# CSC321 Tutorial 5 part B: Assignment 2: neural networks for classifying handwritten digits (part 1 & 2) and Bayesian neural net (part 3)

Yue Li
Email: yueli@cs.toronto.edu

Wed 11-12 Feb 12
Fri 10-11 Feb 14

- In this assignment you will be using neural networks for classifying handwritten digits.
- First copy and unzip the archive below into your directory.

  `http://www.cs.toronto.edu/~bonner/courses/2014s/csc321/assignments/hw2_matlab.zip`
- An easy way to do this is as follows:

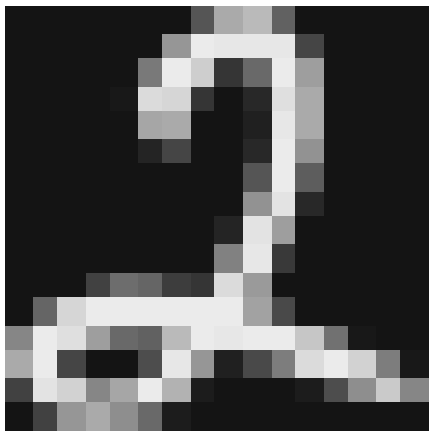
  $ `wget http://www.cs.toronto.edu/~bonner/courses/2014s/csc321/assignments/hw2_matlab.zip`
  $ `unzip hw2_matlab.zip`
  $ `cd hw2_matlab`
  $ `matlab` (or `matlab -nojvm`)

```
>> load assign2data2012.mat;
```

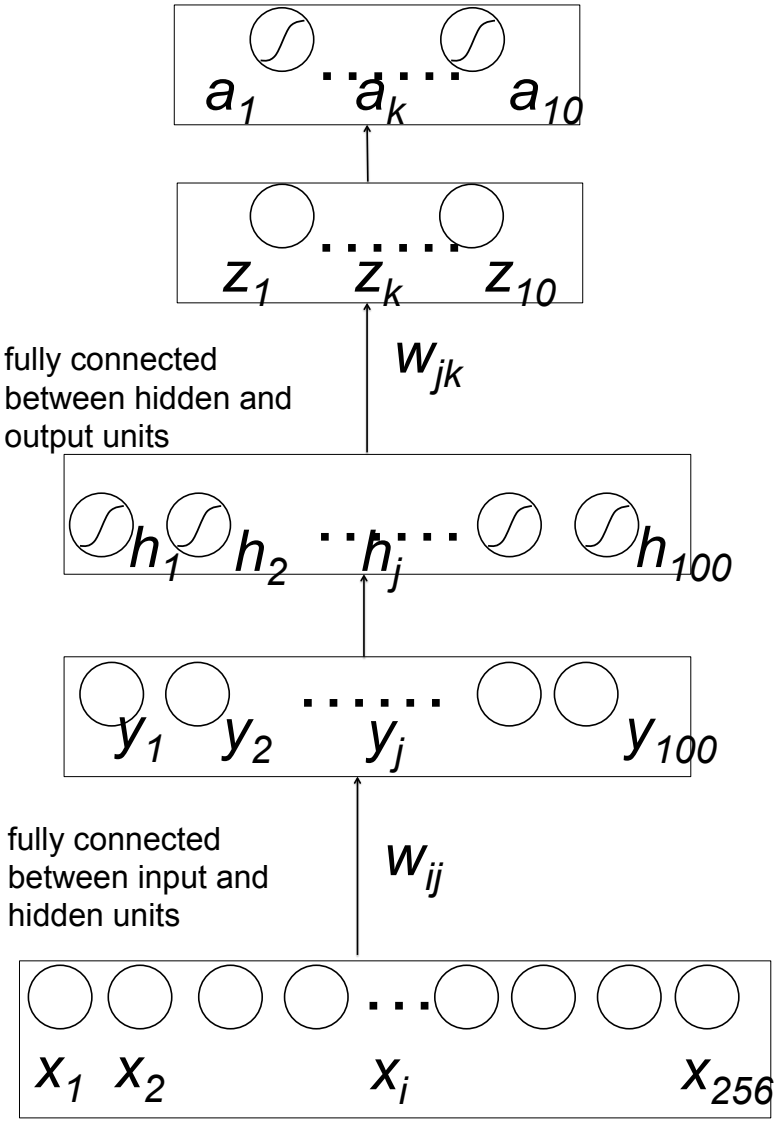
- This will load the training and test data into your workspace.
- The training data consists of 150 images of handwritten digits (15 of each class). Each image is of size $16 \times 16$ pixels and each pixel is represented by a real number between 0 (black) and 1 (white).
- The test data contains 3000 images (300 of each class).

In order to get a feel for what we are dealing with you can see the training data by typing:

```
>> showimages(data);
```



- The network we use consists of a single hidden layer with sigmoidal activation units and a 10-way softmax output unit.
- The network tries to minimize cross-entropy error (the error on a case is the negative log probability that the network gives to the correct answer).
- It uses momentum to speed the learning and weightcost to keep the weights small.

| Neural Network Schematics | Neural Network information | Variable in classbp2.m |
|---|---|---|
|  | softmax output<br>10 output units<br>100 X 10 hidden-to-output $w_{jk}$<br>100 hidden activities<br>100 hidden units<br>256 X 100 input-to-hidden $w_{ij}$<br>16 X 16 = 256 input units | output<br>outsum<br>hidout<br>hidacts<br>hidsum<br>inhid<br>data |

## Forward Pass

| Math | Matlab |
| --- | --- |
| $y_j = \sum_{i=1}^{256} w_{ij} x_i + b_j$ | `hidsum = data*inhid + repmat(hidbiases, numcases, 1);` |
| $h_j = \frac{1}{1+e^{-y_j}}$ | `hidacts = 1./(1+exp(-hidsum));` |
| $z_k = \sum_{j=1}^{60} w_{jk} h_j + b_k$ | `outsum = hidacts*hidout + repmat(outbiases, numcases, 1);` |
| $a_k = \frac{e^{z_k}}{\sum_{k=1}^{10} e^{z_k}}$ | `unnormalisedoutputs=exp(outsum);`<br>`rowsums=sum(unnormalisedoutputs,2);`<br>`outputs=unnormalisedoutputs ./repmat(rowsums,1,numout);` |

## Cross-entropy and misclassification error

| Math | Matlab |
|---|---|
| $E = -\sum_{n=1}^{N^*} \sum_{k=1}^{10} t_k \log a_k$ | `E = - sum(sum(targets .* log(tiny+outputs)));` |
| $\hat{t} = \arg\max_k a_k$ | `testguesses = (toutputs - repmat( (max(toutputs'))', 1, numout)) >= 0;` |
| $E_{misclass} = \sum_{n=1}^{N^*} (t_n - \hat{t}_n)$ | `terrors = sum(sum(testtargets)) - sum(sum(testtargets .* testguesses));` |

*$N = 150$ for training cases; $N = 3000$ for testing cases.

## Backward Pass

| Math | Matlab |
| --- | --- |
| $\frac{\partial E}{\partial h_j} = \sum_{k=1}^{10} \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial z_k}\frac{\partial z_k}{\partial h_j} = (a_k - t_k)w_{jk}$ | `dEbydhidacts = dEbydoutsum*(hidout');` |
| $\frac{\partial E}{\partial y_j} = \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial y_j} = \frac{\partial E}{\partial h_j} h_j(1-h_j)$ | `dEbydhidsum = dEbydhidacts .* hidacts.*(1-hidacts);` |
| $\frac{\partial E}{\partial b_k} = \sum_{n=1}^{N} \frac{\partial E}{\partial a_k}$ | `dEbydoutbiases = sum(dEbydoutsum);` |

## **Backward Pass** cont'd

| Math | Matlab |
| --- | --- |
| $\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial z_k}\frac{\partial z_k}{\partial w_{ij}} = (a_k - t_k)h_j$ | `dEbydhidout = hidacts'*dEbydoutsum;` |
| $\frac{\partial E}{\partial b_j} = \sum_{n=1}^{N^*}\frac{\partial E}{\partial y_k}$ | `dEbydhidbiases = sum(dEbydhidsum);` |
| $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}x_i$ | `dEbydinhid = data' * dEbydhidsum;` |
| | |

## **Weight Update** (related to part 1&2)

| Math | Matlab |
| --- | --- |
| $\Delta w_{ij}^{(t)} = \eta \Delta w_{ij}^{(t-1)} - \epsilon(\frac{\partial E}{\partial w_{ij}} + \lambda w_{ij}^{(t-1)})$ | `inhidinc = momentum*inhidinc - epsilon*(dEbydinhid + weightcost*inhid);` |
| $\Delta w_{jk}^{(t)} = \eta \Delta w_{jk}^{(t-1)} - \epsilon(\frac{\partial E}{\partial w_{jk}} + \lambda w_{jk}^{(t-1)})$ | `hidoutinc = momentum*hidoutinc - epsilon*(dEbydhidout + weightcost*hidout);` |

## **Weight Update** cont'd

| Math | Matlab |
| --- | --- |
| $\Delta b_j^{(t)} = \eta \Delta b_j^{(t-1)} - \epsilon \frac{\partial E}{b_j}$ | `hidbiasesinc = momentum*hidbiasesinc - epsilon*dEbydhidbiases;` |
| $\Delta b_k^{(t)} = \eta \Delta b_k^{(t-1)} - \epsilon \frac{\partial E}{b_k}$ | `outbiasesinc = momentum*outbiasesinc - epsilon*dEbydoutbiases;` |

## **Weight Update** cont'd

| Math | Matlab |
|---|---|
| $w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t)}$ | `inhid = inhid + inhidinc;` |
| $w_{jk}^{(t)} = w_{jk}^{(t-1)} + \Delta w_{jk}^{(t)}$ | `hidout = hidout + hidoutinc;` |
| $b_j^{(t)} = b_j^{(t-1)} + \Delta b_j^{(t)}$ | `hidbiases = hidbiases + hidbiasesinc;` |
| $b_k^{(t)} = b_k^{(t-1)} + \Delta b_k^{(t)}$ | `outbiases = outbiases + outbiasesinc;` |

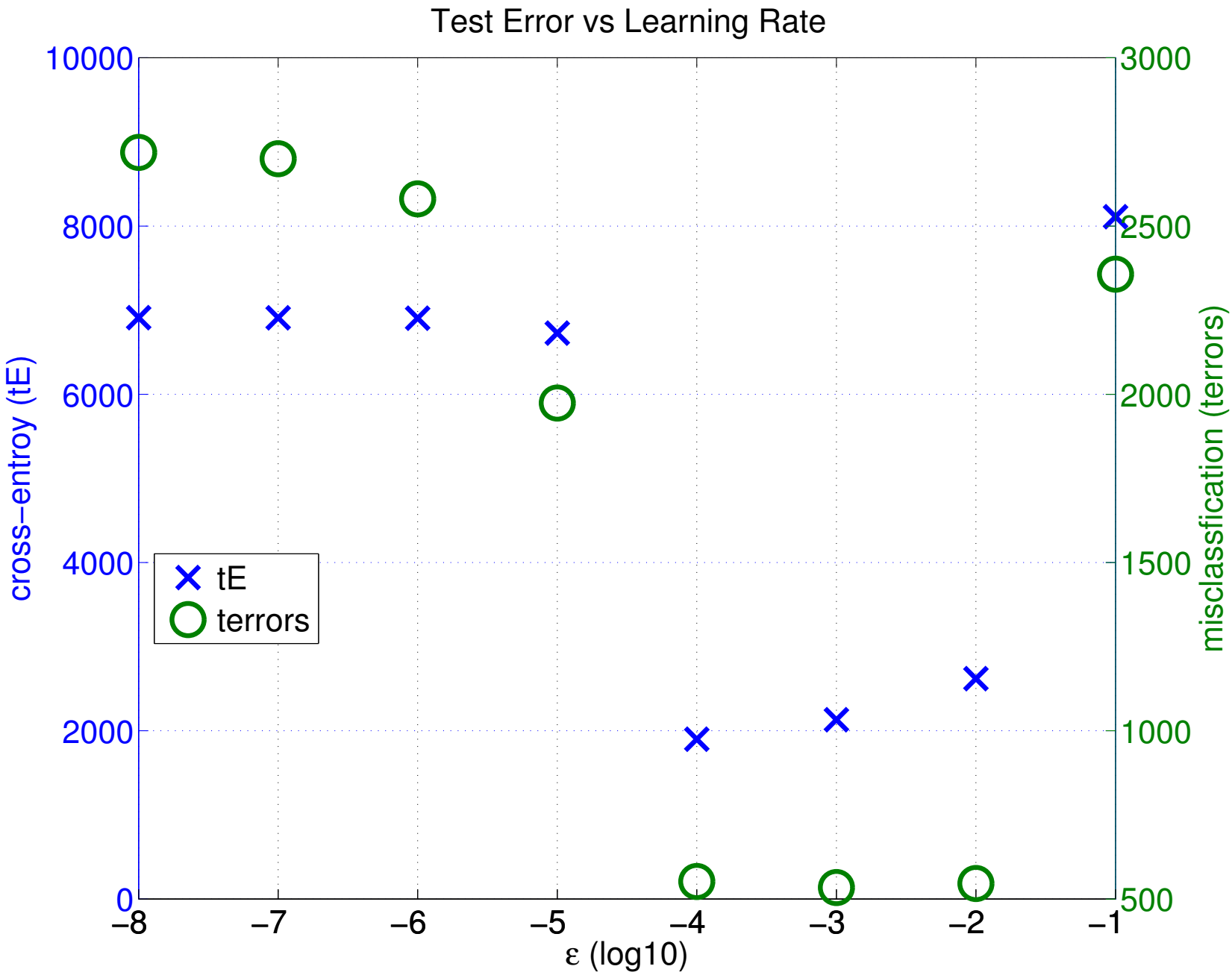
The main matlab file "classbp2.m" expects you to set several global variables that control these behaviours by hand (see the code file). This makes it easy to set up experiments in which you try many different settings:

```
>> restart = 1;
>> maxepoch = 2000;
>> numhid = 100;
>> epsilon = .01;
>> finalmomentum = 0.8;
>> weightcost = 0;
>> classbp2;
```

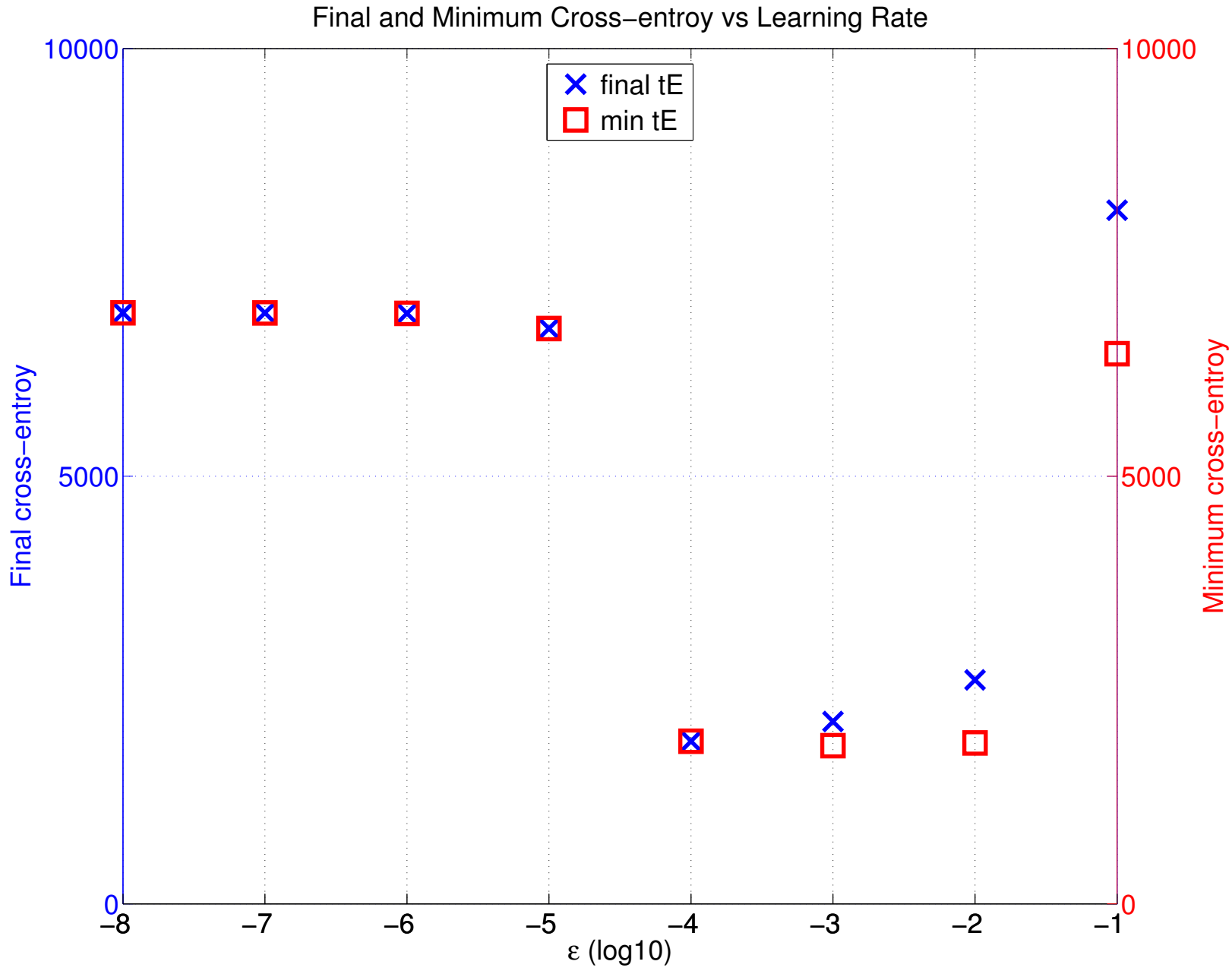
- `classbp2` trains the network.
- It prints out the cross-entropy (E) on the training set and on the test set.
- It also prints out the number of errors (it chooses the maximum output as the right answer).
- When it has finished it plots graphs of the number of test errors and of the cross-entropy cost on the test set.
- You can set the variable `errorprintfreq` in `classbp2` to make it measure the error as frequently as you want.
- You must always reset the restart variable to 1 whenever you want to redo the training from random initial weights. If you don't the network will start learning from where it left off.
- If you call `showweights(inhid)` you can see the input-to-hidden weights as gray-level images.

PART 1 (3 points)

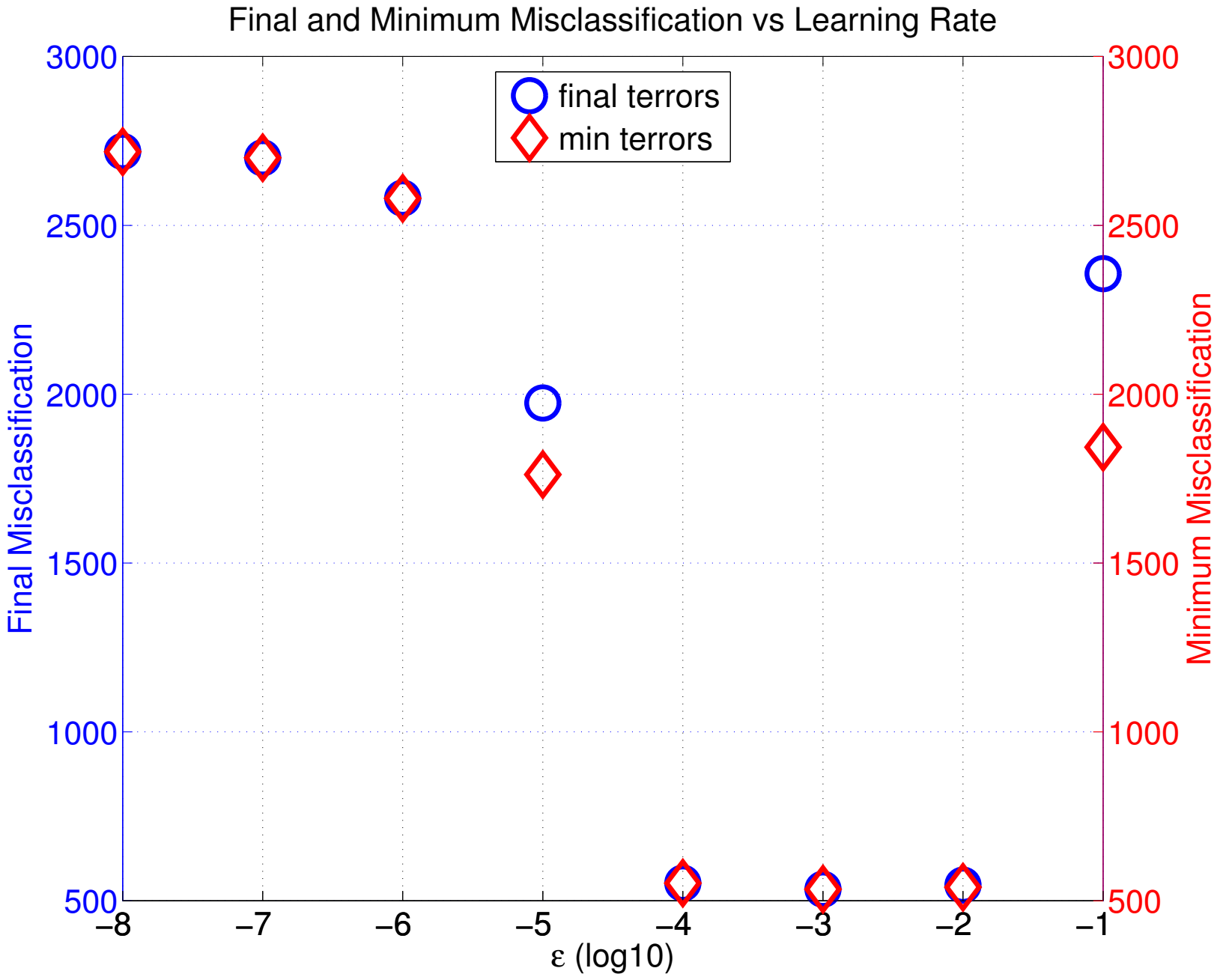
- Using `numhid=100` and `maxepoch=2000` and `weightcost=0`, play around with `epsilon` and `finalmomentum` to find settings that make `tE` low after 2000 epochs.
- Briefly report what you discover. Include the values of `epsilon` and `finalmomentum` that work best and say what values they produce for the **test** errors and the cross-entropy error.
- If you were instead asked to find the `epsilon` that produced the **best minimum** value (not the best final value) for test set cross entropy, would you expect to find a larger or smaller `epsilon` (assuming the minimum value occurs *before* the last epoch)? In a sentence, justify your answer.
- Suggestion: experiment with `epsilon` and `finalmomentum` within a reasonable range by fixing one and experimenting with the other variable: try a large range first (e.g. $\epsilon = 10^x$, where $x \in [-8, -1]$) in order to find a smaller range which achieves the best `tE`. Then refine the search within that smaller range to find the best `epsilon` and `finalmomentum`.




`finalmomentum` fixed to 0.8; (DEMO ONLY; try do better than it)




`finalmomentum` fixed to 0.8; (DEMO ONLY; try do better than it)

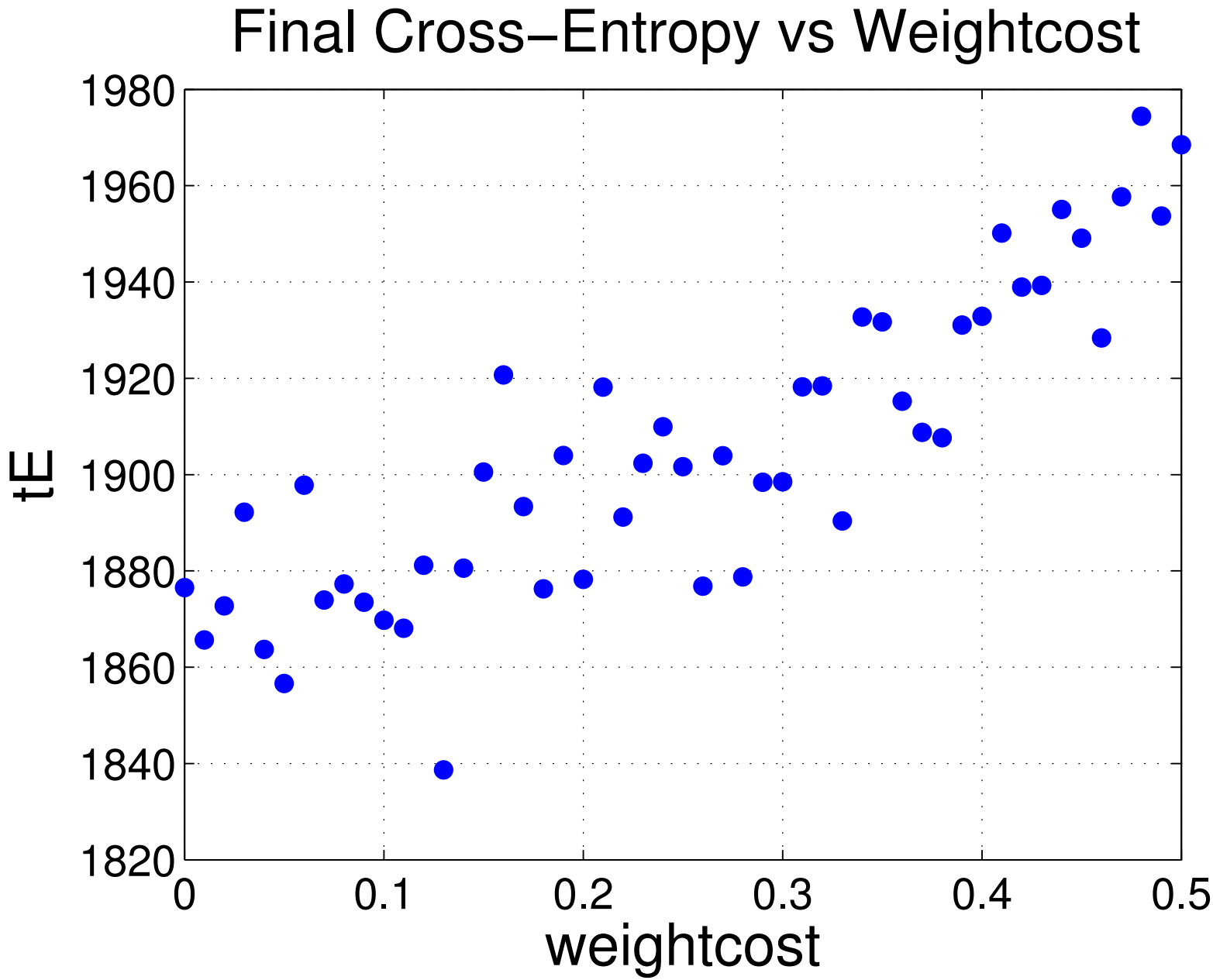



`finalmomentum` fixed to 0.8; (DEMO ONLY; try do better than it)

PART 2 (2 points)

- Using `numhid=100` and `maxepoch=2000` and `finalmomentum=0.7` set `epsilon` to a sensible value based on your experiments in part 1 and then try various values for weightcost to see how it affects the final value of `tE`.
- You may find the file `experiment.m` useful, but you will have to edit it.
- Briefly report what you discovered and include a plot of the final value of `tE` against the `weightcost`.

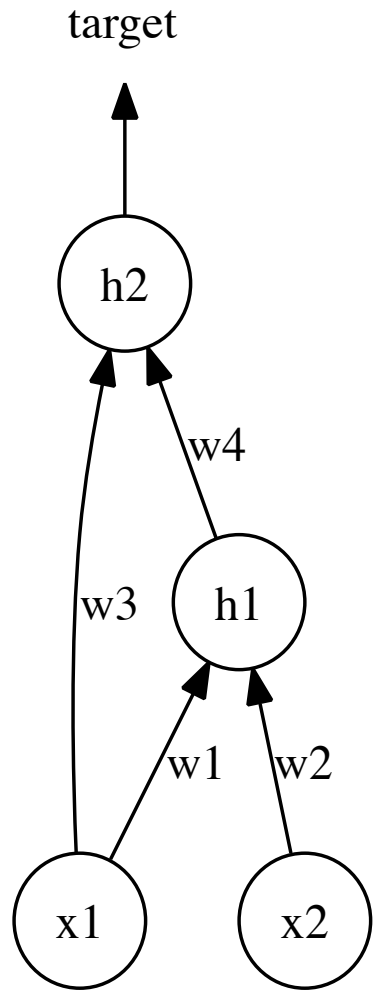
Your report on Parts 1 and 2 combined should be NOT MORE THAN ONE PAGE long, but graphs and printouts of runs can be attached.




`epsilon` fixed to $10^{-4}$; `finalmomentum` fixed to 0.8. (DEMO ONLY - try do better)

## PART 3: (5 points)

- In this part we will try to find a **Bayesian** solution to a small toy problem. The net is on the right.
- Each data point will consist of 2 real valued numbers $x_1$ and $x_2$.
- $h_1$ and $h_2$ represent sigmoidal units.
- We are interested in learning $\mathbf{W} = (w_1, w_2, w_3, w_4)$.



# PART 3: (5 points)

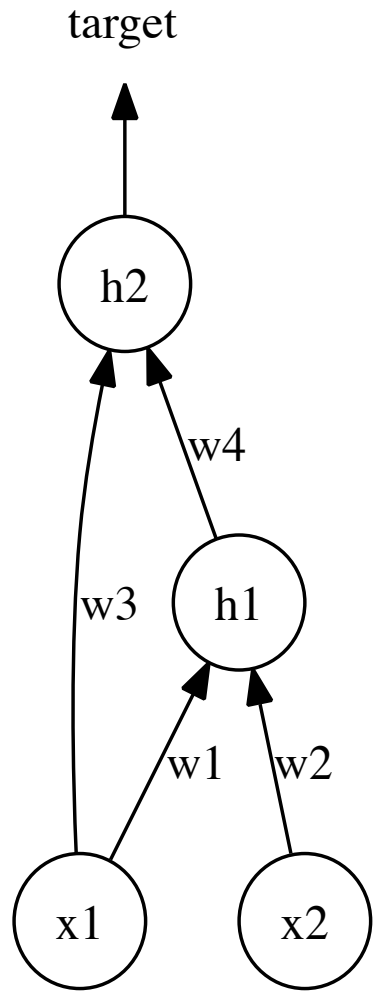
$$\hat{t} = h_2$$

$$h_2 = \frac{1}{1 + \exp(-z_2)}$$

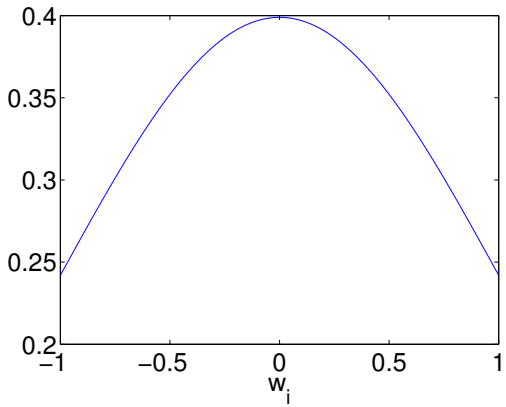
$$z_2 = w_3 x_1 + w_4 h_1$$

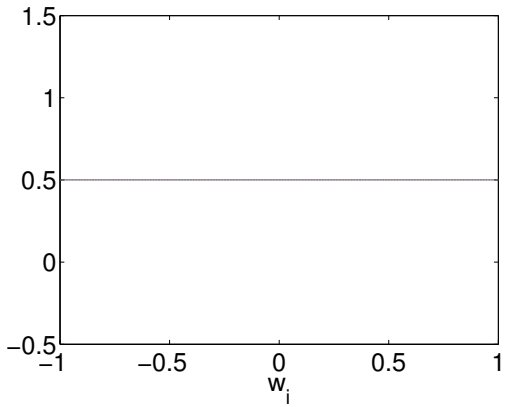
$$h_1 = \frac{1}{1 + \exp(-z_1)}$$

$$z_1 = w_1 x_1 + w_2 x_2$$



- Recall, that in a Bayesian setting, we have a prior probability distribution over parameters **W**. Each setting of parameters to values constitutes a hypothesis.
- We have some prior belief about what hypotheses are more likely to be true. One such belief could be that the hypotheses which have small values of w's are more likely (also called a Gaussian Prior; bottom left graph).
- Another belief could be that all hypotheses are equally likely (a Uniform Prior; bottom right graph). In this assignment we will be working with a **uniform prior** over **W**.



We then observe the data and modify our belief by weighting each hypothesis by its *likelihood*. In other words, the modified belief about hypothesis $\mathcal{H}$ is the prior about $\mathcal{H}$ multiplied by the likelihood that the data could be explained by $\mathcal{H}$:

$$p(\mathbf{W}|\mathbf{D}) = \frac{p(\mathbf{W}) \cdot p(\mathbf{D}|\mathbf{W})}{p(\mathbf{D})} \qquad \text{(Bayes Rule)}$$

where

- the weight vector is $\mathbf{W} = (w_1, w_2, w_3, w_4)$;
- the input data matrix $\mathbf{D}$ contains $N$ training cases of $(x_1, x_2)$ (i.e. an $N \times 2$ matrix);
- $p(\mathbf{W})$ is the prior (belief) for the weight $\mathbf{W}$;
- $p(\mathbf{D}|\mathbf{W}) = \exp(-cost)$ is the likelihood under the weights $\mathbf{W}$, where $cost = -\sum_{i=1}[t_i \log \hat{t}_i + (1 - t_i) \log(1 - \hat{t}_i)]$
- $p(\mathbf{D}) = \sum_{\mathbf{W}} p(\mathbf{W})p(\mathbf{D}|\mathbf{W})$ probability of the data, considered as a normalization factor;
- $p(\mathbf{W}|\mathbf{D})$ is the *posterior* probability of the weights.

- We then use the weighted average of the hypotheses to make decisions. The model so obtained is called a **Bayesian estimate**.
- Note that another way of making decisions could be to choose the hypothesis with the highest modified belief (instead of taking a weighted average of all of them). This is called the **Maximum a-posteriori (MAP)** estimate.
- Still another way could be to choose the hypothesis with the largest likelihood (and ignore our prior beliefs). This is called the **Maximum Likelihood (ML)** estimate.
- In this assignment we will be comparing Bayesian and ML/MAP estimators. Since we are using a uniform prior, ML and MAP estimates are the same.

Prediction on test data $\mathbf{X}_j$:

Bayesian estimate:

$$\hat{t}_j^{(\text{Bayes})} = p(t_j|\mathbf{X}_j) = \sum_{\mathbf{W}} p(\mathbf{W}|\mathbf{D})p(t|\mathbf{X}_j, \mathbf{W})$$

MAP:

$$\begin{aligned}
\mathbf{W}_{MAP} &= \arg\max_{\mathbf{W}} p(\mathbf{W}|\mathbf{D}) \\
&= \arg\max_{\mathbf{W}} \frac{p(\mathbf{W})p(\mathbf{D}|\mathbf{W})}{p(\mathbf{D})} \\
&= \arg\max_{\mathbf{W}} p(\mathbf{W})p(\mathbf{D}|\mathbf{W}) \\
\hat{t}_j^{(\text{MAP})} &= p(t_j|\mathbf{X}_j) = p(t_j|\mathbf{X}_j, \mathbf{W}_{MAP})
\end{aligned}$$

ML:

$$\begin{aligned}
\mathbf{W}_{ML} &= \arg\max_{\mathbf{W}} p(\mathbf{D}|\mathbf{W}) \\
\hat{t}_j^{(\text{ML})} &= p(t_j|\mathbf{X}_j) = p(t_j|\mathbf{X}_j, \mathbf{W}_{ML})
\end{aligned}$$

For $p(\mathbf{W}) \sim \text{Uniform}(\mathbf{W})$

$$\begin{aligned}\mathbf{W}_{MAP} &= \arg\max_{\mathbf{W}} p(\mathbf{W})p(\mathbf{D}|\mathbf{W}) \\ &= \arg\max_{\mathbf{W}} p(\mathbf{D}|\mathbf{W}) \\ &= \mathbf{W}_{ML}\end{aligned}$$

**Note**: the above is true for part 3 despite changing the true $\mathbf{W}$ distribution (i.e., `teachernet`) in the second part of part 3.

Evaluation (Contrastive Divergence):

$$ErrorSum = -\sum_{i=1}^{N'} \left[ t_n \log(\hat{t}_i) + (1 - t_i) \log(1 - \hat{t}_i) \right] + \sum_{i} \left[ t \log(t_i) + (1 - t_i) \log(1 - t_i) \right]$$

$$\textbf{ErrorPerBit} = \frac{ErrorSum}{N' \log(2)}$$

where $N'$ is the number of test cases (i.e., `testnumcases`).

**Note:** Use **ErrorPerBit** to evaluate Bayesian and MAP estimate rather than ErrorSum.

- We will generate our training data randomly. Each dimension of each data point will be a sample drawn from a normal distribution with mean 0 and standard deviation ('inputstandev', currently set to '4'):

```
data=inputstandev*randn(numcases,2);
```

- The weights w will be set to uniform random values in [-1, 1] and we assert that this is the 'true' net (also called the teacher net):

```
prior=ones(9^4,1)/(9^4);
```

- The targets are binary valued and will be sampled from a Bernoulli distribution defined by the predictions made by the teacher net (i.e., given input x, the target will be 1 with probability p, where p is the output of the teacher network on input x):

```
targets = .001 + .998*(rand(1,numcases) <
applyweights(wteacher, data));
```

>> makeallvecs;

- This makes a matrix in which each row is a possible weight vector. That is, we have all possible hypotheses with us in the matrix `allvecs`. Each $w_i$ can take on 9 possible values (-1 to 1 in intervals of 0.25). The prior is uniform over these 9^4 possible weight vectors.

>> maketeacher;

- This makes a teacher network by sampling the four weights from a uniform distribution from [-1, 1]:
- `wteacher = 2*rand(1,4) - 1;`
- Suggestion: try different prior distributions such as a standard normal $\mathcal{N}(0, 1)$ with the Matlab built-in function `randn`

>> `numcases=10;`

- `numcases` is the number of training cases that the model will use. The number of `testcases` is fixed at 100.
- Suggestion: Try larger `numcases` for the number of training cases and compare the results from Bayesian and MAP/ML estimations. Try for the same number of training cases multiple times and take the average to obtain more reliable error estimate.

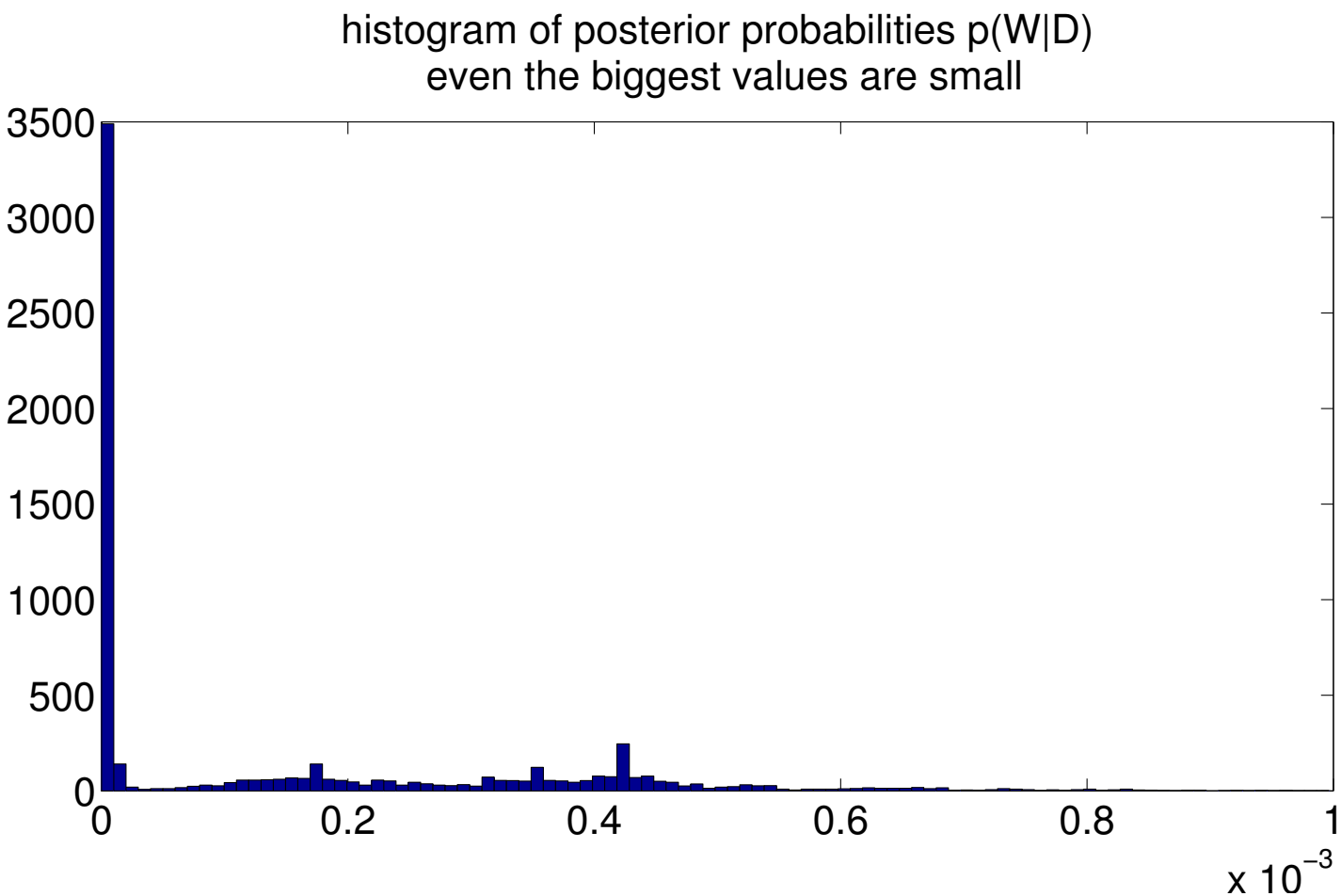
>> `bayeswithbest;`

- The code will print out the Bayesian estimate's error on the training and test sets as well as the error of the MAP/ML estimate

- Figure 1 will show you how well the outputs of the teacher on the training data can be predicted by bayes-averaging the outputs of all possible nets.
- "Bayes-averaging" means weighting the prediction of each net by the posterior probability of that net given the training data and the prior (which is flat in this example).
- Each column corresponds to a data point. The first row gives the output of the teacher net. The second row gives the output of the Bayesian estimate. The third row gives the output of the MAP/ML estimate. The size of the square is proportional to the output value.
- Figure 2 will show the same on the test data.
- Figure 3 shows a histogram of the posterior probability distribution across all 9^4 weight vectors. Notice that the posterior can be very spread out so that even the best net gets a very small posterior probability.

## Bayesian estimate (`bayespredictions`):

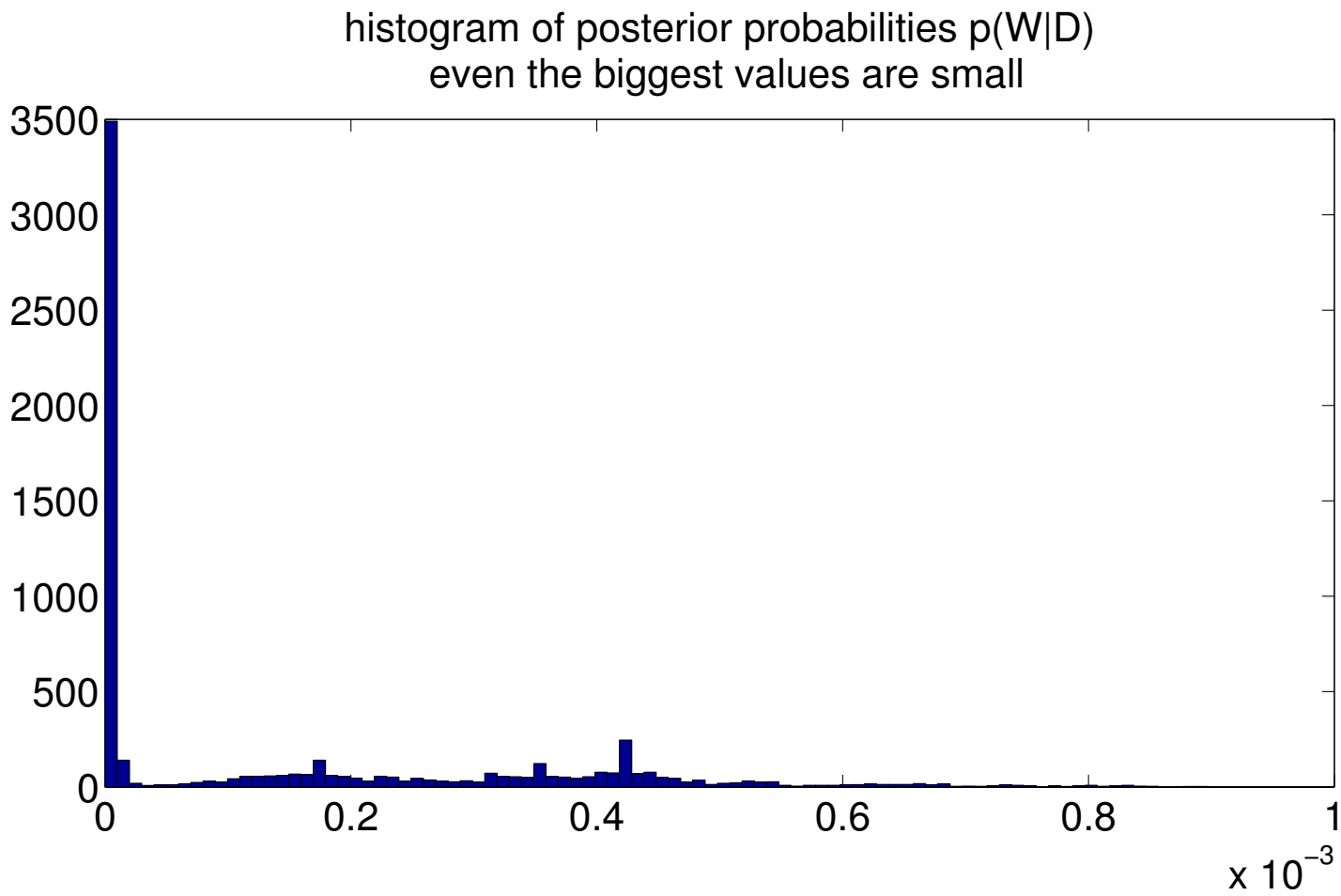
$$p(t|D) = \sum_{\forall \mathbf{W}} p(\mathbf{W}|\mathbf{D}) p(t|\mathbf{D}, \mathbf{W})$$



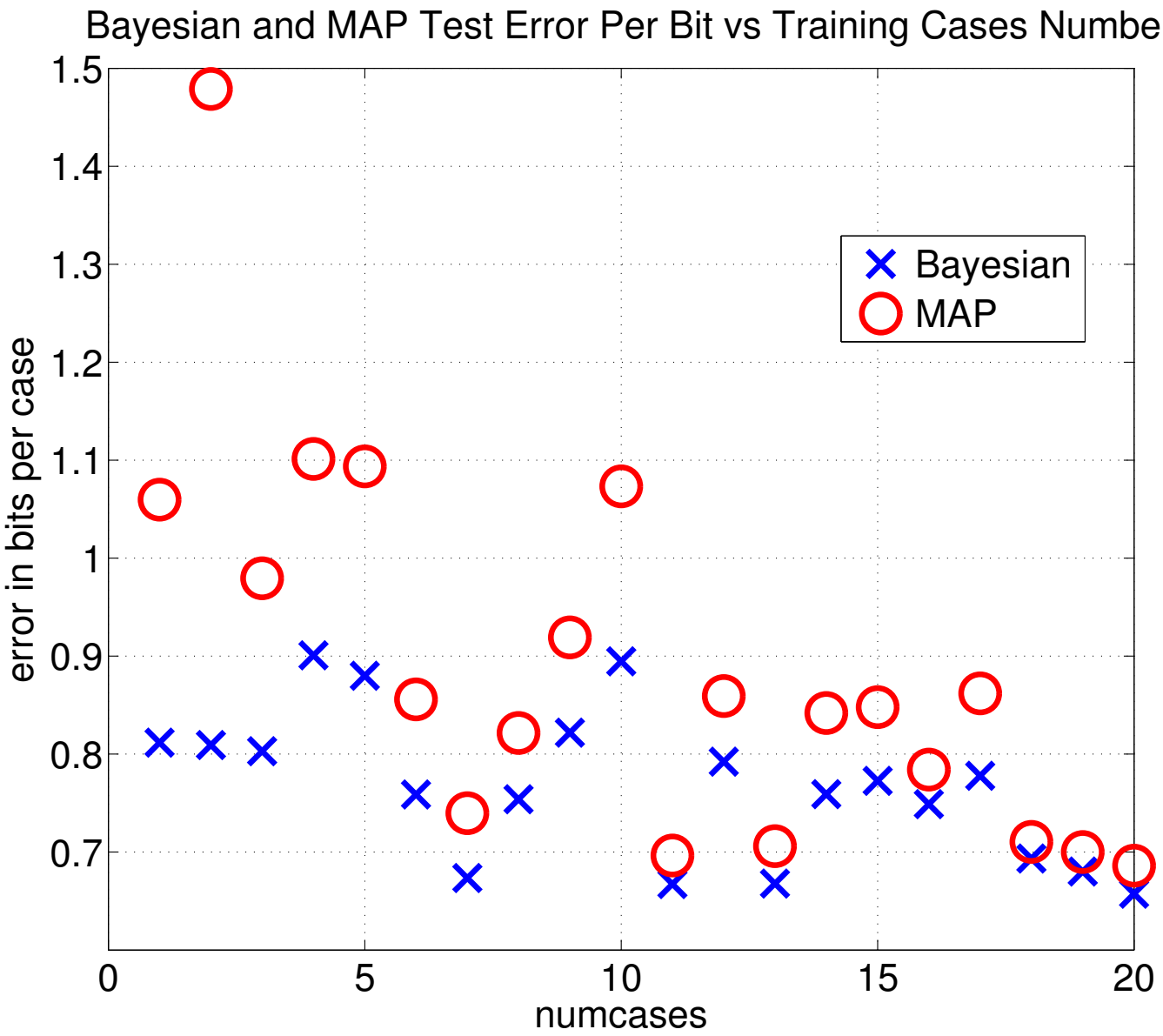
MAP (`bestpredictions`):

$$\mathbf{W}_{MAP} = \arg\max_{\mathbf{W}} p(\mathbf{W}|\mathbf{D})$$

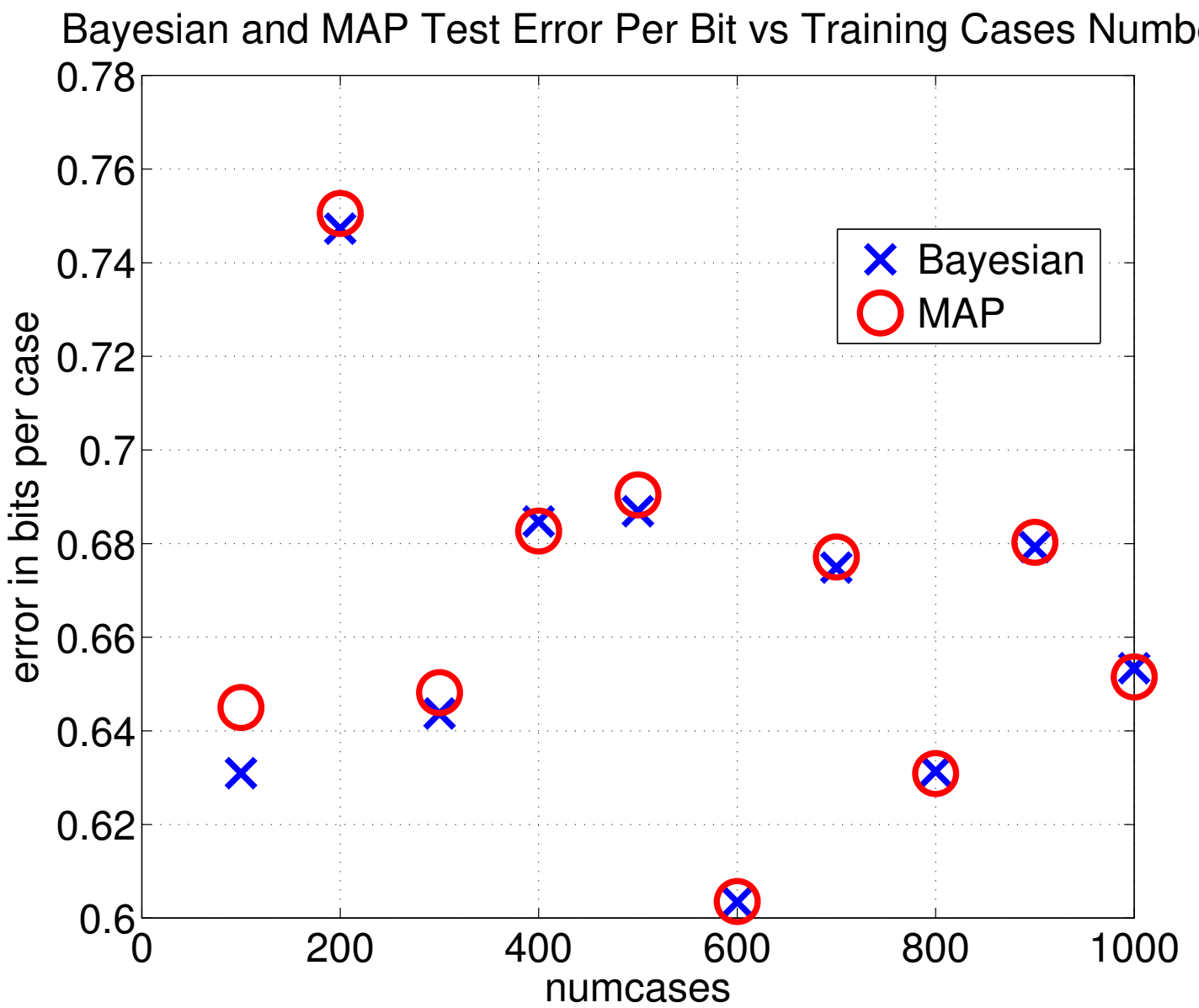
$$p(t|\mathbf{D}) \equiv p(t|\mathbf{D}, W_{MAP})$$



Bayesian vs MAP using small number training cases:



Bayesian vs MAP using large number training cases:



- Your report should be AT MOST HALF A PAGE and should describe the effects of changing the number of training cases.

  Suggestion: Try larger `numcases` for the number of training cases and compare the results from Bayesian and MAP/ML estimations. Try for the same number of training cases multiple times and take the average to obtain more reliable error estimate.
- You should also try modifying maketeacher to set different kinds of teacher nets and see how the results depend on the particular teacher net. (Changing maketeacher amounts to changing the distribution that the teacher weights are chosen from, e.g., by changing the parameters of the uniform distribution or by using a different kind of distribution altogether.)

  Suggestion: What performance would you expect when using a "wrong" prior; e.g., when the true weight is sampled from a standard normal distribution while we are using uniform prior? How do Bayesian and MAP/ML compare now?
- **A2 Due on Feb 25 at 3pm**