CS 156a Learning Systems - Homework #1

Ling Li, ling@cs.caltech.edu

October 10, 2000

## 1.1 Bins and Marbles

(i) For one sample, the probability that $\nu = 0$ is

$$\Pr[\nu = 0] = (1-\pi)^{10}. \tag{1}$$

(ii) For $N$ independent samples, $R_i$ is the event that $\nu = 0$ in $i$th sample.[1] Since all events are i.i.d., we have the probability that (at least) one of the sample has $\nu = 0$ is

$$\Pr\left[\left(\bigcap_{i=1}^{N} R_i^c\right)^c\right] = 1 - \prod_{i=1}^{N}\left(1 - \Pr[R_i]\right) = 1 - \left[1 - (1-\pi)^{10}\right]^N. \tag{2}$$

Easy to see that (1) is a special case of (2) when $N = 1$.

(iii) We list the (approximate) probabilities for different $N$ and $\pi$ below

| $N$ \ $\pi$ | 0.05 | 0.5 | 0.8 |
|---|---|---|---|
| 1 | 0.5987 | $9.766 \times 10^{-4}$ | $1.024 \times 10^{-7}$ |
| $10^3$ | 1 | 0.6236 | $1.024 \times 10^{-4}$ |
| $10^6$ | 1 | 1 | 0.09733 |

## 1.2 Gradient Descent

(i) At point $(u, v) = (1, 1)$, we have

$$\begin{aligned}
\frac{\partial E}{\partial u} &= 2\left(ue^v - 2ve^{-u}\right)\left(e^v + 2ve^{-u}\right) = 2\left(e^2 - 4e^{-2}\right) \approx 13.695 \\
\frac{\partial E}{\partial v} &= 2\left(ue^v - 2ve^{-u}\right)\left(ue^v - 2e^{-u}\right) = 2\left(e - 2e^{-1}\right)^2 \approx 7.8608
\end{aligned}$$

and the gradient direction is

$$\Delta_{u,v} = \left(\frac{\frac{\partial E}{\partial u}}{\sqrt{\left(\frac{\partial E}{\partial u}\right)^2 + \left(\frac{\partial E}{\partial v}\right)^2}}, \frac{\frac{\partial E}{\partial v}}{\sqrt{\left(\frac{\partial E}{\partial u}\right)^2 + \left(\frac{\partial E}{\partial v}\right)^2}}\right).$$

Note that $|\Delta_{u,v}| = 1$.

[1] Thanks to Amir Atiya. I did not know how to denote the probability when $N$ samples are involved. He told me to use notion like event.

(ii) The first-order approximation is like

$$E(u+\Delta u, v+\Delta v) = E(x,y) + \frac{\partial E}{\partial u}\Delta u + \frac{\partial E}{\partial v}\Delta v.$$

Descending along the gradient, i.e., $(\Delta u, \Delta v) = -0.1\Delta_{u,v}$, we have

$$\Delta E = \frac{\partial E}{\partial u}\Delta u + \frac{\partial E}{\partial v}\Delta v = -0.1\sqrt{\left(\frac{\partial E}{\partial u}\right)^2 + \left(\frac{\partial E}{\partial v}\right)^2} \approx -1.5791$$

(iii) Without approximation, we have the new point is $(u+\Delta u, v+\Delta v) \approx (0.913271, 0.950220)$. Denoting it by $(\hat{u}, \hat{v})$, we have

$$\Delta E = E(\hat{u}, \hat{v}) - E(u,v) \approx -1.3720$$

(iv) If descending along the $u$-coordinate, i.e., $(\Delta u, \Delta v) = (-0.1, 0)$, we have

$$\Delta E = E(0.9, 1) - E(1,1) \approx -1.2627$$

(v) If descending along the $v$-coordinate, i.e., $(\Delta u, \Delta v) = (0, -0.1)$, we have

$$\Delta E = E(1, 0.9) - E(1,1) \approx -0.6997$$

(vi) At new point $(\hat{u}, \hat{v})$,

$$\begin{aligned}
\frac{\partial E}{\partial u} &= 2\left(\hat{u}e^{\hat{v}} - 2\hat{v}e^{-\hat{u}}\right)\left(e^{\hat{v}} + 2\hat{v}e^{-\hat{u}}\right) \approx 10.713 \\
\frac{\partial E}{\partial v} &= 2\left(\hat{u}e^{v} - 2\hat{v}e^{-u}\right)\left(\hat{u}e^{\hat{v}} - 2e^{-\hat{u}}\right) \approx 4.9890
\end{aligned}$$

Compared to the values at $(u,v)$, the partial derivatives here are smaller. Since we took those bigger values as the gradient for the whole descent step when using the first-order approximation, the result should be greater (in sense of absolute value) than the actual value. That is why we get $-1.5791$ instead of $-1.3720$. A more precise approximation can be done by averaging the partial derivatives at point $(u,v)$ and $(\hat{u}, \hat{v})$:

$$\Delta E = \left(\left.\frac{\partial E}{\partial u}\right|_{(u,v)} + \left.\frac{\partial E}{\partial u}\right|_{(\hat{u},\hat{v})}\right)\frac{\Delta u}{2} + \left(\left.\frac{\partial E}{\partial v}\right|_{(u,v)} + \left.\frac{\partial E}{\partial v}\right|_{(\hat{u},\hat{v})}\right)\frac{\Delta v}{2} \approx -1.3783$$

That is much closer to the actual value.

## 1.3 Backpropagation

To test the performance of the back-propagation algorithm with various parameters, an 'auto-stop' training program is designed.[2] This program (`trainnet.m`) keeps an eye on the average performance in recent epochs, analyzes whether the learning is going on well or stuck in a local minimum, and then stops the training when it thinks that the learning can not continue. Thus we do not need to specify number of epochs for every combination of parameters, which could be very boring.

Specifically, the program inspects

[2] Thanks to Gentian Buzi. He discussed the idea to 'automate' all the training tasks with me.

- *Training error.* If the training error is less than a predefined value $E_{\min}$, then learning stops.
- *Epochs.* If the maximum epochs reached, learning stops.
- *Average training error (ATE).* Every 50 epochs (we call that a period), the program will compare the ATE of the last period with that of the second last period. If the ATE decreases (decays) too little, say, within 1%, the program will argue that the learning can not continue and stop.
- *Average maximum* $\left|\Delta w_{ij}^l\right|$. In every epoch, weights are adjusted and there's a maximum $\left|\Delta w_{ij}^l\right|$. The average maximum $\left|\Delta w_{ij}^l\right|$ is calculated for a period and if this value is too small, the program will also think that the learning can not continue and stop.

For all combinations of parameters, we take $E_{\min} = 0.0004$, the maximum epochs as 2000, the minimum ATE decay as 1%, and the minimum average maximum $\left|\Delta w_{ij}^l\right|$ is 0.001. These are empirical data by observing the performance of the algorithm with some combinations.

`run.m` carried out 4 tests for each combination of parameters, since we want to find out the overall performance from random weights. It seems that quite a few combinations can do a good job. The best combinations are (judged by how fast they sucessfully learned the training data)

| $d^1$ | $\eta$ | initial range of $w$ |
|---|---|---|
| 2 | 0.1 | $(-0.2, 0.2)$, $(-0.02, 0.02)$ |
| 3 | 0.1 | $(-0.2, 0.2)$, $(-0.02, 0.02)$ |
| 4 | 0.1 | $(-0.2, 0.2)$, $(-2, 2)$ |
| 5 | 0.1 | $(-0.2, 0.2)$ |

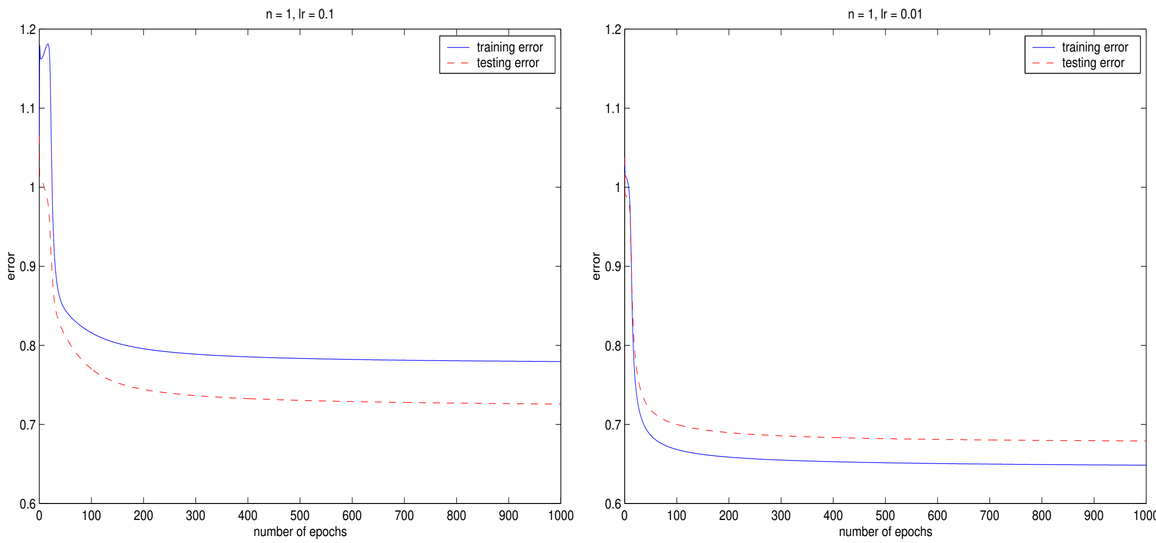
It happens that for $d^1 = 1$ or $\eta = 1$, all tests failed to learn. And the critical parameters are $d^1$ and $\eta$. The initial range of $w$ should not be too small. However, randomness helps to avoid the wrong setting of initial $w$.

### 1.4 Generalization

All plots are listed in Figure 1 on page 4, Figure 3 on page 5, and Figure 4 on page 6. They were produced by `traintst.m`, which is similar to `trainnet.m` but without some stopping conditions. From the plots, we get

- The plots in Figure 1 show that only 1 hidden neuron is not sufficient for learning the data. This can be found out in Figure 2 on page 4. Approximately, inputs with same output form 3 regions in the plane, and 2 lines are needed to separate these regions. So at least 2 hidden neurons are needed.
- Over-training exits. For example, the testing error takes its minimum value (0.0941) at epoch 224 in Figure 3(a). After that, it inceases and takes 0.1229 at epoch 1000. This happens for $d^1 = 2..5$.
- For some plots, the errors oscillate at the beginning part of learning, in that the learning rate $\eta$ is a little large. Experiments on $d^1 = 2$ and $\eta = 0.05$ show no oscillation.

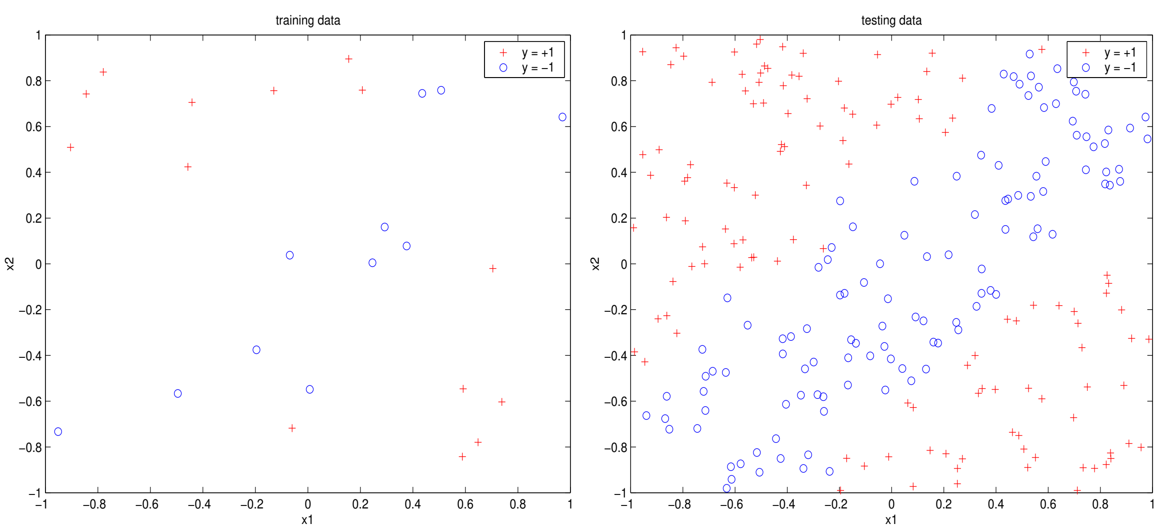
- While reversing the role of the training and testing data, learning rate should be decreased since now we have more training data. This happens to sequential learning since if we make much effort in learning a single datum, the network will be over-trained even in one epoch. This is no good to learning the whole data set.




(a) $\eta = 0.1$, errors: 0.7795 and 0.7257

(b) $\eta = 0.01$, errors: 0.6484 and 0.6790

Figure 1: With $d^1 = 1$ hidden neuron, training and testing errors are high. (a) Using training data for training; (b) Using testing data for training. Initial $w \in (-0.2, 0.2)$ for both plots.

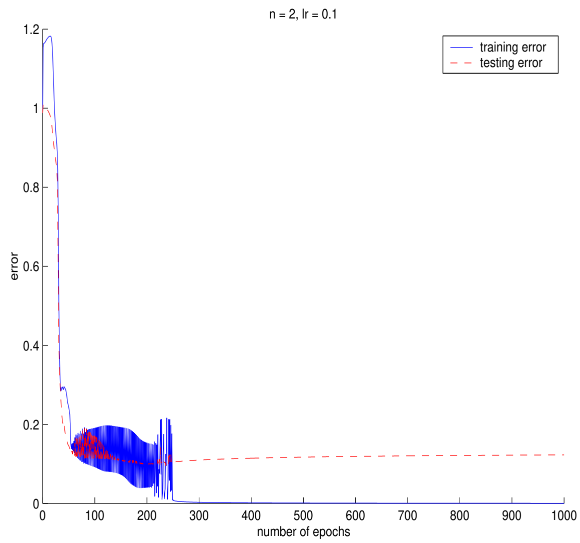



(a) Training data

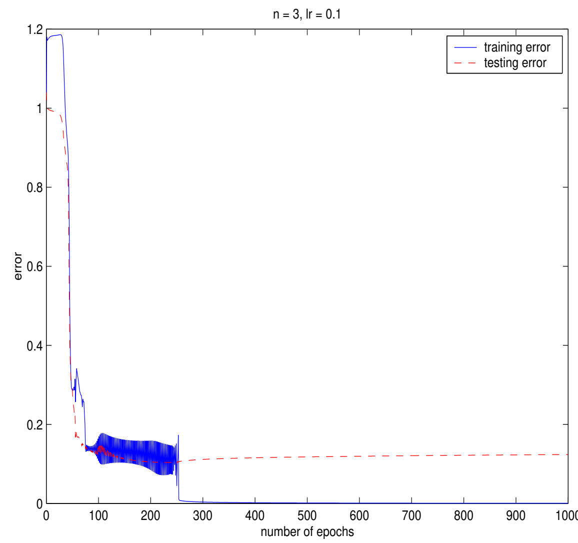
(b) Testing data

Figure 2: Training data and testing data. Points (input pair $(x_1^0, x_2^0)$) with output +1 are marked by '+', while −1 by a circle.

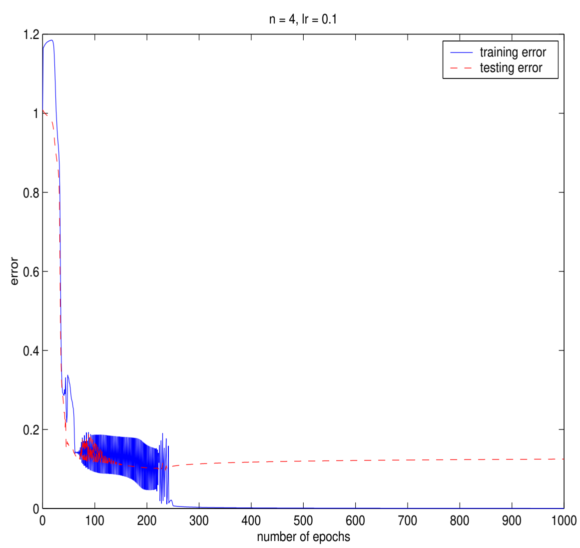
- The number of hidden neurons $d^1$ controls the complexity of our learning model. With the increasing of the model complexity, there should be some trends such as slower learning or smaller training error. However, these are not so substantial since the learning problem here is not so difficult.



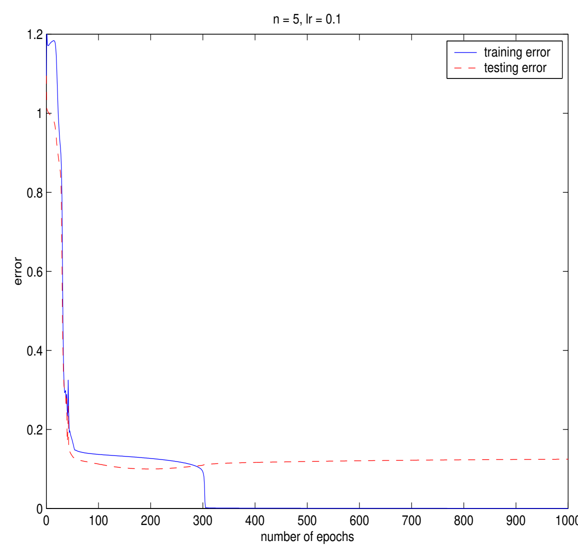

(a) $d^1 = 2$, errors: $3.910 \times 10^{-4}$ and 0.1229




(b) $d^1 = 3$, errors: $4.371 \times 10^{-4}$ and 0.1237

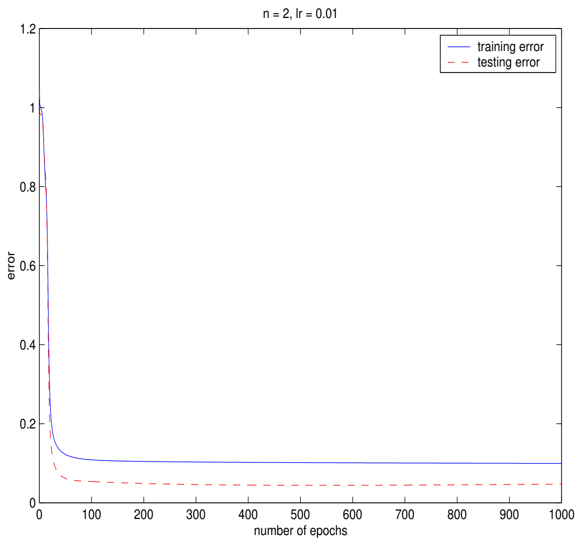



(c) $d^1 = 4$, errors: $3.906 \times 10^{-4}$ and 0.1250

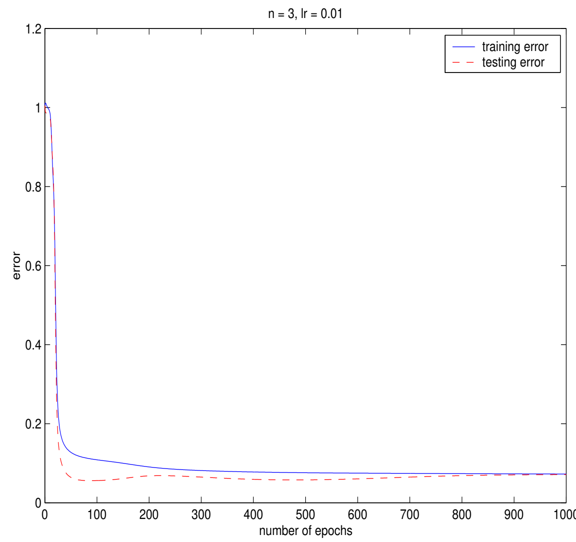



(d) $d^1 = 5$, errors: $3.192 \times 10^{-4}$ and 0.1247

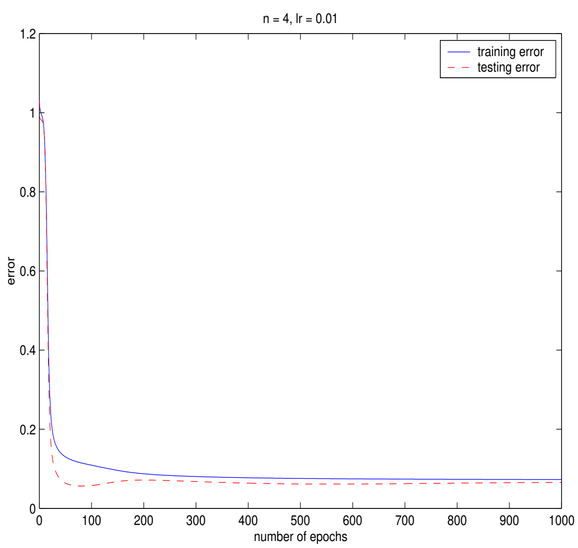
Figure 3: Using training data for training, testing data for testing, with $\eta$ fixed to 0.1, and the initial $w \in (-0.2, 0.2)$. The number of hidden neurons ($d^1$ or $n$ in the plots), average training error and testing error at epoch 1000 are listed under each plot.



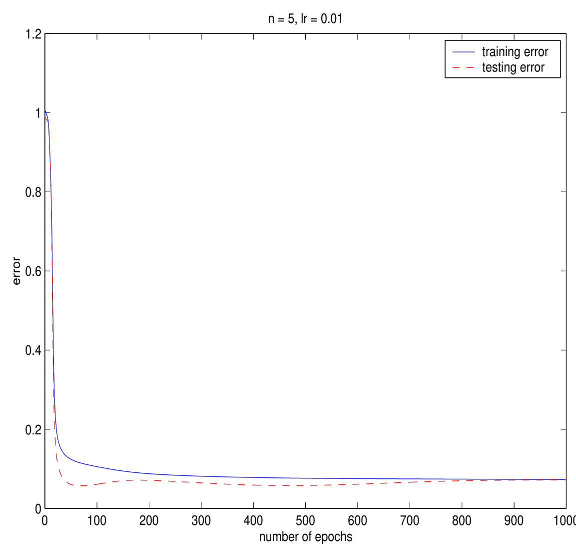

(a) $d^1 = 2$, errors: 0.0996 and 0.0471




(b) $d^1 = 3$, errors: 0.0728 and 0.0712




(c) $d^1 = 4$, errors: 0.0726 and 0.0654




(d) $d^1 = 5$, errors: 0.0728 and 0.0718

Figure 4: (Data reversing) Using testing data for training, training data for testing, with $\eta$ fixed to 0.01, and the initial $w \in (-0.2, 0.2)$.