CS 188 Spring 2022 Introduction to Artificial Intelligence — Written HW 10

Due: Friday 04/22/2022 at 10:59pm (submit via Gradescope).

Policy: Can be solved in groups (acknowledge collaborators) but must be written up individually

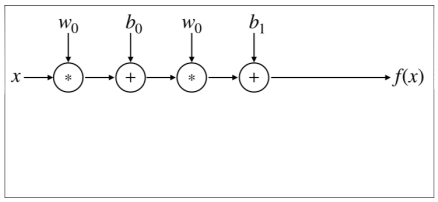
Submission: It is recommended that your submission be a PDF that matches this template. You may also fill out this template digitally (e.g. using a tablet). **However, if you do not use this template, you will still need to write down the below four fields on the first page of your submission.**

First name	Qing jing
Last name	Zhang
SID	3037581086
Collaborators	None

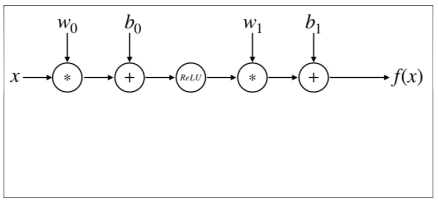
For staff use only:

Q1.	Neural Networks and MLE	/38
	Total	/38

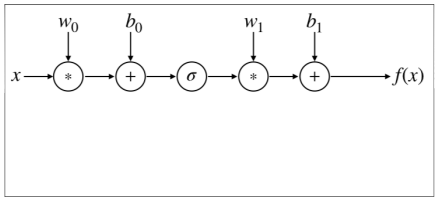
Q1. [38 pts] Neural Networks and MLE



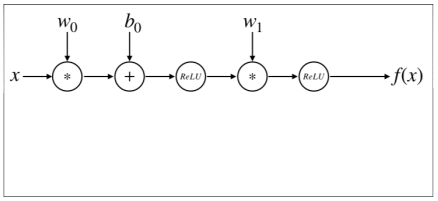
(a) Neural Network 1



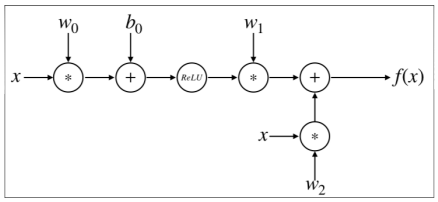
(b) Neural Network 2



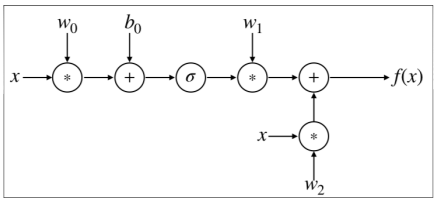
(c) Neural Network 3



(d) Neural Network 4



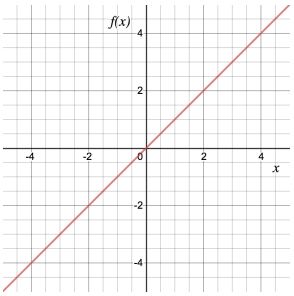
(e) Neural Network 5



(f) Neural Network 6

(a) We first investigate what functions different neural network architectures can represent. For each of the six following graphs, select the neural networks that can represent the function **exactly** on the range $x \in (-\infty, \infty)$. In the networks above, *ReLU* represents the rectified linear unit and σ represents the sigmoid function: $ReLU(x) = \max(0, x)$, $\sigma(x) = \frac{1}{1+e^{-x}}$.

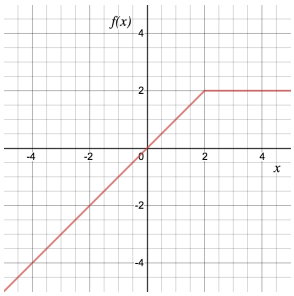
(i) [3 pts]



$y = x$

- [x] Neural Network 1
- [ ] Neural Network 2
- [ ] Neural Network 3
- [ ] Neural Network 4
- [ ] Neural Network 5
- [ ] Neural Network 6
- ( ) None of the above

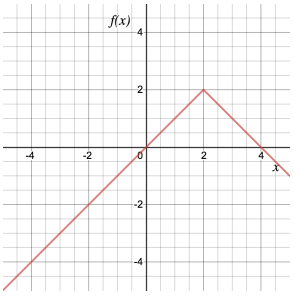
(ii) [3 pts]



$y = \begin{cases} x & x \leq 2 \\ 2 & x > 2 \end{cases}$

- [ ] Neural Network 1
- [x] Neural Network 2
- [ ] Neural Network 3
- [ ] Neural Network 4
- [x] Neural Network 5
- [ ] Neural Network 6
- ( ) None of the above

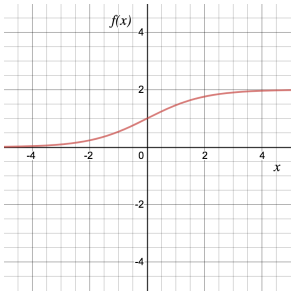
(iii) [3 pts]



$y = \begin{cases} x & x \leq 2 \\ 2-(x-2) & x > 2 \end{cases}$

- [ ] Neural Network 1
- [ ] Neural Network 2
- [ ] Neural Network 3
- [ ] Neural Network 4
- [x] Neural Network 5
- [ ] Neural Network 6
- ( ) None of the above

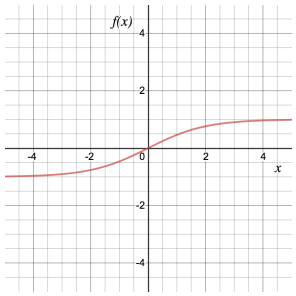
(iv) [3 pts]



$y = \frac{2}{1+e^{-x}}$

- [ ] Neural Network 1
- [ ] Neural Network 2
- [x] Neural Network 3
- [ ] Neural Network 4
- [ ] Neural Network 5
- [ ] Neural Network 6
- ( ) None of the above

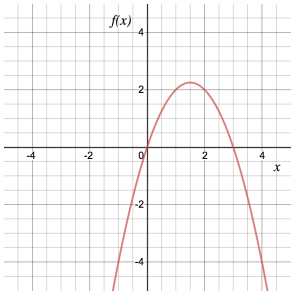
(v) [3 pts]



- ☐ Neural Network 1
- ☐ Neural Network 2
- ☑ Neural Network 3
- ☐ Neural Network 4
- ☐ Neural Network 5
- ☑ Neural Network 6
- ○ None of the above

$y = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ $= 1 - \frac{2}{e^{2x}+1}$

(vi) [3 pts]



- ☐ Neural Network 1
- ☐ Neural Network 2
- ☐ Neural Network 3
- ☐ Neural Network 4
- ☐ Neural Network 5
- ☐ Neural Network 6
- ○ None of the above

$y = 3x - x^2$

(b) Now we'll try to utilize a neural network to play Go! Recall from Homework 1 Question 4 that we can estimate the value of certain states by playing random simulated games starting from those states. Now we will go one step further and try to learn the simulated games to learn a utility function, which takes in state (board configuration) as input. Once we've acquired enough data to learn this utility function, we can use this learned function to estimate the value of new states that we've never simulated games from before! We will represent this utility function as a neural network and learn the parameters of the network by gradient descent on the collected data (i.e. simulated games).

(i) [4 pts] Given some training data, for what kinds of states would we expect this learned utility function to yield accurate estimates? Conversely for what kinds of states would we expect this learned utility function to yield poor estimates? *Hint: Do not over-think this question. There are multiple correct answers*

the States that covers all possibles future steps or state that may occurs in GO competition

the state that only cover fews fixed future steps

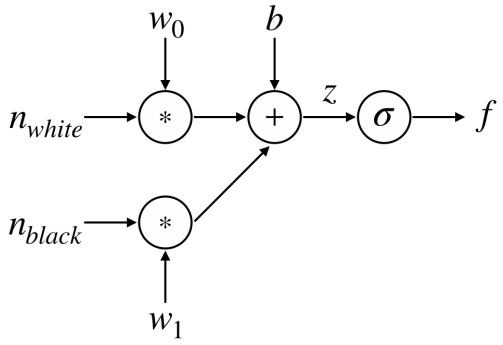
(ii) [4 pts] We can also recognize that many states may look different but have equal utility (for example a board can be rotated 90 degrees and represent a different state, but have the same value). To make use of this insight, we will featurize the states and try to learn a utility function with those features as inputs.

What are potential benefits of learning a utility function as a function of hand chosen features instead of from raw state? What are potential cons? Name at least one benefit and at least one con.

pros: chosen feature reduce the dimension of inputs and then the efficiency improved.

cons: if feature chosen is not irrelavent, then the outcome can be bad.

(iii) [6 pts] Suppose we choose our feature vector to be $x = [n_{white}, n_{black}]$, the number of white and black pieces on the board, respectively and we design our 1-layer neural network $f(x;\theta) = \sigma(z)$, where $z = w_0 n_{white} + w_1 n_{black} + b$, $\theta = [w_0, w_1, b]$ and σ is the sigmoid function.



We preprocess the data and organize it by features. The organized data comes in the form of k tuples: (x_i, n_i, m_i) for $i = 1, \ldots, k$, where x_i is the feature vector, n_i is the number of games played starting from states with those features, and m_i is the number of games won of those n_i games.

We decide that a reasonable utility function should be the probability of success and the output of our neural network will be the the probability p of winning each game starting from the given state. Our goal is to learn the parameters θ of $p = f(x;\theta)$ which outputs the most likely p for a given feature vector x, given our training data.

What is the log likelihood function that we are trying to maximize? Keep your answer in terms of $n_i, m_i, x_i, f(x_i;\theta)$. *Hint: You may want to use rules for logs to expand out the log-likelihood to make the next part easier.*

$$L = f(x_i;\theta)^{m_i} \cdot [1 - f(x_i;\theta)]^{(n_i - m_i)}$$

$$ll = \log L = m_i \log f(x_i;\theta) + (n_i - m_i) \log (1 - f(x_i;\theta))$$

$f = w_0 n_{white} + w_1 \cdot n_{black} + b_0$ $\quad \frac{d\delta}{dz} = \delta(1-\delta)$

(iv) [6 pts] Compute the partial derivatives $\frac{\partial ll}{\partial w_0}, \frac{\partial ll}{\partial w_1}, \frac{\partial ll}{\partial b_0}$ for one training example (x_i, n_i, m_i) where ll is the log likelihood function from the previous part. Feel free to include intermediate terms such as $f(x_i;\theta)$ and z in your answer. *Hint: Use chain rule,* $\frac{\partial ll}{\partial w_0} = \frac{dll}{df}\frac{df}{dz}\frac{\partial z}{\partial w_0}$. *The other two partial derivatives should follow very similar computation.*

$$\frac{dll}{dw_0} = \frac{dll}{df} \cdot \frac{df}{dz} \cdot \frac{dz}{dw_0}$$

$$= m_i \cdot \frac{1}{f(x_i;\theta)} z(1-z) \cdot n_{white} + (n_i - m_i) \frac{1}{1-f(x_i;\theta)} z(1-z) n_{white}$$

$$= \left(\frac{m_i}{f(x_i;\theta)} + \frac{n_i - m_i}{1-f(x_i;\theta)}\right) z(1-z) \cdot n_{white}$$

$$\frac{dll}{dw_1} = \frac{dll}{df} \cdot \frac{df}{dz} \cdot \frac{dz}{dw_1}$$

$$= \left(\frac{m_i}{f(x_i;\theta)} + \frac{n_i - m_i}{1-f(x_i;\theta)}\right) z(1-z) \cdot n_{black}$$

$$\frac{dll}{db_0} = \frac{dll}{df} \cdot \frac{df}{dz} \cdot \frac{dz}{db_0}$$

$$\left(\frac{m_i}{f(x_i;\theta)} + \frac{n_i - m_i}{1-f(x_i;\theta)}\right) z(1-z)$$