Assignment #2

Name: Kuei-Yu Tsai

NetID: kxt230002

1

1.1

Backpropagation Algorithm

Initialize all weights to small random numbers

Until convergence, Do

For each training example, Do

1. *Input it to network and compute network outputs*
2. *For each output unit k, we used the derivative of the sigmoid function*

$\sigma(x) = \frac{1}{1+e^{-x}}$, *which is* $\sigma'(x) = \sigma(x)\big(1-\sigma(x)\big)$, *and have the result*:

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

3. *For each hidden unit h, we also used the derivative function*

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{h,k}\delta_k$$

4. *Update each network weight* $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}, where\ \Delta w_{i,j} = \eta\delta_j x_{i,j}$$

a.

Revised Backpropagation Algorithm with tanh activation function

Initialize all weights to small random numbers

Until convergence, Do

For each training example, Do

1. *Input it to network and compute network outputs*
2. *For each output unit k, we used the derivative of the* tanh *function*

$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, *which is* $tanh'(x) = 1 - tanh^2(x)$

$$\delta_k \leftarrow (1-tanh^2(o_k))(t_k-o_k)$$

3. *For each hidden unit h*

$$\delta_h \leftarrow (1-tanh^2(o_h))\sum_{k\in outputs} w_{h,k}\delta_k$$

4. *Update each network weight* $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}, where\ \Delta w_{i,j} = \eta\delta_j x_{i,j}$$

b.

Revised Backpropagation Algorithm with ReLu activation function

Initialize all weights to small random numbers

Until convergence, Do

For each training example, Do

1. *Input it to network and compute network outputs*
2. *For each output unit k, we used the derivative of the ReLu function $ReLu(x) = \max(0, x)$,*
   however, the derivative is not continuous at $x = 0$, so we need to consider the derivative of ReLu seperately for positive and non − positive values of x.

$$\delta_k \leftarrow \begin{cases} 0 & if\ o_k \le 0 \\ (t_k - o_k) & if\ o_k > 0 \end{cases}$$

3. *For each hidden unit h*

$$\delta_h \leftarrow \begin{cases} 0 & if\ o_h \le 0 \\ \sum_{k \in outputs} w_{h,k}\delta_k & if\ o_h > 0 \end{cases}$$

4. *Update each network weight $w_{i,j}$*

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}, where\ \Delta w_{i,j} = \eta \delta_j x_{i,j}$$

1.2
Derivation

1. *Compute the error:*

$$E = \frac{1}{2}(t - o)^2, where\ t\ is\ the\ target\ output$$

2. *Compute the gradient of the error with respect to each weight parameter:*

$$\frac{\partial E}{\partial w_i} = -(t - o)\frac{\partial o}{\partial w_i} = -(t - o)\frac{\partial}{\partial w_i}\left(w_0 + w_1(x_1 + x_1^2) + \cdots + w_n(x_n + x_n^2)\right)$$
$$= -(t - o)\left(x_i + x_i^2\right)$$

3. *Update rule for each weight parameter:*

$$w_i \leftarrow w_i - \eta\frac{\partial E}{\partial w_i} \leftarrow w_i + \eta(t - o)\left(x_i + x_i^2\right)$$

1.3

a. $y_5 = h(w_{53}h(w_{31}x_1 + w_{32}x_2) + w_{54}h(w_{41}x_1 + w_{42}x_2))$

b. $Y = h(W^{(2)}h(W^{(1)}X))$

c. $h_s(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-x}} * \frac{e^x+e^{-x}}{e^x+e^{-x}} = \frac{e^x+e^{-x}}{e^x+e^{-x}+e^x+e^{-x}} = \frac{1+e^{-2x}}{1+e^{-x}+e^x+e^{-x}} = \frac{1}{2} + \frac{1}{2} * \frac{e^{-2x}}{1+e^{-x}+e^x+e^{-x}}$

$= \frac{1}{2} + \frac{1}{2} * h_t(2x)$

Now, by substituting this relationship into the output function, we can see that neural nets using the sigmoid and tanh activation functions will generate the same function, differing only by linear transformations and constants.

2.
dataset:
https://archive.ics.uci.edu/dataset/545/rice+cammeo+and+osmancik

The analysis supposes that the dataset is an accurate representation of the population it seeks to model. Any inherent biases or inconsistencies within the dataset could potentially impact the performance of the neural network models. Moreover, the examination of parameters was limited to a predetermined range, suggesting that there may be other permutations that enhance model efficiency.
In summary, the results indicate that the Sigmoid activation function generally outperformed Tanh and ReLu for the classification task at hand. Lower learning rates tended to produce more stable and accurate models, while larger hidden layer sizes contributed to improved performance, particularly with the sigmoid activation function. Furthermore, up to a certain point, more iterations improved the model's convergence; after that, only slight gains were seen. These insights shed light on the pivotal role of hyperparameters in shaping the efficacy of neural network models, underscoring the need for systematic exploration and optimization to achieve optimal performance.