Solutions to Take-Home Quiz Two

ECE 271B - Winter 2022

Electrical and Computer Engineering
University of California San Diego

1.

a) The function $f(\mathbf{x})$ is the *XOR*

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

As can be seen from the figure below, where $\times$'s represent $f(\mathbf{x}) = 0$ and $\circ$'s represent $f(\mathbf{x}) = 1$, there is no line that can place the two $\times$'s on one side and the two $\circ$'s on the other.

$$\begin{array}{cc} \times & \circ \\ \circ & \times \end{array}$$

The best that we can do is to have one error. Given that all configurations of $\mathbf{X}$ have the same probability and there are four of them, the probability of each configuration is 0.25. This is, therefore, the minimum probability of error.

b) Let

$$\mathbf{z} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2.$$

Then, if $\mathbf{x}_1, \mathbf{x_2} \in \mathcal{R}_i$,

$$\begin{aligned} g_i(\mathbf{z}) &= \mathbf{w}_i^T \mathbf{z} + b_i \\ &= \lambda(\mathbf{w}_i^T \mathbf{x}_1 + b_i) + (1 - \lambda)(\mathbf{w}_i^T \mathbf{x}_2 + b_i) \\ &= \lambda g_i(\mathbf{x}_1) + (1 - \lambda) g_i(\mathbf{x}_2) \\ &\geq \lambda g_j(\mathbf{x}_1) + (1 - \lambda) g_j(\mathbf{x}_2),\ \forall j \neq i \\ &= \lambda(\mathbf{w}_j^T \mathbf{x}_1 + b_j) + (1 - \lambda)(\mathbf{w}_j^T \mathbf{x}_2 + b_j),\ \forall j \neq i \\ &= \mathbf{w}_j^T \mathbf{z} + b_j,\ \forall j \neq i \\ &= g_j(\mathbf{z}),\ \forall j \neq i \end{aligned}$$

from which $\mathbf{z} \in \mathcal{R}_i$.

2.

a) The Lagrangian for this problem is

$$L = ||\mathbf{x} - \mathbf{x}_a||^2 - \lambda(\mathbf{w}^T\mathbf{x} + b).$$

Setting its derivatives to zero, we obtain

$$\frac{\partial L}{\partial \mathbf{x}} = 0 \;\Leftrightarrow\; 2(\mathbf{x} - \mathbf{x}_a) = \lambda \mathbf{w}$$

and, multiplying by $\mathbf{w}^T$ on both sides, we get

$$\lambda = \frac{2}{||\mathbf{w}||^2}\mathbf{w}^T(\mathbf{x} - \mathbf{x}_a) = \frac{2}{||\mathbf{w}||^2}(g(\mathbf{x}) - g(\mathbf{x}_a))$$

and, using the constraint $g(\mathbf{x}) = 0$,

$$\lambda = -\frac{2}{||\mathbf{w}||^2}g(\mathbf{x}_a).$$

It follows that

$$\mathbf{x} - \mathbf{x}_a = -\frac{g(\mathbf{x}_a)}{||\mathbf{w}||^2}\mathbf{w}$$

and, therefore,

$$||\mathbf{x} - \mathbf{x}_a|| = \frac{|g(\mathbf{x}_a)|}{||\mathbf{w}||}.$$

b) This follows trivially from **a)**, where we have seen that

$$\mathbf{x} = \mathbf{x}_a - \frac{g(\mathbf{x}_a)}{||\mathbf{w}||^2}\mathbf{w}.$$

3.

a) To compute the optimal solution, we set the derivatives of the cost to zero

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{w}} &= -\sum_i 2(y_i - \mathbf{x}_i^T\mathbf{w} - b)\mathbf{x}_i = 0\\ \frac{\partial L}{\partial b} &= -\sum_i 2(y_i - \mathbf{x}_i^T\mathbf{w} - b) = 0\end{aligned}$$

which leads to

$$\begin{aligned}\left(\sum_i \mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{w} + b\sum_i \mathbf{x}_i &= \sum_i y_i\mathbf{x}_i\\ \left(\sum_i \mathbf{x}_i^T\right)\mathbf{w} + nb &= \sum_i y_i\end{aligned}$$

or

$$\mathbf{X}^T\mathbf{X}\mathbf{a} = \mathbf{X}^T\mathbf{y},$$

where

$$\mathbf{a} = (\mathbf{w}, b)^T \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T & 1\\ \mathbf{x}_2^T & 1\\ \vdots & \vdots\\ \mathbf{x}_n^T & 1\end{bmatrix} \qquad \mathbf{y} = (y_1, \dots, y_n)^T.$$

Hence the solution is

$$\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and, computing second-order derivatives, one can check that this solution is indeed a minimum.

b) Stochastic gradient descent algorithms take a step in the direction of the negative gradient for each pattern. From the gradient equations above, it is clear that the step should be

$$\begin{aligned}\mathbf{w}^{k+1} &= \mathbf{w}^k + \eta(y_i - \mathbf{x}_i^T\mathbf{w}^k - b)\mathbf{x}_i\\ b^{k+1} &= b^k + \eta(y_i - \mathbf{x}_i^T\mathbf{w}^k - b).\end{aligned}$$

Unlike the Perceptron, we do not have the notion of “correctly or incorrectly” classified point, so we always have to iterate through all the $\mathbf{x}_i$. The algorithm is, therefore,

```
• set $\mathbf{w}^0 = \mathbf{0}, b^0 = 0, k = 0$
• repeat
    – for $i = 1, \dots, n$
        ∗ set $\mathbf{w}^{k+1} = \mathbf{w}^k + \eta(y_i - \mathbf{x}_i^T\mathbf{w}^k - b^k)\mathbf{x}_i$
        ∗ set $b^{k+1} = b^k + \eta(y_i - \mathbf{x}_i^T\mathbf{w}^k - b^k)$
        ∗ set $k = k + 1$
• until the error $L(\mathbf{w}^k, b^k)$ is less than a threshold.
```

c) Once again we start by setting gradients to zero

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}} &= -\sum_i 2(y_i - \mathbf{x}_i^T\mathbf{w} - b)\mathbf{x}_i + 2\lambda\mathbf{w} = 0 \\
\frac{\partial L}{\partial b} &= -\sum_i 2(y_i - \mathbf{x}_i^T\mathbf{w} - b) = 0.
\end{aligned}$$

From the first equation, we have

$$\mathbf{w} = \frac{1}{\lambda}\sum_i (y_i - \mathbf{x}_i^T\mathbf{w} - b)\mathbf{x}_i$$

which shows that the solution must be of the type

$$a = \sum_i \alpha_i \mathbf{x}_i,$$

with

$$\alpha_i = \frac{1}{\lambda}(y_i - \mathbf{x}_i^T\mathbf{w} - b).$$

Plugging this back into $L(\mathbf{w}, b)$, we get

$$L(\alpha_i) = \sum_i (y_i - \sum_j \alpha_j \mathbf{x}_j^T\mathbf{x}_i - b)^2 + \lambda \sum_{ij} \alpha_i\alpha_j \mathbf{x}_i^T\mathbf{x}_j$$

which can be written as

$$\begin{aligned}
L(\boldsymbol{\alpha}) &= \sum_i (y_i - (\mathbf{K}\boldsymbol{\alpha})_i - b)^2 + \lambda\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} \\
&= ||\mathbf{y} - \mathbf{K}\boldsymbol{\alpha} - b||^2 + \lambda\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha}
\end{aligned}$$

by defining the matrix $\mathbf{K}_{ij} = \mathbf{x}_i^T\mathbf{x}_j$ and vector $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^T$. Setting the gradient with respect to α to zero, we obtain

$$\begin{aligned}
\lambda\mathbf{K}\boldsymbol{\alpha} &= \mathbf{K}(\mathbf{y} - \mathbf{K}\boldsymbol{\alpha} - b) \\
\mathbf{K}(\mathbf{K} + \lambda\mathbf{I})\boldsymbol{\alpha} &= \mathbf{K}(\mathbf{y} - b) \\
\boldsymbol{\alpha} &= (\mathbf{K} + \lambda\mathbf{I})^{-1}(\mathbf{y} - b).
\end{aligned}$$

This leads to

$$\mathbf{w}^* = \mathbf{X}\boldsymbol{\alpha} = \mathbf{X}(\mathbf{K} + \lambda\mathbf{I})^{-1}(\mathbf{y} - b),$$

where $\mathbf{X}$ has the $\mathbf{x}_i$ as columns and

$$f(\mathbf{x}) = (\mathbf{y} - b)^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{x} + b.$$

Note that $\mathbf{K}$ only depends on the dot-products of training points and $\mathbf{X}^T\mathbf{x}$ on the dot-products of the $\mathbf{x}_i$ with $\mathbf{x}$.

4.

a) Note that, at any point in the algorithm, we can express $\mathbf{w}$ and b as

$$\begin{aligned}\mathbf{w} &= \eta \sum_j y_j \mathbf{x}_j \\ b &= \eta \sum_j y_j R^2\end{aligned}$$

and the only contribution of η is to rescale the vector $(\mathbf{w}, b)$. However, both the *if* statement of the Perceptron algorithm and the decision function only depend on the sign of $\mathbf{w}^T\mathbf{x} + b$. Since different η's only scale this quantity, they do not change the sign and therefore do not really change anything. Hence, η is irrelevant.

b) Above, we saw that $\mathbf{w}$ is of the form

$$\mathbf{w} = \sum_{j \in \mathcal{J}} y_j \mathbf{x}_j.$$

We now note that $\mathcal{J}$ is the set of all indices j such that $\mathbf{x}_j$ satisfied the *if* statement (i.e. was misclassified when the iteration took place). Note that the indices cycle forever, so there may be various terms containing the same $\mathbf{x}_j$. We can introduce a variable α_i that is incremented every time the *if* statement is true for pattern $\mathbf{x}_i$ and then rewrite

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i.$$

The condition in the *if* statement then becomes

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \le 0 \Leftrightarrow y_i \left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b \right) \le 0.$$

This leads to the equivalent algorithm, which is known as the *dual* form of the Perceptron.

c) We have already seen that α_i is the number of times the sample point $\mathbf{x}_i$ satisfies the *if* statement, i.e. the number of iterations in which it was misclassified. Therefore, the hardest samples are the ones that have largest α_i.

d) The reason why this is interesting is that it makes the algorithm immediately "kernalizable." If we want to apply the algorithm in a high-dimensional feature space, where the dot-product can be implemented as a kernel function $K(\mathbf{x}, \mathbf{y})$, we simply need to replace, in the dual algorithm, all the $\mathbf{x}_i^T\mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j)$. The decision function can also be written as

$$h(\mathbf{x}) = sgn\left(\sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + R^2 \sum_i \alpha_i y_i \right)$$

and "kernalized" in the same way, i.e. by replacing $\mathbf{x}_i^T\mathbf{x}$ with $K(\mathbf{x}_i, \mathbf{x})$.

5 a)

i)

$$\begin{aligned}
-\frac{\partial E^n(\mathbf{w})}{\partial w_{jk}} &= -\sum_{i=1}^{c} \frac{\partial E^n(\mathbf{w})}{\partial y_i} \frac{\partial y_i}{\partial a_k} \frac{\partial a_k}{\partial w_{jk}} \\
&= \left[\frac{t_k^n}{y_k^n}(1-y_k^n)y_k^n + \sum_{i \neq k} t_i^n(-y_k^n) \right] x_j^n \\
&= \left(t_k^n - y_k^n \sum_i t_i^n \right) x_j^n \\
&= (t_k^n - y_k^n) x_j^n
\end{aligned}$$

ii) The plot of training and test error is as follows. Note that both train and test error decrease with the iteration, showing that there is no overfitting.

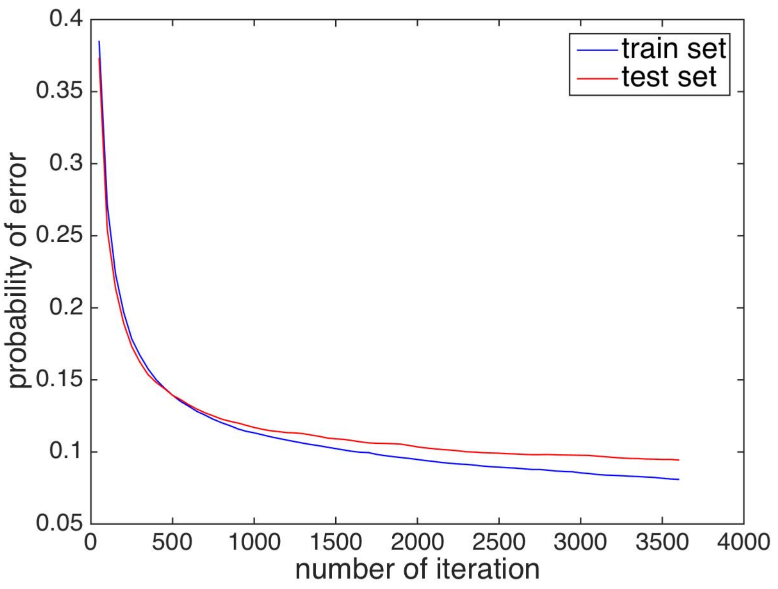


Figure 1: Probabilty of error as a function of backpropagation iteration.

The following table summarizes the final errors.

	training set	test set
probability of error	8.25%	9.83%

5 b)

i) To solve this problem, we use the following notation

- x_i - i^{th} input
- w^1_{ji} - weight between input i and hidden unit j
- $g_j = \sum_l w^1_{jl} x_l$ - input of non-linearity of hidden unit j
- $y_j = \sigma[g_j]$ - output of non-linearity of hidden unit j
- w^2_{kj} - weight between output of hidden unit j and output unit k
- $u_k = \sum_l w^2_{kl} y_l$ - k^{th} softmax input
- $z_k = e^{u_k} / \sum_j e^{u_j}$ - k^{th} softmax output
- $E = \sum_l t_l \ln z_l$ cost of the example of output z.

We also denote the derivative of the cost with respect to the inputs of the non-linearities as

$$\delta^2_k = \frac{\partial E}{\partial u_k} \qquad \delta^1_j = \frac{\partial E}{\partial g_j}. \tag{1}$$

We can then obtain the derivatives as follows

$$\begin{aligned}
-\frac{\partial E^{(n)}}{\partial w^2_{kj}} &= -\frac{\partial E^{(n)}}{\partial u^{(n)}_k}\frac{\partial u^{(n)}_k}{\partial w^2_{kj}} = -[\delta^2_k]\frac{\partial}{\partial w^2_{kj}}\sum_l w^2_{kl} y^{(n)}_l = -[\delta^2_k]^{(n)} y^{(n)}_j \\
-\frac{\partial E^{(n)}}{\partial w^1_{ji}} &= -\frac{\partial E^{(n)}}{\partial g^{(n)}_j}\frac{\partial g^{(n)}_j}{\partial w^1_{ji}} = -[\delta^1_j]^{(n)}\frac{\partial}{\partial w^1_{ji}}\sum_l w^1_{jl} x^{(n)}_l = -[\delta^1_j]^{(n)} x^{(n)}_i.
\end{aligned}$$

To compute the δs, we note that

$$\begin{aligned}
-\frac{\partial E^{(n)}}{\partial w^1_{ji}} &= -\sum_k \frac{\partial E^{(n)}}{\partial u^{(n)}_k}\frac{\partial u^{(n)}_k}{\partial g^{(n)}_j}\frac{\partial g^{(n)}_j}{\partial w^1_{ji}} = -\sum_k [\delta^2_k]^{(n)}\frac{\partial u^{(n)}_k}{\partial y^{(n)}_j}\frac{\partial y^{(n)}_j}{\partial g^{(n)}_j}\frac{\partial}{\partial w^1_{ji}}\sum_l w^1_{jl} x^{(n)}_l \\
&= -\sum_k [\delta^2_k]^{(n)} w^2_{kj} \sigma'[g^{(n)}_j] x^{(n)}_i,
\end{aligned}$$

from which

$$[\delta^1_j]^{(n)} = \sigma'[g^{(n)}_j]\sum_k w^2_{kj}[\delta^2_k]^{(n)}.$$

Similarly,

$$\begin{aligned}
[\delta^2_k]^{(n)} &= \frac{\partial E^{(n)}}{\partial u^{(n)}_k} = \frac{\partial}{\partial u^{(n)}_k}\sum_l t^{(n)}_l \ln\frac{e^{u^{(n)}_l}}{\sum_j e^{u^{(n)}_j}} = \frac{\partial}{\partial u^{(n)}_k} t^{(n)}_k \ln\frac{e^{u^{(n)}_k}}{\sum_j e^{u^{(n)}_j}} + \frac{\partial}{\partial u^{(n)}_k}\sum_{l\neq k} t^{(n)}_l \ln\frac{e^{u^{(n)}_l}}{\sum_j e^{u^{(n)}_j}} \\
&= \frac{t^{(n)}_k}{z^{(n)}_k}\frac{e^{u^{(n)}_k}\sum_j e^{u^{(n)}_j} - e^{2u^{(n)}_k}}{[\sum_j e^{u^{(n)}_j}]^2} - \sum_{l\neq k} t^{(n)}_l \frac{\partial}{\partial u^{(n)}_k}\ln\sum_j e^{u^{(n)}_j} \\
&= t^{(n)}_k \frac{\sum_j e^{u^{(n)}_j} - e^{u^{(n)}_k}}{\sum_j e^{u^{(n)}_j}} - \sum_{l\neq k} t^{(n)}_l z^{(n)}_k = t^{(n)}_k(1 - z^{(n)}_k) - \sum_{l\neq k} t^{(n)}_l z^{(n)}_k \\
&= t^{(n)}_k(1 - z^{(n)}_k) - (1 - t^{(n)}_k) z^{(n)}_k = t^{(n)}_k - z^{(n)}_k.
\end{aligned}$$

In summary, in the forward step, we compute

$$\begin{aligned}
g_j^{(n)} &= \sum_l w_{jl}^1 x_l^{(n)} \\
y_j^{(n)} &= \sigma[g_j^{(n)}] \\
u_k^{(n)} &= \sum_l w_{kl}^2 y_l^{(n)} \\
z_k^{(n)} &= e^{u_k^{(n)}} / \sum_j e^{u_j^{(n)}}
\end{aligned}$$

to compute the network output for each example $x^{(n)}$. In the backprop step, we then compute

$$\begin{aligned}
[\delta_k^2]^{(n)} &= t_k^{(n)} - z_k^{(n)} \\
[\delta_j^1]^{(n)} &= \sigma'[g_j^{(n)}] \sum_k w_{kj}^2 [\delta_k^2]^{(n)}.
\end{aligned}$$

The derivatives are finally

$$\begin{aligned}
-\frac{\partial E^{(n)}}{\partial w_{kj}^2} &= -[\delta_k^2]^{(n)} y_j^{(n)} \\
-\frac{\partial E^{(n)}}{\partial w_{ji}^1} &= -[\delta_j^1]^{(n)} x_i^{(n)}.
\end{aligned}$$

Note that these equations can be easily written in matrix form as

$$\begin{aligned}
-\nabla_{\mathbf{W}^2} E^{(n)} &= -[\delta^2]^{(n)} [y^{(n)}]^T \\
-\nabla_{\mathbf{W}^1} E^{(n)} &= -[\delta^1]^{(n)} [x^{(n)}]^T,
\end{aligned}$$

where

$$\begin{aligned}
[\delta^2]^{(n)} &= t^{(n)} - z^{(n)} \\
[\delta^1]^{(n)} &= \sigma'[g^{(n)}] \circ [\mathbf{W}^2]^T [\delta^2]^{(n)},
\end{aligned}$$

and $\circ$ means the element-wise multiplications (the operator `.*` in Matlab), and

$$\begin{aligned}
g^{(n)} &= \mathbf{W}^1 x^{(n)} \\
u^{(n)} &= \mathbf{W}^2 \sigma[g^{(n)}] \\
z^{(n)} &= softmax(u^{(n)}).
\end{aligned}$$

Hence, the backprop algorithm can be implemented by iterating over 7 equations. The pattern of the equation generalizes to any number of layers, allowing the easy derivation and implementation of the backprop algorithm. These are two reasons that have made it so popular.

ii) The plots of training and test error of the three networks are as follows.

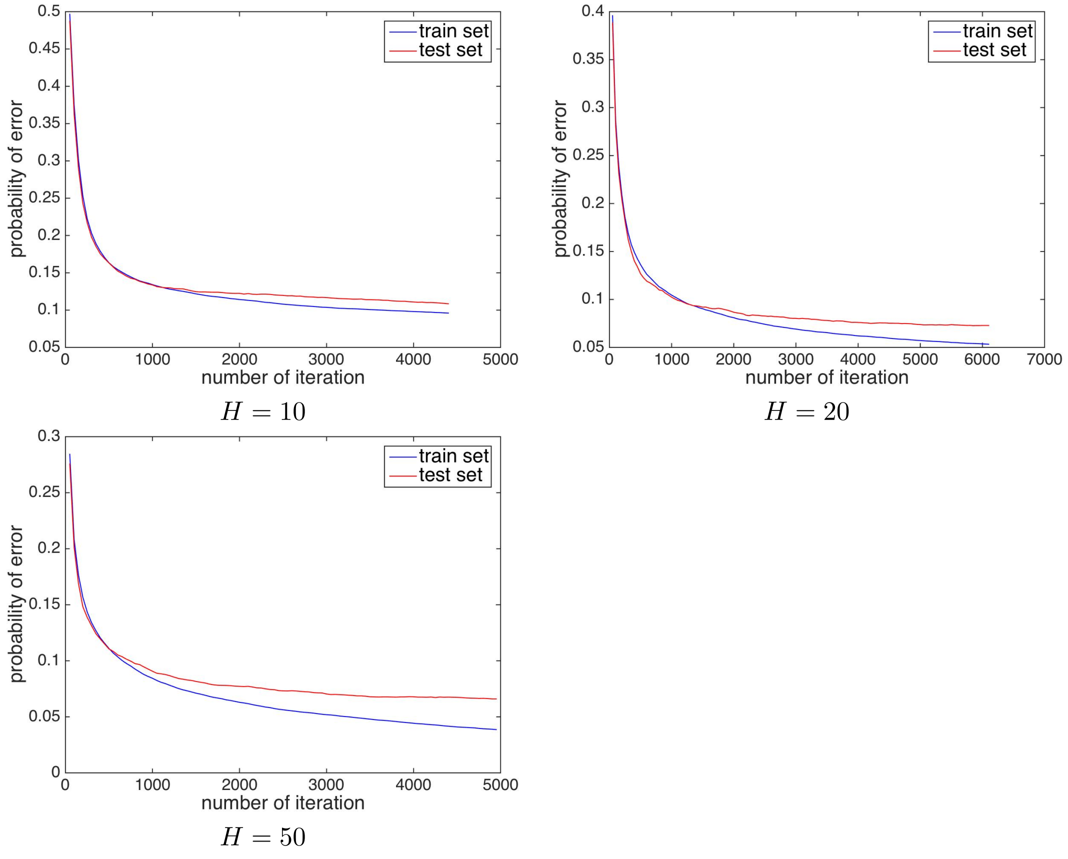
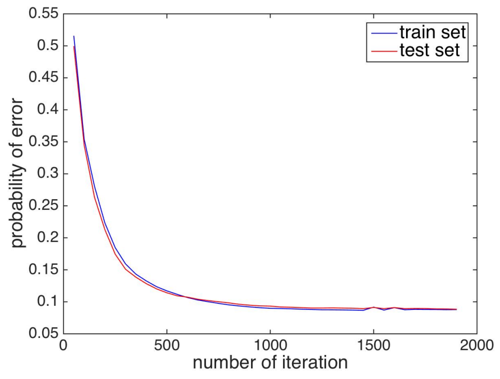


Figure 2: Probabilty of error as a function of backpropagation iteration for the three networks.

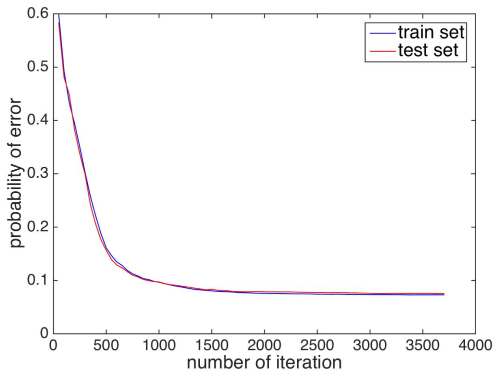
The following table summarizes the final errors. Note that the error is always smaller than that of the single layer networks. Furthermore, increasing the number of hidden units seems to always improve performance. However, if we look at the iteration plots, we can see that the train and test error curves start to diverge. This is a sign that the network is about to overfit. Adding more hidden units would likely make this happen.

	10 units	20 units	50 units
probability of error	8.76%	6.11%	5.68%

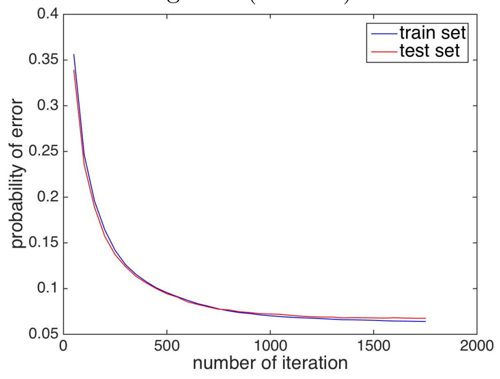
iii) The plots of training and test error of the three networks and two non-linearities are as follows.



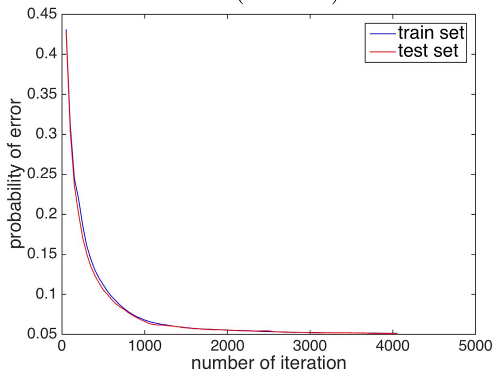
sigmoid ($H = 10$)



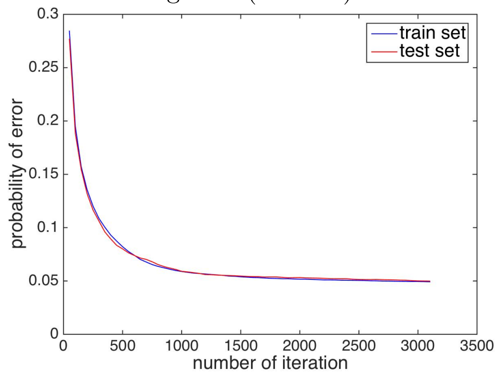
ReLU ($H = 10$)



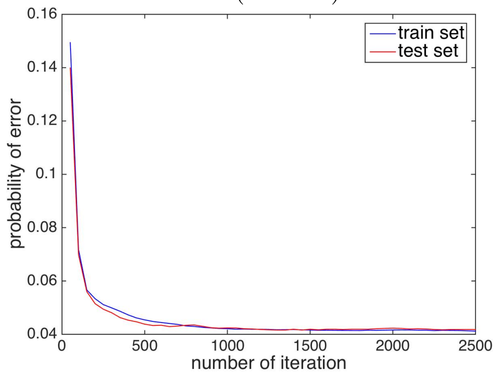
sigmoid ($H = 20$)



ReLU ($H = 20$)



sigmoid ($H = 50$)

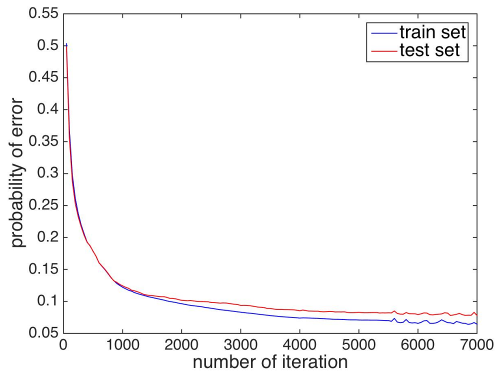


ReLU ($H = 50$)

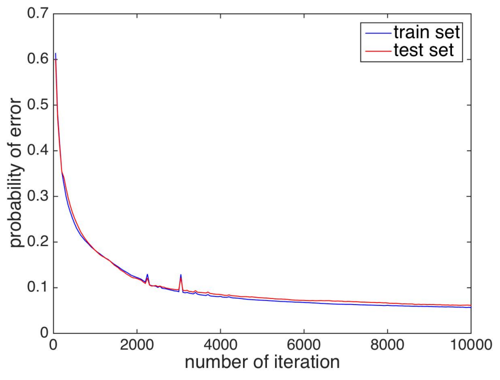
The table summarizes the final errors. As before, more hidden units lead to less error. The ReLU network achieves lower error and converges faster.

	10 units	20 units	50 units
sigmoid	8.61%	6.48%	5.05%
ReLU	7.21%	5.02%	4.51%

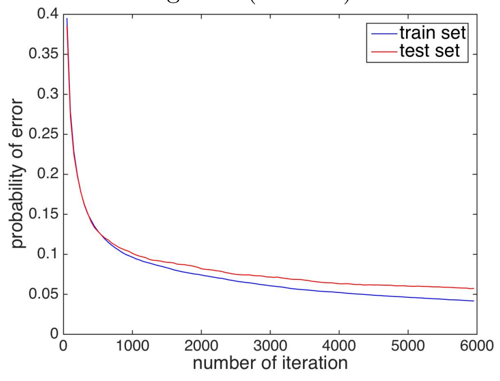
iv) The plots of training and test error of the three networks and two non-linearities are as follows.



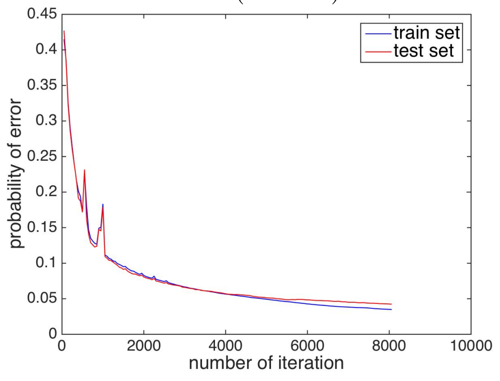
sigmoid ($H = 10$)



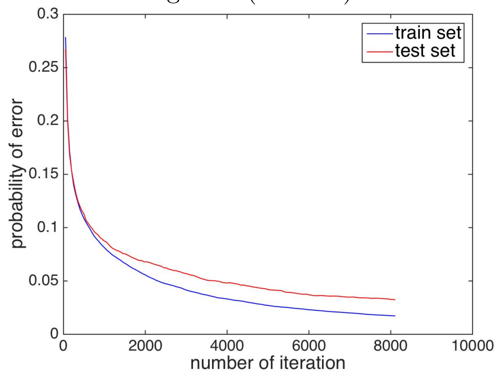
ReLU ($H = 10$)



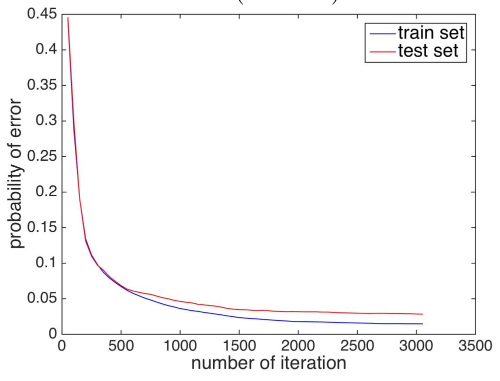
sigmoid ($H = 20$)



ReLU ($H = 20$)



sigmoid ($H = 50$)



ReLU ($H = 50$)

The table summarizes the final errors. As before, more hidden units lead to less error and the ReLU network achieves lower error. The results with this regularization strength are the best. This shows that regularization matters, but we have to get the regularization strength right. This is a common observation in machine learning.

	10 units	20 units	50 units
sigmoid	7.67%	5.74%	3.26%
Relu	6.38%	3.66%	2.98%

5. c)

The table and plot of training and test error for the single layer is as follows. Note that the performance of the network is the same as for batch learning, but learning requires much less computation. Before, we needed thousands of iterations over the entire training set, now we only need a few (1 epoch is one pass through the entire training set).

	training set	test set
probability of error	8.87%	9.21%

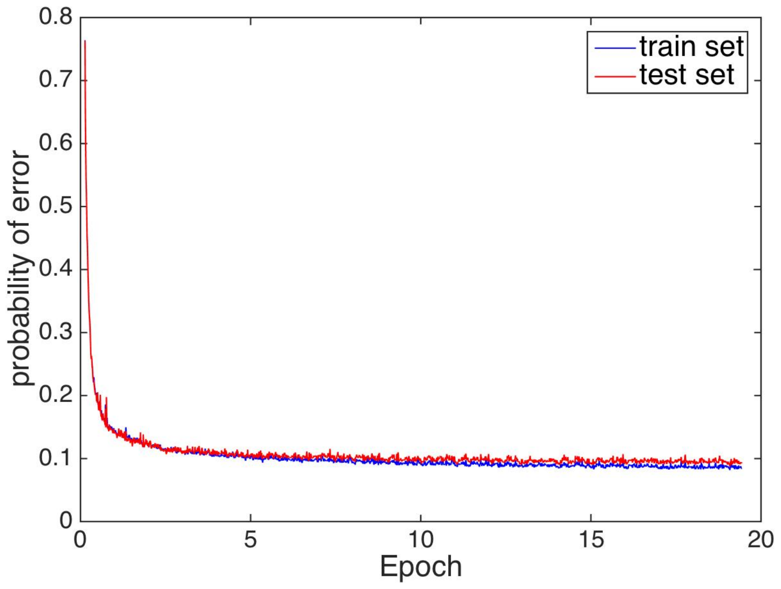
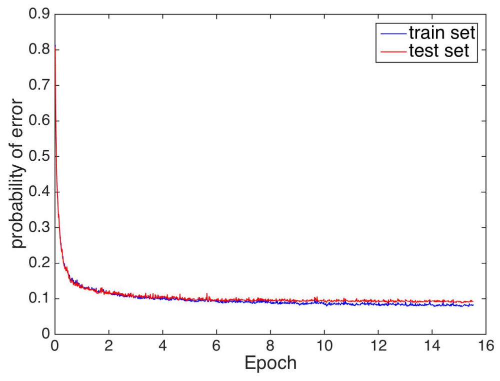


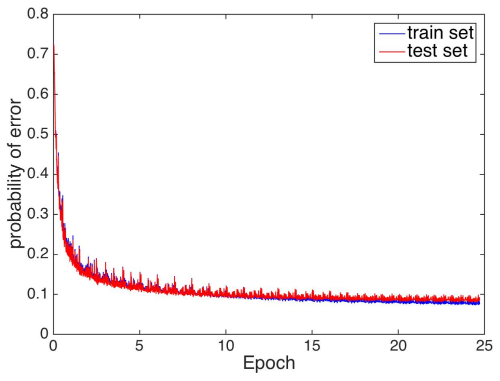
Figure 3: Probabilty of error as a function of backpropagation iteration.

The table and plots of training and test error of the three two-layer networks and two non-linearities are as follows. These results are comparable to those we obtained in **b) iii)**, where we used batch learning (without regularization, like we do here). Again, the computation is much smaller.

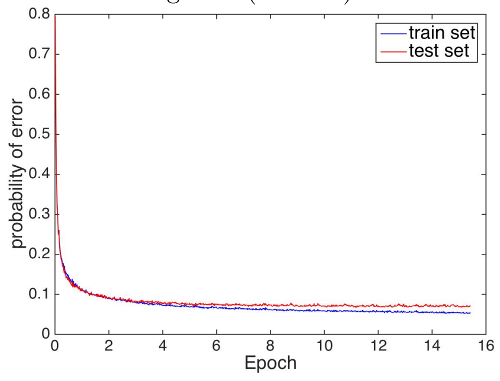
	10 units	20 units	50 units
sigmoid	9.22%	6.99%	5.36%
Relu	8.53%	5.88%	4.89%



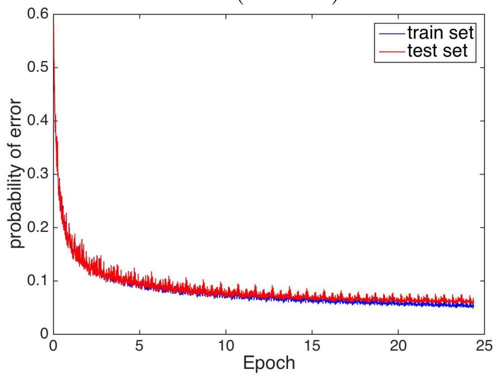
sigmoid ($H = 10$)



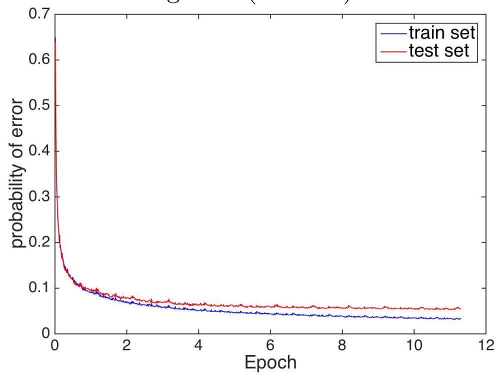
ReLU ($H = 10$)



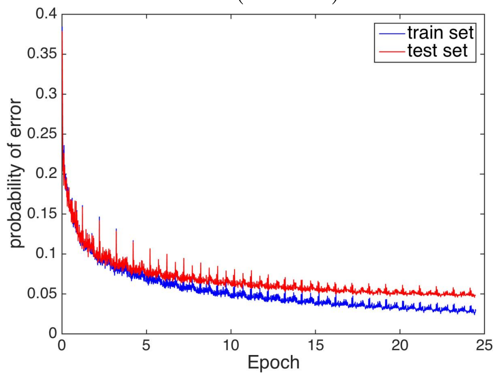
sigmoid ($H = 20$)



ReLU ($H = 20$)



sigmoid ($H = 50$)



ReLU ($H = 50$)

Figure 4: Probabilty of error as a function of backpropagation iteration for the three two-layer networks and two non-linearities.