CS181 Homework 2

Lucas Freitas and Angela Li

March 1, 2013

1 Perceptrons

1. The bright-or-dark feature **cannot** be recognized by a perceptron. Even using weights, the two features will be in a distribution that is not linearly separable. If we make weights be 1, and input for a pixel be 1 if the pixel is on, or -1 if it is off. The distribution would be similar to image below (credit to Lecture slides), considering blue dots to be data that is at least 75% bright or dark, and red to be data that is not.

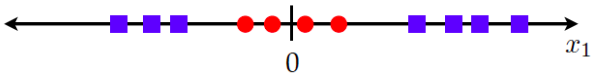


Figure 1: Distribution of points for item 1

It is easy to see that the data is not linearly separable, and the classification cannot be done with perceptrons alone. If we use high-level features, however, such that $\phi(x_i) = x_i^2$, the features become linearly separable (again, credit to Lecture slides):

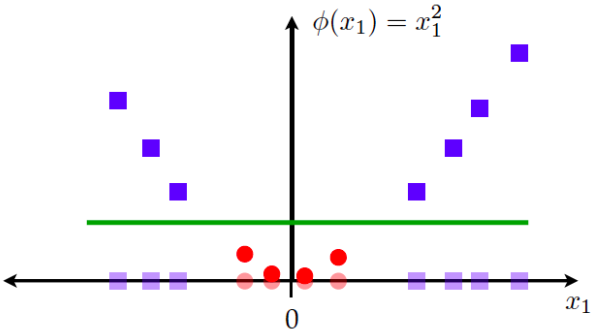


Figure 2: Distribution of points from item 1 after applying $\phi(x_i) = x_i^2$

2. The top-bright feature **can** be recognized by a perceptron. For this classification, we can use the following matrix of weights, considering that x_i for each pixel is 1 if the pixel is on, or 0 if the pixel is off:

$$M = \begin{bmatrix} 2 & 2 & 2 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

That way, if $\sum_k x_k w_k$ is positive, we have a larger fraction of pixels on in the top row; otherwise, we know that there is not a larger fraction of pixels on in the top row. x

3. Just like in item 1, connectedness **cannot** be recognized by a perceptron. Perceptrons are capable of viewing the local features of each pixel (checking whether it is on or off), but not at the relationships between two or more of them. That relationship is essential, however, for this classification problem, since connectedness depends on the combined configuration of adjacent pixels, which makes connectedness a non-linearly separable feature.

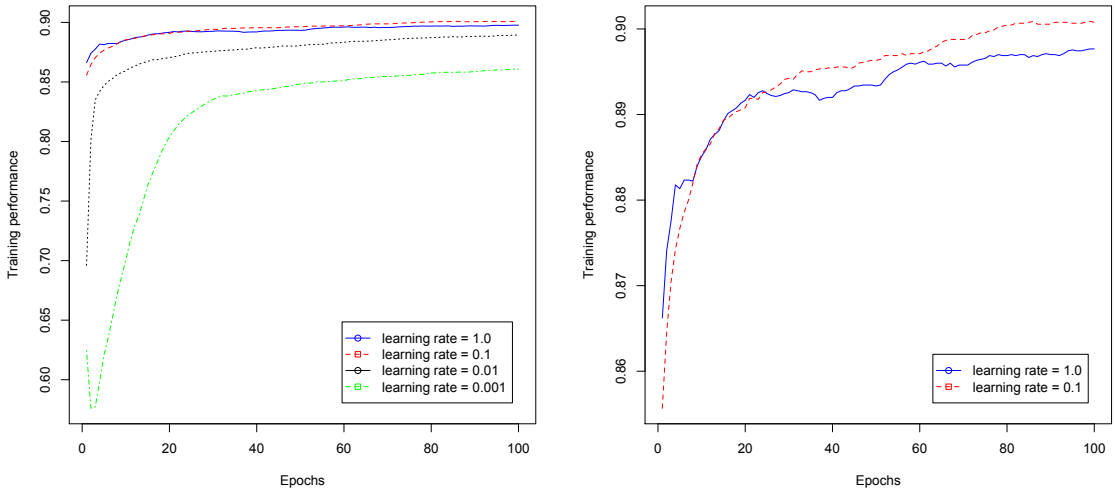
2 Learning Algorithms

- **Decision trees**
  Decision trees are a viable candidate for the digit classification task.
- **Boosted decision stumps**
  Boosted decision stumps are not a good candidate for the digit classification task.
- **Perceptrons**
  Perceptrons are a viable candidate for the digit classification task.
- **Multi-layer feed-forward neural networks**
  Multi-layer feed-forward neural networks are a great candidate for the digit classification task.

3 Neural Networks

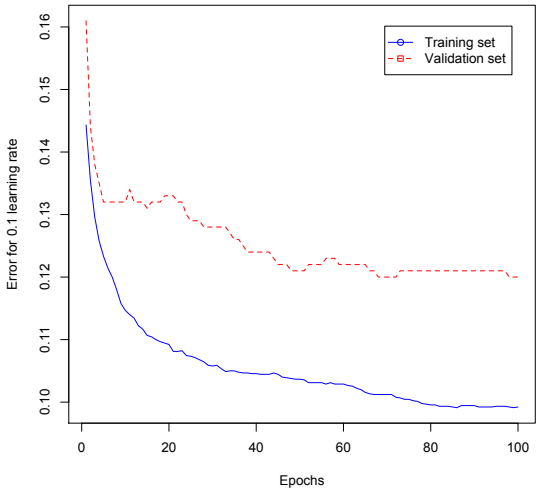
1. See `FeedForward()`.
2. See `Backprop()`.
3. See `Train()`.
4. See `EncodedNetworkFramework()`.
5. In normalizing the input values from [0, 255] to between 0 and 1, we adapt them to the outputs of the sigmoid function, which is useful in calculating intermediary values. Normalization is also good because it preserves all relationships between inputs without introducing bias.

6. After trying learning rates of 1.0, 0.1, 0.01, and 0.001, we found that the learning rate exhibiting best performance was 0.1:



(a) The learning rate used is 0.1.

(b) The graph of the training set and validation set *error* against the number of epochs can be seen below:



i. It does not appear that we are in danger of overfitting by training for too many epochs, as performance continues to increase (up to an apparent asymptotic upper bound) as the number of epochs we train for increases.

ii. We thought that 50 is a good number of epochs to train for since for larger number of epochs, the decrease in the error is not very significant, and running more epochs will just take a lot more time in the program execution, not actually improving its performance too much.

iii.

(c) For learning rate 0.1 and 50 epochs, our network had a training, validation, and test performance of 0.8964, 0.879, and 0.9160, respectively.

7. (a) Running the algorithm for 15 and 30 hidden layers, and for a learning rate of $1.0, 0.1, 0.01$, and 0.001, we realized that the optimal learning rate would be 0.1, just like in $6(a)$, for both 15 or 30 hidden layers.

   (b) One way that we can go about doing that is by keeping track of the last k, let's say 10, performances by epoch. Consider that we are looking at the i_{th} performance, P_i. Let's also consider that $i > 10$. We just need to check for each performance if:

$$\left| P_i - \frac{1}{10} \sum_{j=i-10}^{i-1} P_j \right| < t$$

   Where t is a threshold we pick. That is, we will perform the algorithm until $i == 100$ or until we reach that threshold, which would mean that the increase in are seeing in the performance is now small, and we should stop iterating.

   (c) Using a threshold $t = 0.001$, we get 26 epochs for 15 hidden units, and 38 epochs for 30 hidden units. The graphs can be seen below:

   (d)

   (e)

   (f)

8. (a)

   (b)

4 Error Functions

1.

2.