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](https://arxiv.org/abs/1207.0580)

```
# 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_gradient_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))))
```

```
# 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,)
```

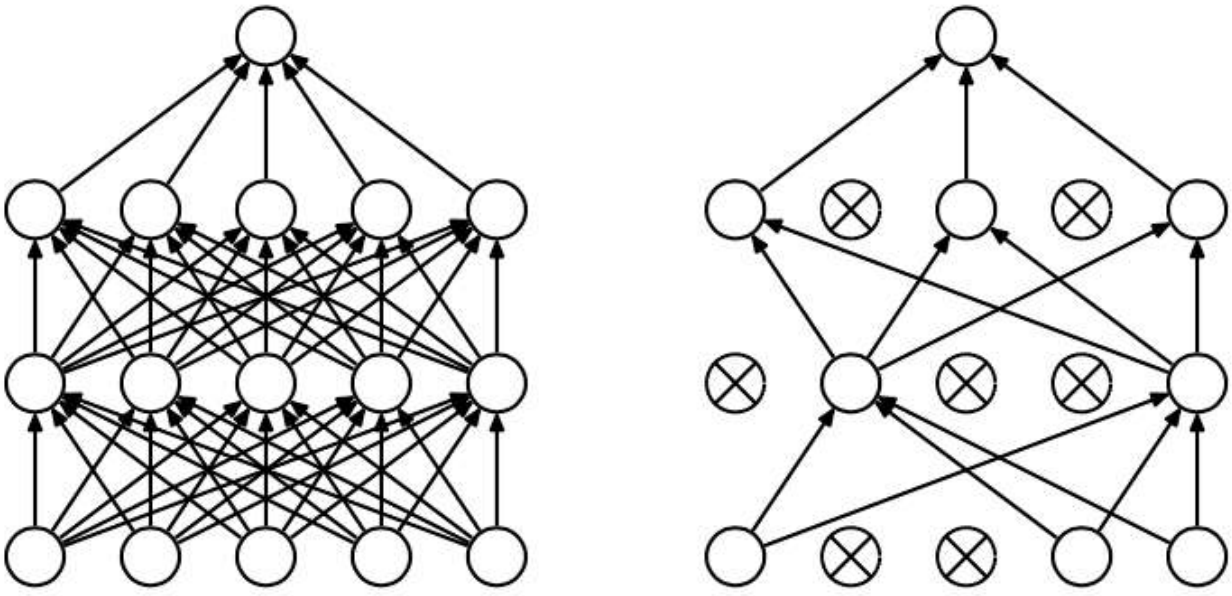
Dropout forward pass

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.

解释：

Dropout 是为了解决过拟合的一种手段。当模型太过复杂的时候，容易学习过度，一种解决方法是训练多个模型做融合，但是这样很费时间，这时我们可以使用Dropout技术从原始网络中找出一个更精简的网络。即对每个神经元以概率P保留它。它强迫一个神经单元，和随机挑选出来的其他神经单元共同工作，达到好的效果。消除减弱了神经元节点间的联合适应性，增强了泛化能力。即从下图左边到右边的转变。



因而，对于一个有n个节点的神经网络，有了dropout后，就可以看做是2^n个模型的集合了，但此时要训练的参数数目却是不变的，这就解决了费时的问题。

虽然训练的时候我们使用了dropout， 但是在测试时，我们不使用dropout （不对网络的参数做任何丢弃，这时dropout layer相当于进来什么就输出什么）。然后，把测试时dropout layer的输出乘以训练时使用的retaining probability **p** 。由于我们在测试时不做任何的参数丢弃，，导致在统计意义下，测试时 每层 dropout layer的输出比训练时的输出多加了【（1 - p）x 100】% units 的输出。 即 【p x 100】% 个units 的和 是同训练时随机采样得到的子网络的输出一致，另【（1 - p）x 100】% 的units的和是本来应该扔掉但是又在测试阶段被保留下来的。所以，为了使得dropout layer 下一层的输入和训练时具有相同的“意义”和“数量级”，我们要对测试时的伪dropout layer的输出（即下层的输入）做 rescale： 乘以一个p，表示最后的sum中只有这么大的概率，或者这么多的部分被保留。这样以来，只要一次测试，就将原2^n个子网络的参数全部考虑进来了，并且最后的 rescale 保证了后面一层的输入仍然符合相应的物理意义和数量级。

假设x是dropout layer的输入，y是dropout layer的输出，W是上一层的所有weight parameters， Wp是以 retaining probability 为p 采样得到的weight parameter子集。可得（忽略bias）：

train：$\boldsymbol{y = Wp * x}$

test: $\boldsymbol{y = W * p * x}$

但是一般写程序的时候，我们想直接在test时用 $\boldsymbol{y = W^{/} * x}$的形式（$\boldsymbol{W^{/} = W * p}$）因此我们就在训练的时候就直接训练$\boldsymbol{y = W^{/}}$。所以训练时，第一个公式修正为 $\boldsymbol{y = \frac{Wp}{p} * x = Wp * \frac{x}{p}}$。即把dropout的输入乘以p 再进行训练，这样得到的训练得到的weight 参数就是$\boldsymbol{W^{/}}$,测试的时候除了不使用dropout外，不需要再做任何rescale。

前向传播代码如下：

```
#Train:
mask = (np.random.rand(*x.shape) <= p ) / p
out = x * mask

#Test:
out = x
```

- Tips:dropout率的选择
  • 经过交叉验证，隐含节点dropout率等于0.5的时候效果最好，原因是0.5的时候dropout随机生成的网络结构最多。
  • dropout也可以被用作一种添加噪声的方法，直接对input进行操作。输入层设为更接近1的数。使得输入变化不会太大（0.8）

```
np.random.seed(231)
x = np.random.randn(500, 500) + 10

for p in [0.25, 0.4, 0.7]:
  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 =  0.25
Mean of input:  10.000207878477502
Mean of train-time output:  10.014059116977283
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.749784
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.4
Mean of input:  10.000207878477502
Mean of train-time output:  9.977917658761159
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.600796
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.7
Mean of input:  10.000207878477502
Mean of train-time output:  9.987811912159426
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.30074

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.

解释：

在训练时，我们只需要调整未被dropout节点的权重即可，所以后一层往回传过来的结果去掉跟dropout的点相关的梯度，代码如下：

```
dx = dout * mask
```

在测试时，就不需要变了：

```
dx = dout
```

```
np.random.seed(231)
x = np.random.randn(10, 10) + 10
dout = np.random.randn(*x.shape)

dropout_param = {'mode': 'train', 'p': 0.2, '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)

# Error should be around e-10 or less
print('dx relative error: ', rel_error(dx, dx_num))
```

```
dx relative error:  5.44560814873387e-11
```

Inline Question 1:

What happens if we do not divide the values being passed through inverse dropout by `p` in the dropout layer? Why does that happen?

Answer:

这样得到结果的期望是不对，设原期望为 EX，则经过dropout后的期望就是：$EX * p + 0 * (1 - p)$，假设没$*p$，那么期望会变大

Fully-connected nets with Dropout

In the file `cs231n/classifiers/fc_net.py`, modify your implementation to use dropout. Specifically, if the constructor of the net receives a value that is not 1 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.

注解：

在每一层结束之后（除了输出层），将这一层的输出调用已经写好函数dropout_forward来使用dropout。反向传播时也一样，但这时要先通过dropout层再进行梯度计算，调用的接口是dropout_backward，完整代码详见`cs231n/classifiers/fc_net.py`

```
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 [1, 0.75, 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)

  # Relative errors should be around e-6 or less; Note that it's fine
  # if for dropout=1 you have W2 error be on the order of e-5.
  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 =  1
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 =  0.75
Initial loss:  2.302371489704412
W1 relative error: 1.90e-07
W2 relative error: 4.76e-06
W3 relative error: 2.60e-08
b1 relative error: 4.73e-09
b2 relative error: 1.82e-09
b3 relative error: 1.70e-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 keep probability of 0.25. We will then visualize the training and validation accuracies of the two networks over time.

```
# 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 = [1, 0.25]
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
```

```
1
(Iteration 1 / 125) loss: 7.856643
(Epoch 0 / 25) train acc: 0.260000; val_acc: 0.184000
(Epoch 1 / 25) train acc: 0.416000; val_acc: 0.258000
(Epoch 2 / 25) train acc: 0.482000; val_acc: 0.276000
(Epoch 3 / 25) train acc: 0.532000; val_acc: 0.277000
(Epoch 4 / 25) train acc: 0.600000; val_acc: 0.271000
(Epoch 5 / 25) train acc: 0.708000; val_acc: 0.299000
(Epoch 6 / 25) train acc: 0.722000; val_acc: 0.282000
(Epoch 7 / 25) train acc: 0.832000; val_acc: 0.255000
(Epoch 8 / 25) train acc: 0.878000; val_acc: 0.269000
(Epoch 9 / 25) train acc: 0.902000; val_acc: 0.275000
(Epoch 10 / 25) train acc: 0.890000; val_acc: 0.261000
(Epoch 11 / 25) train acc: 0.930000; val_acc: 0.282000
(Epoch 12 / 25) train acc: 0.958000; val_acc: 0.300000
(Epoch 13 / 25) train acc: 0.964000; val_acc: 0.305000
(Epoch 14 / 25) train acc: 0.962000; val_acc: 0.314000
(Epoch 15 / 25) train acc: 0.964000; val_acc: 0.304000
(Epoch 16 / 25) train acc: 0.982000; val_acc: 0.309000
(Epoch 17 / 25) train acc: 0.972000; val_acc: 0.323000
(Epoch 18 / 25) train acc: 0.992000; val_acc: 0.316000
(Epoch 19 / 25) train acc: 0.984000; val_acc: 0.303000
(Epoch 20 / 25) train acc: 0.986000; val_acc: 0.313000
(Iteration 101 / 125) loss: 0.011629
(Epoch 21 / 25) train acc: 0.994000; val_acc: 0.306000
(Epoch 22 / 25) train acc: 0.976000; val_acc: 0.312000
(Epoch 23 / 25) train acc: 0.962000; val_acc: 0.319000
(Epoch 24 / 25) train acc: 0.978000; val_acc: 0.306000
(Epoch 25 / 25) train acc: 0.982000; val_acc: 0.308000
0.25
(Iteration 1 / 125) loss: 17.318480
(Epoch 0 / 25) train acc: 0.230000; val_acc: 0.177000
(Epoch 1 / 25) train acc: 0.378000; val_acc: 0.243000
(Epoch 2 / 25) train acc: 0.402000; val_acc: 0.254000
(Epoch 3 / 25) train acc: 0.502000; val_acc: 0.276000
(Epoch 4 / 25) train acc: 0.528000; val_acc: 0.298000
(Epoch 5 / 25) train acc: 0.562000; val_acc: 0.296000
(Epoch 6 / 25) train acc: 0.626000; val_acc: 0.291000
(Epoch 7 / 25) train acc: 0.622000; val_acc: 0.297000
(Epoch 8 / 25) train acc: 0.688000; val_acc: 0.313000
(Epoch 9 / 25) train acc: 0.712000; val_acc: 0.297000
(Epoch 10 / 25) train acc: 0.724000; val_acc: 0.308000
(Epoch 11 / 25) train acc: 0.768000; val_acc: 0.308000
(Epoch 12 / 25) train acc: 0.772000; val_acc: 0.285000
(Epoch 13 / 25) train acc: 0.824000; val_acc: 0.310000
(Epoch 14 / 25) train acc: 0.804000; val_acc: 0.342000
(Epoch 15 / 25) train acc: 0.848000; val_acc: 0.345000
(Epoch 16 / 25) train acc: 0.838000; val_acc: 0.298000
(Epoch 17 / 25) train acc: 0.850000; val_acc: 0.307000
(Epoch 18 / 25) train acc: 0.858000; val_acc: 0.328000
(Epoch 19 / 25) train acc: 0.878000; val_acc: 0.316000
(Epoch 20 / 25) train acc: 0.872000; val_acc: 0.312000

(Iteration 101 / 125) loss: 4.141964
```

```
(Epoch 21 / 25) train acc: 0.912000; val_acc: 0.315000
(Epoch 22 / 25) train acc: 0.898000; val_acc: 0.300000
(Epoch 23 / 25) train acc: 0.910000; val_acc: 0.297000
(Epoch 24 / 25) train acc: 0.912000; val_acc: 0.317000
(Epoch 25 / 25) train acc: 0.896000; val_acc: 0.323000
```

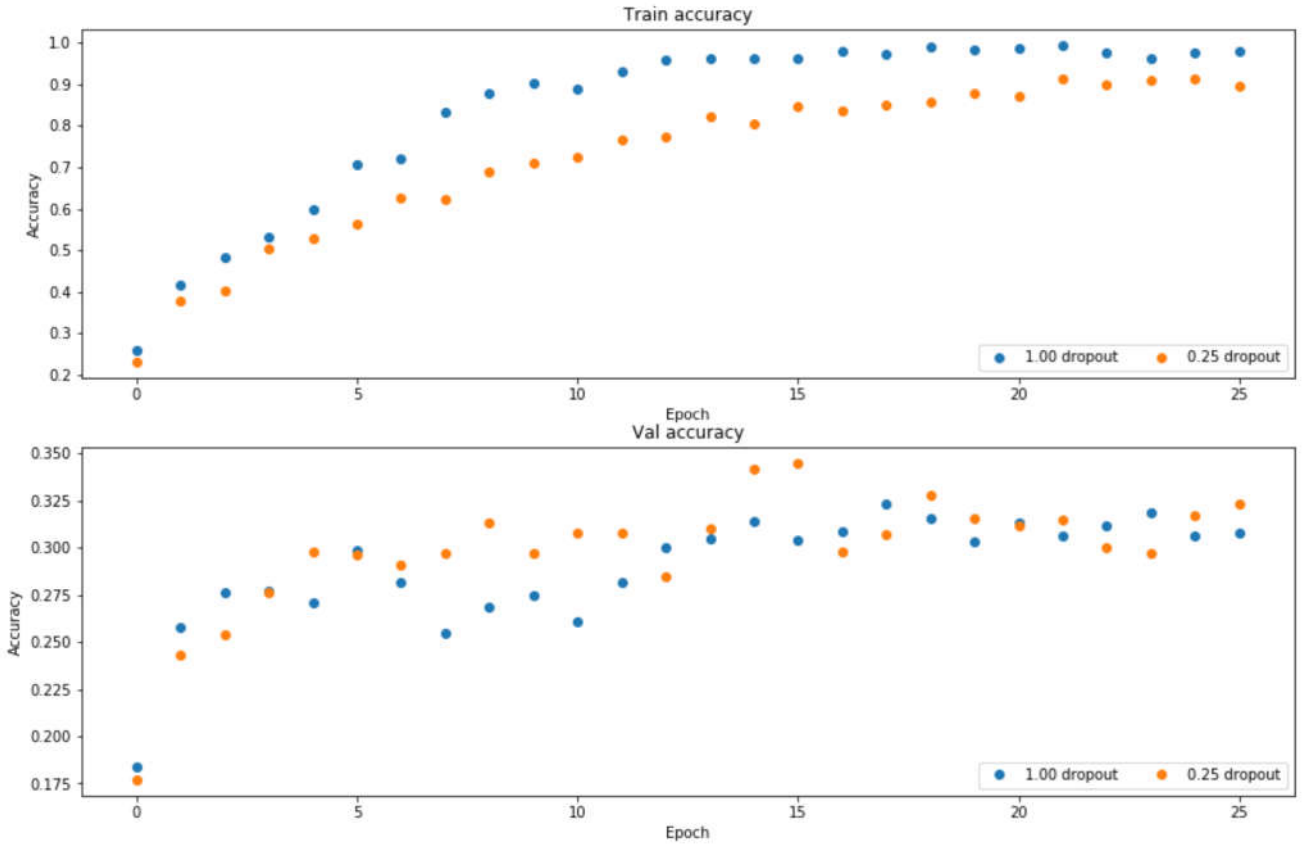
```
# Plot train and validation accuracies of the two models

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' % 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 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()
```



Inline Question 2:

Compare the validation and training accuracies with and without dropout -- what do your results suggest about dropout as a regularizer?

Answer:

通过上面图我们发现，使用dropout的验证集的正确率稍微比没使用的高一点。但是训练集的正确率却比没使用dropout的低，这说明dropout的确可以防止过拟合。我们在网络结构复杂的情况下最后加入dropout技术来防止过拟合。

Inline Question 3:

Suppose we are training a deep fully-connected network for image classification, with dropout after hidden layers (parameterized by keep probability p). How should we modify p, if at all, if we decide to decrease the size of the hidden layers (that is, the number of nodes in each layer)?

Answer:

Dropout的p应该全局都一样，不然减少隐藏层节点数会让p的设计很困难。