Question

.

As discussed in class, neural networks are built out of units with real-valued inputs $X_1 \ldots X_n$, where the unit output Y is given by

$$Y = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

Here we will explore the expressiveness of neural nets, by examining their ability to represent boolean functions. Here the inputs X_i will be 0 or 1. Of course the output Y will be real-valued, ranging anywhere between 0 and 1. We will interpret Y as a boolean value by interpreting it to be a boolean 1 if $Y > 0.5$, and interpreting it to be 0 otherwise.

1. Give 3 weights for a single unit with two inputs X_1 and X_2, that implements the logical OR function $Y = X_1 \vee X_2$.

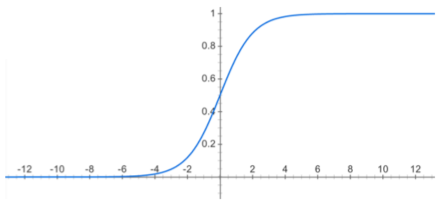

Figure 1: $\frac{1}{1+e^{-x}}$.

★ **SOLUTION:** Figure 1 shows the value of $y = \frac{1}{1+e^{-x}}$ for different values of x. Note that $y \geq 0.5$ if $x \geq 0$, and $y \leq 0.5$ if $x \leq 0$. Given this, we need to choose w_i so that $w_0 + w_1 * x_1 + w_2 * x_2$ will be greater than 0 when $x_1 \vee x_2$ is equal to 1. One candidate solution is $[w_0 = -0.5, w_1 = 1, w_2 = 1]$.

2. Can you implement the logical AND function $Y = X_1 \wedge X_2$ in a single unit? If so, give weights that achieve this. If not, explain the problem.

   ★ **SOLUTION:** Similar to previous part, we can obtain $[w_0 = -1.5, w_1 = 1, w_2 = 1]$

3. It is impossible to implement the EXCLUSIVE-OR function $Y = X_1 \oplus X_2$ in a single unit. However, you can do it using a multiple unit neural network. Please do. Use the smallest number of units you can. Draw your network, and show all weights of each unit.

   ★ **SOLUTION:** It can be represented by a neural network with two nodes in the hidden layer. Input weights for node 1 in the hidden layer would be $[w_0 = -0.5, w_1 = 1, w_2 = -1]$, input weights for node 2 in the hidden layer would be $[w_0 = -0.5, w_1 = -1, w_2 = 1]$, and input weights for the output node would be $[w_0 = -0.8, w_1 = 1, w_2 = 1]$.

4. Create a neural network with only one hidden layer (of any number of units) that implements $(A \vee \neg B) \oplus (\neg C \vee \neg D)$. Draw your network, and show all weights of each unit.

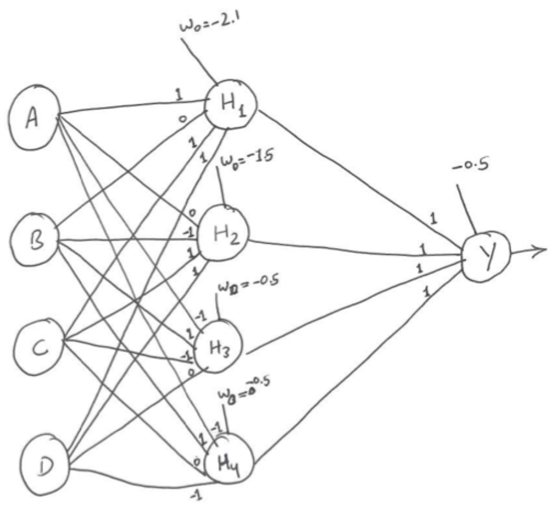

Figure 2: An example of neural network for problem 1.4

★ **SOLUTION:** Note that XOR operation can be written in terms of AND and OR operations: $p \oplus q = (p \wedge \neg q) \vee (\neg p \wedge q)$. Given this, we can rewrite the formula as $(A \wedge C \wedge D) \vee (\neg B \wedge C \wedge D) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg D)$. This formula can be represented by a neural network with one hidden layer and four nodes in the hidden layer (one unit for each parenthesis). An example is shown in Figure 2.

Q. Answer TRUE or FALSE, midterm_sol.pdf

The decision boundary learned by a neural network is always non-linear

False as the decision boundary can be linear if all threshold units are linear. Regardless of the size of the neural network, the backpropagation algorithm can always find the globally optimal weights for the neural network.

False It needn't since for a general multi-layer NN with non-linear threshold units, the function optimized by the backpropagation algorithm is not convex and has lots of local minimal points.

Q. Assume we have a set of data from patients who have visited UPMC hospital during the year 2011. A set of features (e.g., temperature, height) have been also extracted for each patient. Our goal is to decide whether a new visiting patient has any of diabetes, heart disease, or Alzheimer (a patient can have one or more of these diseases). midterm_solutions

(a) [3 points] We have decided to use a neural network to solve this problem. We have two choices: either to train a *separate* neural network for each of the diseases or to train a single neural network with one output neuron for each disease, but with a shared hidden layer. Which method do you prefer? Justify your answer.

(b) [3 points] Some patient features are expensive to collect (e.g., brain scans) whereas others are not (e.g., temperature). Therefore, we have decided to first ask our classification algorithm to predict whether a patient has a disease, and if the classifier is 80% confident that the patient has a disease, then we will do additional examinations to collect additional patient features In this case, which classification methods do you recommend: neural networks, decision tree, or naive Bayes? Justify your answer in one or two sentences.

Solution:

1- Neural network with a shared hidden layer can capture dependencies between diseases. It can be shown that in some cases, when there is a dependency between the output nodes, having a shared node in the hidden layer can improve the accuracy.
2- If there is no dependency between diseases (output neurons), then we would prefer to have a separate neural network for each disease.

Solution:

We expect students to explain how each of these learning techniques can be used to output a confidence value (any of these techniques can be modified to provide a confidence value). In addition, Naive Bayes is preferable to other cases since we can still use it for classification when the value of some of the features are unknown.

We gave partial credits to those who mentioned neural network because of its non-linear de- cision boundary, or decision tree since it gives us an interpretable answer.

(c) Assume that we use a logistic regression learning algorithm to train a classifier for each disease. The classifier is trained to obtain MAP estimates for the logistic regression weights W . Our MAP estimator optimizes the objective

$$W \leftarrow \arg\max_W \ln[P(W)\prod_l P(Y^l|X^l,W)]$$

where l refers to the lth training example. We adopt a Gaussian prior with zero mean for the weights $W = \langle w_1 \ldots w_n \rangle$, making the above objective equivalent to:

$$W \leftarrow \arg\max_W \ -C\sum_i w_i + \sum_l \ln P(Y^l|X^l,W)$$

Note C here is a constant, and we re-run our learning algorithm with different values of C. Please answer each of these true/false questions, and explain/justify your answer in no more than 2 sentences.

i. [2 points] The average log-probability of the *training data* can never increase as we increase C. True False

Solution:

True. As we increase C, we give more weight to constraining the predictor. Thus it makes our predictor less flexible to fit to training data (over constraining the predictor, makes it unable to fit to training data).

ii. [2 points] If we start with C = 0, the average log-probability of *test data* will likely decrease as we increase C.

True False

iii. [2 points] If we start with a very large value of C, the average log-probability of *test data* can never decrease as we increase C.

True False

Solution:

False. As we increase the value of C (starting from C = 0), we avoid our predictor to over fit to training data and thus we expect the accuracy of our predictor to be increased on the test data.

Solution:

False. Similar to the previous parts, if we over constraint the predictor (by choosing very large value of C), then it wouldn't be able to fit to training data and thus makes it to perform worst on the test data.

5.2 Regularized Neural Networks

[8 points]

One method for preventing the neural networks' weights from overfitting is to add regularization terms. You will now derive the update rules for the regularized neural network.
Note: $Y = out(x)$

Recall that the non-regularized gradient descent update rule for w_1 is:

$$w_1^{t+1} \leftarrow w_1^t + \eta \sum_j \left[\left(y^{(j)} - out(x^{(j)}) \right) out(x^{(j)})(1 - out(x^{(j)})) * V_1(x^{(j)}) \right] \tag{1}$$

[4 points] Derive the update rule for w_1 in the regularized neural net loss function which penalizes based on the square of each weight. Use λ to denote the magic regularization parameter.

★ **SOLUTION:** The regularization term is $\lambda(\sum_i w_i^2)$. Differentiating with respect to w_1 yields $2\lambda w_1$. The update rule is

$$w_1^{t+1} \leftarrow w_1^t + \eta(\sum_j \left[\left(y^{(j)} - out(x^{(j)}) \right) out(x^{(j)})(1 - out(x^{(j)})) * V_1(x^{(j)}) \right] - 2\lambda w_1)$$

[4 points] Now, re-express the regularized update rule so that the only difference between the regularized setting and the unregularized setting above is that the old weight w_1^t is scaled by some constant. Explain how this scaling prevents overfitting.

★ **SOLUTION:**

$$w_1^{t+1} \leftarrow w_1^t(1 - 2\eta\lambda) + \eta \sum_j \left[\left(y^{(j)} - out(x^{(j)}) \right) out(x^{(j)})(1 - out(x^{(j)})) * V_1(x^{(j)}) \right]$$

At each update the weight is kept closer to zero by the $(1 - 2\eta\lambda)$ term. This prevents the weights from becoming very large, which corresponds to overfitting.