Problem 1 (Concepts) - 1 point

In addition to the sigmoid, another popular soft threshold is the hyperbolic tangent:

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

(a) How is $\tanh$ related to the logistic function $\theta$? [Hint: Shift and scale.]

(b) Show that $\tanh(s)$ converges to a hard threshold for large $|s|$, and converges to no threshold for small $|s|$. [Hint: Formalize the figure below.]

![Figure 1: Hard and soft thresholds](image)

Problem 2 (Concepts) - 1 point

(a) If we are learning from $\pm 1$ data to predict a noisy target $P(y|x)$ with candidate hypothesis $h$, show that the maximum likelihood method reduces to the task of finding an $h$ that minimizes

$$E_{\text{in}}(w) = \sum_{n=1}^{N} \begin{cases} \ln \frac{1}{h(x_n)} + \ln \frac{1}{1 - h(x_n)} & \text{if } y_n = +1 \\ -\ln \frac{1}{h(x_n)} & \text{if } y_n = -1 \end{cases}.$$ 

(b) For the case $h(x) = \theta(w^T x)$, argue that minimizing the in-sample error in part (a) is equivalent to minimizing

$$E_{\text{in}}(w) = \frac{1}{N} \sum_{n=1}^{N} \ln \left(1 + e^{-y_n w^T x_n}\right).$$
For the two probability distributions \( \{ p, 1 - p \} \) and \( \{ q, 1 - q \} \) with binary outcomes, the cross-entropy (from information theory) is

\[
p \log \frac{1}{q} + (1 - p) \log \frac{1}{1 - q}.
\]

The in-sample error in part (a) corresponds to a cross-entropy error measure on the data point \((x_n, y_n)\), with \( p = \mathbb{I}[y_n = +1] \) and \( q = h(x_n) \).

**Problem 3 (Concepts) - 1 point**

For logistic regression, show that

\[
\nabla E_{in}(w) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n x_n}{1 + e^{y_n w^T x_n}}
\]

(1)

Argue that a ‘misclassified’ example contributes more to the gradient than a correctly classified one.

**Problem 4 (Concepts) - 1 point**

In gradient descent, the claim that \( \hat{v} \) is the direction which gives the largest decrease in \( E_{in} \) only holds for small step size, \( \eta \). Why?

**Problem 5 (Practice) - 3 points**

(a) **Backpropagation**: Following the lecture notes, implement the backpropagation algorithm that takes as input a network architecture \((d^{(0)} = d, d^{(1)}, \ldots, d^{(L)} = 1)\) and a set of examples \((x_1, y_1), \ldots, (x_N, y_N)\), where \( x_n \in \mathbb{R}^d \) and \( y \in \mathbb{R} \), and produces as output the network weights. The algorithm should perform gradient descent on one example at a time, but should also keep track of the average error for all the examples in each epoch. Train your algorithm on the dataset at http://work.caltech.edu/train.dat (the first two columns are the input and the third column is the output). Test the convergence behavior for architectures with one hidden layer \((L = 2)\) and 1 to 5 neurons \((d^{(1)} = 1, 2, 3, 4, 5)\), with combinations of the following parameters:

(i) The initial weight values chosen independently and randomly from the range \((-0.02, 0.02)\), or the range \((-2, 2)\).

(ii) The learning rate \( \eta \) fixed at 0.01, 0.1, or 1.

(iii) A sufficient number of epochs to get the training error to converge (within reason).

Turn in your code (from scratch - no PyTorch!) and a single parameter combination that resulted in good convergence for the above architectures.

(b) **Generalization**: Using your backpropagation program and data from part (a), train different neural networks with \( L = 2 \) (an input layer, one ‘hidden’ layer, and an output layer), where the number of neurons in the hidden layer is 1, 2, 3, 4, or 5. Use the out-of-sample data at http://work.caltech.edu/test.dat to test your networks.
Plot the training and test error for each network as a function of the epoch number (hence the 'intermediate' networks are evaluated using the test data, but the test data is not used in the backpropagation).

Repeat the experiment by reversing the roles of the training and test sets (Note: You may need to readjust the parameter combination from the previous problem). Plot the training and test errors again. Briefly analyze your results.

Problem 6 (Practice) - 3 points

In this problem, you will dive into PyTorch to get familiar with tensors and the power of automatic differentiation. You’ll also construct your first neural network in PyTorch, defining its layers and optimizer. You’ll then train and test it to see if you can achieve state-of-the-art performance on handwritten digit classification.

Questions 6(a) through 6(e) are posed in notebook 'tutorial4-mlp-sgd-mnist.ipynb' on the course website. Please feel free to answer all questions in the notebook using a combination of code output and markdown.

*** IMPORTANT: No hard copies accepted. Submit typed or written answers in PDF format for the theoretical portion. Submit answers to the programming questions in PDF format, along with plots. You may choose to markdown your Jupyter Notebook output. In addition, you must upload your source code files. ***