Dropout

In this notebook, you will implement dropout. Then we will ask you to train a network with batchnorm and dropout, and acheive over 60% accuracy on CIFAR-10.

CS231n has built a solid API for building these modular frameworks and training them, and we will use their very well implemented framework as opposed to "reinventing the wheel." This includes using their Solver, various utility functions, and their layer structure. This also includes nndl.fc_net, nndl.layers, and nndl.layer_utils. As in prior assignments, we thank Serena Yeung & Justin Johnson for permission to use code written for the CS 231n class (cs231n.stanford.edu).

In [1]:
```
## Import and setups

import time
import numpy as np
import matplotlib.pyplot as plt
from nndl.fc_net import *
from nndl.layers import *
from cs231n.data_utils import get_CIFAR10_data
from cs231n.gradient_check import eval_numerical_gradient, eval_numerical_gradien
t_array
from cs231n.solver import Solver

%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))))
```

In [2]:
```
# Load the (preprocessed) CIFAR10 data.

data = get_CIFAR10_data()
for k in data.keys():
  print('{}: {} '.format(k, data[k].shape))
```

```
X_train: (49000, 3, 32, 32)
y_train: (49000,)
X_val: (1000, 3, 32, 32)
y_val: (1000,)
X_test: (1000, 3, 32, 32)
y_test: (1000,)
```

Dropout forward pass

Implement the training and test time dropout forward pass, `dropout_forward`, in `nndl/layers.py`. After that, test your implementation by running the following cell.

```
In [15]: x = np.random.randn(500, 500) + 10

         for p in [0.3, 0.6, 0.75]:
           out, _ = dropout_forward(x, {'mode': 'train', 'p': p})
           out_test, _ = dropout_forward(x, {'mode': 'test', 'p': p})

           print('Running tests with p = ', p)
           print('Mean of input: ', x.mean())
           print('Mean of train-time output: ', out.mean())
           print('Mean of test-time output: ', out_test.mean())
           print('Fraction of train-time output set to zero: ', (out == 0).mean())
           print('Fraction of test-time output set to zero: ', (out_test == 0).mean())
```

```
Running tests with p =  0.3
Mean of input:  9.999024747460385
Mean of train-time output:  9.973523548843803
Mean of test-time output:  9.999024747460385
Fraction of train-time output set to zero:  0.700644
Fraction of test-time output set to zero:  0.0
Running tests with p =  0.6
Mean of input:  9.999024747460385
Mean of train-time output:  9.980716350624634
Mean of test-time output:  9.999024747460385
Fraction of train-time output set to zero:  0.401164
Fraction of test-time output set to zero:  0.0
Running tests with p =  0.75
Mean of input:  9.999024747460385
Mean of train-time output:  9.99619481359815
Mean of test-time output:  9.999024747460385
Fraction of train-time output set to zero:  0.25018
Fraction of test-time output set to zero:  0.0
```

Dropout backward pass

Implement the backward pass, `dropout_backward`, in `nndl/layers.py`. After that, test your gradients by running the following cell:

```
In [16]: x = np.random.randn(10, 10) + 10
         dout = np.random.randn(*x.shape)

         dropout_param = {'mode': 'train', 'p': 0.8, 'seed': 123}
         out, cache = dropout_forward(x, dropout_param)
         dx = dropout_backward(dout, cache)
         dx_num = eval_numerical_gradient_array(lambda xx: dropout_forward(xx, dropout_par
         am)[0], x, dout)

         print('dx relative error: ', rel_error(dx, dx_num))
```

```
dx relative error:  5.4456103870489836e-11
```

Implement a fully connected neural network with dropout layers

Modify the `FullyConnectedNet()` class in `nndl/fc_net.py` to incorporate dropout. A dropout layer should be incorporated after every ReLU layer. Concretely, there shouldn't be a dropout at the output layer since there is no ReLU at the output layer. You will need to modify the class in the following areas:

(1) In the forward pass, you will need to incorporate a dropout layer after every relu layer.

(2) In the backward pass, you will need to incorporate a dropout backward pass layer.

Check your implementation by running the following code. Our W1 gradient relative error is on the order of 1e-6 (the largest of all the relative errors).

```
In [18]: N, D, H1, H2, C = 2, 15, 20, 30, 10
         X = np.random.randn(N, D)
         y = np.random.randint(C, size=(N,))

         for dropout in [0, 0.25, 0.5]:
           print('Running check with dropout = ', dropout)
           model = FullyConnectedNet([H1, H2], input_dim=D, num_classes=C,
                                     weight_scale=5e-2, dtype=np.float64,
                                     dropout=dropout, seed=123)

           loss, grads = model.loss(X, y)
           print('Initial loss: ', loss)

           for name in sorted(grads):
             f = lambda _: model.loss(X, y)[0]
             grad_num = eval_numerical_gradient(f, model.params[name], verbose=False, h=1e
         -5)
             print('{} relative error: {}'.format(name, rel_error(grad_num, grads[name])))
           print('\n')
```

```
Running check with dropout =  0
Initial loss:  2.303043161170242
W1 relative error: 4.795196837428236e-07
W2 relative error: 1.9717710574314515e-07
W3 relative error: 1.5587099943922404e-07
b1 relative error: 2.033615668300116e-08
b2 relative error: 1.6863157228196606e-09
b3 relative error: 1.1144421861081857e-10


Running check with dropout =  0.25
Initial loss:  2.302354247831908
W1 relative error: 1.0017417771677944e-07
W2 relative error: 2.2591355266316633e-09
W3 relative error: 2.5553916705214888e-05
b1 relative error: 9.368619593647787e-10
b2 relative error: 0.2134290559158092
b3 relative error: 1.2466815697171467e-10


Running check with dropout =  0.5
Initial loss:  2.304242617164796
W1 relative error: 1.2078835464919935e-07
W2 relative error: 2.454128122918086e-08
W3 relative error: 8.057427967417468e-07
b1 relative error: 2.276255983500775e-08
b2 relative error: 6.836023099402868e-10
b3 relative error: 1.2833211144563337e-10
```

Dropout as a regularizer

In class, we claimed that dropout acts as a regularizer by effectively bagging. To check this, we will train two small networks, one with dropout and one without dropout.

In [19]:
```
# Train two identical nets, one with dropout and one without

num_train = 500
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

solvers = {}
dropout_choices = [0, 0.6]
for dropout in dropout_choices:
  model = FullyConnectedNet([100, 100, 100], dropout=dropout)

  solver = Solver(model, small_data,
                  num_epochs=25, batch_size=100,
                  update_rule='adam',
                  optim_config={
                    'learning_rate': 5e-4,
                  },
                  verbose=True, print_every=100)
  solver.train()
  solvers[dropout] = solver
```

```
(Iteration 1 / 125) loss: 2.300804
(Epoch 0 / 25) train acc: 0.220000; val_acc: 0.168000
(Epoch 1 / 25) train acc: 0.188000; val_acc: 0.147000
(Epoch 2 / 25) train acc: 0.266000; val_acc: 0.200000
(Epoch 3 / 25) train acc: 0.338000; val_acc: 0.262000
(Epoch 4 / 25) train acc: 0.378000; val_acc: 0.278000
(Epoch 5 / 25) train acc: 0.428000; val_acc: 0.297000
(Epoch 6 / 25) train acc: 0.468000; val_acc: 0.323000
(Epoch 7 / 25) train acc: 0.494000; val_acc: 0.287000
(Epoch 8 / 25) train acc: 0.566000; val_acc: 0.328000
(Epoch 9 / 25) train acc: 0.572000; val_acc: 0.322000
(Epoch 10 / 25) train acc: 0.622000; val_acc: 0.324000
(Epoch 11 / 25) train acc: 0.670000; val_acc: 0.279000
(Epoch 12 / 25) train acc: 0.710000; val_acc: 0.338000
(Epoch 13 / 25) train acc: 0.746000; val_acc: 0.319000
(Epoch 14 / 25) train acc: 0.792000; val_acc: 0.307000
(Epoch 15 / 25) train acc: 0.834000; val_acc: 0.297000
(Epoch 16 / 25) train acc: 0.876000; val_acc: 0.327000
(Epoch 17 / 25) train acc: 0.886000; val_acc: 0.320000
(Epoch 18 / 25) train acc: 0.918000; val_acc: 0.314000
(Epoch 19 / 25) train acc: 0.922000; val_acc: 0.290000
(Epoch 20 / 25) train acc: 0.944000; val_acc: 0.306000
(Iteration 101 / 125) loss: 0.156105
(Epoch 21 / 25) train acc: 0.968000; val_acc: 0.302000
(Epoch 22 / 25) train acc: 0.978000; val_acc: 0.302000
(Epoch 23 / 25) train acc: 0.976000; val_acc: 0.289000
(Epoch 24 / 25) train acc: 0.986000; val_acc: 0.285000
(Epoch 25 / 25) train acc: 0.978000; val_acc: 0.311000
(Iteration 1 / 125) loss: 2.301328
(Epoch 0 / 25) train acc: 0.154000; val_acc: 0.143000
(Epoch 1 / 25) train acc: 0.214000; val_acc: 0.195000
(Epoch 2 / 25) train acc: 0.252000; val_acc: 0.216000
(Epoch 3 / 25) train acc: 0.276000; val_acc: 0.200000
(Epoch 4 / 25) train acc: 0.308000; val_acc: 0.254000
(Epoch 5 / 25) train acc: 0.316000; val_acc: 0.241000
(Epoch 6 / 25) train acc: 0.322000; val_acc: 0.282000
(Epoch 7 / 25) train acc: 0.354000; val_acc: 0.273000
(Epoch 8 / 25) train acc: 0.364000; val_acc: 0.276000
(Epoch 9 / 25) train acc: 0.408000; val_acc: 0.282000
(Epoch 10 / 25) train acc: 0.454000; val_acc: 0.302000
(Epoch 11 / 25) train acc: 0.472000; val_acc: 0.296000
(Epoch 12 / 25) train acc: 0.496000; val_acc: 0.318000
(Epoch 13 / 25) train acc: 0.512000; val_acc: 0.310000
(Epoch 14 / 25) train acc: 0.532000; val_acc: 0.318000
(Epoch 15 / 25) train acc: 0.558000; val_acc: 0.331000
(Epoch 16 / 25) train acc: 0.574000; val_acc: 0.300000
(Epoch 17 / 25) train acc: 0.624000; val_acc: 0.324000
(Epoch 18 / 25) train acc: 0.608000; val_acc: 0.326000
(Epoch 19 / 25) train acc: 0.622000; val_acc: 0.326000
(Epoch 20 / 25) train acc: 0.664000; val_acc: 0.342000
(Iteration 101 / 125) loss: 1.293077
(Epoch 21 / 25) train acc: 0.688000; val_acc: 0.320000
(Epoch 22 / 25) train acc: 0.708000; val_acc: 0.326000
(Epoch 23 / 25) train acc: 0.734000; val_acc: 0.343000
(Epoch 24 / 25) train acc: 0.756000; val_acc: 0.322000
(Epoch 25 / 25) train acc: 0.788000; val_acc: 0.345000
```

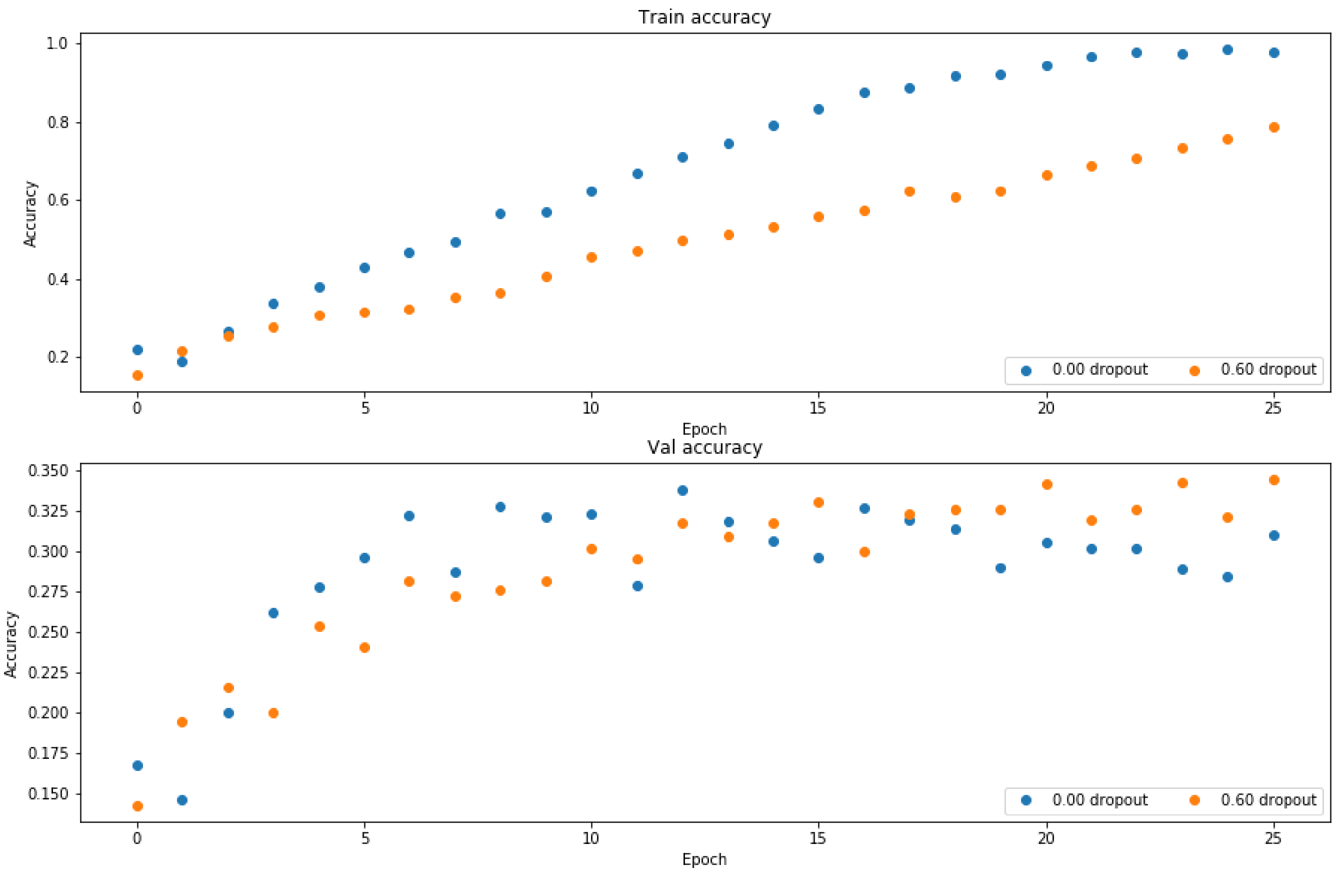
In [20]:
```
# Plot train and validation accuracies of the two models

train_accs = []
val_accs = []
for dropout in dropout_choices:
  solver = solvers[dropout]
  train_accs.append(solver.train_acc_history[-1])
  val_accs.append(solver.val_acc_history[-1])

plt.subplot(3, 1, 1)
for dropout in dropout_choices:
  plt.plot(solvers[dropout].train_acc_history, 'o', label='%.2f dropout' % dropou
t)
plt.title('Train accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(ncol=2, loc='lower right')

plt.subplot(3, 1, 2)
for dropout in dropout_choices:
  plt.plot(solvers[dropout].val_acc_history, 'o', label='%.2f dropout' % dropout)
plt.title('Val accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(ncol=2, loc='lower right')

plt.gcf().set_size_inches(15, 15)
plt.show()
```



Question

Based off the results of this experiment, is dropout performing regularization? Explain your answer.

Answer:

Yes, because the difference between the training and validation error is less when using dropout.

Final part of the assignment

Get over 60% validation accuracy on CIFAR-10 by using the layers you have implemented. You will be graded according to the following equation:

min(floor((X - 32%)) / 28%, 1) where if you get 60% or higher validation accuracy, you get full points.

```
In [49]: # ================================================================ #
         # YOUR CODE HERE:
         #   Implement a FC-net that achieves at least 60% validation accuracy
         #   on CIFAR-10.
         # ================================================================ #
         layer_dims = [600, 600, 600, 600]
         weight_scale = 0.01
         learning_rate = 1e-3
         lr_decay = 0.90

         model = FullyConnectedNet(layer_dims, weight_scale=weight_scale,
                                   dropout=0.65, reg=0.0, use_batchnorm=True)

         solver = Solver(model, data,
                         num_epochs=50, batch_size=100,
                         update_rule='adam',
                         optim_config={
                           'learning_rate': learning_rate,
                         },
                         lr_decay=lr_decay,
                         verbose=True, print_every=50)
         solver.train()
         # ================================================================ #
         # END YOUR CODE HERE
         # ================================================================ #
```

```
(Iteration 1 / 24500) loss: 2.305405
(Epoch 0 / 50) train acc: 0.163000; val_acc: 0.155000
(Iteration 51 / 24500) loss: 1.895922
(Iteration 101 / 24500) loss: 1.807972
(Iteration 151 / 24500) loss: 1.760907
(Iteration 201 / 24500) loss: 1.654568
(Iteration 251 / 24500) loss: 1.774943
(Iteration 301 / 24500) loss: 1.716539
(Iteration 351 / 24500) loss: 1.561507
(Iteration 401 / 24500) loss: 1.624929
(Iteration 451 / 24500) loss: 1.647275
(Epoch 1 / 50) train acc: 0.468000; val_acc: 0.463000
(Iteration 501 / 24500) loss: 1.367936
(Iteration 551 / 24500) loss: 1.555204
(Iteration 601 / 24500) loss: 1.483475
(Iteration 651 / 24500) loss: 1.356630
(Iteration 701 / 24500) loss: 1.536303
(Iteration 751 / 24500) loss: 1.720222
(Iteration 801 / 24500) loss: 1.723152
(Iteration 851 / 24500) loss: 1.568987
(Iteration 901 / 24500) loss: 1.266345
(Iteration 951 / 24500) loss: 1.577630
(Epoch 2 / 50) train acc: 0.518000; val_acc: 0.503000
(Iteration 1001 / 24500) loss: 1.331066
(Iteration 1051 / 24500) loss: 1.541900
(Iteration 1101 / 24500) loss: 1.410722
(Iteration 1151 / 24500) loss: 1.367694
(Iteration 1201 / 24500) loss: 1.413612
(Iteration 1251 / 24500) loss: 1.578218
(Iteration 1301 / 24500) loss: 1.257078
(Iteration 1351 / 24500) loss: 1.445430
(Iteration 1401 / 24500) loss: 1.489271
(Iteration 1451 / 24500) loss: 1.424285
(Epoch 3 / 50) train acc: 0.516000; val_acc: 0.509000
(Iteration 1501 / 24500) loss: 1.378358
(Iteration 1551 / 24500) loss: 1.361894
(Iteration 1601 / 24500) loss: 1.469139
(Iteration 1651 / 24500) loss: 1.356522
(Iteration 1701 / 24500) loss: 1.451606
(Iteration 1751 / 24500) loss: 1.440173
(Iteration 1801 / 24500) loss: 1.413091
(Iteration 1851 / 24500) loss: 1.312794
(Iteration 1901 / 24500) loss: 1.540010
(Iteration 1951 / 24500) loss: 1.465006
(Epoch 4 / 50) train acc: 0.563000; val_acc: 0.515000
(Iteration 2001 / 24500) loss: 1.361087
(Iteration 2051 / 24500) loss: 1.133883
(Iteration 2101 / 24500) loss: 1.361684
(Iteration 2151 / 24500) loss: 1.377699
(Iteration 2201 / 24500) loss: 1.345573
(Iteration 2251 / 24500) loss: 1.205908
(Iteration 2301 / 24500) loss: 1.329005
(Iteration 2351 / 24500) loss: 1.517518
(Iteration 2401 / 24500) loss: 1.401187
(Epoch 5 / 50) train acc: 0.579000; val_acc: 0.542000
(Iteration 2451 / 24500) loss: 1.349786
(Iteration 2501 / 24500) loss: 1.179071
(Iteration 2551 / 24500) loss: 1.292837
(Iteration 2601 / 24500) loss: 1.353099
(Iteration 2651 / 24500) loss: 1.202831
(Iteration 2701 / 24500) loss: 1.382101
```

```
(Iteration 2751 / 24500) loss: 1.543414
(Iteration 2801 / 24500) loss: 1.254617
(Iteration 2851 / 24500) loss: 1.235397
(Iteration 2901 / 24500) loss: 1.343249
(Epoch 6 / 50) train acc: 0.616000; val_acc: 0.552000
(Iteration 2951 / 24500) loss: 1.313147
(Iteration 3001 / 24500) loss: 1.352385
(Iteration 3051 / 24500) loss: 1.040495
(Iteration 3101 / 24500) loss: 1.312170
(Iteration 3151 / 24500) loss: 1.159872
(Iteration 3201 / 24500) loss: 1.151120
(Iteration 3251 / 24500) loss: 1.246808
(Iteration 3301 / 24500) loss: 1.202060
(Iteration 3351 / 24500) loss: 1.120763
(Iteration 3401 / 24500) loss: 1.390110
(Epoch 7 / 50) train acc: 0.613000; val_acc: 0.535000
(Iteration 3451 / 24500) loss: 1.157195
(Iteration 3501 / 24500) loss: 1.174080
(Iteration 3551 / 24500) loss: 1.411040
(Iteration 3601 / 24500) loss: 1.153144
(Iteration 3651 / 24500) loss: 1.175108
(Iteration 3701 / 24500) loss: 1.235538
(Iteration 3751 / 24500) loss: 1.281613
(Iteration 3801 / 24500) loss: 1.209705
(Iteration 3851 / 24500) loss: 0.884508
(Iteration 3901 / 24500) loss: 1.189045
(Epoch 8 / 50) train acc: 0.634000; val_acc: 0.549000
(Iteration 3951 / 24500) loss: 1.321094
(Iteration 4001 / 24500) loss: 1.150332
(Iteration 4051 / 24500) loss: 1.435022
(Iteration 4101 / 24500) loss: 0.994263
(Iteration 4151 / 24500) loss: 1.110363
(Iteration 4201 / 24500) loss: 1.063714
(Iteration 4251 / 24500) loss: 1.161106
(Iteration 4301 / 24500) loss: 1.366952
(Iteration 4351 / 24500) loss: 1.268125
(Iteration 4401 / 24500) loss: 1.280903
(Epoch 9 / 50) train acc: 0.680000; val_acc: 0.557000
(Iteration 4451 / 24500) loss: 1.130212
(Iteration 4501 / 24500) loss: 1.152019
(Iteration 4551 / 24500) loss: 0.991677
(Iteration 4601 / 24500) loss: 1.159109
(Iteration 4651 / 24500) loss: 1.210210
(Iteration 4701 / 24500) loss: 1.154131
(Iteration 4751 / 24500) loss: 0.984839
(Iteration 4801 / 24500) loss: 0.986202
(Iteration 4851 / 24500) loss: 1.406942
(Epoch 10 / 50) train acc: 0.655000; val_acc: 0.563000
(Iteration 4901 / 24500) loss: 1.340959
(Iteration 4951 / 24500) loss: 0.970882
(Iteration 5001 / 24500) loss: 1.190687
(Iteration 5051 / 24500) loss: 1.196977
(Iteration 5101 / 24500) loss: 1.253798
(Iteration 5151 / 24500) loss: 1.079625
(Iteration 5201 / 24500) loss: 1.251764
(Iteration 5251 / 24500) loss: 1.108914
(Iteration 5301 / 24500) loss: 1.070281
(Iteration 5351 / 24500) loss: 1.207383
(Epoch 11 / 50) train acc: 0.652000; val_acc: 0.569000
(Iteration 5401 / 24500) loss: 1.116703
(Iteration 5451 / 24500) loss: 1.018663
(Iteration 5501 / 24500) loss: 0.968404
```

```
(Iteration 5551 / 24500) loss: 1.160531
(Iteration 5601 / 24500) loss: 1.149543
(Iteration 5651 / 24500) loss: 1.108142
(Iteration 5701 / 24500) loss: 1.040542
(Iteration 5751 / 24500) loss: 1.241276
(Iteration 5801 / 24500) loss: 1.050371
(Iteration 5851 / 24500) loss: 0.945394
(Epoch 12 / 50) train acc: 0.678000; val_acc: 0.584000
(Iteration 5901 / 24500) loss: 0.985231
(Iteration 5951 / 24500) loss: 1.058509
(Iteration 6001 / 24500) loss: 1.320623
(Iteration 6051 / 24500) loss: 1.152069
(Iteration 6101 / 24500) loss: 0.863892
(Iteration 6151 / 24500) loss: 0.976334
(Iteration 6201 / 24500) loss: 1.091637
(Iteration 6251 / 24500) loss: 1.045518
(Iteration 6301 / 24500) loss: 1.049506
(Iteration 6351 / 24500) loss: 1.165354
(Epoch 13 / 50) train acc: 0.695000; val_acc: 0.580000
(Iteration 6401 / 24500) loss: 1.087945
(Iteration 6451 / 24500) loss: 1.185851
(Iteration 6501 / 24500) loss: 1.073525
(Iteration 6551 / 24500) loss: 0.814498
(Iteration 6601 / 24500) loss: 1.023440
(Iteration 6651 / 24500) loss: 1.239899
(Iteration 6701 / 24500) loss: 1.166954
(Iteration 6751 / 24500) loss: 0.965370
(Iteration 6801 / 24500) loss: 1.286594
(Iteration 6851 / 24500) loss: 1.113082
(Epoch 14 / 50) train acc: 0.665000; val_acc: 0.580000
(Iteration 6901 / 24500) loss: 0.839494
(Iteration 6951 / 24500) loss: 0.838638
(Iteration 7001 / 24500) loss: 0.895560
(Iteration 7051 / 24500) loss: 0.967985
(Iteration 7101 / 24500) loss: 1.033964
(Iteration 7151 / 24500) loss: 1.073965
(Iteration 7201 / 24500) loss: 1.231775
(Iteration 7251 / 24500) loss: 1.005998
(Iteration 7301 / 24500) loss: 0.933446
(Epoch 15 / 50) train acc: 0.691000; val_acc: 0.590000
(Iteration 7351 / 24500) loss: 0.952628
(Iteration 7401 / 24500) loss: 0.952984
(Iteration 7451 / 24500) loss: 0.958054
(Iteration 7501 / 24500) loss: 1.042820
(Iteration 7551 / 24500) loss: 1.212823
(Iteration 7601 / 24500) loss: 0.915462
(Iteration 7651 / 24500) loss: 1.112218
(Iteration 7701 / 24500) loss: 1.054046
(Iteration 7751 / 24500) loss: 0.901583
(Iteration 7801 / 24500) loss: 1.155146
(Epoch 16 / 50) train acc: 0.730000; val_acc: 0.588000
(Iteration 7851 / 24500) loss: 0.926943
(Iteration 7901 / 24500) loss: 0.750936
(Iteration 7951 / 24500) loss: 0.961800
(Iteration 8001 / 24500) loss: 0.943148
(Iteration 8051 / 24500) loss: 0.823362
(Iteration 8101 / 24500) loss: 0.931198
(Iteration 8151 / 24500) loss: 0.986520
(Iteration 8201 / 24500) loss: 0.839939
(Iteration 8251 / 24500) loss: 1.148176
(Iteration 8301 / 24500) loss: 1.103267
(Epoch 17 / 50) train acc: 0.732000; val_acc: 0.586000
```

```
(Iteration 8351 / 24500) loss: 1.013542
(Iteration 8401 / 24500) loss: 1.033530
(Iteration 8451 / 24500) loss: 0.924949
(Iteration 8501 / 24500) loss: 0.995163
(Iteration 8551 / 24500) loss: 1.072750
(Iteration 8601 / 24500) loss: 1.038492
(Iteration 8651 / 24500) loss: 1.219584
(Iteration 8701 / 24500) loss: 1.007752
(Iteration 8751 / 24500) loss: 0.925297
(Iteration 8801 / 24500) loss: 0.940192
(Epoch 18 / 50) train acc: 0.729000; val_acc: 0.576000
(Iteration 8851 / 24500) loss: 1.077766
(Iteration 8901 / 24500) loss: 1.003778
(Iteration 8951 / 24500) loss: 0.784537
(Iteration 9001 / 24500) loss: 0.818034
(Iteration 9051 / 24500) loss: 0.951942
(Iteration 9101 / 24500) loss: 0.891193
(Iteration 9151 / 24500) loss: 1.045544
(Iteration 9201 / 24500) loss: 0.914384
(Iteration 9251 / 24500) loss: 0.907780
(Iteration 9301 / 24500) loss: 0.966635
(Epoch 19 / 50) train acc: 0.756000; val_acc: 0.588000
(Iteration 9351 / 24500) loss: 0.883855
(Iteration 9401 / 24500) loss: 1.157534
(Iteration 9451 / 24500) loss: 0.968482
(Iteration 9501 / 24500) loss: 0.881435
(Iteration 9551 / 24500) loss: 1.023181
(Iteration 9601 / 24500) loss: 0.985334
(Iteration 9651 / 24500) loss: 0.977189
(Iteration 9701 / 24500) loss: 0.989856
(Iteration 9751 / 24500) loss: 1.021420
(Epoch 20 / 50) train acc: 0.749000; val_acc: 0.574000
(Iteration 9801 / 24500) loss: 0.908041
(Iteration 9851 / 24500) loss: 0.898181
(Iteration 9901 / 24500) loss: 0.764440
(Iteration 9951 / 24500) loss: 1.000073
(Iteration 10001 / 24500) loss: 0.744144
(Iteration 10051 / 24500) loss: 0.988009
(Iteration 10101 / 24500) loss: 0.953059
(Iteration 10151 / 24500) loss: 0.912375
(Iteration 10201 / 24500) loss: 1.080402
(Iteration 10251 / 24500) loss: 0.734249
(Epoch 21 / 50) train acc: 0.763000; val_acc: 0.580000
(Iteration 10301 / 24500) loss: 0.922739
(Iteration 10351 / 24500) loss: 0.885038
(Iteration 10401 / 24500) loss: 0.747679
(Iteration 10451 / 24500) loss: 1.126683
(Iteration 10501 / 24500) loss: 0.891979
(Iteration 10551 / 24500) loss: 0.954819
(Iteration 10601 / 24500) loss: 0.940721
(Iteration 10651 / 24500) loss: 0.861106
(Iteration 10701 / 24500) loss: 0.874840
(Iteration 10751 / 24500) loss: 0.909941
(Epoch 22 / 50) train acc: 0.746000; val_acc: 0.592000
(Iteration 10801 / 24500) loss: 0.801810
(Iteration 10851 / 24500) loss: 0.837487
(Iteration 10901 / 24500) loss: 1.056188
(Iteration 10951 / 24500) loss: 0.938667
(Iteration 11001 / 24500) loss: 0.775945
(Iteration 11051 / 24500) loss: 0.898042
(Iteration 11101 / 24500) loss: 0.892971
(Iteration 11151 / 24500) loss: 0.746334
```

```
(Iteration 11201 / 24500) loss: 0.842717
(Iteration 11251 / 24500) loss: 0.795465
(Epoch 23 / 50) train acc: 0.781000; val_acc: 0.593000
(Iteration 11301 / 24500) loss: 0.986391
(Iteration 11351 / 24500) loss: 1.040719
(Iteration 11401 / 24500) loss: 0.778256
(Iteration 11451 / 24500) loss: 1.139385
(Iteration 11501 / 24500) loss: 0.729800
(Iteration 11551 / 24500) loss: 0.931683
(Iteration 11601 / 24500) loss: 0.890840
(Iteration 11651 / 24500) loss: 0.945107
(Iteration 11701 / 24500) loss: 0.752788
(Iteration 11751 / 24500) loss: 0.824466
(Epoch 24 / 50) train acc: 0.791000; val_acc: 0.587000
(Iteration 11801 / 24500) loss: 0.912047
(Iteration 11851 / 24500) loss: 0.807942
(Iteration 11901 / 24500) loss: 0.750175
(Iteration 11951 / 24500) loss: 0.814072
(Iteration 12001 / 24500) loss: 0.892791
(Iteration 12051 / 24500) loss: 0.993192
(Iteration 12101 / 24500) loss: 0.911538
(Iteration 12151 / 24500) loss: 0.672663
(Iteration 12201 / 24500) loss: 1.237944
(Epoch 25 / 50) train acc: 0.773000; val_acc: 0.582000
(Iteration 12251 / 24500) loss: 0.773281
(Iteration 12301 / 24500) loss: 1.026120
(Iteration 12351 / 24500) loss: 0.782128
(Iteration 12401 / 24500) loss: 0.784366
(Iteration 12451 / 24500) loss: 0.760001
(Iteration 12501 / 24500) loss: 0.904179
(Iteration 12551 / 24500) loss: 0.781266
(Iteration 12601 / 24500) loss: 0.727542
(Iteration 12651 / 24500) loss: 1.005716
(Iteration 12701 / 24500) loss: 0.925121
(Epoch 26 / 50) train acc: 0.780000; val_acc: 0.591000
(Iteration 12751 / 24500) loss: 0.984700
(Iteration 12801 / 24500) loss: 0.786625
(Iteration 12851 / 24500) loss: 1.019540
(Iteration 12901 / 24500) loss: 0.721131
(Iteration 12951 / 24500) loss: 0.923106
(Iteration 13001 / 24500) loss: 0.984795
(Iteration 13051 / 24500) loss: 0.909641
(Iteration 13101 / 24500) loss: 0.755178
(Iteration 13151 / 24500) loss: 0.700706
(Iteration 13201 / 24500) loss: 0.833847
(Epoch 27 / 50) train acc: 0.764000; val_acc: 0.589000
(Iteration 13251 / 24500) loss: 0.858937
(Iteration 13301 / 24500) loss: 0.856274
(Iteration 13351 / 24500) loss: 0.912349
(Iteration 13401 / 24500) loss: 1.048571
(Iteration 13451 / 24500) loss: 0.820849
(Iteration 13501 / 24500) loss: 1.022572
(Iteration 13551 / 24500) loss: 0.795427
(Iteration 13601 / 24500) loss: 0.985009
(Iteration 13651 / 24500) loss: 0.933283
(Iteration 13701 / 24500) loss: 0.844097
(Epoch 28 / 50) train acc: 0.797000; val_acc: 0.590000
(Iteration 13751 / 24500) loss: 0.905174
(Iteration 13801 / 24500) loss: 0.843156
(Iteration 13851 / 24500) loss: 0.794416
(Iteration 13901 / 24500) loss: 0.819769
(Iteration 13951 / 24500) loss: 0.978043
```

```
(Iteration 14001 / 24500) loss: 0.965652
(Iteration 14051 / 24500) loss: 0.852074
(Iteration 14101 / 24500) loss: 0.845987
(Iteration 14151 / 24500) loss: 0.747855
(Iteration 14201 / 24500) loss: 0.795871
(Epoch 29 / 50) train acc: 0.792000; val_acc: 0.593000
(Iteration 14251 / 24500) loss: 0.801941
(Iteration 14301 / 24500) loss: 0.884767
(Iteration 14351 / 24500) loss: 0.731136
(Iteration 14401 / 24500) loss: 0.794959
(Iteration 14451 / 24500) loss: 1.021526
(Iteration 14501 / 24500) loss: 0.738700
(Iteration 14551 / 24500) loss: 0.828544
(Iteration 14601 / 24500) loss: 0.759870
(Iteration 14651 / 24500) loss: 0.745240
(Epoch 30 / 50) train acc: 0.767000; val_acc: 0.598000
(Iteration 14701 / 24500) loss: 0.971425
(Iteration 14751 / 24500) loss: 0.844364
(Iteration 14801 / 24500) loss: 0.836075
(Iteration 14851 / 24500) loss: 0.773092
(Iteration 14901 / 24500) loss: 1.104318
(Iteration 14951 / 24500) loss: 0.738051
(Iteration 15001 / 24500) loss: 0.981469
(Iteration 15051 / 24500) loss: 0.948608
(Iteration 15101 / 24500) loss: 1.010112
(Iteration 15151 / 24500) loss: 0.828965
(Epoch 31 / 50) train acc: 0.776000; val_acc: 0.585000
(Iteration 15201 / 24500) loss: 0.884839
(Iteration 15251 / 24500) loss: 0.808028
(Iteration 15301 / 24500) loss: 0.814889
(Iteration 15351 / 24500) loss: 0.842192
(Iteration 15401 / 24500) loss: 0.775567
(Iteration 15451 / 24500) loss: 0.868179
(Iteration 15501 / 24500) loss: 0.888033
(Iteration 15551 / 24500) loss: 0.675218
(Iteration 15601 / 24500) loss: 0.809412
(Iteration 15651 / 24500) loss: 0.795681
(Epoch 32 / 50) train acc: 0.802000; val_acc: 0.592000
(Iteration 15701 / 24500) loss: 0.874625
(Iteration 15751 / 24500) loss: 0.776042
(Iteration 15801 / 24500) loss: 0.905111
(Iteration 15851 / 24500) loss: 0.875138
(Iteration 15901 / 24500) loss: 0.782644
(Iteration 15951 / 24500) loss: 1.008065
(Iteration 16001 / 24500) loss: 1.049957
(Iteration 16051 / 24500) loss: 0.593335
(Iteration 16101 / 24500) loss: 0.959024
(Iteration 16151 / 24500) loss: 1.022140
(Epoch 33 / 50) train acc: 0.803000; val_acc: 0.597000
(Iteration 16201 / 24500) loss: 0.581478
(Iteration 16251 / 24500) loss: 0.930686
(Iteration 16301 / 24500) loss: 0.953515
(Iteration 16351 / 24500) loss: 0.870605
(Iteration 16401 / 24500) loss: 0.783915
(Iteration 16451 / 24500) loss: 0.683314
(Iteration 16501 / 24500) loss: 0.817522
(Iteration 16551 / 24500) loss: 1.043455
(Iteration 16601 / 24500) loss: 0.736522
(Iteration 16651 / 24500) loss: 0.765296
(Epoch 34 / 50) train acc: 0.805000; val_acc: 0.591000
(Iteration 16701 / 24500) loss: 0.803287
(Iteration 16751 / 24500) loss: 1.021465
```

```
(Iteration 16801 / 24500) loss: 0.857296
(Iteration 16851 / 24500) loss: 0.862635
(Iteration 16901 / 24500) loss: 0.894754
(Iteration 16951 / 24500) loss: 0.864463
(Iteration 17001 / 24500) loss: 0.984065
(Iteration 17051 / 24500) loss: 0.729083
(Iteration 17101 / 24500) loss: 0.834994
(Epoch 35 / 50) train acc: 0.810000; val_acc: 0.601000
(Iteration 17151 / 24500) loss: 0.862277

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-49-da0f2753ada5> in <module>()
     20                 lr_decay=lr_decay,
     21                 verbose=True, print_every=50)
---> 22 solver.train()
     23 # ================================================================ #
     24 # END YOUR CODE HERE

~/Desktop/UCLA-ECE239AS-W19/hw4/code/cs231n/solver.py in train(self)
    262 
    263         for t in range(num_iterations):
--> 264             self._step()
    265 
    266             # Maybe print training loss

~/Desktop/UCLA-ECE239AS-W19/hw4/code/cs231n/solver.py in _step(self)
    185             dw = grads[p]
    186             config = self.optim_configs[p]
--> 187             next_w, next_config = self.update_rule(w, dw, config)
    188             self.model.params[p] = next_w
    189             self.optim_configs[p] = next_config

~/Desktop/UCLA-ECE239AS-W19/hw4/code/nndl/optim.py in adam(w, dw, config)
    190   v_tilda = config['v'] / (1-config['beta1'] ** config['t'])
    191   a_tilda = config['a'] / (1-config['beta2'] ** config['t'])
--> 192   next_w = w - config['learning_rate'] / (np.sqrt(a_tilda) + config['eps
ilon']) * v_tilda
    193   # ================================================================ #
    194   # END YOUR CODE HERE

KeyboardInterrupt: 
```