HW

CS156, Kai Chang

Question 1

Answer Choice: **B**, In general, deterministic noise will increase.
Reasoning: We know that deterministic noise depends on H, and because H' is a subset of H, and remember, H is the hypothesis, and can be considered as the guess fit / model, a subset of it (assuming H is the best possible model learned) will most likely be worse of a fit / model, as it contains less information than the original.

Question 2

Answer Choice: **A**, 0.03, 0.08
Reasoning: see code.

```
In [259]: def transform(x_):
              '''
              Returns transformed data used for regression.

              Parameters
              ----------
              x_ : two-dimensional input (x1, x2), numpy array

              Returns
              -------
              phi_ : (1, x1, x2, x1^2, x2^2, x1x2, |x1 - x2|, |x1 + x2|), numpy array
              '''
              n = x_.shape[1]
              phi_ = np.ones((n,1))
              phi_ = np.insert(phi_, 1, x_[0:2], 1)
              phi_ = np.insert(phi_, phi_.shape[1], x_[0]**2, 1)
              phi_ = np.insert(phi_, phi_.shape[1],  x_[1]**2, 1)
              phi_ = np.insert(phi_, phi_.shape[1], x_[0]*x_[1], 1)
              phi_ = np.insert(phi_, phi_.shape[1], np.abs(x_[0] - x_[1]), 1)
              phi_ = np.insert(phi_, phi_.shape[1], np.abs(x_[0] + x_[1]), 1)


          #     phi_ = np.append(phi_, x_[0:2], axis=0)
          #     phi_ = np.append(phi_, x_[0]**2)
          #     phi_ = np.append(phi_, x_[1]**2)
          #     phi_ = np.append(phi_, x_[0]*x_[1])
          #     phi_ = np.append(phi_, np.abs(x_[0] - x_[1]))
          #     phi_ = np.append(phi_, np.abs(x_[0] + x_[1])).reshape(n, 8)

              return phi_
```

In [288]:
```
def calc_suedo(x_train):
    return np.dot( np.linalg.inv( np.dot( x_train.T, x_train) ), x_train.T)

def lin_reg(x_pseudo, y):
    ''' Doesnt work when we have structure of [[1,1,1,1,1], [x,x,x,x,x], [xy,xy,x
y,xy,xy], etc..],
    even with switch of transpose. Not sure why.'''
    print y
    return np.dot(x_pseudo, y)
```

In [289]:
```
def calc_error(w, x_, y_):
    '''
    Calculates the E_out (error of out sample)

    Parameters
    ----------
    w  : weights learned from linear regression, numpy array
    x_ : dataset, multi-dimensional numpy array
    y_ : output (solution), numpy array

    Returns
    -------
    E_out
    '''
#     print np.sign(np.dot(w.T, x_))
#     print np.max(0, np.sign(-np.dot(np.dot(w.T, x_),y_)))/len(x_[0])
#     return np.sum(np.sign(np.dot(w.T, x_)) - y_)/ len(x_[0])

    N = 0

    for i in xrange(x_.shape[0]):
        N += max(0, np.sign(-np.dot(w.T, x_[i])*y_[i]))

    return N/float(x_.shape[0])
```

I HAVE HAD SO MUCH TROUBLE WITH THESE 2 .DTA FILES, WOULD YOU KINDLY USE A CSV OR A NUMPY FILE OR EVEN A .DAT, OR SOMETHING IN THE FUTURE? .DTA WOULD BE FINE IF THE CONVERSION WAS UNIVERSAL, BUT IT WAS NOT AS I COULDN'T READ IT EVEN USING PANDAS FUNCTION TO HANDLE DTA FILES AND RSTUDIOS. :((((((((((((

In [290]:
```
import numpy as np
import pandas as pd

df = pd.read_csv('in.csv', sep='   ', header=None)
y_train = np.array(df[2])
x_train = np.append(np.array(df[0]), np.array(df[1])).reshape(2, len(y_train))


df = pd.read_csv('out.csv', sep='   ', header=None)
y_test = np.array(df[2])
x_test = np.append(np.array(df[0]), np.array(df[1])).reshape(2, len(y_test))
```

```
In [291]: phi_train = transform(x_train)
          phi_suedo = calc_suedo(phi_train)
          phi_w = lin_reg(phi_suedo, y_train)

          [ 1.  1.  1.  1. -1.  1. -1.  1.  1. -1. -1.  1.  1. -1. -1.  1. -1.  1.
           -1.  1. -1. -1.  1.  1. -1. -1.  1.  1.  1. -1.  1. -1.  1. -1.  1.]
```

```
In [292]: phi_test = transform(x_test)
```

I had a mistake in my code -- was doing the error as a physical error (mean squared), but in reality, it is supposed to be the sign function, as we are doing classification still.

```
In [294]: err_train = calc_error(phi_w, phi_train, y_train)
          err_test = calc_error(phi_w, phi_test, y_test)

          print 'Training classification error: ', err_train
          print 'Test classification error: ', err_test

          Training classification error:  0.0285714285714
          Test classification error:  0.084
```

Question 3

Answer Choice: **D**, 0.03, 0.08
Reasoning: see code. Note, this decay is actually the regularization linear regression weight developed in class. This is from our augmented error, the squared error with the addition of our weight decay:
$E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}[h(x_n) - y_n]^2 + \frac{\lambda}{N}\sum_{q=0}^{Q}\omega_i^2$, and because we want to minimize this error function, we arrive at $\omega = (Z^T Z + \lambda I)^{-1} Z^T = \omega_{reg}$. See page 11, Yaser's lecture slide 12 for more comprehensive derivation.

```
In [296]: def calc_suedo_reg(x_train, lmda):
              ''' With weight decay.'''
              return np.dot( np.linalg.inv( np.dot( x_train.T, x_train) + lmda*np.identity(
          x_train.shape[1])), x_train.T)

          def lin_reg_reg(x_pseudo, y):
              ''' With weight decay.'''
              return np.dot(x_pseudo, y)
```

```
In [305]: phi_suedo_reg = calc_suedo_reg(phi_train, 10**-3)
          phi_w_reg = lin_reg_reg(phi_suedo_reg, y_train)
```

```
In [306]: err_train_reg = calc_error(phi_w_reg, phi_train, y_train)
          err_test_reg = calc_error(phi_w_reg, phi_test, y_test)

          print 'Training classification error: ', err_train_reg
          print 'Test classification error: ', err_test_reg

          Training classification error:  0.0285714285714
          Test classification error:  0.08
```

Note the near identical error sampling :0.

Question 4

Answer Choice: **E**, 0.4, 0.4
Reasoning: see code.

In [307]:
```
phi_suedo_reg = calc_suedo_reg(phi_train, 10**3)
phi_w_reg = lin_reg_reg(phi_suedo_reg, y_train)

err_train_reg = calc_error(phi_w_reg, phi_train, y_train)
err_test_reg = calc_error(phi_w_reg, phi_test, y_test)

print 'Training classification error: ', err_train_reg
print 'Test classification error: ', err_test_reg
```

```
Training classification error:  0.371428571429
Test classification error:  0.436
```

Question 5

Answer Choice: **D**, -1
Reasoning: see code. I plugged in all 5 k's.

In [308]:
```
min_err_test_reg = 1
k_value = 0
for i in xrange(-2,3,1):
    phi_suedo_reg = calc_suedo_reg(phi_train, 10**i)
    phi_w_reg = lin_reg_reg(phi_suedo_reg, y_train)

#     err_train_reg = calc_error(phi_w_reg, phi_train, y_train)
    err_test_reg = calc_error(phi_w_reg, phi_test, y_test)
    if err_test_reg < min_err_test_reg:
        min_err_test_reg = err_test_reg
        k_value = i


print 'Local minimum k: ', k_value
print 'Test classification error: ', min_err_test_reg
```

```
Local minimum k:  -1
Test classification error:  0.056
```

Question 6

Answer Choice: **B**, 0.06
Reasoning: see code. Turns out our local minima was the global minima.

In [309]:
```
min_err_test_reg = 1
k_value = 0

# limiting to integer values
for i in xrange(-5,5,1):
    phi_suedo_reg = calc_suedo_reg(phi_train, 10**i)
    phi_w_reg = lin_reg_reg(phi_suedo_reg, y_train)

    err_test_reg = calc_error(phi_w_reg, phi_test, y_test)
    if err_test_reg < min_err_test_reg:
        min_err_test_reg = err_test_reg
        k_value = i


print 'Global minimum k: ', k_value
print 'Test classification error: ', min_err_test_reg
```

```
Global minimum k:  -1
Test classification error:  0.056
```

Question 7

Answer Choice: **C**, $H(10, 0, 3) \cap H(10, 0, 4) = H_2$
Reasoning: See paper for explicit writing out and calculation of H set. We note that for $H(10, 0, 3)$, we have H_0, H_1, H_2, for $H(10, 0, 4)$, we have H_0, H_1, H_2, H_3, for $H(10, 1, 3)$, we have up to H_{10}, and for $H(10, 1, 4)$, it is up to H_{10}.

So in case a, the union of these two hypothesis should have the H_Q (the max H) leaves us H_3, as H_3 contains the set of H_0, H_1, H_2, H_3. This is incorrect. In case b, the union of this is H_{10} because of the condition $w_q = C$, for $q \geq Q_0$. So, this is incorrect. Case d is incorrect because with the $C = 1$ case, the intersection is the subset H_4 to H_{10} (because we do not know whether $w_q = 0$ for cases $q < 3, 4$). However, seeing that there is only 1 limiting restraint, we can generalized w_q, such that both contain the entire set H_0 to H_{10}, just the restraint that w_q is limited to 1 for the condition of $q \geq Q_0$. This then leaves us H_{10}, which again covers all H_{q_i} prior. Thus d is incorrect. Choice c has the intersection with $C = 0$, meaning we intersect the set H_0, H_1, H_2, and H_0, H_1, H_2, H_3, and so we can clearly see that the intersection is of H_0, H_1, H_2, which is just H_2 (again covers all subsets). Thus, statement c is correct and statement e is incorrect.

Sorry if my notation is kind of off or weird, I know H_q isn't quite valid, but the point to get across is the idea of H' and H. Remember, Q is the maximum order.

Question 8

Answer Choice: **D**, 45
Reasoning: For more detailed calculations, see attached paper. If we consider just looking at the nodes, then we have essentially 5 inputs to 3 inputs to 1 ouput (2 depth network, 3 layers). However, because we have to consider a node of value 1 in our **x**, then we actually have 6 inputs to 4 inputs to 1 output. There is 4 inputs on the middle layer instead of 3 because we are again feeding it to another network to the final layer of 1 output. Thus, using the products of form $w_{ij}^{(l)} x_i^{(l-1)}$, we get 6 mapping to 3 and 4 mapping to 1 yielding 18 + 4 = 22. We note that this matches the formula regarding network operations on page 14 of Yaser's slide 10. In a more mathematical term, we can consider the summation over the input as each item is a operation count, and we then span that across the output for each layer, so 6 in sum $\times$ 3 span (3 output layer) + 4 in sum $\times$ 1 span (final output layer).

Now, we also again have to consider the backward propagation operation $w_{ij}^{(l)} \delta_j^{(l)}$, which when you look at page 19 of Yaser's slide 10, you will see that this can be re-written in propotion to the input backpropagation strictly (ie. $\delta_i^{(l-1)}$). So, given $l = 1, 2$, we have $\delta_i^{(0,1)}$, and if we look at the node mapping (backwards for backpropgagtion), we see that the input of layer 0 has 0 operations, and the input layer 1 has 3 backpropagations (3 nodes to map back to, where the arrows join together). So we get a value of 3. Finally, considering $x_i^{(l-1)} \delta_j^{(l)}$, which represents the weight update, we get an update on all the weights (strings) so following the weight operation on page 14, we get 22, identical to the one above. Note that each weight needs an operation.

Thus, 22 + 3 + 22 = 47. Choice d is closest to this value.

The ghost problem really helped on EdX forum.

Question 9

Answer Choice: **A**, 46
Reasoning: To minimize the weights, we need to consider the minimum # of connections, so having more nodes per layer results in more combination of connections. Thus, the structure of our neural network that reduces the # of connections or weights is one of the identity, or having 2 nodes in each hidden layer. So, starting out with 10 input units ($x_0^{(0)}$ counted as a unit, so technically 9 units with 1 bias unit), we get then 10 weights to attach to 2 output node (remember we have a value 1 node, the bias, on our input as well as our output, but the output bias doesn't get any connections). So, we have 18 layers for the 36 hidden units (as $x_0^{(l)} = 1$ counts as a unit). For each of these layers, we have 2 connections and thus 2 weights, giving a total of $36 - 2 = 34$ weights for all 18 layers (it is -2 because we are only looking at connections in between these layers). Finally, for our final network consistening of the hidden layer to output layer, we have 2 connections.

So, 10 + 34 + 2 = 46.

Look at diagram attached for more clear visualization.

Question 10

Answer Choice: **E**, 510
Reasoning: To maximize the weights, we want the most # of connections, so having many nodes per layer results in more combinations of connections. However, this is not as simple as the previous problem because of the nodes per layer tradeoff.

If we consider a single hidden layer of all 36 nodes, we get 350 weights for the first layer and 36 for the second, giving us 396 weights. However, if we split this into 2 hidden layers of 18 and 18, then we get 170 for the first layer, 306 for the second layer, and 18 for the third layer, giving us 494 weights. And if we do 3 hidden layers, we attempt 12 x 12 x 12, we get $110 + 132 + 132 + 12 = 386$ weights. So, this strongly suggests that 2 hidden layers is the optimal amount.

So using Mathematica, we plug in the following function:

- Maximize[10*(n - 1) + (n)*(36 - n - 1) + (36 - n), n]

which returned to us 510 weights for 2 hidden layers, the first of 22 nodes and the second of 12 nodes. We can also check any $2+$ hidden layers by using Python to run a MCMC simulation, but since 510 was the maximum number choice given, we do not need to do that.