scores_test, axis=1) == y_test))
```

```
Test accuracy:  0.495
```