Overfitting and Deterministic Noise

1. b

Deterministic Noise will increase. Since we know that Deterministic noise depends on H and that H' is a subset of H, the noise will only get bigger.

Regularization with Weight Decay

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression


train_data = []
with open('train.txt','r') as train_file:
    for line in train_file:
        train_data.append([float(index) for index in line.split()])
train_data = np.asarray(train_data)


test_data = []
with open('test.txt','r') as test_file:
    for line in test_file:
        test_data.append([float(index) for index in line.split()])
test_data = np.asarray(test_data)


def nonlinear_transform(X):
    new_X = []
    for x in X:
        x1 = x[0]
        x2 = x[1]
        new_x = [1, x1, x2, x1**2, x2**2, x1*x2, np.abs(x1-x2), np.abs(x1+x2)]
        new_X.append(new_x)
    return np.asarray(new_X)


def lin_reg(X, y):
    X_plus = np.linalg.inv(X.transpose().dot(X)).dot(X.transpose())
    w = X_plus.dot(y)
    return(w)


def class_err(X, w, y):
    correct_idx = []
    count = 0
    for x in X:
        if np.sign(w.dot(x)) == y[count]:
            correct_idx.append(count)
        count += 1
    class_err = 1-len(correct_idx)/float(count)
    return class_err
```

2. a

```
X = train_data[:,:2]
y = train_data[:,2]
Z = nonlinear_transform(X)
w = lin_reg(Z,y)
in_sample_err = class_err(Z,w,y)
print(f'In Sample Error: {in_sample_err}')

test_X = test_data[:,:2]
test_Z = nonlinear_transform(test_X)
test_y = test_data[:,2]
```

```
test_err = class_err(test_Z,w,test_y)
print(f'Out of Sample Error: {test_err}')
```

```
In Sample Error: 0.02857142857142858
Out of Sample Error: 0.08399999999999996
```

3. d

```
def lin_reg_weights(X, y, l):
    X_plus = np.linalg.inv(X.transpose().dot(X) + l*np.eye(X.shape[1])).dot(X.transpose())
    w = X_plus.dot(y)
    return(w)
```

```
k = -3
l = 10**k
X = train_data[:,:2]
X = nonlinear_transform(X)
y = train_data[:,2]
w = lin_reg_weights(X,y,l)
in_sample_err = class_err(X,w,y)
print(f'In Sample Error: {in_sample_err}')

test_X = nonlinear_transform(test_data[:,:2])
test_y = test_data[:,2]
test_err = class_err(test_X,w,test_y)
print(f'Out of Sample Error: {test_err}')
```

```
In Sample Error: 0.02857142857142858
Out of Sample Error: 0.07999999999999996
```

4. e

```
k = 3
l = 10**k
X = train_data[:,:2]
X = nonlinear_transform(X)
y = train_data[:,2]
w = lin_reg_weights(X,y,l)
in_sample_err = class_err(X,w,y)
print(f'In Sample Error: {in_sample_err}')

test_X = nonlinear_transform(test_data[:,:2])
test_y = test_data[:,2]
test_err = class_err(test_X,w,test_y)
print(f'Out of Sample Error: {test_err}')
```

```
In Sample Error: 0.37142857142857144
Out of Sample Error: 0.43600000000000005
```

5. d

```
k = -2
for k in [2,1,0,-1,-2]:
    l = 10**k
    X = train_data[:,:2]
    X = nonlinear_transform(X)
    y = train_data[:,2]
    w = lin_reg_weights(X,y,l)
    in_sample_err = class_err(X,w,y)

    test_X = nonlinear_transform(test_data[:,:2])
    test_y = test_data[:,2]
    test_err = class_err(test_X,w,test_y)
    print(f'k: {k}, Error: {test_err}')
```

```
k: 2, Error: 0.22799999999999998
k: 1, Error: 0.124
k: 0, Error: 0.09199999999999997
k: -1, Error: 0.05600000000000005
k: -2, Error: 0.08399999999999996
```

6. b

Value closest when $k = -1$. Error is roughly 0.06.

Regularization for Polynomials

7. c

Using the above definitions, we know that $H(10, 0, 3) = \{h | h(x) = \sum_{q=0}^{10} w_q L_q(x), w_q = 0$ for $q \geq 3\} = \{h | h(x) = \sum_{q=0}^{2} w_q L_q(x)\} = H_2$. We also know that $H(10, 0, 4) = H_3$ by the same process. Since $H(10, 0, 3)$ is inlcu

Neural Networks

8. d

The total number of operations is given by the forward propagation + backpropagation + updating weights.

Forward propagation is given by:

$x_j^{(l)} = \theta(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)})$. For $l = 1$, we have $6 \cdot 3 = 18$ operations and for $l = 2$, we have $4 \cdot 1 = 4$ operations. The total forward propagation operations is 22.

Back propagation is given by:

$\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2)(\sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \delta_j^{(l)}). For$l = 2, $we have$3 \cdot 1 = 3$ operations.

Updating weights is given by:

$w_{ij}^{(l)} = w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_J^{(l)}$. For each l, we have 1 operation. There are 22 in total.

Thus the total operations is given by $22 + 3 + 22 = 47$.

9. a

To get the minimum possible number of weights for the network, we must have 2 units for every hidden layer. After the 10 input units, we have 18 total hidden layers for the 36 hidden units. Each layer has 2 units and therefore 2 connections and 2 weights. Therefore the minimum number of weights is given by $10 + 18 \cdot 2 = 46$.

10. e

To find the maximum possible number of weights, we need to maximize the equation: $w = 10(h - 1) + h \cdot (36 - h - 1) + (36 - h)$ where h is the number of nodes in the first layer. Using Mathematica to maximize this function, we get a maximum when $h = 22$ nodes. In this case $w = 510$.