Dropout

Dropout [1] 是一种通过在正向传播中将一些输出随机设置为零，神经网络正则化的方法。在这个练习中，你将实现一个dropout层，并修改你的全连接网络使其可选择的使用dropout

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

In [2]:

```
# As usual, a bit of setup
from __future__ import print_function
import time
import numpy as np
import matplotlib.pyplot as plt
from daseCV.classifiers.fc_net import *
from daseCV.data_utils import get_CIFAR10_data
from daseCV.gradient_check import eval_numerical_gradient, eval_numerical_gradient_ar
from daseCV.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))))
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [3]:

```
# 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 正向传播

在文件 `daseCV/layers.py` 中完成dropout的正向传播过程。由于dropout在训练和测试期间的行为是不同的，因此请确保两种模式下都实现完成。

完成此操作后，运行下面的cell以测试你的代码。

In [4]:

```
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 反向传播

在文件 `daseCV/layers.py` 中完成dropout的反向传播。完成之后运行以下cell以对你的实现代码进行梯度检查。

In [5]:

```
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)[

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

```
dx relative error:  5.445608148733387e-11
```

问题 1:

在dropout层，如果不让inverse dropout技术过的数据除以p，会发生什么？为什么会这样呢？

回答:

如果我们不将值除以p，那么在测试时我们将不会考虑训练输出的平均值。因此，我们将偏向于考虑可能导致大值（爆炸梯度）的所有可能子网络的总和。test time的值也将变为原来的p倍。

除了Mean of train-time output 和Mean of test-time output 会变，其他结果，反向传播的结果都不会变，因为对训练输出做整体的拉伸只要在test 的时候拉伸回来，就相当于没操作。写成 inverteddropout 只是方便在test 阶段不操作。

全连接网络的Dropout

修改 `daseCV/classifiers/fc_net.py` 文件完成使用dropout的部分。具体来说，如果网络的构造函数收到的 `dropout` 参数值不为1，则应在每个ReLU之后添加一个dropout层。完成之后，运行以下命令以对你的代码进行梯度检查。

In [6]:

```
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: 5.37e-09
b2 relative error: 2.99e-09
b3 relative error: 1.13e-10
```

正则化实验

作为实验，我们将在500个样本上训练一对双层网络：一个不使用dropout，另一个使用概率为0.25的dropout。之后，我们将可视化这两个网络训练和验证的准确度。

In [7]:

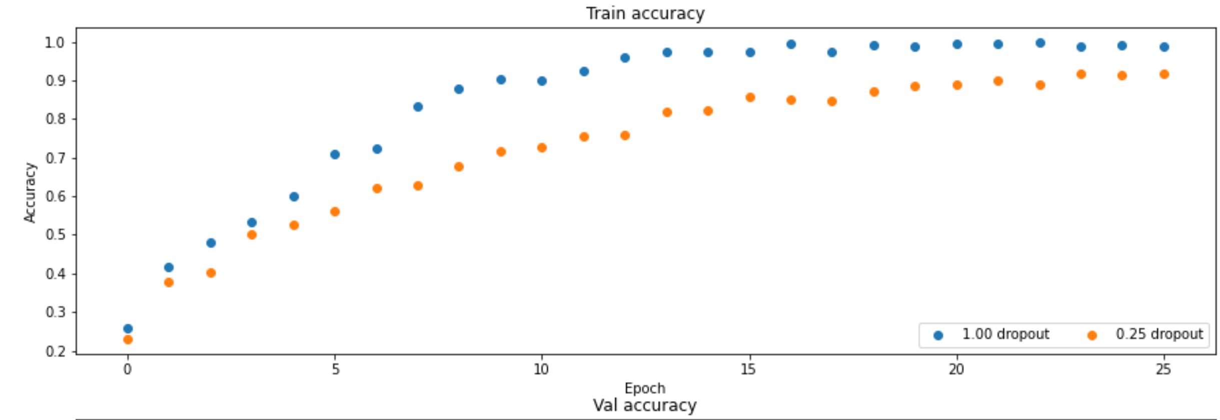
```
# Train two identical nets, one with dropout and one without
np.random.seed(231)
num_train = 500
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

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
  print()
```

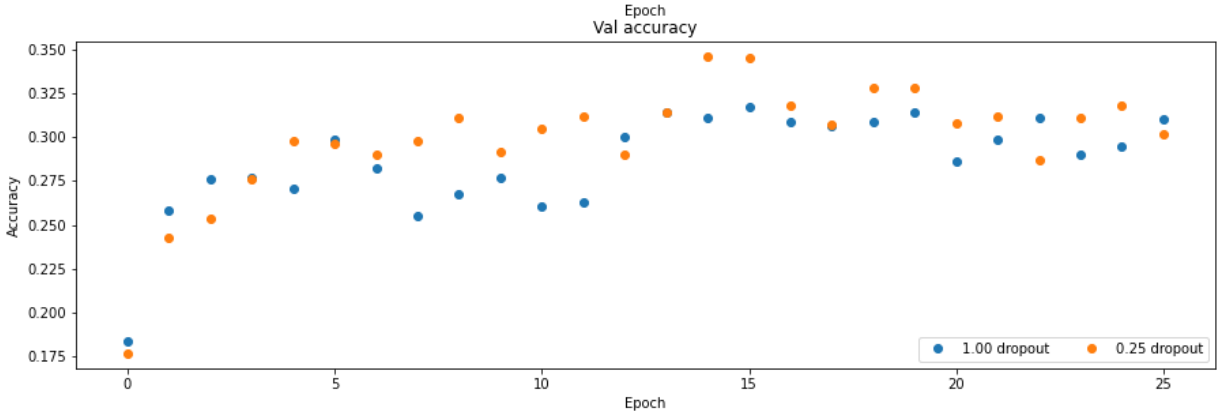
```
1
(Iteration 1 / 125) loss: 7.856643
(Epoch 0 / 25) train acc: 0.260000; val_acc: 0.184000
(Epoch 1 / 25) train acc: 0.416000; val_acc: 0.258000
(Epoch 2 / 25) train acc: 0.482000; val_acc: 0.276000
(Epoch 3 / 25) train acc: 0.532000; val_acc: 0.277000
(Epoch 4 / 25) train acc: 0.600000; val_acc: 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: 0.880000; val_acc: 0.268000
(Epoch 9 / 25) train acc: 0.902000; val_acc: 0.277000
(Epoch 10 / 25) train acc: 0.898000; val_acc: 0.261000
(Epoch 11 / 25) train acc: 0.924000; val_acc: 0.263000
(Epoch 12 / 25) train acc: 0.960000; val_acc: 0.300000
(Epoch 13 / 25) train acc: 0.972000; val_acc: 0.314000
(Epoch 14 / 25) train acc: 0.972000; val_acc: 0.311000
(Epoch 15 / 25) train acc: 0.972000; val_acc: 0.317000
(Epoch 16 / 25) train acc: 0.994000; val_acc: 0.309000
(Epoch 17 / 25) train acc: 0.974000; val_acc: 0.306000
(Epoch 18 / 25) train acc: 0.990000; val_acc: 0.309000
(Epoch 19 / 25) train acc: 0.986000; val_acc: 0.314000
(Epoch 20 / 25) train acc: 0.994000; val_acc: 0.286000
(Iteration 101 / 125) loss: 0.036931
(Epoch 21 / 25) train acc: 0.994000; val_acc: 0.299000
```

```
(Epoch 22 / 25) train acc: 0.998000; val_acc: 0.311000
(Epoch 23 / 25) train acc: 0.988000; val_acc: 0.290000
(Epoch 24 / 25) train acc: 0.992000; val_acc: 0.295000
(Epoch 25 / 25) train acc: 0.986000; val_acc: 0.310000

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: 0.620000; val_acc: 0.290000
(Epoch 7 / 25) train acc: 0.628000; val_acc: 0.298000
(Epoch 8 / 25) train acc: 0.678000; val_acc: 0.311000
(Epoch 9 / 25) train acc: 0.718000; val_acc: 0.292000
(Epoch 10 / 25) train acc: 0.728000; val_acc: 0.305000
(Epoch 11 / 25) train acc: 0.754000; val_acc: 0.312000
(Epoch 12 / 25) train acc: 0.760000; val_acc: 0.290000
(Epoch 13 / 25) train acc: 0.818000; val_acc: 0.314000
(Epoch 14 / 25) train acc: 0.822000; val_acc: 0.346000
(Epoch 15 / 25) train acc: 0.856000; val_acc: 0.345000
(Epoch 16 / 25) train acc: 0.852000; val_acc: 0.318000
(Epoch 17 / 25) train acc: 0.848000; val_acc: 0.307000
(Epoch 18 / 25) train acc: 0.870000; val_acc: 0.328000
(Epoch 19 / 25) train acc: 0.886000; val_acc: 0.328000
(Epoch 20 / 25) train acc: 0.890000; val_acc: 0.308000
(Iteration 101 / 125) loss: 3.571001
(Epoch 21 / 25) train acc: 0.898000; val_acc: 0.312000
(Epoch 22 / 25) train acc: 0.890000; val_acc: 0.287000
(Epoch 23 / 25) train acc: 0.916000; val_acc: 0.311000
(Epoch 24 / 25) train acc: 0.914000; val_acc: 0.318000
(Epoch 25 / 25) train acc: 0.918000; val_acc: 0.302000
```

In [8]:

```
# 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()
```





问题 2:

对比有无dropout的验证和训练的精度，你对使用dropout作为正则化有何建议?

回答:

使用dropout后模型在训练集上的效果会降低；但是验证集上的效果则提升了，也就是泛化能力提高了。

所以为了提高模型的泛化能力(防止过拟合)可以使用dropout作为正则化。

由于前几个epoch 学到的还是渐层特征，所以dropout 的负面影响远小于防止过拟合的正面影响，准确率显著增高，但是到后面训练深层特征时再使用高p 的dropout，会影响识别些比较重要的特征，所以建议后层的dropout 可采用类似学习率算法之类动态变化。另外，dropout 的思想真的不错。CV 预处理图片时使用的cutout 方法就借鉴了dropout 的思想，随机将图片挖掉一小块，会提升训练效果。我在矿石识别项目上将图片切成小块重新排列，也在最初的模型中提升了2% 左右的准确率，这样避免了识别岩石的轮廓。

问题三 3:

假设我们正在训练一个深层的全连接网络用以进行图像分类，并隐层之后dropout（通过使用概率p进行参数化）。如果我们担心过度拟合而决定减小隐层的大小（即每层中的节点数）时，应该如何修改p（如果有的话）？

回答:

在隐层大小还算大的时候，不需要修改p，因为将被丢弃的神经元数量将与隐藏层的大小成正比。当然如果隐层的大小过小时还是需要稍微减小p的，否则就比较随机了。

例如，假设我们在隐藏层中有n个神经元，并且我们使用$p = 0.5$。因此，丢弃的神经元的期望数量是$p * n$。如果我们将隐藏层中的神经元数量减少到$n/2$并使用相同的$p = 0.5$，那么丢弃神经元的期望数量将为$p * n/2$。因此,当我们改变隐藏层的大小时，我们不需要修改保持概率p。

减小p，本来模型就小，维持p 会损失太多重要特征信息。

In [ ]:

```

```