Dropout

October 27, 2019

1 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

[1]:
```
# As usual, a bit of setup
from __future__ import print_function
import time
import numpy as np
import matplotlib.pyplot as plt
from ie590.classifiers.fc_net import *
from ie590.data_utils import get_CIFAR10_data
from ie590.gradient_check import eval_numerical_gradient,␣
 ↪eval_numerical_gradient_array
from ie590.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 ie590 directory and try again:
python setup.py build_ext --inplace
You may also need to restart your iPython kernel
```

[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 `ie590/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.

[3]:
```
np.random.seed(220)
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:  10.000073983317677
Mean of train-time output:  23.942958248076366
Mean of test-time output:  10.000073983317677
Fraction of train-time output set to zero:  0.401236
Fraction of test-time output set to zero:  0.0

Running tests with p =  0.4
Mean of input:  10.000073983317677
Mean of train-time output:  16.39074194873602
Mean of test-time output:  10.000073983317677
Fraction of train-time output set to zero:  0.344464
Fraction of test-time output set to zero:  0.0
```

```
Running tests with p =  0.7
Mean of input:  10.000073983317677
Mean of train-time output:  10.830890239522626
Mean of test-time output:  10.000073983317677
Fraction of train-time output set to zero:  0.24172
Fraction of test-time output set to zero:  0.0
```

3 Dropout backward pass

In the file `ie590/layers.py`, implement the backward pass for dropout. After doing so, run the following cell to numerically gradient-check your implementation.

[4]:
```
np.random.seed(220)
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:  5.445606263132224e-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:

Missing /p in inversed dropout is equivalent to setting dropout probability p at training stage while dropout probability 1 at test stage. A direct result would be the prediction value at test is a wrong value y instead of correct value p*y. This means we set randomness at training but ignore the randomness when use the trained network to predict. The reason is that the /p is moved from test stage to simplify original vanilla dropout.

4 Fully-connected nets with Dropout

In the file `ie590/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` 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.

[5]:
```
np.random.seed(220)
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-3 or less; Note that it's fine

  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:  2.3109271789130945
W1 relative error: 1.84e-05
W2 relative error: 1.59e-03
W3 relative error: 1.72e-06
b1 relative error: 4.36e-07
b2 relative error: 1.66e-07
b3 relative error: 8.72e-11

Running check with dropout =  0.75
Initial loss:  2.297789151503058
W1 relative error: 2.80e-03
W2 relative error: 4.47e-04
W3 relative error: 2.14e-06
b1 relative error: 2.60e-03
b2 relative error: 5.67e-08
b3 relative error: 1.50e-10

Running check with dropout =  0.5
Initial loss:  2.306176853993195
W1 relative error: 1.92e-06
W2 relative error: 9.62e-04
```

```
W3 relative error: 2.35e-06
b1 relative error: 2.10e-07
b2 relative error: 1.09e-07
b3 relative error: 9.55e-11
```

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 0.3. We will then visualize the training and validation accuracies of the two networks over time.

[6]:
```
# Train two identical nets, one with dropout and one without
np.random.seed(220)
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, 0.3]
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: 8.304420
(Epoch 0 / 25) train acc: 0.192000; val_acc: 0.195000
(Epoch 1 / 25) train acc: 0.388000; val_acc: 0.228000
(Epoch 2 / 25) train acc: 0.480000; val_acc: 0.249000
(Epoch 3 / 25) train acc: 0.584000; val_acc: 0.273000
(Epoch 4 / 25) train acc: 0.716000; val_acc: 0.285000
(Epoch 5 / 25) train acc: 0.796000; val_acc: 0.297000
(Epoch 6 / 25) train acc: 0.834000; val_acc: 0.265000
```

```
(Epoch 7 / 25) train acc: 0.818000; val_acc: 0.262000
(Epoch 8 / 25) train acc: 0.868000; val_acc: 0.259000
(Epoch 9 / 25) train acc: 0.914000; val_acc: 0.268000
(Epoch 10 / 25) train acc: 0.950000; val_acc: 0.251000
(Epoch 11 / 25) train acc: 0.958000; val_acc: 0.259000
(Epoch 12 / 25) train acc: 0.940000; val_acc: 0.281000
(Epoch 13 / 25) train acc: 0.968000; val_acc: 0.283000
(Epoch 14 / 25) train acc: 0.928000; val_acc: 0.280000
(Epoch 15 / 25) train acc: 0.944000; val_acc: 0.262000
(Epoch 16 / 25) train acc: 0.952000; val_acc: 0.281000
(Epoch 17 / 25) train acc: 0.938000; val_acc: 0.276000
(Epoch 18 / 25) train acc: 0.950000; val_acc: 0.289000
(Epoch 19 / 25) train acc: 0.958000; val_acc: 0.276000
(Epoch 20 / 25) train acc: 0.964000; val_acc: 0.269000
(Iteration 101 / 125) loss: 0.213629
(Epoch 21 / 25) train acc: 0.990000; val_acc: 0.265000
(Epoch 22 / 25) train acc: 0.978000; val_acc: 0.278000
(Epoch 23 / 25) train acc: 0.984000; val_acc: 0.287000
(Epoch 24 / 25) train acc: 0.986000; val_acc: 0.290000
(Epoch 25 / 25) train acc: 0.980000; val_acc: 0.292000

0.3
(Iteration 1 / 125) loss: 22.585506
(Epoch 0 / 25) train acc: 0.214000; val_acc: 0.157000
(Epoch 1 / 25) train acc: 0.394000; val_acc: 0.240000
(Epoch 2 / 25) train acc: 0.456000; val_acc: 0.297000
(Epoch 3 / 25) train acc: 0.542000; val_acc: 0.243000
(Epoch 4 / 25) train acc: 0.626000; val_acc: 0.267000
(Epoch 5 / 25) train acc: 0.688000; val_acc: 0.268000
(Epoch 6 / 25) train acc: 0.722000; val_acc: 0.283000
(Epoch 7 / 25) train acc: 0.778000; val_acc: 0.288000
(Epoch 8 / 25) train acc: 0.806000; val_acc: 0.286000
(Epoch 9 / 25) train acc: 0.772000; val_acc: 0.269000
(Epoch 10 / 25) train acc: 0.882000; val_acc: 0.288000
(Epoch 11 / 25) train acc: 0.888000; val_acc: 0.314000
(Epoch 12 / 25) train acc: 0.914000; val_acc: 0.332000
(Epoch 13 / 25) train acc: 0.952000; val_acc: 0.331000
(Epoch 14 / 25) train acc: 0.948000; val_acc: 0.287000
(Epoch 15 / 25) train acc: 0.948000; val_acc: 0.310000
(Epoch 16 / 25) train acc: 0.972000; val_acc: 0.316000
(Epoch 17 / 25) train acc: 0.954000; val_acc: 0.329000
(Epoch 18 / 25) train acc: 0.978000; val_acc: 0.322000
(Epoch 19 / 25) train acc: 0.982000; val_acc: 0.337000
(Epoch 20 / 25) train acc: 0.958000; val_acc: 0.313000
(Iteration 101 / 125) loss: 1.010401
(Epoch 21 / 25) train acc: 0.970000; val_acc: 0.334000
(Epoch 22 / 25) train acc: 0.974000; val_acc: 0.289000
(Epoch 23 / 25) train acc: 0.980000; val_acc: 0.301000
```

```
(Epoch 24 / 25) train acc: 0.968000; val_acc: 0.304000
(Epoch 25 / 25) train acc: 0.946000; val_acc: 0.309000
```

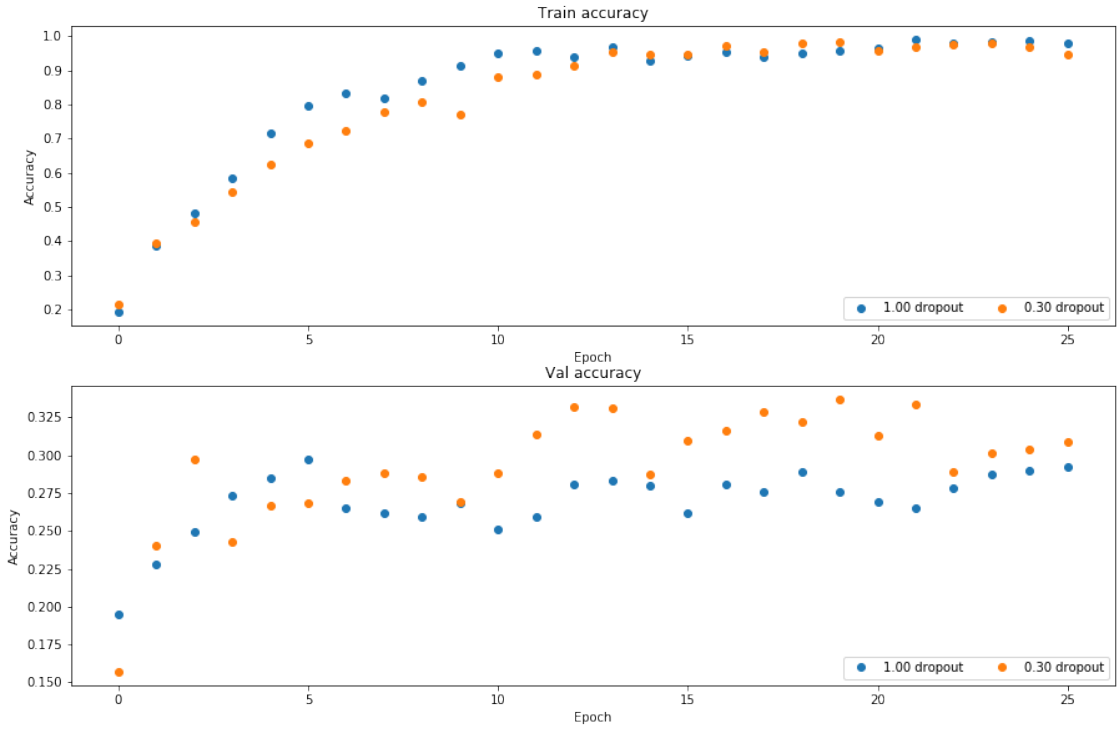
[7]:
```
# Plot train and validation accuracies of the two models

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:

From training accuracy, dropout can preserve the training accuracy though it may converge slower. From validation accuracy, dropout can increase the validation accuracy.

The results suggest that dropout as a regularizer can help generalizing the network to unknown dataset and can reduce the overfitting.

5.3 Inline Question 3:

Should we use dropout in the input layer? Why / Why not?

5.4 Answer:

[No, we should not use dropout in input layer. Becaue it will directly reduce the true data we can use, which is similar to using a smaller dataset thus is bad. The dropout layer is to reduce redundant information that we *learned* in the betwork, so it should apply to learned patterns inside the network, not at the beginning when nothing has been learned yet.]

5.5 Inline Question 4:

Do the weights of the network change when trained using dropout as compared to not using dropout? If they change, do they increase or decrease as compared to not using dropout?

5.6 Answer:

Yes, the weightes of the network will change. The weights will increase as compared to not using dropout. Because only part of the weights will be counted using dropout, to catch the output value, the weights will tend to increse.

5.7 Inline Question 5:

Should we use dropout while testing? Why/Why not?

5.8 Answer:

No, we should not use dropout at test. Because the dropout is like training a large ensemble of models that share parameters. All the models in the ensemble have good generality that can predict better at test stage. To do that, we should averge out the predictions of all models in the ensemble. Using dropout while testing is like choosign one model from the ensemble, thus can degrade the accuracy a lot.

[ ]: