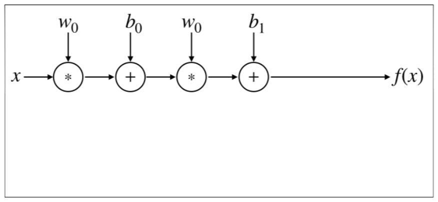
CS 188 Spring 2024 Introduction to Artificial Intelligence

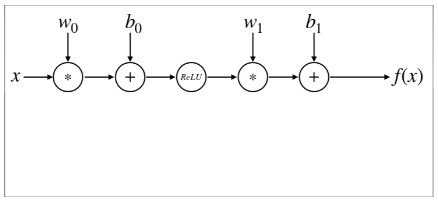
HW 9B Sol.

Solutions for HW 9B

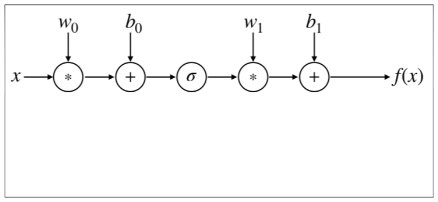
Q1. [38 pts] Neural Networks and MLE



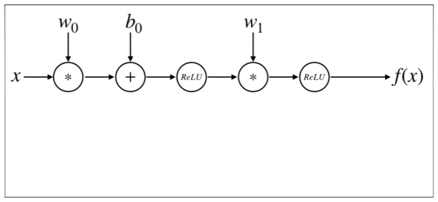
(a) Neural Network 1



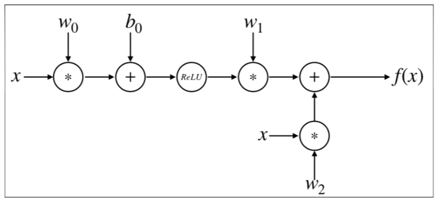
(b) Neural Network 2



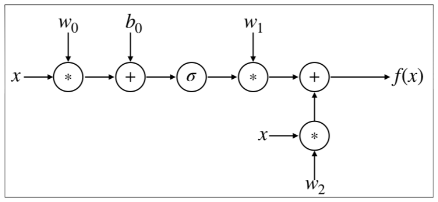
(c) Neural Network 3



(d) Neural Network 4



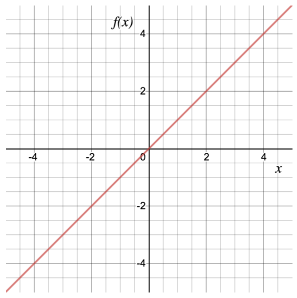
(e) Neural Network 5



(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: $ReLU(x) = \max(0, x)$, $\sigma(x) = \frac{1}{1+e^{-x}}$.

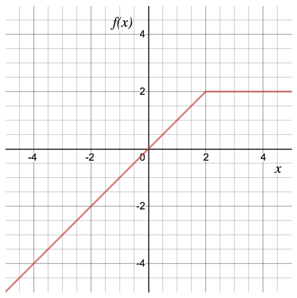
(i) [3 pts]



$y = x$

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

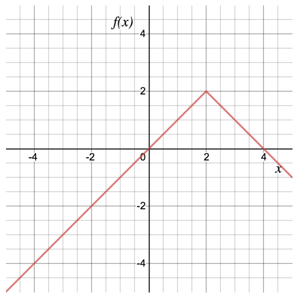
(ii) [3 pts]



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

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

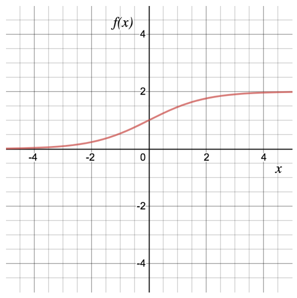
(iii) [3 pts]



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

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

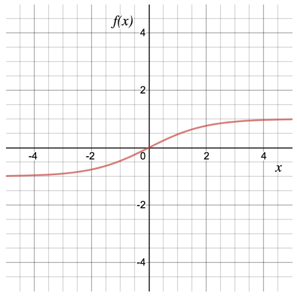
(iv) [3 pts]



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

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

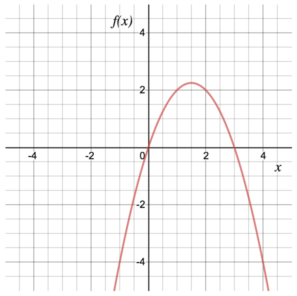
(v) [3 pts]



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

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

(vi) [3 pts]



$y = 3x - x^2$

- ☐ Neural Network 1
- ☐ Neural Network 2
- ☐ Neural Network 3
- ☐ Neural Network 4
- ☐ Neural Network 5
- ☐ Neural Network 6
- [✓ None of the above]

(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* We would expect accurate estimates for states that are well represented in the training data (or "look like" states that are well represented in the training data), and inaccurate estimates for states that we have little training data on. Also states that are close to winning states would likely have more accurate estimates since it is easier to estimate the value of later states than earlier states.

(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.

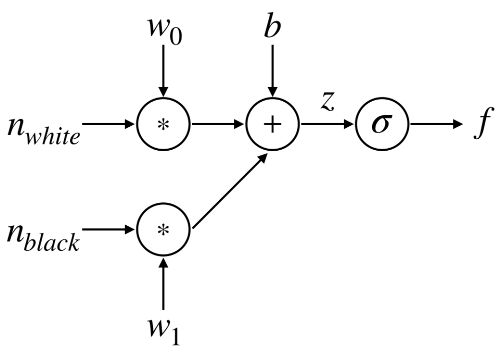
Benefits:

1. Allows us to be more data efficient by only focusing on information that will be relevant for likelihood of winning and ignoring irrelevant information such as orientation of the board.
2. May make learning faster / easier since our data is more compressed and lower-dimensional

Cons:

1. We may unknowingly choose to exclude information that could be useful for determining win rates (e.g. perhaps we don't think orientation is important but in reality our opponent gets flustered when all the pieces are far away from them and plays better when the pieces are closer to them).
2. We may not know what good features to pick are.
3. It may be difficult to extract features that are useful (e.g. image classification).

(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.*

$$\max_\theta ll(x,n,m;\theta) = \max_\theta \log p(x,n,m;\theta) = \max_\theta \log \prod_{i=1}^{k} p(x_i,n_i,m_i;\theta) = \max_\theta \sum_{i=1}^{k} \log p(x_i,n_i,m_i;\theta)$$

$$= \max_\theta \sum_{i=1}^{k} \log \left(\binom{n_i}{m_i} f(x_i;\theta)^{m_i} (1 - f(x_i;\theta))^{n_i - m_i} \right)$$

$$= \max_\theta \sum_{i=1}^{k} \log \binom{n_i}{m_i} + m_i \log f(x_i;\theta) + (n_i - m_i) \log(1 - f(x_i;\theta))$$

$$= \max_\theta \sum_{i=1}^{k} m_i \log f(x_i;\theta) + (n_i - m_i) \log(1 - f(x_i;\theta))$$

(Note: it's fine to entirely leave out $\binom{n_i}{m_i}$. This shouldn't cost the student any points.)

(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.*

$$\frac{d}{df} ll(x_i, n_i, m_i; \theta) = \frac{m_i}{f(x_i;\theta)} - \frac{n_i - m_i}{1 - f(x_i;\theta)}$$

$$\frac{df}{dz} = \frac{d}{dz}\sigma(z) = \sigma(z)(1-\sigma(z)) = \sigma(z)(1-\sigma(z))$$

$$\frac{\partial z}{\partial w_0} = n_{white}$$

$$\frac{\partial z}{\partial w_1} = n_{black}$$

$$\frac{\partial z}{\partial b} = 1$$

$$\frac{\partial ll}{\partial w_0} = \frac{dll}{df}\frac{df}{dz}\frac{\partial z}{\partial w_0} = \left(\frac{m_i}{f(x_i;\theta)} - \frac{n_i - m_i}{1 - f(x_i;\theta)}\right)\sigma(z)(1-\sigma(z))n_{white}$$

$$\frac{\partial ll}{\partial w_1} = \frac{dll}{df}\frac{df}{dz}\frac{\partial z}{\partial w_1} = \left(\frac{m_i}{f(x_i;\theta)} - \frac{n_i - m_i}{1 - f(x_i;\theta)}\right)\sigma(z)(1-\sigma(z))n_{black}$$

$$\frac{\partial ll}{\partial b} = \frac{dll}{df}\frac{df}{dz}\frac{\partial z}{\partial b} = \left(\frac{m_i}{f(x_i;\theta)} - \frac{n_i - m_i}{1 - f(x_i;\theta)}\right)\sigma(z)(1-\sigma(z))$$