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_numer
ical_gradient_array
from cs231n.solver import Solver

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

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-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_val: (1000, 3, 32, 32)
X_train: (49000, 3, 32, 32)
X_test: (1000, 3, 32, 32)
y_val: (1000,)
y_train: (49000,)
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 [3]:
```
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).mea
n())
  print('Fraction of test-time output set to zero: ', (out_test ==
0).mean())
```

```
('Running tests with p = ', 0.3)
('Mean of input: ', 10.001082235485809)
('Mean of train-time output: ', 9.999702017393306)
('Mean of test-time output: ', 10.001082235485809)
('Fraction of train-time output set to zero: ', 0.300028)
('Fraction of test-time output set to zero: ', 0.0)
('Running tests with p = ', 0.6)
('Mean of input: ', 10.001082235485809)
('Mean of train-time output: ', 10.006773002777756)
('Mean of test-time output: ', 10.001082235485809)
('Fraction of train-time output set to zero: ', 0.599804)
('Fraction of test-time output set to zero: ', 0.0)
('Running tests with p = ', 0.75)
('Mean of input: ', 10.001082235485809)
('Mean of train-time output: ', 10.012936754081702)
('Mean of test-time output: ', 10.001082235485809)
('Fraction of train-time output set to zero: ', 0.749844)
('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 [4]:
```
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_param)[0], x, dout)

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

```
('dx relative error: ', 1.892903972810276e-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 [5]:
```
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, gr
ads[name])))
  print('\n')
```

```
('Running check with dropout = ', 0)
('Initial loss: ', 2.3051948273987857)
W1 relative error: 5.25426264222e-07
W2 relative error: 1.88756029969e-05
W3 relative error: 2.91609738888e-07
b1 relative error: 1.34135249104e-07
b2 relative error: 7.09286957083e-08
b3 relative error: 1.4926760615e-10


('Running check with dropout = ', 0.25)
('Initial loss: ', 2.3052077546540826)
W1 relative error: 2.61384694481e-07
W2 relative error: 1.00340102207e-07
W3 relative error: 4.45631607704e-08
b1 relative error: 1.79278481749e-07
b2 relative error: 5.03584968497e-09
b3 relative error: 1.00397473212e-10


('Running check with dropout = ', 0.5)
('Initial loss: ', 2.3035667586595423)
W1 relative error: 1.93342115799e-06
W2 relative error: 7.42499917861e-08
W3 relative error: 7.40458236465e-09
b1 relative error: 7.42143754193e-08
b2 relative error: 4.4872977417e-10
b3 relative error: 1.45584710338e-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 [6]:
```
# 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: 2.298716
(Epoch 0 / 25) train acc: 0.132000; val_acc: 0.146000
(Epoch 1 / 25) train acc: 0.118000; val_acc: 0.131000
(Epoch 2 / 25) train acc: 0.220000; val_acc: 0.214000
(Epoch 3 / 25) train acc: 0.206000; val_acc: 0.180000
(Epoch 4 / 25) train acc: 0.220000; val_acc: 0.193000
(Epoch 5 / 25) train acc: 0.264000; val_acc: 0.229000
(Epoch 6 / 25) train acc: 0.268000; val_acc: 0.203000
(Epoch 7 / 25) train acc: 0.266000; val_acc: 0.212000
(Epoch 8 / 25) train acc: 0.282000; val_acc: 0.236000
(Epoch 9 / 25) train acc: 0.310000; val_acc: 0.255000
(Epoch 10 / 25) train acc: 0.320000; val_acc: 0.267000
(Epoch 11 / 25) train acc: 0.338000; val_acc: 0.273000
(Epoch 12 / 25) train acc: 0.346000; val_acc: 0.278000
(Epoch 13 / 25) train acc: 0.332000; val_acc: 0.279000
(Epoch 14 / 25) train acc: 0.328000; val_acc: 0.284000
(Epoch 15 / 25) train acc: 0.354000; val_acc: 0.271000
(Epoch 16 / 25) train acc: 0.386000; val_acc: 0.277000
(Epoch 17 / 25) train acc: 0.388000; val_acc: 0.297000
(Epoch 18 / 25) train acc: 0.402000; val_acc: 0.280000
(Epoch 19 / 25) train acc: 0.388000; val_acc: 0.274000
(Epoch 20 / 25) train acc: 0.386000; val_acc: 0.274000
(Iteration 101 / 125) loss: 1.919649
(Epoch 21 / 25) train acc: 0.402000; val_acc: 0.272000
(Epoch 22 / 25) train acc: 0.440000; val_acc: 0.286000
(Epoch 23 / 25) train acc: 0.458000; val_acc: 0.295000
(Epoch 24 / 25) train acc: 0.462000; val_acc: 0.311000
(Epoch 25 / 25) train acc: 0.466000; val_acc: 0.297000
```

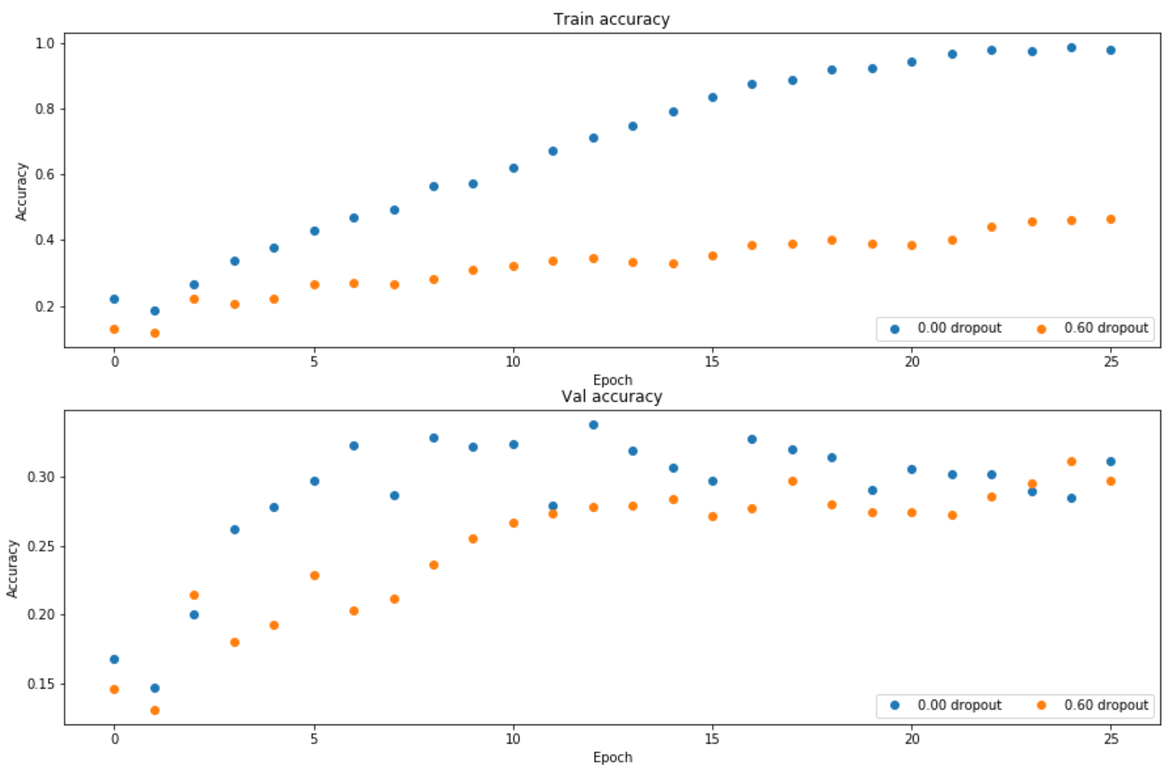
In [7]:
```
# Plot train and validation accuracies of the two models

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 dropo
ut' % dropout)
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 dropou
t' % 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:

By the plots, it looks like it is. You can see that the training accuracy gets very far ahead of the validation accuracy with 0 dropout, suggesting the model is overfitting. With .6 dropout, the validation accuracy and training accuracy are more or less in step.

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 [11]:
```
# ================================================================ #
# YOUR CODE HERE:
#   Implement a FC-net that achieves at least 60% validation accuracy
#   on CIFAR-10.
# ================================================================ #


hidden_dims = [500, 500, 500, 500]
learning_rates = [1e-2, 1e-3, 1e-4]
optimizer = ['adam', 'sgd_nesterov_momentum']
weight_scale = [1e-2, 1e-3]
lr_decay = [.9, .95]
bs  = 200
epochs = 30
dropout = [.2, .3]
use_bn = ['True', 'False']

best_val_acc = -1
best_params = None
best_settings = None

"""

for lr in learning_rates:
    for op in optimizer:
        for ws in weight_scale:
            for dec in lr_decay:
                for do in dropout:
                    for bn in use_bn:

                    model = FullyConnectedNet(hidden_dims, dropout=d
o,
```

```
                                             weight_scale=ws, use_bat
chnorm = bn)

                    solver = Solver(model, data,
                                   num_epochs=epochs, batch_size =bs,
                                   update_rule = op,
                                   optim_config= {
                                       'learning_rate' : lr
                                   },
                                   lr_decay=dec,
                                   print_every=400)

                    solver.train()


                    print solver.best_val_acc

                    if solver.best_val_acc > best_val_acc:
                        best_val_acc = solver.best_val_acc
                        best_params = solver.best_params
                        best_settings = (lr, op, ws, dec, do)
                        best_model = model







"""




model = FullyConnectedNet(hidden_dims, dropout=.35,
                         weight_scale=1e-2, use_batchnorm=True)

solver = Solver(model, data,
               num_epochs=epochs, batch_size=bs,
               update_rule = 'adam',
               optim_config= {
                   'learning_rate' : 5e-4
               },
               lr_decay=.95,
               print_every=400)

time_start = time.time()

solver.train()

print 'Training time: {}'.format(time.time()-time_start)




# ================================================================ #
# END YOUR CODE HERE
# ================================================================ #
```

```
(Iteration 1 / 7350) loss: 2.349134
(Epoch 0 / 30) train acc: 0.144000; val_acc: 0.146000
(Epoch 1 / 30) train acc: 0.455000; val_acc: 0.456000
(Iteration 401 / 7350) loss: 1.511353
(Epoch 2 / 30) train acc: 0.496000; val_acc: 0.495000
(Epoch 3 / 30) train acc: 0.542000; val_acc: 0.521000
(Iteration 801 / 7350) loss: 1.495701
(Epoch 4 / 30) train acc: 0.553000; val_acc: 0.521000
(Iteration 1201 / 7350) loss: 1.305079
(Epoch 5 / 30) train acc: 0.592000; val_acc: 0.528000
(Epoch 6 / 30) train acc: 0.568000; val_acc: 0.530000
(Iteration 1601 / 7350) loss: 1.346184
(Epoch 7 / 30) train acc: 0.592000; val_acc: 0.557000
(Epoch 8 / 30) train acc: 0.583000; val_acc: 0.535000
(Iteration 2001 / 7350) loss: 1.166619
(Epoch 9 / 30) train acc: 0.607000; val_acc: 0.554000
(Iteration 2401 / 7350) loss: 1.082266
(Epoch 10 / 30) train acc: 0.638000; val_acc: 0.570000
(Epoch 11 / 30) train acc: 0.650000; val_acc: 0.562000
(Iteration 2801 / 7350) loss: 1.148503
(Epoch 12 / 30) train acc: 0.674000; val_acc: 0.564000
(Epoch 13 / 30) train acc: 0.663000; val_acc: 0.568000
(Iteration 3201 / 7350) loss: 1.081492
(Epoch 14 / 30) train acc: 0.667000; val_acc: 0.571000
(Iteration 3601 / 7350) loss: 1.148346
(Epoch 15 / 30) train acc: 0.688000; val_acc: 0.584000
(Epoch 16 / 30) train acc: 0.704000; val_acc: 0.574000
(Iteration 4001 / 7350) loss: 1.077373
(Epoch 17 / 30) train acc: 0.697000; val_acc: 0.576000
(Iteration 4401 / 7350) loss: 0.881307
(Epoch 18 / 30) train acc: 0.691000; val_acc: 0.586000
(Epoch 19 / 30) train acc: 0.725000; val_acc: 0.582000
(Iteration 4801 / 7350) loss: 0.826899
(Epoch 20 / 30) train acc: 0.715000; val_acc: 0.580000
(Epoch 21 / 30) train acc: 0.745000; val_acc: 0.584000
(Iteration 5201 / 7350) loss: 0.943966
(Epoch 22 / 30) train acc: 0.734000; val_acc: 0.592000
(Iteration 5601 / 7350) loss: 0.772600
(Epoch 23 / 30) train acc: 0.756000; val_acc: 0.593000
(Epoch 24 / 30) train acc: 0.738000; val_acc: 0.592000
(Iteration 6001 / 7350) loss: 0.953089
(Epoch 25 / 30) train acc: 0.748000; val_acc: 0.591000
(Epoch 26 / 30) train acc: 0.761000; val_acc: 0.599000
(Iteration 6401 / 7350) loss: 0.984805
(Epoch 27 / 30) train acc: 0.780000; val_acc: 0.600000
(Iteration 6801 / 7350) loss: 0.946586
(Epoch 28 / 30) train acc: 0.761000; val_acc: 0.584000
(Epoch 29 / 30) train acc: 0.792000; val_acc: 0.604000
(Iteration 7201 / 7350) loss: 0.825021
(Epoch 30 / 30) train acc: 0.768000; val_acc: 0.592000
Training time: 1063.93349099
```

In [12]:
```
print solver.best_val_acc
plt.subplot(2, 1, 1)
plt.title('Training loss')
plt.xlabel('Iteration')
plt.plot(solver.loss_history, 'o')

plt.subplot(2, 1, 2)
plt.plot(solver.train_acc_history, '-o', label='train')
plt.plot(solver.val_acc_history, '-o', label='val')
plt.plot(.6*np.ones_like(solver.val_acc_history), '--', label='targe
t')
plt.title('Accuracy History')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.gcf().set_size_inches(15, 15)
plt.show()
```

```
0.604
```