Dropout

Dropout [1] is a technique for regularizing neural networks by randomly setting some features to zero during the forward pass. In this exercise you will implement a dropout layer and modify your fully-connected network to optionally use dropout.

[1] Geoffrey E. Hinton et al, "Improving neural networks by preventing co-adaptation of feature detectors", arXiv 2012

In [3]:

```
# As usual, a bit of setup
from __future__ import print_function
import time
import numpy as np
import matplotlib.pyplot as plt
from cs231n.classifiers.fc_net import *
from cs231n.data_utils import get_CIFAR10_data
from cs231n.gradient_check import eval_numerical_gradient, eval_numerical_grad
ient_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-ipyt
hon
%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 autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [5]:

```
# Load the (preprocessed) CIFAR10 data.

data = get_CIFAR10_data()
for k, v in data.items():
  print('%s: ' % k, v.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

In the file `cs231n/layers.py` , implement the forward pass for dropout. Since dropout behaves differently during training and testing, make sure to implement the operation for both modes.

Once you have done so, run the cell below to test your implementation.

In [6]:

```
np.random.seed(231)
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())
  print()
```

```
Running tests with p =  0.3
Mean of input:  10.000207878477502
Mean of train-time output:  10.035072797050494
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.699124
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.6
Mean of input:  10.000207878477502
Mean of train-time output:  9.976910758765856
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.401368
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.75
Mean of input:  10.000207878477502
Mean of train-time output:  9.993068588261146
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.250496
Fraction of test-time output set to zero:  0.0
```

Dropout backward pass

In the file `cs231n/layers.py` , implement the backward pass for dropout. After doing so, run the following cell to numerically gradient-check your implementation.

In [7]:

```
np.random.seed(231)
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:  5.445612718272284e-11
```

Fully-connected nets with Dropout

In the file `cs231n/classifiers/fc_net.py` , modify your implementation to use dropout. Specificially, if the constructor the the net receives a nonzero value for the `dropout` parameter, then the net should add dropout immediately after every ReLU nonlinearity. After doing so, run the following to numerically gradient-check your implementation.

In [8]:

```
np.random.seed(231)
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('%s relative error: %.2e' % (name, rel_error(grad_num, grads[name]))
)
  print()
```

```
Running check with dropout =  0
Initial loss:  2.3004790897684924
W1 relative error: 1.48e-07
W2 relative error: 2.21e-05
W3 relative error: 3.53e-07
b1 relative error: 5.38e-09
b2 relative error: 2.09e-09
b3 relative error: 5.80e-11

Running check with dropout =  0.25
Initial loss:  2.2924325088330475
W1 relative error: 2.74e-08
W2 relative error: 2.98e-09
W3 relative error: 4.29e-09
b1 relative error: 7.78e-10
b2 relative error: 3.36e-10
b3 relative error: 1.65e-10

Running check with dropout =  0.5
Initial loss:  2.3042759220785896
W1 relative error: 3.11e-07
W2 relative error: 1.84e-08
W3 relative error: 5.35e-08
b1 relative error: 2.58e-08
b2 relative error: 2.99e-09
b3 relative error: 1.13e-10
```

Regularization experiment

As an experiment, we will train a pair of two-layer networks on 500 training examples: one will use no dropout, and one will use a dropout probability of 0.75. We will then visualize the training and validation accuracies of the two networks over time.

In [9]:

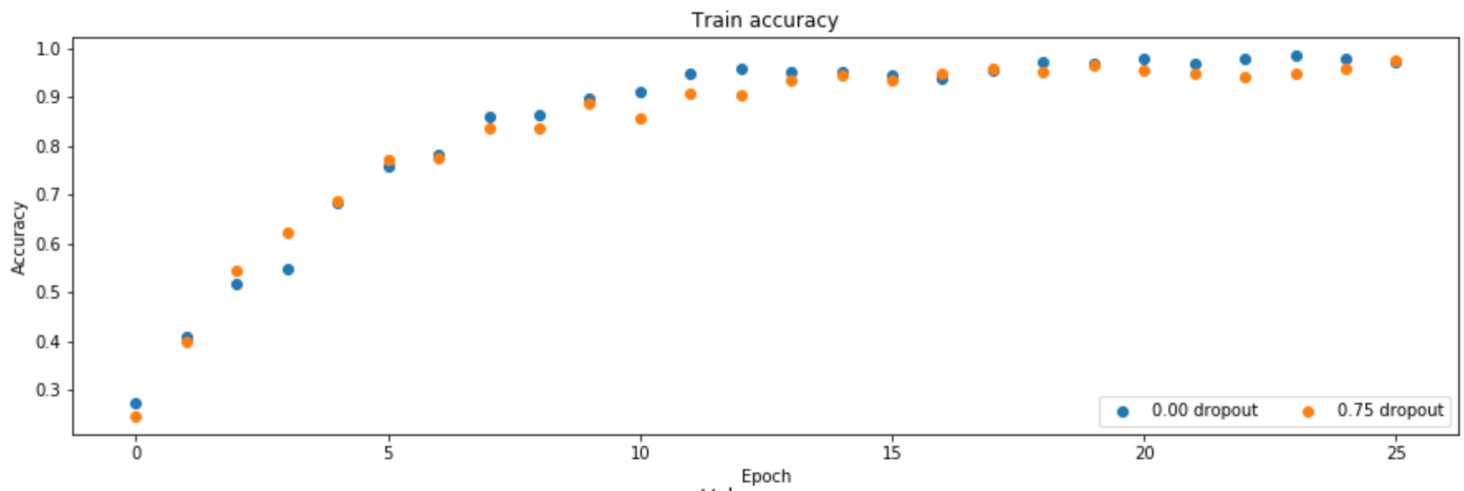
```
# Train two identical nets, one with dropout and one without
np.random.seed(231)
num_train = 500
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

solvers = {}
dropout_choices = [0, 0.75]
for dropout in dropout_choices:
  model = FullyConnectedNet([500], dropout=dropout)
  print(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
```

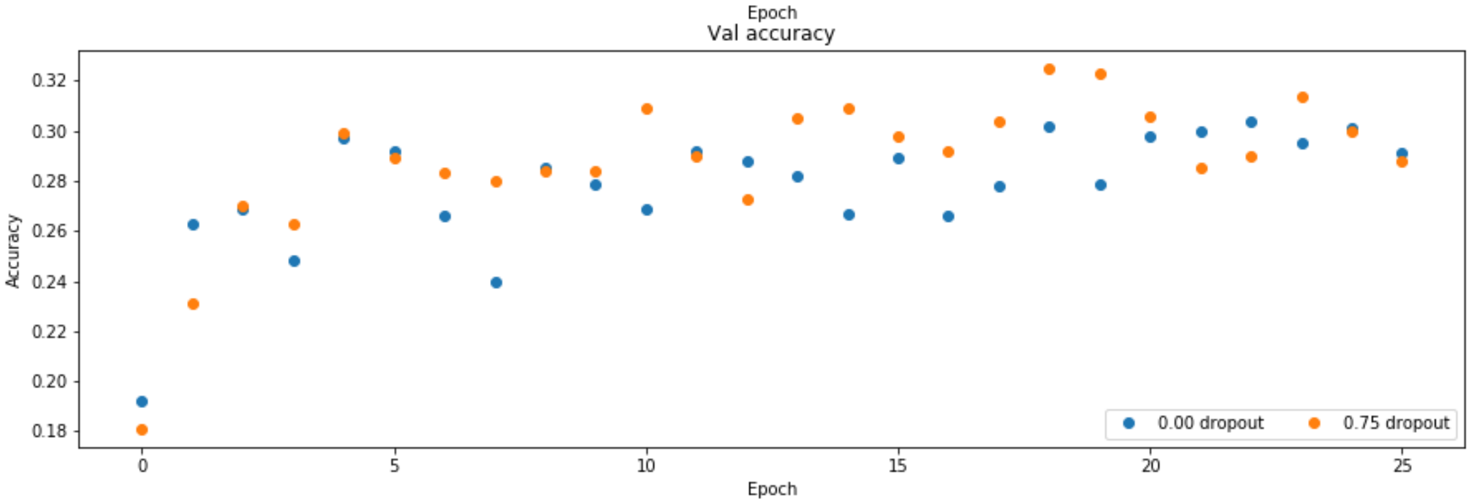
```
0
(Iteration 1 / 125) loss: 7.856644
(Epoch 0 / 25) train acc: 0.274000; val_acc: 0.192000
(Epoch 1 / 25) train acc: 0.410000; val_acc: 0.263000
(Epoch 2 / 25) train acc: 0.518000; val_acc: 0.269000
(Epoch 3 / 25) train acc: 0.550000; val_acc: 0.248000
(Epoch 4 / 25) train acc: 0.684000; val_acc: 0.297000
(Epoch 5 / 25) train acc: 0.758000; val_acc: 0.292000
(Epoch 6 / 25) train acc: 0.782000; val_acc: 0.266000
(Epoch 7 / 25) train acc: 0.860000; val_acc: 0.240000
(Epoch 8 / 25) train acc: 0.864000; val_acc: 0.285000
(Epoch 9 / 25) train acc: 0.898000; val_acc: 0.279000
(Epoch 10 / 25) train acc: 0.910000; val_acc: 0.269000
(Epoch 11 / 25) train acc: 0.948000; val_acc: 0.292000
(Epoch 12 / 25) train acc: 0.960000; val_acc: 0.288000
(Epoch 13 / 25) train acc: 0.952000; val_acc: 0.282000
(Epoch 14 / 25) train acc: 0.952000; val_acc: 0.267000
(Epoch 15 / 25) train acc: 0.944000; val_acc: 0.289000
(Epoch 16 / 25) train acc: 0.940000; val_acc: 0.266000
(Epoch 17 / 25) train acc: 0.956000; val_acc: 0.278000
(Epoch 18 / 25) train acc: 0.972000; val_acc: 0.302000
(Epoch 19 / 25) train acc: 0.968000; val_acc: 0.279000
(Epoch 20 / 25) train acc: 0.980000; val_acc: 0.298000
(Iteration 101 / 125) loss: 0.240881
(Epoch 21 / 25) train acc: 0.968000; val_acc: 0.300000
(Epoch 22 / 25) train acc: 0.980000; val_acc: 0.304000
(Epoch 23 / 25) train acc: 0.986000; val_acc: 0.295000
(Epoch 24 / 25) train acc: 0.980000; val_acc: 0.301000
(Epoch 25 / 25) train acc: 0.974000; val_acc: 0.291000
```

```
0.75
(Iteration 1 / 125) loss: 11.299055
(Epoch 0 / 25) train acc: 0.246000; val_acc: 0.181000
(Epoch 1 / 25) train acc: 0.400000; val_acc: 0.231000
(Epoch 2 / 25) train acc: 0.544000; val_acc: 0.270000
(Epoch 3 / 25) train acc: 0.622000; val_acc: 0.263000
(Epoch 4 / 25) train acc: 0.688000; val_acc: 0.299000
(Epoch 5 / 25) train acc: 0.774000; val_acc: 0.289000
(Epoch 6 / 25) train acc: 0.776000; val_acc: 0.283000
(Epoch 7 / 25) train acc: 0.836000; val_acc: 0.280000
(Epoch 8 / 25) train acc: 0.838000; val_acc: 0.284000
(Epoch 9 / 25) train acc: 0.888000; val_acc: 0.284000
(Epoch 10 / 25) train acc: 0.858000; val_acc: 0.309000
(Epoch 11 / 25) train acc: 0.908000; val_acc: 0.290000
(Epoch 12 / 25) train acc: 0.904000; val_acc: 0.273000
(Epoch 13 / 25) train acc: 0.934000; val_acc: 0.305000
(Epoch 14 / 25) train acc: 0.944000; val_acc: 0.309000
(Epoch 15 / 25) train acc: 0.934000; val_acc: 0.298000
(Epoch 16 / 25) train acc: 0.950000; val_acc: 0.292000
(Epoch 17 / 25) train acc: 0.958000; val_acc: 0.304000
(Epoch 18 / 25) train acc: 0.954000; val_acc: 0.325000
(Epoch 19 / 25) train acc: 0.966000; val_acc: 0.323000
(Epoch 20 / 25) train acc: 0.956000; val_acc: 0.306000
(Iteration 101 / 125) loss: 0.412030
(Epoch 21 / 25) train acc: 0.948000; val_acc: 0.285000
(Epoch 22 / 25) train acc: 0.942000; val_acc: 0.290000
(Epoch 23 / 25) train acc: 0.948000; val_acc: 0.314000
(Epoch 24 / 25) train acc: 0.958000; val_acc: 0.300000
(Epoch 25 / 25) train acc: 0.976000; val_acc: 0.288000
```

In [10]:

```
# 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' % dro
pout)
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' % dropo
ut)
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

Explain what you see in this experiment. What does it suggest about dropout?

Answer

During training their doesn't seem to be any difference, but we can see that the validation accuracy with dropout is mostly higher than without which suggest better generalization.