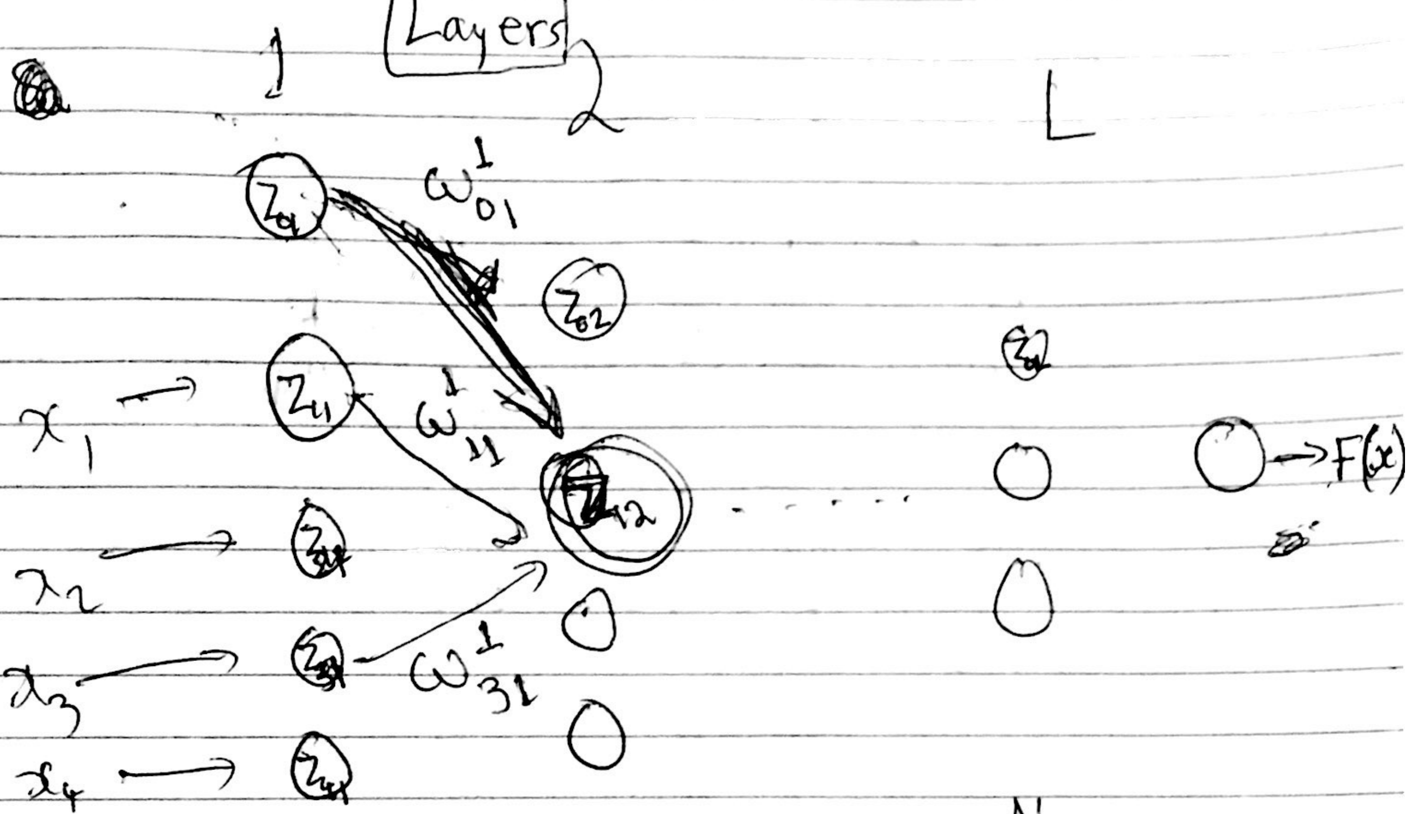
STATS 315B Final



Suppose cost function $C(\omega) = \sum_{i=1}^{N} c_i(\omega)$ where $c_i(\omega) = \frac{1}{2}[y_i - F(x_i)]^2$. Further, let node outputs for layer k be $\vec{Z}_k = (Z_{1k}, \dots, Z_{m_k k})$ with input node $\vec{x} = (x_1, \dots, x_n)$. Finally, let the edge from Z_{ik} to $Z_{j(k+1)}$ have weight $\omega^{(k)}_{ij}$. Then, using sigmoid function $S(x) = \frac{1}{1+e^{-x}}$

$$Z_{j1} = x_j \quad \forall j \in [1, n]$$

$$Z_{0k} = 1 \quad \forall k \in [1, L]$$

$$Z_{m(k+1)} = S\left(\omega^{(k)}_{0m} + \sum_{i=1}^{m_k} \omega^{(k)}_{im} Z_{ik}\right) \quad \forall k \in [1, L-1]$$

$$Z_1^{L+1} = \omega^{(L)}_{01} + \sum_{i=1}^{m_L} \omega^{(L)}_{i1} Z_{iL} = F(x)$$

Then, for each node Z_{mk}, error e_{mk} is

$$e_{1(L+1)} = F(x_i) - y_i \quad (m=1)$$

$$e_{mk} = Z_{mk}(1-Z_{mk})\left(\sum_{i=1}^{m_k} \omega_{mi}^{(k)} e_{i(k+1)}\right) \quad (m \neq 1)$$

The e_{mk} for $m \neq 1$ is computed recursively starting from $k=L$ and decrementing by 1 till $k=1$ is reached.

Now, we propose:

$$\boxed{1} \quad e_{mk} = Z_{mk}(1-Z_{mk})\frac{\partial c_i}{\partial Z_{mk}}$$

~~*~~ We prove this via induction:

Base Case: $k=L$,

$$e_{mL} = \omega_{m1}^{(L)} Z_{mL}(1-Z_{mL})\, e_{1(L+1)}$$

$$= \omega_{m1}^{(L)} Z_{mL}(1-Z_{mL})(F(x_i)-y_i)$$

$$= Z_{mL}(1-Z_{mL})\frac{\partial c_i}{\partial Z_{mL}} \quad \checkmark$$

Inductive step: Before proceeding, we use the sigmoid function property $S'(x) = S(x)(1-S(x))$, which is easily verified. So now, assuming [1] holds for $k+1$, we show it holds for k.

$$e_{mk} = Z_{mk}(1-Z_{mk})\left(\sum_{i=1}^{m_{k+1}} \omega_{mi}^{(k)} e_{i(k+1)}\right)$$

$$= Z_{mk}(1-Z_{mk})\left(\sum_{j=1}^{m_{k+1}} \omega_{mj}^{(k)} S'\left(\omega_{0j}^{(k+1)} + \sum_{i=1}^{m_k} \omega_{ij}^{(k)} Z_{ik}\right)\frac{\partial C_i}{\partial Z_{j(k+1)}}\right)$$

Using aforementioned property

$$= Z_{mk}(1-Z_{mk})\left(\sum_{j=1}^{m_{k+1}} \frac{\partial Z_{j(k+1)}}{\partial Z_{mk}} \cdot \frac{\partial C_i}{\partial Z_{j(k+1)}}\right)$$

$$= Z_{mk}(1-Z_{mk})\frac{\partial C_i}{\partial Z_{mk}}$$

$\therefore$ we are done

So to compute the gradient $G(\bar{\omega})$, we have

$$\frac{\partial C_i}{\partial \omega_{rs}^{(k)}} = Z_{rk} \cancel{S'\left(\omega_{0s}^{(k)} + \sum_{j=1}^{m_k} \omega_{js}^{(k)} Z\right)}$$

$$= Z_{rk} S'\left(\omega_{0s}^{(k)} + \sum_{j=1}^{m_k} \omega_{js}^{(k)} Z_{jk}\right) \frac{\partial C_i}{\partial Z_{s(k+1)}}$$

$$= Z_{rk} Z_{s(k+1)}(1-Z_{s(k+1)}) \frac{\partial C_i}{\partial Z_{s(k+1)}}$$

$$= Z_{rk}\, e_{s(k+1)}$$

So, the algorithm works as follows:

1.) Set $e_{i(L+1)} = F(x_i) - y_i$

2.) For $k = L$ to 1, $k--$,

- Compute $\frac{\partial C_i}{\partial \omega_{rs}^{(k)}} = z_{rk} e_{s(k+1)}$ for all edges.
- For $m = 1$ to m_k, $m++$
  - Compute $e_{mk} = z_{mk}(1 - z_{mk})\left(\sum_{i=1}^{m_{k+1}} \omega_{mi}^{(k)} e_{i(k+1)}\right)$

After repeating for each training sample i,

$$G_i(w) = \frac{\partial C}{\partial \omega_{rs}^{(k)}} = \sum_{i=1}^{n} \frac{\partial C_i}{\partial \omega_{rs}^{(k)}}$$

Scanned by CamScanner

Using sum-of-squares error criterion,

$$G_i(w) = \frac{1}{2}\left[y_i - \hat{F}(x_0)\right]^2 \quad \text{for output } \vec{y}$$

(different G)

$$\therefore \frac{\partial G_i}{\partial a_0} = \hat{F}(x_0) - y_i$$

$$\frac{\partial G_i}{\partial a_m} = (\hat{F}(x_0) - y_i)\, B(x \mid \mu_m, \sigma_m)$$

$$\frac{\partial G_i}{\partial \mu_{jm}} = a_m(\hat{F}(x_0) - y_i)\frac{\partial B}{\partial \mu_{jm}}$$

$$= \frac{a_m(\hat{F}(x_0) - y_i)\, B(x \mid \mu_m, \sigma_m)\,(x_{ji} - \mu_{jm})}{\sigma_m^2}$$

$$\frac{\partial G_i}{\partial \sigma_m} = \frac{a_m(\hat{F}(x_0) - y_i)\, B(x \mid \mu_m, \sigma_m)\sum_{j=1}^{n}(x_{ji} - \mu_{jm})^2}{\sigma_m^3}$$

$$\frac{\partial G}{\partial w} = \sum_{i=1}^{N} \frac{\partial G_i}{\partial w}$$

Here, we employed chain rule and each

$\vec{x}_i = (x_{1i}, \ldots x_{ni}) \quad \forall i \in [1, N]$

Since Σ is a (symmetric) positive definite matrix, it has a Cholesky decomposition LL^T.

Let $y = x - \mu_m$, then

$$(x-\mu_m)^T \Sigma (x-\mu_m)$$

$$= y^T \Sigma y = y^T L L^T y$$

$$= (L^T y)^T (L^T y) = Z^T Z \quad \therefore \text{ spherically symmetric}$$

$$\therefore \boxed{\tilde{x} = L^T x \text{ and } \tilde{\mu}_m = L^T \mu_m}$$

4. We claim that this property holds ~~for~~ if and only if Σ_m has exactly one eigenvalue λ_m with multiplicity 1.

$\Leftarrow$: Assuming Σ_m has exactly one eigenvalue λ_k with multiplicity one, then we can again write (using eigendecomposition) $\Sigma_m = S^T D S$:

$$(x-\mu_m)^T \Sigma_m (x-\mu_m)$$

$$= (x-\mu_m)^T S^T D S (x-\mu_m)$$

$$= \left(S(x-\mu_m)\right)^T \begin{bmatrix} 0 & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & & \lambda_m & \\ & & & & 0 \\ & & & & & \ddots \\ & & & & & & 0 \end{bmatrix} S(x-\mu_m)$$

$\therefore B(x \mid \mu_m, \Sigma_m)$ only varies in direction of eigenvector corresponding to λ_k

$\Rightarrow$: Proceeding by contradiction, suppose the function varies in one direction only in the input space but Σ_m has more than one eigenvalue. If this is the case, then the function can vary in any direction that is a linear combination of the eigenvectors corresponding to the eigenvalues, implying more than one direction, hence a contradiction. Therefore initial assumption is wrong and Σ_m has one eigenvalue w/ multiplicity 1. This completes the proof.

5

a.)

//Loading and labeling data.

```
> spam_all<-read.csv(file.choose())
> spam_test<-read.csv(file.choose())
> spam_train<-read.csv(file.choose())
> rflabs<-c("make","address","all","3d", "our", "over", "remove", "internet", "order", "mail", "receive", "
will", "people", "report", "addresses", "free", "business","email", "you", "credit", "your", "front", "000",
"money", "hp", "hpl", "george", "650", "lab", "labs","telnet","857", "data", "415", "85","technology","19
99", "parts", "pm", "direct", "cs", "meeting", "original", "project", "re", "edu", "table", "conference", ";",
"{", "[", "!", "$", "#", "CAPAVE", "CAPMAX" , "CAPTOT", "type")
> colnames(spam_all) = rflabs
> colnames(spam_test) = rflabs
> colnames(spam_train) = rflabs
```

//Standardizing predictor values in training data and isolating output.

```
> scaled_all = scale(spam_all)
> scaled_test = scale(spam_test)
> scaled_train = scale(spam_train)
> scaled_all = scale(spam_all)
> scaled_test = scale(spam_test)
> scaled_train = scale(spam_train)
> train_x = scaled_train[,c(1:57)]
>train_y = spam_train[,58]
```

//Running neural nets with number of edges being 59*(size of hidden layer) + 1

```
> library(nnet)

> nnet_1 = nnet(train_x, train_y, size = 1, Wts = runif(60,-.5,.5), linout = T)
> nnet_2 = nnet(train_x, train_y, size = 1, Wts = runif(119,-.5,.5), linout = T)
> nnet_3 = nnet(train_x, train_y, size = 1, Wts = runif(178,-.5,.5), linout = T)
> nnet_4 = nnet(train_x, train_y, size = 1, Wts = runif(237,-.5,.5), linout = T)
> nnet_5 = nnet(train_x, train_y, size = 1, Wts = runif(296,-.5,.5), linout = T)
> nnet_6 = nnet(train_x, train_y, size = 1, Wts = runif(355,-.5,.5), linout = T)
> nnet_7 = nnet(train_x, train_y, size = 1, Wts = runif(414,-.5,.5), linout = T)
> nnet_8 = nnet(train_x, train_y, size = 1, Wts = runif(473,-.5,.5), linout = T)
> nnet_9 = nnet(train_x, train_y, size = 1, Wts = runif(532,-.5,.5), linout = T)
> nnet_10 = nnet(train_x, train_y, size = 1, Wts = runif(591,-.5,.5), linout = T)
```

Best model is, as expected, the one with 10 hidden units (lowest value after 100 iterations).

b.)

```
//Defining error matrix for all possible sizes of hidden layer and decay rate combinations.

err = matrix(rep(0,110), nrows = 10, ncols = 11)

//Standardizing predictor values in test data and isolating output.

test_ x = scaled_test[,c(1:57)]

test_y = spam_test[,58]

//First loop is for units in hidden layer, the second is the decaying, and the third is the number of runs

for ( i in 1:10){

 for (j in 1:11){

  err_new = 0

  for (k in 1:10){

   nnet_l = nnet(train_x, train_y, size = i, Wts = runif(59*i+1,-.5,.5), linout = F, decay = 0.1*(j-1))

   y0 = predict(nnet_l, test_x, type = "raw")

//Converts into classified data

   y0 = ifelse(y0 < 0.5, 0, 1)

   err_new  = err_new + mean(y0 != test_y)

  }

//Averaging error after runs and storing resulting value in matrix

  err[i,j]=err_new/10

 }

}

//Finding minimum error value in matrix. Row index corresponds to optimal size and column index corresponds to optimal regularization.

which (err == min(err),arr.ind = TRUE)
```

This results in 8 units and decay of 0.2 as optimal model.