Dropout

May 4, 2018

1 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)

```
In [1]: # 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_arra
        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))))
```

```
run the following from the cs231n directory and try again:
python setup.py build_ext --inplace
You may also need to restart your iPython kernel
```

```
In [2]: # 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,)
```

2 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 [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:  7.496693099233182
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.250216
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:  6.00904081497304
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.399204
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:  3.008739539965901
Mean of test-time output:  10.000207878477502
Fraction of train-time output set to zero:  0.69926
Fraction of test-time output set to zero:  0.0
```

3 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 [10]: 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:  1.892896957390533e-11
```

3.1 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?

3.2 Answer:

4 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.

```
In [11]: 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,))
```

```python
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:  3.4046031324121993
W1 relative error: 3.72e-07
W2 relative error: 8.64e-06
W3 relative error: 8.01e-07
b1 relative error: 2.10e-08
b2 relative error: 1.23e-09
b3 relative error: 3.94e-10

Running check with dropout =  0.75
Initial loss:  3.401186562367183
W1 relative error: 1.73e-08
W2 relative error: 3.07e-07
W3 relative error: 2.29e-05
b1 relative error: 5.22e-10
b2 relative error: 1.00e+00
b3 relative error: 2.58e-10

Running check with dropout =  0.5
Initial loss:  3.400803690996306
W1 relative error: 1.11e-06
W2 relative error: 8.84e-08
W3 relative error: 1.66e-06
b1 relative error: 2.08e-08
b2 relative error: 1.41e-09
b3 relative error: 3.57e-10
```

5 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.

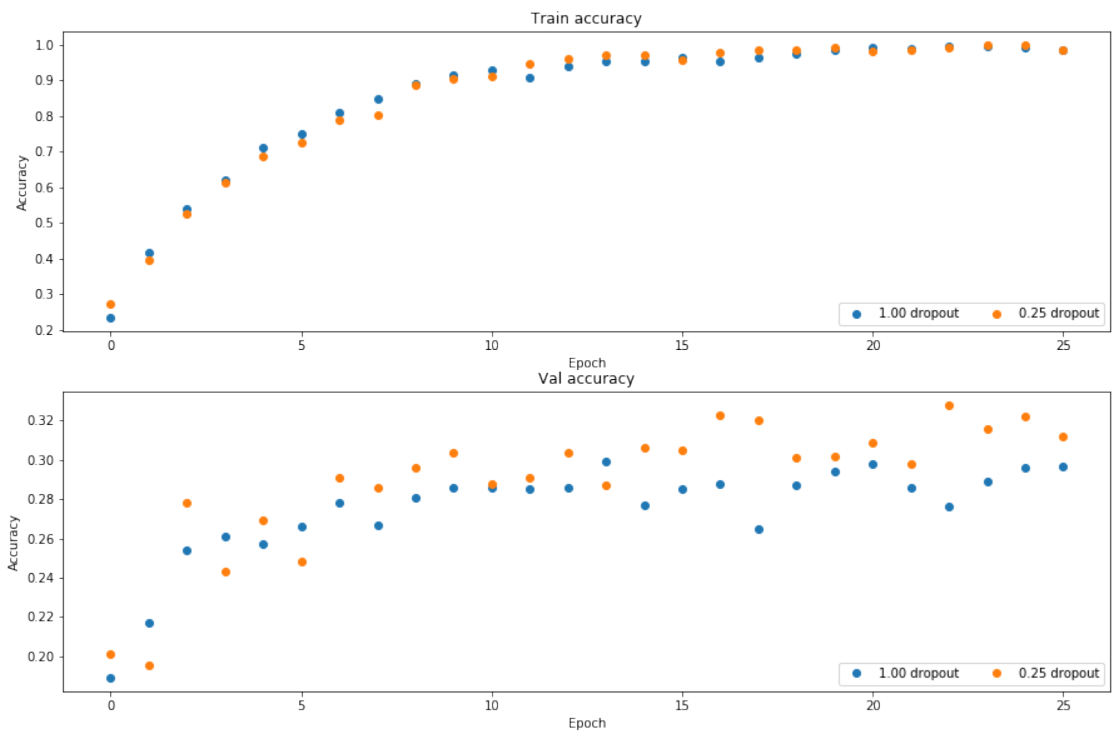
In [12]:
```
# Train two identical nets, one with dropout and one without
np.random.seed(231)
num_train = 500
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

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: 18.076194
(Epoch 0 / 25) train acc: 0.234000; val_acc: 0.189000
(Epoch 1 / 25) train acc: 0.416000; val_acc: 0.217000
(Epoch 2 / 25) train acc: 0.540000; val_acc: 0.254000
(Epoch 3 / 25) train acc: 0.620000; val_acc: 0.261000
(Epoch 4 / 25) train acc: 0.712000; val_acc: 0.257000
(Epoch 5 / 25) train acc: 0.750000; val_acc: 0.266000
(Epoch 6 / 25) train acc: 0.810000; val_acc: 0.278000
(Epoch 7 / 25) train acc: 0.848000; val_acc: 0.267000
(Epoch 8 / 25) train acc: 0.888000; val_acc: 0.281000
(Epoch 9 / 25) train acc: 0.914000; val_acc: 0.286000
(Epoch 10 / 25) train acc: 0.928000; val_acc: 0.286000
(Epoch 11 / 25) train acc: 0.906000; val_acc: 0.285000
(Epoch 12 / 25) train acc: 0.938000; val_acc: 0.286000
(Epoch 13 / 25) train acc: 0.954000; val_acc: 0.299000
```

```
(Epoch 14 / 25) train acc: 0.952000; val_acc: 0.277000
(Epoch 15 / 25) train acc: 0.964000; val_acc: 0.285000
(Epoch 16 / 25) train acc: 0.952000; val_acc: 0.288000
(Epoch 17 / 25) train acc: 0.964000; val_acc: 0.265000
(Epoch 18 / 25) train acc: 0.974000; val_acc: 0.287000
(Epoch 19 / 25) train acc: 0.986000; val_acc: 0.294000
(Epoch 20 / 25) train acc: 0.990000; val_acc: 0.298000
(Iteration 101 / 125) loss: 0.018257
(Epoch 21 / 25) train acc: 0.988000; val_acc: 0.286000
(Epoch 22 / 25) train acc: 0.994000; val_acc: 0.276000
(Epoch 23 / 25) train acc: 0.996000; val_acc: 0.289000
(Epoch 24 / 25) train acc: 0.990000; val_acc: 0.296000
(Epoch 25 / 25) train acc: 0.986000; val_acc: 0.297000
0.25
(Iteration 1 / 125) loss: 14.008363
(Epoch 0 / 25) train acc: 0.272000; val_acc: 0.201000
(Epoch 1 / 25) train acc: 0.394000; val_acc: 0.195000
(Epoch 2 / 25) train acc: 0.526000; val_acc: 0.278000
(Epoch 3 / 25) train acc: 0.612000; val_acc: 0.243000
(Epoch 4 / 25) train acc: 0.686000; val_acc: 0.269000
(Epoch 5 / 25) train acc: 0.726000; val_acc: 0.248000
(Epoch 6 / 25) train acc: 0.788000; val_acc: 0.291000
(Epoch 7 / 25) train acc: 0.802000; val_acc: 0.286000
(Epoch 8 / 25) train acc: 0.886000; val_acc: 0.296000
(Epoch 9 / 25) train acc: 0.902000; val_acc: 0.304000
(Epoch 10 / 25) train acc: 0.910000; val_acc: 0.288000
(Epoch 11 / 25) train acc: 0.946000; val_acc: 0.291000
(Epoch 12 / 25) train acc: 0.960000; val_acc: 0.304000
(Epoch 13 / 25) train acc: 0.970000; val_acc: 0.287000
(Epoch 14 / 25) train acc: 0.972000; val_acc: 0.306000
(Epoch 15 / 25) train acc: 0.958000; val_acc: 0.305000
(Epoch 16 / 25) train acc: 0.976000; val_acc: 0.323000
(Epoch 17 / 25) train acc: 0.986000; val_acc: 0.320000
(Epoch 18 / 25) train acc: 0.986000; val_acc: 0.301000
(Epoch 19 / 25) train acc: 0.990000; val_acc: 0.302000
(Epoch 20 / 25) train acc: 0.982000; val_acc: 0.309000
(Iteration 101 / 125) loss: 0.245039
(Epoch 21 / 25) train acc: 0.986000; val_acc: 0.298000
(Epoch 22 / 25) train acc: 0.992000; val_acc: 0.328000
(Epoch 23 / 25) train acc: 0.998000; val_acc: 0.316000
(Epoch 24 / 25) train acc: 0.998000; val_acc: 0.322000
(Epoch 25 / 25) train acc: 0.984000; val_acc: 0.312000
```

In [13]:
```
# 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()
```



5.1 Inline Question 2:

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

5.2 Answer:

In our results, the training accuracies are the same but the validation accuracies are different. It seems that use of dropout does not increase training accuracy but it does increase validation accuracy.

5.3 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)?

5.4 Answer:

p should be decreased in proportion to the decrease of the size of the hidden layers.