In [1]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'Colab Notebooks/assignment2'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd /content/drive/MyW Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/MyW Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/Colab Notebooks/assignment2/cs231n/datasets
/content/drive/My Drive/Colab Notebooks/assignment2
```

Dropout

Dropout [1] is a technique for regularizing neural networks by randomly setting some output activations 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 [2]:
```
# Setup cell.
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_a
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"

%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))))
```

```
========== You can safely ignore the message below if you are NOT working on Convolut
ionalNetworks.ipynb ==========
        You will need to compile a Cython extension for a portion of this assignment.
        The instructions to do this will be given in a section of the notebook below.
```

In [3]:
```
# Load the (preprocessed) CIFAR-10 data.
data = get_CIFAR10_data()
for k, v in list(data.items()):
    print(f"{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 [5]:
```
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.

In [6]:
```
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:  5.44560814873387e-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:

Test 과정에서 p를 곱해줘야 한다. 우리는 Train과 Test에서 expectation이 동일한 것을 원하지만 dropout을 주게 되면 1/p만큼의 차이가 나게 되므로 output으로 p를 곱해야한다.

Fully Connected Networks with Dropout

In the file `cs231n/classifiers/fc_net.py`, modify your implementation to use dropout. Specifically, if the constructor of the network receives a value that is not 1 for the `dropout_keep_ratio` parameter, then the net should add a dropout layer immediately after every ReLU nonlinearity. After doing so, run the following to numerically gradient-check your implementation.

In [14]:
```
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_keep_ratio in [1, 0.75, 0.5]:
    print('Running check with dropout = ', dropout_keep_ratio)
    model = FullyConnectedNet(
        [H1, H2],
        input_dim=D,
        num_classes=C,
        weight_scale=5e-2,
        dtype=np.float64,
        dropout_keep_ratio=dropout_keep_ratio,
        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_keep_ratio=1 you have W2 error be on the orde
    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
        print('%s relative error: %.2e' % (name, rel_error(grad_num, grads[name])))
    print()
```

```
Running check with dropout =  1
Initial loss:  2.300479089768492
W1 relative error: 1.03e-07
W2 relative error: 2.21e-05
W3 relative error: 4.56e-07
b1 relative error: 4.66e-09
b2 relative error: 2.09e-09
b3 relative error: 1.69e-10

Running check with dropout =  0.75
Initial loss:  2.302371489704412
W1 relative error: 1.85e-07
W2 relative error: 2.15e-06
W3 relative error: 4.56e-08
b1 relative error: 1.16e-08
b2 relative error: 1.82e-09
b3 relative error: 1.48e-10

Running check with dropout =  0.5
Initial loss:  2.30427592207859
W1 relative error: 3.11e-07
W2 relative error: 2.48e-08
W3 relative error: 6.43e-08
b1 relative error: 5.37e-09
b2 relative error: 1.91e-09
b3 relative error: 1.85e-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.

In [15]:

```
# 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_keep_ratio in dropout_choices:
    model = FullyConnectedNet(
        [500],
        dropout_keep_ratio=dropout_keep_ratio
    )
    print(dropout_keep_ratio)

    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_keep_ratio] = 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: 0.414000; val_acc: 0.261000
(Epoch 2 / 25) train acc: 0.482000; val_acc: 0.278000
(Epoch 3 / 25) train acc: 0.538000; val_acc: 0.274000
(Epoch 4 / 25) train acc: 0.604000; val_acc: 0.266000
(Epoch 5 / 25) train acc: 0.740000; val_acc: 0.301000
(Epoch 6 / 25) train acc: 0.738000; val_acc: 0.287000
(Epoch 7 / 25) train acc: 0.832000; val_acc: 0.261000
(Epoch 8 / 25) train acc: 0.856000; val_acc: 0.270000
(Epoch 9 / 25) train acc: 0.896000; val_acc: 0.282000
(Epoch 10 / 25) train acc: 0.922000; val_acc: 0.267000
(Epoch 11 / 25) train acc: 0.926000; val_acc: 0.271000
(Epoch 12 / 25) train acc: 0.944000; val_acc: 0.299000
(Epoch 13 / 25) train acc: 0.964000; val_acc: 0.301000
(Epoch 14 / 25) train acc: 0.982000; val_acc: 0.299000
(Epoch 15 / 25) train acc: 0.978000; val_acc: 0.284000
(Epoch 16 / 25) train acc: 0.974000; val_acc: 0.298000
(Epoch 17 / 25) train acc: 0.974000; val_acc: 0.302000
(Epoch 18 / 25) train acc: 0.958000; val_acc: 0.288000
(Epoch 19 / 25) train acc: 0.952000; val_acc: 0.297000
(Epoch 20 / 25) train acc: 0.982000; val_acc: 0.312000
(Iteration 101 / 125) loss: 0.142865
(Epoch 21 / 25) train acc: 0.992000; val_acc: 0.310000
(Epoch 22 / 25) train acc: 0.992000; val_acc: 0.314000
(Epoch 23 / 25) train acc: 0.998000; val_acc: 0.322000
(Epoch 24 / 25) train acc: 0.980000; val_acc: 0.305000
(Epoch 25 / 25) train acc: 0.994000; val_acc: 0.304000

0.25
(Iteration 1 / 125) loss: 17.318478
(Epoch 0 / 25) train acc: 0.230000; val_acc: 0.176000
(Epoch 1 / 25) train acc: 0.376000; val_acc: 0.244000
(Epoch 2 / 25) train acc: 0.410000; val_acc: 0.251000
(Epoch 3 / 25) train acc: 0.480000; val_acc: 0.275000
(Epoch 4 / 25) train acc: 0.518000; val_acc: 0.297000
(Epoch 5 / 25) train acc: 0.574000; val_acc: 0.295000
(Epoch 6 / 25) train acc: 0.660000; val_acc: 0.294000
(Epoch 7 / 25) train acc: 0.636000; val_acc: 0.295000
(Epoch 8 / 25) train acc: 0.722000; val_acc: 0.318000
(Epoch 9 / 25) train acc: 0.732000; val_acc: 0.298000
(Epoch 10 / 25) train acc: 0.754000; val_acc: 0.314000
(Epoch 11 / 25) train acc: 0.788000; val_acc: 0.322000
(Epoch 12 / 25) train acc: 0.796000; val_acc: 0.287000
(Epoch 13 / 25) train acc: 0.812000; val_acc: 0.310000
(Epoch 14 / 25) train acc: 0.804000; val_acc: 0.319000
(Epoch 15 / 25) train acc: 0.866000; val_acc: 0.318000
(Epoch 16 / 25) train acc: 0.808000; val_acc: 0.296000
(Epoch 17 / 25) train acc: 0.850000; val_acc: 0.301000
(Epoch 18 / 25) train acc: 0.824000; val_acc: 0.326000
(Epoch 19 / 25) train acc: 0.898000; val_acc: 0.340000
(Epoch 20 / 25) train acc: 0.892000; val_acc: 0.304000
(Iteration 101 / 125) loss: 3.965150
(Epoch 21 / 25) train acc: 0.904000; val_acc: 0.317000
(Epoch 22 / 25) train acc: 0.882000; val_acc: 0.314000
```

```
(Epoch 23 / 25) train acc: 0.900000; val_acc: 0.312000
(Epoch 24 / 25) train acc: 0.920000; val_acc: 0.311000
(Epoch 25 / 25) train acc: 0.930000; val_acc: 0.321000
```

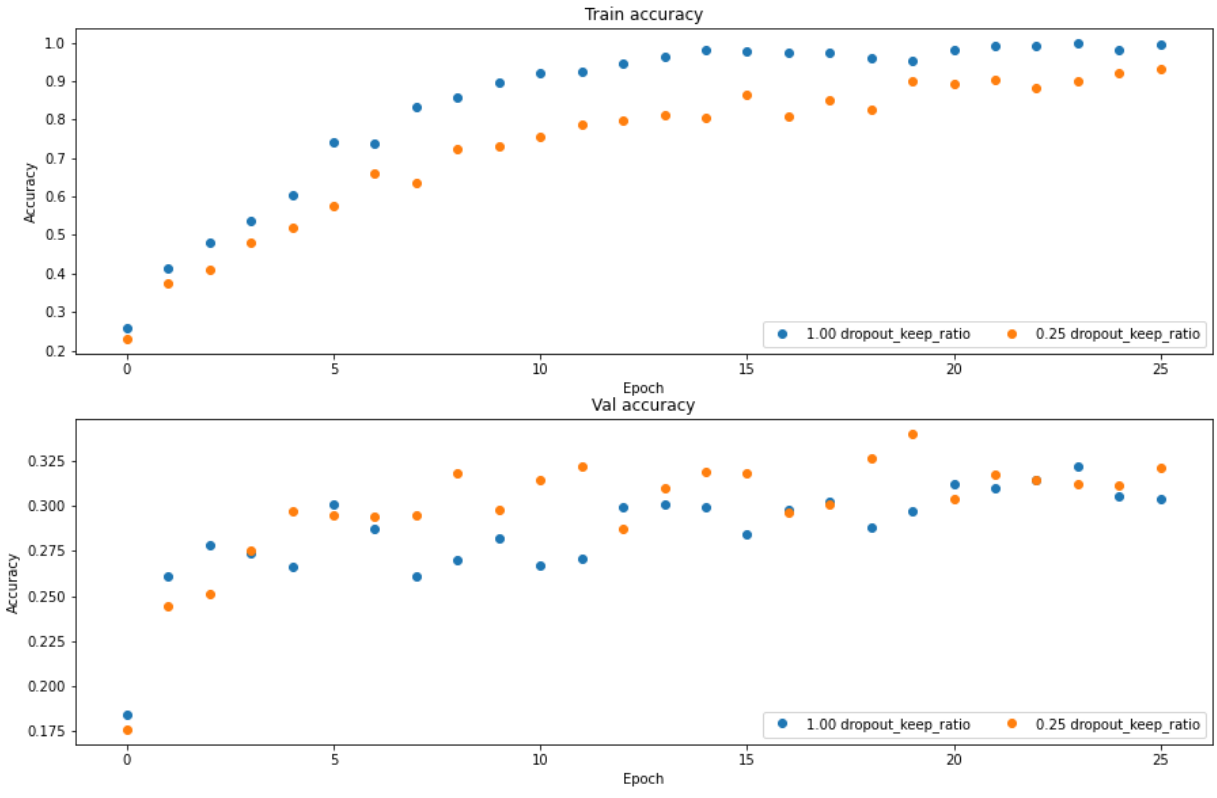
In [16]:

```
# Plot train and validation accuracies of the two models.
train_accs = []
val_accs = []
for dropout_keep_ratio in dropout_choices:
    solver = solvers[dropout_keep_ratio]
    train_accs.append(solver.train_acc_history[-1])
    val_accs.append(solver.val_acc_history[-1])

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

plt.subplot(3, 1, 2)
for dropout_keep_ratio in dropout_choices:
    plt.plot(
        solvers[dropout_keep_ratio].val_acc_history, 'o', label='%.2f dropout_keep_ra
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:

Dropout을 사용하게 되면 train의 accuracy는 사용하지 않았을 때보다 낮아진다. 하지만, validation의 accuracy는 더 높아지는 것을 볼 수 있다. 즉, dropout을 regularizer로써 사용할 수 있다는 결론을 내릴 수 있다.

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). If we are concerned about overfitting, how should we modify p (if at all) when we decide to decrease the size of the hidden layers (that is, the number of nodes in each layer)?

Answer:

만약 우리가 hidden layers의 사이즈를 감소시키기로 결정했다면(즉, 각 레이어의 노드 수를 감소), dropout의 keep probability 'p'를 증가시켜야 한다.