ESL: 11.1

From The Neural network side, since there is ~~not~~ one hidden layer.

$$f_k(X) = \beta_0 + \beta_k^T \theta(\alpha_{0n} + \alpha_n^T X)$$

From The PPR side we know:

$$f(x) = \sum_{m=1}^{N} g_n(w_n^T x)$$

Also we know:

$$g_n(w_m^T x) = \beta_n \theta(\alpha_{0n} + S_n(w_n^T x))$$

Then since is a PPR we know that $w = \frac{\hat{w}}{\|\hat{w}\|}$,

Then, we need to find some S_n and $\hat{w}$, such will allow us to equiparate both ~~[illegible]~~; this is $S = \|\alpha_n\|$ and

$$W_n^T = \frac{\alpha_n}{\|\alpha_n\|}$$

Then $g_n(w^T x) = F_n(x) = \beta_0 + \beta_n \theta(\alpha_{0n} + \alpha_n^T x)$ and

$$f(x) = \sum_{k=1}^{K} f_n(x)$$

The same argument could be done if we use a logistic regression for the $g_n(w_n^T x)$ of PPR and a classification network from the neural network where $g(T) = \frac{e^{T_k}}{\sum_{l=1} e^{T_l}}$.

ESL: 11.2

If the function is linear we know that $f'' = 0$.

Then if $y = \frac{1}{1+e^{-v}}$

$$\frac{\partial y}{\partial v} = \frac{e^{-v}}{(1+e^{-v})^2}$$

$$\frac{\partial^2 y}{\partial v^2} = e^{-v}(1+e^{-v})^2\left(2\cdot(1+e^{-v})^{-1} - 1\right)$$

Then if $v=0$.

$$\left.\frac{\partial^2 y}{\partial v^2}\right|_{v=0} = \frac{1}{4}\cdot\left(\frac{2}{2} - 1\right)$$

$= 0 \quad \Rightarrow$ The we could argue that is linear at $v=0$.

ESL 11.4

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

$$f_k(X) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

Since there is no hidden layer

$$T_k = \beta_{0k} + \beta_k^T X.$$

And now we assume that the $K=1$ is Base category. Then

$$T_k - T_1 = (\beta_{0k} - \beta_{01}) + (\beta_k - \beta_1)^T X$$

$$\therefore \Pr(K = k) = f_k(X) = \frac{e^{T_k - T_1}}{1 + e^{T_k - T_1}}$$

Wich transform for multiple K and using cross entropy.

$$-\sum y_{ik} \log(f_k(X))$$

$\Rightarrow$ wich is the multilogistic objective function.

5.1 Bishop

First we need to found the conversion or relation Between $\tanh(a)$ and $\sigma(a)$

$$\frac{e^a - e^{-a}}{e^a + e^{-a}} = \frac{e^a}{e^a + e^{-a}} - \frac{e^{-a}}{e^a + e^{-a}} = \frac{e^a}{e^a + e^{-a}} - \frac{e^{-a} + e^a - e^a}{e^a + e^{-a}}$$

$$= \frac{e^a}{e^a + e^{-a}} - \left(1 - \frac{e^a}{e^a + e^{-a}}\right)$$

$$= \frac{2e^a}{e^a + e^{-a}} \cdot \frac{e^{-a}}{e^{-a}} - 1$$

$$= \frac{2}{1 + e^{-2a}} - 1$$

$$\frac{e^a - e^{-a}}{e^a + e^{-a}} = 2\,\sigma(2a) - 1$$

Then if we replace this into

$$a_k = \sum_{j=1}^{n} w_{kj}^{(k)} \cdot (2\sigma(2a) - 1) + w_{k0}^{(k)}$$

$$a_k = \sum_{j=1}^{n} 2w^{(k)} \sigma(2a) - 2w_{kj}^{k} + w_{k0}^{(k)}$$

$$a_k = \sum_{j}^{n} 2w_{kj}^{(k)} \sigma(2a) + \sum_{j}^{n} \left[-2w_{kj}^{k} + w_{k0}^{k} \right]$$

Then if for a regular neural network we have to convert into:

$\omega_p^{(2p)} = 2\omega_k^{(2)}$, $\quad a_j^{(p)} = 2a_j^{(2)}$, $\omega_{k0}^{(2p)} = -\sum_{j=1}^{n} \omega_{kj}^{(2)} + \omega_{k0}^{2}$

Using Thes equalities we transform a_k into

$$\boxed{a_p = \sum_{j=1}^{n} \omega_p^{(2p)} \theta(a) + \omega_{p0}^{(2p)}}$$

Bishop 5.5

$$P(T/W_1, \ldots, W_K) = \prod_{k=1}^{K} y_k^{t_k}$$ $\Rightarrow$ Network output for k class.

Then for N data point we have that.

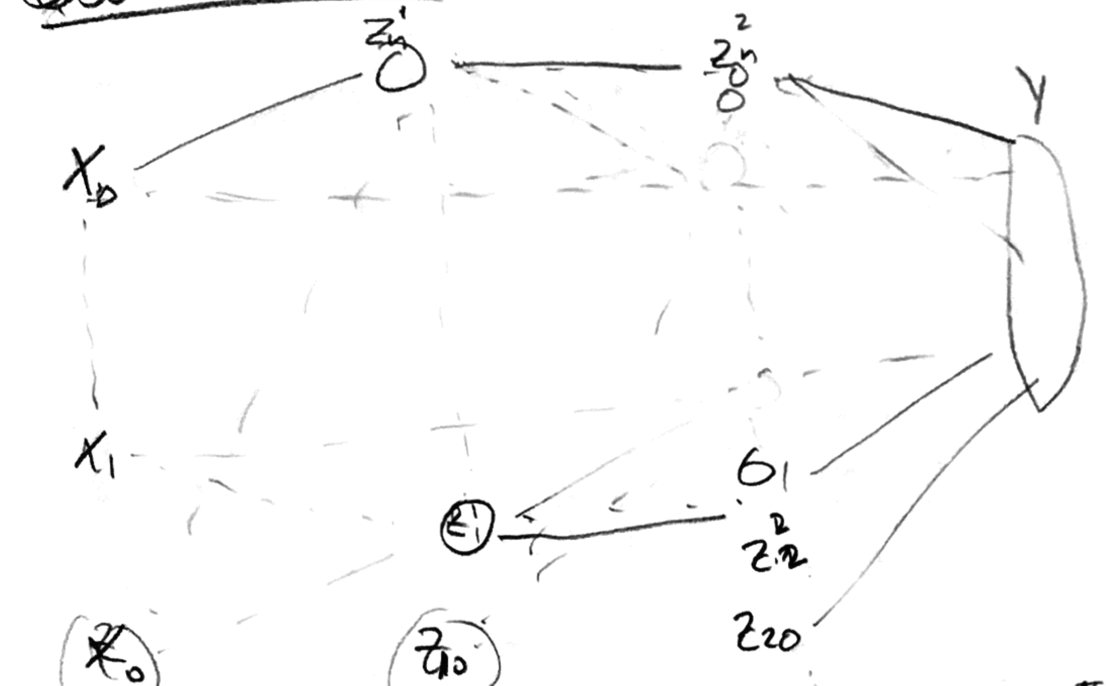
$$p(T/W_1, \ldots, W_K) = \prod_{k=1}^{K} \prod_{n=1}^{N} y_{nk}^{t_{nk}}$$

Then if we compare for the log function.

$$l(T/W_1, \ldots, W_K) = -\sum_{k=1}^{K} \sum_{n=1}^{N} t_{nk} \cdot \log(y_{nk}).$$

Which is the log likelihood function of the 5.24.

Question 6



$$Z_n^2 = \sigma\left(\beta_{0k}^{(2)} + \beta^{T(2)} Z_n^1\right)$$

Equations:

$$Z_n^1 = \sigma(\underbrace{\alpha_{0n} + \alpha_n^T X}_{p})$$

$$\hat{Y}_k = \sigma\left(\beta_{0k}^{1} + \beta_k^{(1)T} Z^2 + \gamma_k^T X + \gamma_{0k}\right)$$

$$R = \frac{1}{n}\sum_{i=1}^{K} \left(\hat{Y}_k - y\right)^2$$

$$\frac{\partial R}{\partial \beta_k} = \delta_{ki} Z_{ni}, \quad s_{mi} = \sigma'(\alpha_n^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

δ_{ki} => are errors in output layer

$$\frac{\partial R}{\partial \alpha_n} = s_{ni} x_{il}$$

$$\frac{\partial R}{\partial \gamma_k} = 2(\hat{y}_n - y) \cdot \hat{y}_k (1 - \hat{y}_k) \cdot X$$

* The main change is that we have a another set of parameters that go from the input layer to the output layer, creating a new partial derivative.

hw8 q7

June 1, 2020

1 Question 7

1.1 Part A

[60]:
```
import numpy as np

lambda_list = [0.001,0.01,0.1,.25,.5,.75,1,2,3,5]
hiddenSize = 10
epoch_list = list(range(0,10000,100))
hiddenSize_list = list(range(1,11,1))

def sigmoid(x):
    rv = 1/(1+np.exp(-x))
    return rv



def deriv_sigmoid(output):
    return output*(1-output)

def predict(X,alpha,beta):
        l_0 = X
        l_1 = sigmoid(np.dot(l_0,alpha))
        l_2 = sigmoid(np.dot(l_1,beta))
        return l_2

def Neural_backprop(itera, hiddenSize, X, y, eta, Lambda):

    alpha_t = 2*np.random.random((3,hiddenSize)) - 1
    beta_t = 2*np.random.random((hiddenSize,1)) - 1
    alpha_t_1 = alpha_t.copy()
    beta_t_1 = beta_t.copy()
    for j in range(itera):
        l_0 = X
        l_1 = sigmoid(np.dot(l_0,alpha_t))
        l_2 = sigmoid(np.dot(l_1,beta_t))
        l_2_error = l_2 - y
```

```
        l_2_delta = l_2_error*deriv_sigmoid(l_2)
        l_1_error = l_2_delta.dot(beta_t.T)
        l_1_delta = l_1_error * deriv_sigmoid(l_1)
        beta_t= beta_t_1*(1- 2*Lambda*eta) + eta * (l_1.T.dot(l_2_delta))
        alpha_t = alpha_t_1*(1- 2*Lambda*eta) + eta*(l_0.T.dot(l_1_delta))
    #mean_sq_error = (l_2_error.copy())**2
    #mean_sq_error = np.mean(mean_sq_error)
    return alpha_t,beta_t
```

[ ]:

1.2 Part B

[8]:

```
def dgp(n):
    x1 = np.random.normal(0,1,n)
    x2 = np.random.normal(0,1,n)
    X = np.concatenate((x1.reshape(n,1),x2.reshape(n,1)),axis=1)
    Z  = np.random.normal(0,1,n)
    a2 = np.zeros((2,1))
    a2[0][0]=3
    a2[1][0]=-3
    Y=sigmoid(3*X@np.ones((2,1))) +(X@a2)**2 + .3*Z.reshape(n,1)
    return (Y,X)
y_train, X_train = dgp(100)
X_train= np.concatenate((X_train,np.ones((100,1))),axis=1)
y_test, X_test = dgp(1000)
X_test= np.concatenate((X_test,np.ones((1000,1))),axis=1)
```

[57]:

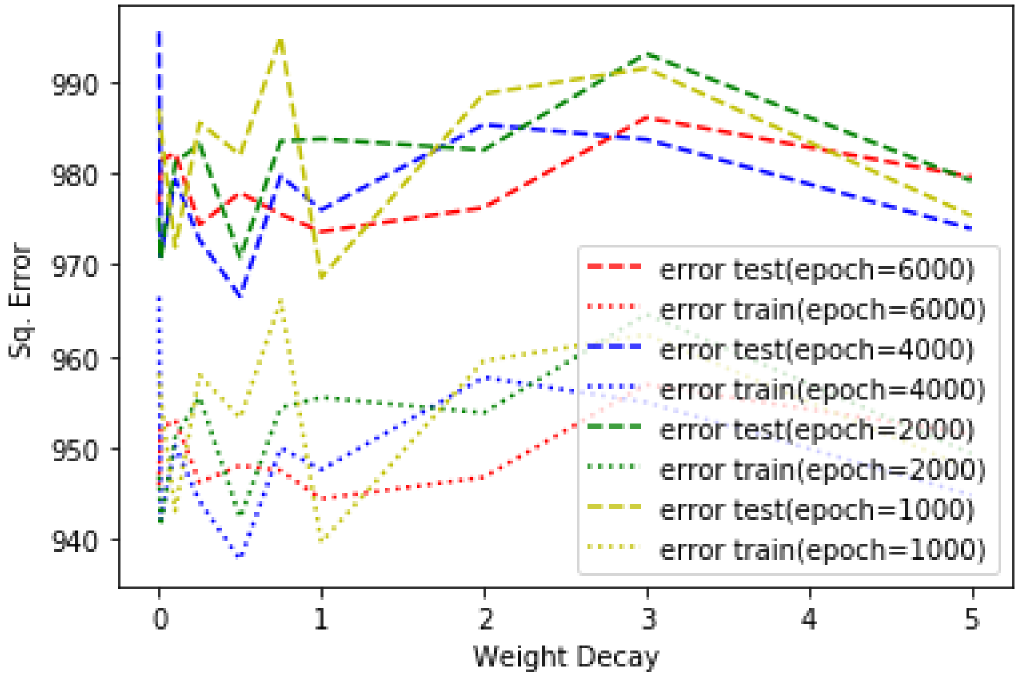
```
error_train_6000 = []
error_test_6000 = []
for i in lambda_list:
    alpha, beta = Neural_backprop(6000, 10, X_train, y_train, 0.01,i)
    sq_error_test_mean = np.mean((y_test - predict(X_test,alpha,beta))**2)
    sq_error_train_mean = np.mean((y_train - predict(X_train,alpha,beta))**2)
    error_test_6000.append(sq_error_test_mean)
    error_train_6000.append(sq_error_train_mean)

error_train_4000 = []
error_test_4000 = []
for i in lambda_list:
    alpha, beta = Neural_backprop(4000, 10, X_train, y_train, 0.01,i)
    sq_error_test_mean = np.mean((y_test - predict(X_test,alpha,beta))**2)
    sq_error_train_mean = np.mean((y_train - predict(X_train,alpha,beta))**2)
    error_test_4000.append(sq_error_test_mean)
    error_train_4000.append(sq_error_train_mean)
```

```
error_train_2000 = []
error_test_2000 = []
for i in lambda_list:
    alpha, beta = Neural_backprop(2000, 10, X_train, y_train, 0.01,i)
    sq_error_test_mean = np.mean((y_test - predict(X_test,alpha,beta))**2)
    sq_error_train_mean = np.mean((y_train - predict(X_train,alpha,beta))**2)
    error_test_2000.append(sq_error_test_mean)
    error_train_2000.append(sq_error_train_mean)

error_train_1000 = []
error_test_1000 = []
for i in lambda_list:
    alpha, beta = Neural_backprop(1000, 10, X_train, y_train, 0.01,i)
    sq_error_test_mean = np.mean((y_test - predict(X_test,alpha,beta))**2)
    sq_error_train_mean = np.mean((y_train - predict(X_train,alpha,beta))**2)
    error_test_1000.append(sq_error_test_mean)
    error_train_1000.append(sq_error_train_mean)
```

[58]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.plot(lambda_list,error_test_6000,color='r', ls='--' ,label='error␣
 ↪test(epoch=6000)')
plt.plot(lambda_list,error_train_6000,color='r', ls=':', label='error␣
 ↪train(epoch=6000)')
plt.plot(lambda_list,error_test_4000,color='b', ls='--', label='error␣
 ↪test(epoch=4000)')
plt.plot(lambda_list,error_train_4000,color='b', ls=':', label='error␣
 ↪train(epoch=4000)')
plt.plot(lambda_list,error_test_2000,color='g',ls='--', label='error␣
 ↪test(epoch=2000)')
plt.plot(lambda_list,error_train_2000,color='g',ls=':', label='error␣
 ↪train(epoch=2000)')
plt.plot(lambda_list,error_test_1000,color='y',ls='--', label='error␣
 ↪test(epoch=1000)')
plt.plot(lambda_list,error_train_1000,color='y',ls=':', label='error␣
 ↪train(epoch=1000)')
plt.xlabel('Weight Decay')
plt.ylabel("Sq. Error")
leg = ax.legend();
```

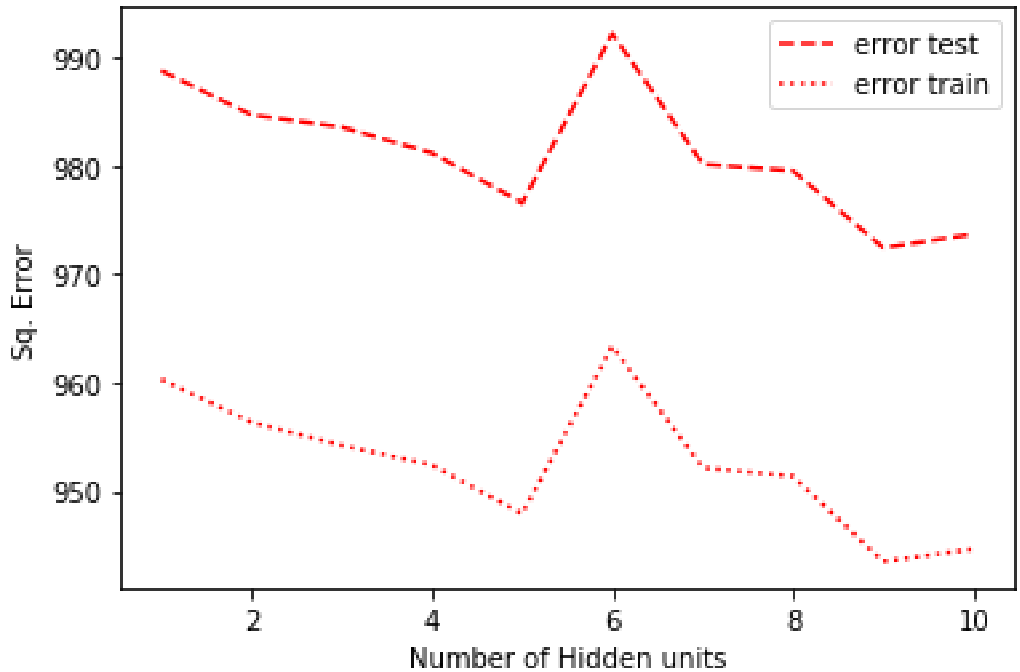


We can see that as increase the weight decay have different behaviours depending how the amount of epochs we pass. For example between 0 and 1 we can see that very sharp movements for the train and test error levels for the simulations with lower number epochs which means that are more suspetible to overfitting.

1.3 Part C

[61]:
```
error_train_hn = []
error_test_hn = []
for i in hiddenSize_list:
    alpha, beta = Neural_backprop(6000, i, X_train, y_train, 0.01,.5)
    sq_error_test_mean = np.mean((y_test - predict(X_test,alpha,beta))**2)
    sq_error_train_mean = np.mean((y_train - predict(X_train,alpha,beta))**2)
    error_test_hn.append(sq_error_test_mean)
    error_train_hn.append(sq_error_train_mean)
```

[62]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.plot(hiddenSize_list,error_test_hn,color='r', ls='--' ,label='error test')
plt.plot(hiddenSize_list,error_train_hn,color='r', ls=':', label='error train')
plt.xlabel('Number of Hidden units')
plt.ylabel("Sq. Error")
leg = ax.legend();
```



We can see that minimize at 9.