Assignment 3

Julien De Mori

June 6, 2013

1 Problem 1

For this problem we want to derive a recursive algorithm for updating all of the weights in the neural network. We use the following update algorithm for each weight,

$$w_{jk}^{(h),(t)} = w_{jk}^{(h),(t-1)} - \eta \frac{\partial R^{(t-1)}}{\partial w_{jk}^{(h)}} \tag{1}$$

where $R = \sum_{i=0}^{N} R_i$ is the sum of the squared-errors from each observation i and $w_{jk}^{(h),(t)}$ is the weight from node j in layer $h-1$ to node k in layer h, at iteration t. There are $H+1$ values that h can take on, where H is the number of hidden layers in the neural network. We also let $y_k^{(h)} = \sigma_k^{(h)}$ be the output of the k-th "squashing" function for layer h. So to determine the recursive relation for updating weights, we need to determine $\frac{\partial R_i}{\partial w_{jk}^{(h)}}$.

I also denote $f^{(h)}$ to be the number of nodes that layer h feeds into (i.e. the number of nodes in layer $h+1$). In addition $y_{k,i}^{(h)}$ denotes the output from node k in layer h from the i^{th} observation.

$$\frac{\partial R_i}{\partial w_{jk}^{(h)}} = \frac{\partial R_i}{\partial y_{k,i}^{(h)}} \frac{\partial y_{k,i}^{(h)}}{\partial w_{jk}^{(h)}} \tag{2}$$

$$= \left[\sum_{l=0}^{f^{(h)}} \frac{\partial R_i}{\partial y_{l,i}^{(h+1)}} \frac{\partial y_{l,i}^{(h+1)}}{\partial y_{k,i}^{(h)}} \right] \frac{\partial y_{k,i}^{(h)}}{\partial w_{jk}^{(h)}} \tag{3}$$

To get from equations (2) to (3), we have evaluated the expression $\frac{\partial R_i}{\partial y_{k,i}^{(h)}}$ using the chain rule and summing over all possible outputs to the following layer $h+1$. We are effectively expanding the total derivative as a function of the multiple outputs of the nodes that node k in layer h can connect to.

We now need to evaluate the second and third derivatives in equation (3), which we know how to do since each output is a sigmoid function whose argument is a linear combination of all of the inputs (other sigmoid functions for all cases except $h = 1$). That is,

$$y_k^{(h)} = \sigma\left(\sum_{p=0}^{f^{(h)}} w_{pk}^{(h)} y_p^{(h-1)}\right) = \sigma(z). \tag{4}$$

Therefore, using the fact that $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$, we compute

$$\frac{\partial y_{k,i}^{(h)}}{\partial w_{jk}^{(h)}} = \frac{\partial y_k^{(h)}}{\partial z}\frac{\partial z}{\partial w_{jk}^{(h)}} \tag{5}$$

$$= \sigma(z)(1-\sigma(z)) \cdot \frac{\partial}{\partial w_{jk}^{(h)}}\left[\sum_{p=0}^{f^{(h)}} w_{pk}^{(h)} y_p^{(h-1)}\right] \tag{6}$$

$$= y_k^{(h)}(1-y_k^{(h)}) \cdot y_j^{(h-1)} \tag{7}$$

and similarly,

$$\frac{\partial y_{l,i}^{(h+1)}}{\partial y_{k,i}^{(h)}} = y_l^{(h+1)}(1-y_l^{(h+1)}) \cdot w_{jl}^{(h+1)} \tag{8}$$

since the second derivative is the only term that changes, and $\frac{\partial y}{\partial w_{jk}^{(h)}} = w_{jl}^{(h)}$. Plugging these back into equation (3), we obtain

$$\frac{\partial R_i}{\partial w_{jk}^{(h)}} = \left[\sum_{l=0}^{f^{(h)}} \frac{\partial R_i}{\partial y_{l,i}^{(h+1)}} \cdot y_l^{(h+1)}(1-y_l^{(h+1)}) \cdot w_{jl}^{(h+1)}\right] \cdot y_k^{(h)}(1-y_k^{(h)}) \cdot y_j^{(h-1)}. \tag{9}$$

I then define $\delta_{jl,i}^{(h)} = \frac{\partial R_i}{\partial w_{jl,i}^{(h)}} \cdot \frac{1}{y_{j,i}^{(h-1)}}$, so that due to equation (2), we can rewrite

$$\frac{\partial R_i}{\partial y_{l,i}^{(h+1)}} = \frac{\delta_{jl,i}^{(h+1)}}{y_l^{(h+1)}(1-y_l^{(h+1)})}. \tag{10}$$

Substituting this all back into equation (9) we obtain

$$\frac{\partial R_i}{\partial w_{jk}^{(h)}} = \left[\sum_{l=0}^{f^{(h)}} \delta_{jl,i}^{(h+1)} \cdot w_{kl}^{(h)}\right] \cdot y_{k,i}^{(h)}(1-y_{k,i}^{(h)}) y_{j,i}^{(h-1)}. \tag{11}$$

Now if we denote $\delta_{jk,i}^{(h)} = \left[\sum_{l=0}^{f^{(h)}} \delta_{jl,i}^{(h+1)} \cdot w_{kl}^{(h)}\right] \cdot y_{k,i}^{(h)}(1 - y_{k,i}^{(h)})$, then equation (11) reduces to

$$\frac{\partial R_i}{\partial w_{jk}^{(h)}} = \delta_{jk,i}^{(h)} \cdot y_{j,i}^{(h-1)} \tag{12}$$

Note that the $\delta_{jk,i}^{(h)}$ can be considered an effective "error" term associated with the respective weight from the i^{th} observation. Therefore, equation (1) becomes

$$w_{jk}^{(h),(t)} = w_{jk}^{(h),(t-1)} - \eta \cdot \sum_{i=1}^{N} \delta_{jk,i}^{(h)} \cdot y_{j,i}^{(h-1)}. \tag{13}$$

So we have determined a *recursive* relationship for updating each of the possible weights in the multi-hidden layer neural network, where the update for a given weight only depends on the output from another node that it is weighting, and an error term (the δ), which are computed for each observation. Note that the "error term" is computed recursively as it depends on all of the error terms in the hidden layers closer to the final output, as well as the weights from the previous batch update, and the outputs which were all computed in the forward pass. Therefore, the algorithm effectively makes a forward pass with all of the observations and computes the outputs y for each node in each layer, given the current set of weights, and then updates each weight, using the implicitly recursive relation in equation (13), for which the base case is the final node. Note that each iteration is a batch update, since the above relations are using the values of $w_{jk}^{(h)}$ from the previous time step. This whole process is repeated until whatever convergence criterion used is met.

2 Problem 2

In this problem we want to determine the gradient of

$$\mathbf{B}(\bar{x}|\mu_m, \sigma_m) = \exp\left(-\frac{1}{2\sigma_m}\sum_{i=1}^{N}(x_i - \mu_{im})^2\right) \tag{14}$$

with respect to all relevant parameters, where N is the number of features used in the statistical learning problem. First of all, we denote $\mathbf{B}_m \equiv \mathbf{B}(\bar{x}|\mu_m, \sigma_m)$, and $\hat{F}(\bar{x}) = a_0 + \sum_{m=1}^{M} a_m \mathbf{B}_m$, and $R = \sum_{j=1}^{n} R_j = \frac{1}{2}\sum_{j=1}^{n}(y_j - \hat{F}(\bar{x}^{(j)}))^2$. Here n is the number of observations made, and $\bar{x}^{(j)}$ denotes the vector containing all the data values for the j^{th} observation. Also, for shorthand, we denote $\hat{F}(\bar{x}^{(j)}) \equiv \hat{F}_j$. Now I derive the gradients $\nabla_{a_m} R$, $\nabla_{\sigma_m} R$, and $\nabla_{\mu_{im}} R$ by obtaining one of the components of each gradient, and noting

that the gradient itself would simply be all of the components vertically concatenated into a vector.

We start with the easiest, and for each case only compute the derivative of R_i, and add them all together at the end.

$$\frac{\partial R_j}{\partial a_m} = \frac{\partial R_i}{\partial \hat{F}_j} \cdot \frac{\partial \hat{F}_j}{\partial a_m} \tag{15}$$

$$= -(y_j - \hat{F}_j) \cdot B_{m,j} \tag{16}$$

where $B_{m,j}$ is the output of the m^{th} basis function for the j^{th} observation. We used the fact that $\frac{\partial R_j}{\partial \hat{F}_j} = -(y_j - \hat{F}_j)$. Then we have that

$$\frac{\partial R_j}{\partial \sigma_m} = \frac{\partial R_j}{\partial \hat{F}_j} \cdot \frac{\partial \hat{F}_j}{\partial B_m} \cdot \frac{\partial B_m}{\partial \sigma_m} \tag{17}$$

$$= -a_m(y_i - \hat{F}_j) \cdot B_{m,j} \cdot \left(\frac{1}{\sigma_m^3} \sum_{i=1}^{N} (x_i - \mu_{im})^2 \right) \tag{18}$$

since $\frac{\partial B_m}{\partial \sigma_m} = B_{m,j} \cdot \left(\frac{1}{\sigma_m^3} \sum_{i=1}^{N} (x_i - \mu_{im})^2 \right)$ and $\frac{\partial \hat{F}_j}{\partial B_m} = a_m$. Note that the x_i values in the previous equation clearly correspond to observation j.

Lastly,

$$\frac{\partial R_j}{\partial \mu_{im}} = \frac{\partial R_j}{\partial \hat{F}_j} \cdot \frac{\partial \hat{F}_j}{\partial B_m} \cdot \frac{\partial B_m}{\partial \mu_{im}} \tag{19}$$

$$= -a_m(y_j - \hat{F}_j) \cdot \frac{\partial B_{m,j}}{\partial \mu_{im}} \tag{20}$$

$$= -a_m(y_j - \hat{F}_j) \cdot B_{m,j} \cdot \left(-\frac{1}{2\sigma_m^2}(-2)(x_i - \mu_{im}) \right) \tag{21}$$

$$= -a_m(y_j - \hat{F}_j) \cdot B_{m,j} \cdot \left(\frac{x_i - \mu_{im}}{\sigma_m^2} \right). \tag{22}$$

Now that we have determined the partial derivatives of R_j with respect to all the parameters of interest, the components of each of the different gradient vectors will be $\frac{\partial R}{\partial \mu_{im}} = \sum_{j=1}^{n} \frac{\partial R_j}{\partial \mu_{im}}$, $\frac{\partial R}{\partial \sigma_m} = \sum_{j=1}^{n} \frac{\partial R_j}{\partial \sigma_m}$, and $\frac{\partial R}{\partial a_m} = \sum_{j=1}^{n} \frac{\partial R_j}{\partial a_m}$.

3 Problem 3

We now consider elliptically symmetric Gaussian basis functions for our neural network, of the form

$$\hat{\mathbf{B}}(\bar{x}|\mu_m, \Sigma) = \exp\left(-\frac{1}{2}(\bar{x} - \mu_m)^T \Sigma (\bar{x} - \mu_m)\right) \tag{23}$$

and want to determine linear transformations for $\bar{x}$ and μ_m so that the output of a neural network composed of these basis functions is the same as the output of one composed of spherical Gaussian basis functions as in Problem 2. This amounts to determining linear transformations such that $\hat{\mathbf{B}}(\bar{x}|\mu_m, \Sigma) = \mathbf{B}(\bar{x}|\mu_m, 1)$ (we can set $\sigma_m = 1$), since the feed-forward steps would yield the same output, and the gradients would also be the same, yielding the same updates for the relevant parameters during the back-propagation phase. We denote the transformed vectors as $\bar{x}'$ and μ'_m respectively, and postulate that for the desired property to hold, $\bar{x}' = \Sigma^{-1/2}\bar{x}$ and $\mu'_m = \Sigma^{-1/2}\mu_m$. Note that $\Sigma^T = \Sigma$ since it is a symmetric matrix. In addition, due to its symmetry, we can do eigenvalue decomposition on it so that $\Sigma = U\Lambda U^{-1}$, where U is a matrix of eigenvectors that are orthonormal to each other, and obviously $UU^{-1} = I$, the identity matrix. D is a diagonal matrix containing the eigenvalues of Σ. Recall also that $(XY)^T = Y^T X^T$. Substituting these two expressions into equation (23) and using the above properties, we obtain the following:

$$\begin{aligned}
\hat{\mathbf{B}}(\bar{x}'|\mu'_m, \Sigma) &= \exp\left(-\frac{1}{2}(\bar{x}' - \mu'_m)^T \Sigma (\bar{x}' - \mu'_m)\right) && (24)\\
&= \exp\left(-\frac{1}{2}(\Sigma^{-1/2}(\bar{x} - \mu_m))^T \Sigma (\Sigma^{-1/2}(\bar{x} - \mu_m))\right) && (25)\\
&= \exp\left(-\frac{1}{2}(\bar{x} - \mu_m)^T \Sigma^{-T/2}\Sigma\Sigma^{-1/2}(\bar{x} - \mu_m)\right) && (26)\\
&= \exp\left(-\frac{1}{2}(\bar{x} - \mu_m)^T U\Lambda^{-1/2}U^{-1}U\Lambda U^{-1}U\Lambda^{-1/2}U^{-1}(\bar{x} - \mu_m)\right) && (27)\\
&= \exp\left(-\frac{1}{2}(\bar{x} - \mu_m)^T U\Lambda^{-1}\Lambda U^{-1}(\bar{x} - \mu_m)\right) && (28)\\
&= \exp\left(-\frac{1}{2}(\bar{x} - \mu_m)^T (\bar{x} - \mu_m)\right) && (29)\\
&= \mathbf{B}(\bar{x}|\mu_m, 1) && (30)
\end{aligned}$$

which is what we wanted to prove. Note that multiplication of diagonal matrices is commutative, which was used to go from (27) to (28).

4 Problem 4

For this problem we consider a more general class of neural network where each elliptical Gaussian basis function is allowed its own Σ_m. Therefore, the basis functions are

$$\mathbf{B}(\bar{x}|\mu_m, \Sigma_m) = \exp\left(-\frac{1}{2}(\bar{x}-\mu_m)^T\Sigma_m(\bar{x}-\mu_m)\right). \tag{31}$$

We want to determine a condition on these symmetric matrices such that the basis functions only vary in one direction in the input space. I begin by making a linear transformation $x \to x - \mu_m$, to simplify the computation while preserving the generality of the problem. Note that the eigen-decomposition of a symmetric matrix can be written $\Sigma_m = U\Lambda U^T = \sum j = 1^m \lambda_j u_j u_j^T$, where λ_j and u_j are the individual eigenvalues and eigenvectors respectively.
I posit that the condition for each basis function to vary in only one direction is that is have only one non-zero eigenvalue (i.e. it is rank 1) and can therefore be written in the form $\Sigma_m = \lambda u u^T$, where λ is the only non-zero eigenvalue, and u is the corresponding eigenvector. In order for the basis function to only vary in one direction, we require that $\mathbf{B}(\bar{x}|\mu_m, \Sigma_m) = \mathbf{B}(\bar{x} + \bar{z}|\mu_m, \Sigma_m)$, for a vector $\bar{z}$ perpendicular to the eigenvector that Σ_m can be decomposed into. I look only at the exponential component, since the desired equation will hold if the exponents are equal.

$$\begin{aligned}(\bar{x}-\bar{z})^T\Sigma_m(\bar{x}-\bar{z}) &= \bar{x}^T\Sigma_m\bar{x} - 2\bar{x}^T\Sigma_m\bar{z} + \bar{z}^T\Sigma_m\bar{z} &(32)\\ &= \bar{x}^T\Sigma_m\bar{x} + (\bar{z}-2\bar{x})^T\lambda u u^T\bar{z} &(33)\\ &= \bar{x}^T\Sigma_m\bar{x} &(34)\end{aligned}$$

since $u^T\bar{z} = 0$. Therefore, we have shown that this condition is sufficient for the basis functions to only vary in one direction.

If we now assume that Σ_m has more than one non-zero eigenvalue, then, since the corresponding eigenvectors must be perpendicular, $\bar{z}$ cannot be perpendicular to both of them. We now write $\Sigma_m = \sum_{j=1}^2 \lambda_j u_j u_j^T$. Therefore, $\Sigma_m\bar{z} = \lambda_1 u_1 u_1^T\bar{z} + \lambda_2 u_2 u_2^T\bar{z} \neq 0$, since u_2 and u_1 cannot both be perpendicular to $\bar{z}$. Therefore, in this case, $\mathbf{B}(\bar{x}|\mu_m, \Sigma_m) \neq \mathbf{B}(\bar{x} + \bar{z}|\mu_m, \Sigma_m)$ for any $\bar{z}$, and the condition that each Σ_m have only one non-zero eigenvalue is also necessary for each basis function to vary in only one direction in the input space.

5 Problem 5

Now we consider a problem with a quadratic error criterion,

$$Q(\bar{w}) = Q_0 + \frac{1}{2}(\bar{w} - w^*)^T H(\bar{w} - w^*) \tag{35}$$

where w^* is the minimum and the constant H (Hessian) is positive-definite, and therefore also symmetric, so that all the properties of the eigen-decomposition of symmetric matrices previously cited will again be used in this problem. We also denote $\tilde{w} = argmin_{\bar{w}}\tilde{Q}(\bar{w})$, where $\tilde{Q}(\bar{w}) = Q(\bar{w}) + \frac{1}{2}\nu\bar{w}^T\bar{w}$. We are interested in determining an equation in terms of $\bar{w}^*$ for the component of $\tilde{w}$ parallel to eigenvector u_i of H. Recall that we can write $H = \sum_{i=1}^{n} \lambda_i u_i u_i^T$, since H is symmetric positive definite, where n is the number of non-zero eigenvalues of H. I also use the linearity of gradient operator.
We first take the gradient of the quadratic error loss with regularization parameter in order to solve for $\tilde{w}$ by setting it equal to zero:

$$\begin{aligned}
\nabla_{\bar{w}}\tilde{Q}(\bar{w}) &= \frac{1}{2}\nabla_{\bar{w}}(\bar{w}^T H\bar{w} - 2\bar{w}^T H\bar{w} + w^{*T} Hw^*) + \nu\tilde{w} && (36)\\
&= \frac{1}{2}\nabla_{\bar{w}}\left[\sum_{i=1}^{n} \lambda_i((\bar{w}^T u_i)^2 - 2(\bar{w}^T u_i)(w^{*T} u_i) + (w^{*T} u_i)^2)\right] + \nu\tilde{w} && (37)\\
&= \left[\sum_{j=i}^{n} \lambda_j(\tilde{w}^T u_i - w^{*T} u_i)u_i\right] + \nu\tilde{w} && (38)\\
&= 0. && (39)
\end{aligned}$$

In the above equations, any of the remaining $\bar{w}$ after the gradient was taken became $\tilde{w}$, since they will be the optimal values once solved for. In order to do so, we set up the following equation from (38) and (39):

$$\begin{aligned}
\sum_{i=1}^{n}(\lambda_i\tilde{w}^T u_i)u_i + \nu\tilde{w} &= \sum_{i=1}^{n}(\lambda_i w^{*T} u_i)u_i && (40)\\
\left[\sum_{i=1}^{n}(\lambda_i\tilde{w}^T u_i)u_i + \nu\tilde{w}\right]^T u_j &= \left[\sum_{i=1}^{n}(\lambda_i w^{*T} u_i)u_i\right]^T u_j && (41)\\
(\lambda_j + \nu)\tilde{w}^T u_j &= \lambda_j w^{*T} u_j && (42)\\
\tilde{w}^T u_j &= \frac{\lambda_j}{\lambda_j + \nu} w^{*T} u_j && (43)
\end{aligned}$$

Equation (43) is what we were trying to demonstrate. Note the most important step was in getting from (41) to (42). In (40) both sides of the equation were transposed and then multiplied on the right by u_j. This is valid because both sides of the equation, once transposed, are simply row vectors of the same dimension of u_j and $\tilde{w}$, given that what is multiplying them are just scalars. The property exploited in (41) to attain (42) is the orthonormality of the eigenvectors, i.e. $u_i^T u_j = \delta_{ij}$, where the right hand side is the Kronecker delta function, taking on the value of 1 if $i = j$ and 0 otherwise.

Result (43) implies that weight decay preferentially shrinks components of the weight vector that are pointing in the direction of smallest variation of $\nabla_{\bar{w}} Q(\bar{w})$. To show this, we note that along directions u_j corresponding to the smallest variation of the gradient, λ_j will be smallest. This is because H is the gradient of the gradient, and the largest eigenvalues correspond to the eigenvectors that point in the direction of greatest gain (variation), and the smallest correspond to those pointing in the direction of smallest variation. Let's assume u_j is the direction of least variation of the gradient, and therefore λ_j is the smallest eigenvalue of H. Therefore, the components of $\bar{w}$ pointing in the direction of u_j, after the next update, will be

$$\tilde{w}^T u_j = \frac{\lambda_j}{\lambda_j + \nu} w^{*T} u_j \tag{44}$$

$$= \frac{1}{1 + \nu/\lambda_j} w^{*T} u_j. \tag{45}$$

If we consider another direction of rapid gradient variation and the associated eigenvalue $\lambda_k > \lambda_j$, then $\frac{1}{1+\nu/\lambda_k} > \frac{1}{1+\nu/\lambda_j}$, which confirms the relative variation of the weight vector component along different directions.

6 Problem 6

In this problem we again consider the quadratic error function $Q(\bar{w})$. We take the initial weight vector $\bar{w}_0$ to be centered at the origin (i.e. equal to 0). We use the standard gradient descent update formula

$$\bar{w}_t = \bar{w}_{t-1} - \eta \bar{G}_{t-1} \tag{46}$$

where η is a small, positive "learning rate". We want to show that after t time steps, the components of the weight vector parallel to the eigenvectors of H can be written

$$w_t^T u_j = [1 - (1 - \eta\lambda_j)^t] w^{*T} u_j. \tag{47}$$

We proceed by induction. We use the formula for the gradient of $Q(\bar{w})$ obtained in Problem 5, except in this case we are evaluating the gradient at the value of the weights for the previous time step:

$$\bar{G}_{t-1} = \nabla_{\bar{w}} Q(\bar{w}_{t-1}) \tag{48}$$

$$= \left[\sum_{i=1}^{n} \lambda_i (\bar{w}_{t-1}^T u_i - w_{t-1}^{*T} u_i) \right] u_i \tag{49}$$

We are now ready to prove equation (47). For the base case, we select $t = 0$. It is trivially shown to hold true:

$$w_0^T u_j = 0 \tag{50}$$

$$[1 - (1 - \eta\lambda_j)^0] w^{*T} u_j = 0 \tag{51}$$

Therefore equation (47) holds for the $t = 0$ case, which we consider the base case. We now assume that it holds for the $t = n$ case, and show that this implies that it also holds for the $t = n + 1$ case.

$$w_{n+1}^T u_j = w_n^T u_j - \eta \bar{G}_n^T u_j \tag{52}$$

$$= [1 - (1 - \eta\lambda_j)^n] w^{*T} u_j - \eta\lambda_j (\bar{w}_n^T u_j - w^{*T} u_j) \tag{53}$$

$$= [1 - (1 - \eta\lambda_j)^n] w^{*T} u_j - \eta\lambda_j [1 - (1 - \eta\lambda_j)^n] w^{*T} u_j + \eta\lambda_j w^{*T} u_j \tag{54}$$

$$= [1 - (1 - \eta\lambda_j)^n] w^{*T} u_j + \eta\lambda_j (1 - \eta\lambda_j)^n w^{*T} u_j \tag{55}$$

$$= [1 - (1 - \eta\lambda_j)(1 - \eta\lambda_j)^n] w^{*T} u_j \tag{56}$$

$$= [1 - (1 - \eta\lambda_j)^{n+1}] w^{*T} u_j \tag{57}$$

By equation (57) we have completed the inductive step, and therefore have shown that equation (47) holds for all time steps t. Note that to get from (52) to (53) we have used the orthonormality of the eigenvectors u_j as in problem 5, to pick out only one component in the sum, as follows:

$$\left[\sum_{i=1}^{n} \lambda_i (\bar{w}_n^T u_i - w^{*T} u_i) \right] u_i^T u_j = \lambda_j (\bar{w}_n^T u_j - w^{*T} u_j) \tag{58}$$

We now want to show that a few interesting relationships hold in different limits. First of all, as $t \to \infty$,

$$w_t^T u_j = [1 - (1 - \eta\lambda_j)^t] w^{*T} u_j \tag{59}$$

$$\approx [1 - 0] w^{*T} u_j \tag{60}$$

$$= w^{*T} u_j \tag{61}$$

since we are told that $|1 - \eta\lambda_j| < 1$, and $\lim_{t\to\infty} x^t = 0$ for all $|x| < 1$.

Next we suppose that training stops after a finite number of steps t.
We first consider the case in which $t \gg 1/(\eta\lambda_j)$. In this case, since t is quite large (which it must be since $\eta\lambda_j$ must be smaller than 2), and $|1 - \eta\lambda_j| < 1$, we have that $(1 - \eta\lambda_j)^t \approx 0$, and therefore, $w_t^T u_j \approx w^{*T} u_j$.
In the case that $t \ll \eta\lambda_j$, we have that $\eta\lambda_j$ is very small, much less than 1, since we have taken some finite number of steps t. Therefore, $(1 - \eta\lambda_j)^t \approx 1^t = 1$. In this case we obtain that $w_t^T u_j \ll w^{*T} u_j$, which are the limiting cases we wanted to demonstrate.

7 Problem 7

In this problem we are looking at how well a single-layer neural network can classify spam and non-spam e-mails, both with and without weight decay. We then look at prioritizing accurate classification of non-spam e-mails, and finally at another criterion for determining an optimal number of iterations for the algorithm, so as not to over fit the data. Before we begin, we standardize all of the data so that each of the data features has mean = 0 and standard deviation = 1. We use the already divided data for training and testing accordingly, since we want to make predictions on data that was not used to generate the model. For all parts of the problem, we have the prediction output be the raw probability that it is a spam e-mail, and then decide on a threshold probability for classification on our own (the standard one being 0.5).

7.1 Part (a)

For this part, we want to determine the optimal number of units in our single-hidden layer neural network. For each of the cases of a neural network with 1-10 units in the hidden layer, we train 10 neural networks with different random starting weights with values in the range $-0.5 \leq w_i \leq 0.5$ for all i, and average their testing error when using them to predict from new data. For this part as well, we output the raw prediction probability, and use a threshold of 0.5 (%50) to discern whether something was classified as spam or

not, where a predicted value greater than 0.5 is spam in this case. Figure (1) shows the average testing error values obtained over the 10 differently initialized cases for each of the 1-10 hidden unit cases. Table 1 contains the actual average testing error values. At this point we have not yet added weight decay to our model.

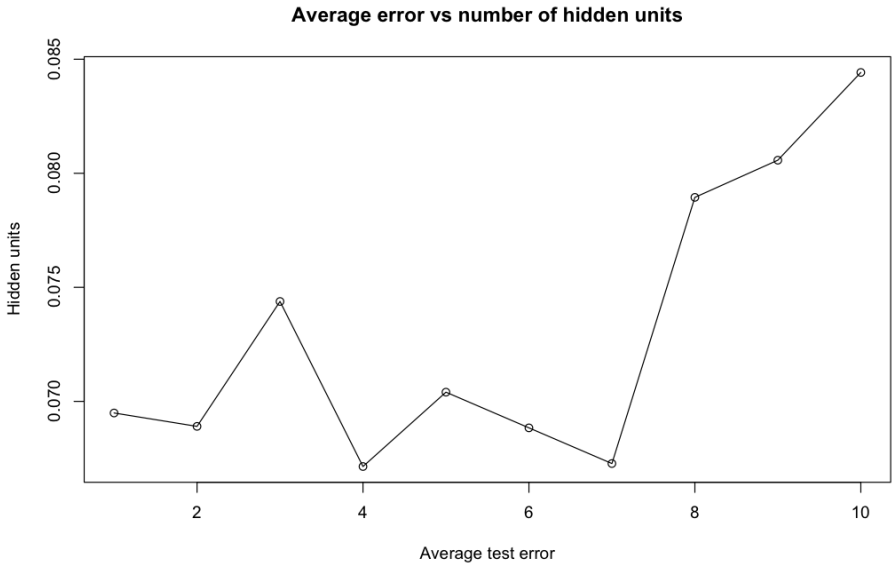

Figure 1: Results for part (a)

Table 1: Actual results from part (a)

Hidden units	1	2	3	4	5	6	7	8	9	10
Av. test err. (%)	6.94	6.89	7.43	6.71	7.04	6.88	6.72	7.89	8.05	8.44

From the plot we can see that the neural-network actually performs fairly well even with 1 hidden unit, and overall, performs very well for 1 - 7 hidden units (with the exception of 3, though this is likely due to the fact that fitting a neural network is a high variance process). It seems that the neural networks with 8 or more hidden units didn't perform as well, perhaps because the spam-classification problem doesn't require that much flexibility, and it may start to over fit with too many hidden units.

From this exercise, I found my optimal number of hidden units to be $N_h = \mathbf{4}$, with the average testing error for neural networks with that number of units being $Av.Test.Err = \mathbf{0.0671}$.

7.2 Part (b)

For this part of the problem, we want to add weight decay to our single layer neural network, and using a neural network with the optimal number of hidden units found in (a), we want to determine the optimal "regularization parameter." We approach this problem in the same manner as in (a), except that now we iterate over "weight parameters" in the range of 0, 0.1, ..., 1. For each case, we average the testing error over 10 trials as before, and then select the parameter which gave the smallest average testing error. So , overall, we train and test using 110 different neural networks. Figure 2 shows the average testing error against the regularization parameter, and Table 2 shows the actual average testing errors obtained for each parameter.

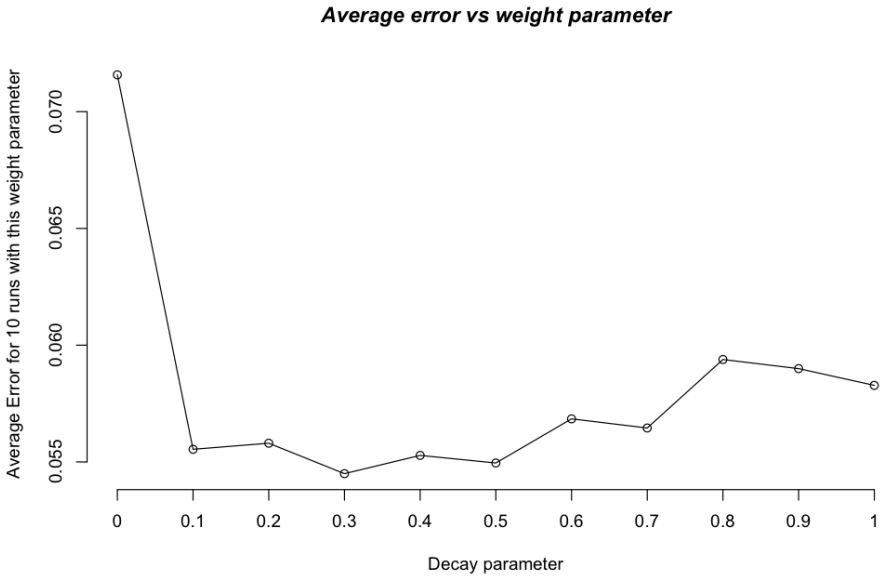

Figure 2: Results for part (b)

Table 2: Actual results from part (b)

Param.	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Av. err. (%)	7.16	5.55	5.58	5.45	5.53	5.50	5.68	5.65	5.94	5.90	5.83

From the results of this part we note that the use of weight decay greatly improves the average testing error. From Figure 2 we see that even with $\nu = 0.1$ (decay parameter), the testing error significantly decreases (by about 0.015). Up until about $\nu = 0.7$ the error

stays about the same, and then it begins to increase gradually. Part of this could just be due to the high variance of the procedure.

The optimal decay parameter obtained by this method is $\nu^* = \mathbf{0.3}$, for which $Av.Test.Err = \mathbf{0.0545}$.

7.3 Part (c)

Our next goal was to actually make our neural network a good spam-filter, which means that we do not want it to misclassify non-spam e-mails, since one does not usually go sifting through one's spam to see if there is anything good in it. To achieve this, I iterated over various possible threshold values for determining whether something is spam or not, and averaged the false positive error, false negative error, and total error for 10 different neural networks for each threshold value P_{th}. All of the neural networks were trained with 4 hidden units and decay parameter = 0.3. The optimal spam filter was selected to be the neural network that misclassified less than 1% of non-spam e-mails. Table 3 shows the data collected for a few different choices of P_{th}, which are all closer to 1 than 0, since we want the neural network to be very confident that something is spam before we classify it as such. Note that, in order to collect results from only the displayed range, I first tested a few threshold probabilities in the lower range and noticed that they don't give small enough non-spam misclassification error. This also makes sense intuitively.

Table 3: This table displays the total, spam, and non-spam, misclassification errors for different threshold probability values, averaged over 10 trials in each case

Threshold	0.7	0.75	0.8	0.85	0.9	0.95
Spam error (%)	11.0	12.5	14.7	17.8	21.5	28.6
Non-Spam error (%)	2.47	2.22	2.07	1.52	1.51	**0.731**
Total error (%)	7.16	5.55	5.58	5.45	5.53	5.50

Note from Table 3 that the non-spam misclassification rate drops to %0.731 for $P_{th} = 0.95$. However, I decided to take 0.01 size steps to see what occurs from 0.9 ¡ P_{th} ¡ 0.95, to see exactly when the non-spam misclassification becomes %1. Doing so, I determined that it occurs at for a threshold probability of $P^*_{th} = \mathbf{0.92}$, for which $Spam - error = \mathbf{0.24}$, $Non - Spam - error = \mathbf{0.0100}$, and $Total - error = \mathbf{0.103}$.

7.4 Bonus

For this part we determine the best early stopping rule for a 57-3-1 neural network that doesn't use weight decay. The way we did this was by running 10 iterations at a time for 10 different neural networks with different starting weights, and obtaining the average testing error after each 10 successive iterations. After every set of iterations, the output weights were passed in as inputs to 10 new neural networks to be trained for 10 iterations and then tested. This process continued until the average testing error for all the neural networks had not improved for 50 iterations. This is a criterion that can be used to determine when to stop training the neural network, though it requires passing in output weights as inputs. Figure 3 is a plot of the average test error versus iteration number. From the figure we can see the sharp increase in accuracy during the first 50 iterations, and then the gradual convergence towards an optimum. According to this plot, the optimal early stopping criterion is **170** *iterations*, since the error ceases to decrease for the following 50 iterations. For that number of iterations, the $Av.Test.Error = \mathbf{0.0602}$.

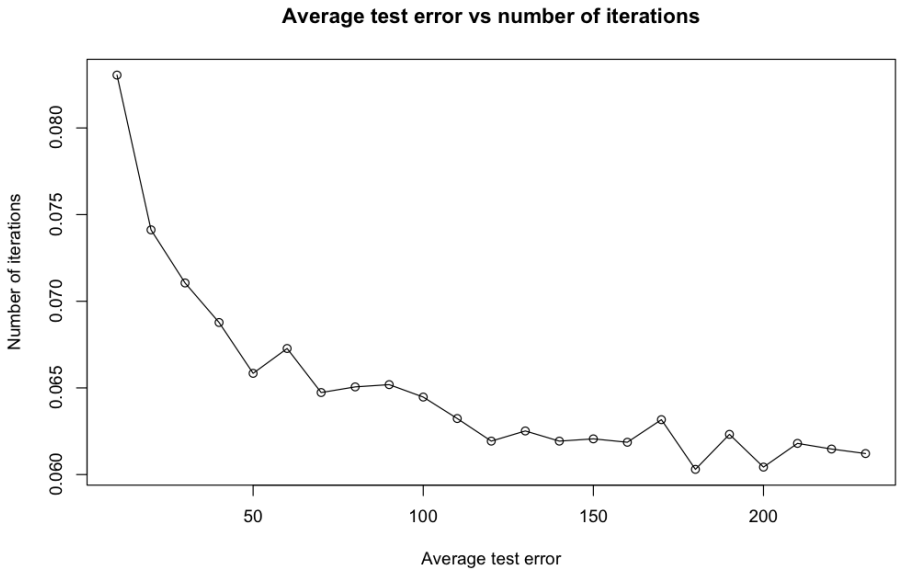

Figure 3: Results for Bonus part