1 Neural Networks: Learning and Representation [30 pt, Amr]

As we discussed in class, one important question we need to ask when learning about a new classifier is what kind of decision boundaries can this classifier learn. We have explored this question in HW1 for the case of 1-KNN and decision trees, and showed that both of them can vary their decision boundary either on a data-driven way for 1-KNN or based on the size of the tree for decision trees. In this question we will explore these issues for the case of a 2-layer Neural Network (NN). Recall from class that the input a_j for a node j is given by:

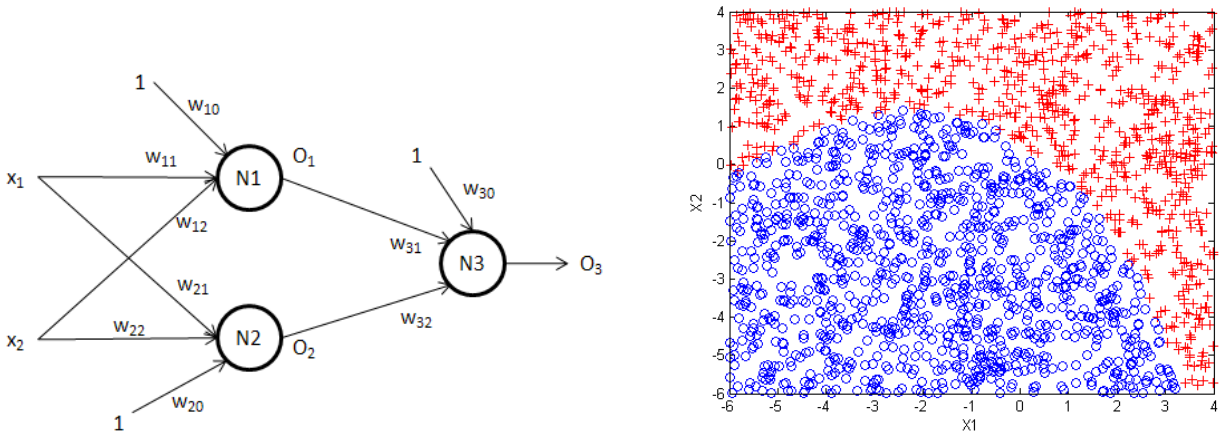
$$a_j = \sum_i w_{ji} o_i$$

Where, w_{ji} is the weight from unit i to unit j, and o_i is the activation/output of unit i. The activation of unit i is the output of a logistic function in this problem (although any differentiable function is allowed):

$$o_i = \sigma(a_i) = \frac{1}{1 + \exp^{(-a_i)}}$$

1.1 Decision Boundary

Consider the classification task shown in figure 1 where '+' and '*o*' denotes positive and negative classes, respectively. This data is available in the file `data.mat, also please read NN_readme.txt which contains a simple code to draw the figure below`. For this part of the problem, you might want to write really a few lines of matlab code to evaluate the network and get the necessary plots. Consider the 2-layer network in Figure 1. This network has 9 weights and a logistic activation function for both the hidden and output layers.



(a) (b)

Figure 1: (a) 2-layer NN with logistic activation functions at both the hidden and output layers. (b) A 2-class dataset: '+' and 'o' marks positive and negative labels respectively.

(1) **[2 points]** For each of the following classifiers, state with a one-line explanation whether or not they can learn the decision boundary illustrated in Figure 1: 1-KNN, decision tree, Naive

Bayes, and logistic regression.

Solution: NB and LR can only learn linear decision boundaries thus they can not learn the decision boundary in Figure 1. 1-KNN can learn that boundary. For decision trees, you can argue either way, if you consider the approach we discussed in HW1, then if you don't constrain the depth of the tree, then definitely DT can learn that boundary with a very large tree (which is not practical), thus you can argue that using a bounded-size tree, the decision boundary in Figure 1 can not be learned.

(2) **[1 point]** Express o_1 and o_2 in terms of $x_1, x_2, w_{10}, w_{11}, w_{12}, w_{20}, w_{21}, w_{22}$.

Solution: $o1 = \sigma(w_{10} + w_{11}x_1 + w_{12}x_2)$ $o_2 = \sigma(w_{20} + w_{21}x_1 + w_{22}x_2)$

(3) **[1 point]** Write down the decision rule for this 2-layer NN classifier.

Solution: If $o_3 >= .5$ then predict class one otherwise predict class 0;

(4) Consider the following set of weights: $w_{10} = -.8; w_{11} = .8; w_{12} = .1; w_{20} = .3; w_{21} = .3; w_{22} = -.4; w_{31} = 1; w_{32} = -1; w_{30} = .2;$

(a) **[2 points]** Draw the representation of the data after the hidden layer, i.e. your dimensions will be o_1 and o_2, and you should label each point with its ground-truth label. State your observation.

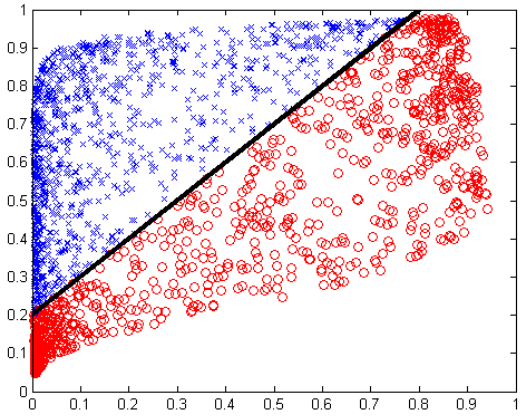


Figure 2:

The data becomes linearly separable in this space.

(b) **[2 points]** Draw the classification result for this 2-layer NN, i.e. re-draw the scatter plot in Figure 1 but label each point as classified by the 2-layer NN. Compare to the ground-truth and state your observations.

Solution: same as in Figure 1.b. Since the final layer is just a logistic regression classifier whose input is (o_1, o_2), the correct classifier can be learned. The boundary in terms

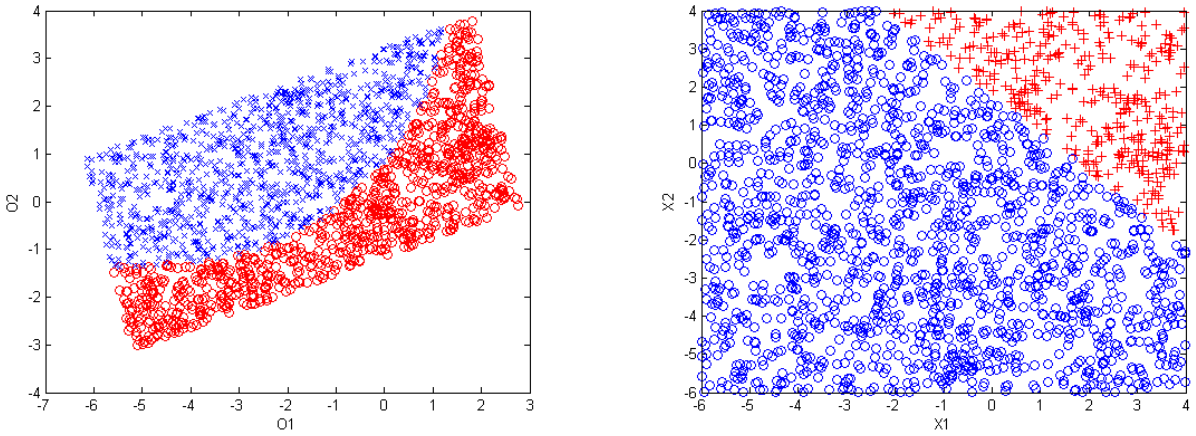
of the input space (x_1, x_2) is non-linear though.

(c) **[2 points]** Overlay the decision boundary **explicitly** over the curve your drew in part (a) in terms of o_1 and o_2. State your observations.

Solution: Note that the last layer is a logistic regression classifier over the input (o_1, o_2), thus the decision rule is: $w_{31}o_1 + w_{32}o_2 + w_{30} = 0$. Which you can easily draw as in Figure 2.

(e) Lets assume that we removed the logistic function form the hidden layer ONLY and instead used an identity function, i.e $o_1 = a_1$ and $o_2 = a_2$, while maintaining the same weight values.

(i) **[2 points]** repeat (a) and (b) and state your observations.



(a) (b)

Figure 3: (a) After first layer. (b) prediction made by the modified NN

Solution:
See figure 3. The data is not linearly separable in terms of (o_1, o_2) thus the final logistic unit can not learn the correct decision boundary. In terms of the input space (x_1, x_2), as shown in figure 3, we got the wrong decision boundary. In fact, this network, as we will show in the next part, can only learn a linear decision boundary over (x_1, x_2) and as shown in figure 3.b, the network learned a liner decision boundary (which is not correct). Note that this is not the best linear boundary that this network can learn, in other words, you can optimize the weights to get a better linear decision boundary, but the network can not still learn the correct decision boundary, nor learn a linear decision boundary that does a good job on this dataset (in terms of classification accuracy)

(ii) **[2 points]** Can you tweak the weights in this case to learn the correct decision boundary? If yes, then find such weights and repeat (i), if NO, then re-express the network in this case using a simpler network (or classifier) and argue why it can not learn this decision boundary.

Solution: The answer is NO, since you can simplify the 2-layer NN into a single-layer NN with a logistic unit (i.e. a logistic regression classifier) which can only a linear decision boundary over (x_1, x_2). To see this:

$$\begin{aligned} o_3 &= \sigma(w_{30} + w_{31}o_1 + w_{32}o_2) \\ &= \sigma\Big(w_{30} + w_{31}(w_{10} + w_{11}x_1 + w_{12}x_2) + w_{32}(w_{20} + w_{21}x_1 + w_{22}x_2)\Big) \\ &= \sigma\Big(w_{30} + w_{31}w_{10} + w_{32}w_{20} + (w_{31}w_{11} + w_{32}w_{21})x_1 + (w_{31}w_{12} + w_{32}w_{22})x_2\Big) \end{aligned}$$

Which is just a logistic regression classifier that can learn ONLY a linear decision boundary over the space (x_1, x_2)

1.2 Backpropagation

In class we discussed that in order to train a NN we need to define an error function E$[W]$ (such as the squared error) that can be minimized using the backpropagation algorithm to find the network weights. We also discussed that this error function can be driven using the M(C)LE principle based on a signal plus noise interpretation in the context of a regression setting. Consider a classification task with Data $\mathcal{D} = (\mathbf{X}, \mathbf{t}) = \{(\mathrm{x}^i, t^i)\}, i = 1, 2, ..., N,$. For example, $\mathbf{x}^i$ might be a face image, and t^i is a binary label equals 0 if the face is for a male and 1 if the face is for a female. Now consider a 2-layer NN based on logistic threshold units at both the hidden and output layers. If we let y denote the real-valued final output of the network, where $y \in [0, 1]$, then we might naturally wish to interpret this output as the probability that the boolean class label t takes on the value $t = 1$; that is, $y = P(t = 1|x; W)$. In this case, as we have done in logistic regression, it is natural to find the NN weights W using the M(C)LE principle as follows:

$$\mathbf{W}_{\mathrm{MLE}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p(t^i|\mathrm{x}^i; \mathbf{w}) \tag{1}$$

(1) **[2 points]** Show that maximizing (1) is equivalent to minimizing the cross-entropy error function given by:

$$\mathrm{E}[W] = -\left[\sum_{i=1}^{N} t^i \ln y^i + (1 - t^i)\ln(1 - y^i)\right]$$

where, y^i is the output of the network corresponding to example i.

Solution:

$$\begin{aligned}
\mathbf{W}_{\text{MLE}} &= \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p(t^i|\mathrm{x}^i;\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \log \prod_{i=1}^{N} p(t^i|\mathrm{x}^i;\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \log \prod_{i=1}^{N} y^{i t^i} (1-y^i)^{(1-t^i)} \qquad (2) \\
&= \arg\max_{\mathbf{w}} \sum_{i=1}^{N} t^i \log y^i + (1-t^i)\log(1-y^i) \\
&= \arg\min_{\mathbf{w}} -\left[\sum_{i=1}^{N} t^i \log y^i + (1-t^i)\log(1-y^i) \right]
\end{aligned}$$

To see why (2) is correct, note that $p(t^i = 1|\mathrm{x}^i;\mathbf{w}) = y^i$ and $p(t^i = 0|\mathrm{x}^i;\mathbf{w}) = 1 - y^i$, which you can double check that (2) will satisfy.

Now we will drive the weight update rule using a stochastic gradient decent using a random example n at each step, thus we will add n to the error function, E_n to make this dependency explicit. The generic update rule for a weight w_{ji} coming from unit (or input i) to unit j is given by:

$$w_{ji}^{\text{new}} = w_{ji}^{\text{old}} - \eta \frac{\partial \mathrm{E}_n}{\partial w_{ji}^{\text{old}}}$$

where η is the learning rate. Lets denote the output unit as k, therefore, $o_k = y$. For this part of the problem, assume there are no bias terms.

(2) **[2 points]** Define $\delta_j = \frac{\partial \mathrm{E}_n}{\partial a_j}$. Show that $\frac{\partial \mathrm{E}_n}{\partial w_{ji}} = \delta_j o_i$

Solution:

$$\begin{aligned}
\frac{\partial \mathrm{E}_n}{\partial w_{ji}} &= \frac{\partial \mathrm{E}_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \\
&= \delta_j \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} o_i \\
&= \delta_j o_i
\end{aligned}$$

(3) **[4 points]** Show that for the output unit $\delta_k = y - t$, where t is the correct label, and $o_k = y \in [0, 1]$ is the network prediction.

Solution:

$$\begin{aligned}
\delta_k = \frac{\partial \mathrm{E}_n}{\partial a_k} &= \frac{\partial}{\partial a_k} - (t\log y + (1-t)\log(1-y)) \\
&= -\left(\frac{t}{y}\frac{\partial}{\partial a_k}y + \frac{(1-t)}{(1-y)}\frac{\partial}{\partial a_k}(1-y)\right) \qquad (3) \\
&= -\left(\frac{t}{\sigma(a_k)}\sigma(a_k)(1-\sigma(a_k)) + \frac{(1-t)}{(1-\sigma(a_k))}(-1)\sigma(a_k)(1-\sigma(a_k))\right) \\
&= -\left(t(1-\sigma(a_k)) - (1-t)\sigma(a_k)\right) \\
&= -\left(t^k - t\sigma(a_k)) - \sigma(a_k) + \sigma(a_k)t)\right) \\
&= -(t-\sigma(a_k)) \qquad (4) \\
&= y - t
\end{aligned}$$

Note that in (3) we substituted $y = \sigma(a_k)$ and used the fact that the derivative of $\sigma(.)$ is $\sigma(.)(1-\sigma(.))$. In (4) we substituted $\sigma(a_k) = y$.

(4) **[2 points]** Write down the update rule for a weight between the output unit k and a hidden unit j.

Solution:

$$\begin{aligned}
w_{kj}^{\text{new}} &= w_{kj}^{\text{old}} - \eta\frac{\partial \mathrm{E}_n}{\partial w_{kj}^{\text{old}}} \\
&= w_{kj}^{\text{old}} - \eta\delta_k o_j \\
&= w_{kj}^{\text{old}} - \eta(y-t)o_j
\end{aligned}$$

(5) **[4 points]** For a hidden unit j, use the chain rule to show that, $\delta_j = \frac{\partial \mathrm{E}_n}{\partial a_j} = \sigma(a_j)(1-\sigma(a_j))(y-t)w_{kj}$. [**Hint**: $\frac{\partial \mathrm{E}_n}{\partial a_j} = \frac{\partial \mathrm{E}_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$, where k is the output unit.]

Solution:

$$\begin{aligned}
\delta_j = \frac{\partial \mathrm{E}_n}{\partial a_j} &= \frac{\partial \mathrm{E}_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\
&= \delta_k\frac{\partial a_k}{\partial a_j} \\
&= \delta_k\frac{\partial}{\partial a_j}(\sum_j w_{kj}o_j) \\
&= \delta_k\frac{\partial}{\partial a_j}(\sum_j w_{kj}\sigma(a_j)) \\
&= \delta_k w_{kj}\frac{\partial}{\partial a_j}\sigma(a_j) \\
&= \delta_k w_{kj}\sigma(a_j)(1-\sigma(a_j)) \\
&= (y-t)w_{kj}\sigma(a_j)(1-\sigma(a_j))
\end{aligned}$$

(6) **[2 points]** Write down the update rule for a weight between the hidden unit j and input x_i.

Solution:

$$\begin{aligned} w_{ji}^{\text{new}} &= w_{ji}^{\text{old}} - \eta \frac{\partial \mathrm{E}_n}{\partial w_{ji}^{\text{old}}} \\ &= w_{ji}^{\text{old}} - \eta \delta_j x_i \\ &= w_{ji}^{\text{old}} - \eta \Big[(y-t) w_{kj} \sigma(a_j)(1-\sigma(a_j)) \Big] x_i \end{aligned} \tag{5}$$