National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester 2, 2024/2025

# Tutorial 8
# Perceptrons and Neural Networks

## Summary of Key Concepts

In this tutorial, we will discuss and explore the following learning points from Lecture:

1. Perceptrons
   (a) Logic Gates
2. Neural Network
   (a) Single-layer Neural Network vs Multi-layer Neural Network
   (b) Forward Pass
   (c) Intermediate Layers
3. Activation Function
   (a) Effects of using activation functions

## A Logic Gates

| AND | | |
|---|---|---|
| x1 | x2 | y |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| OR | | |
|---|---|---|
| x1 | x2 | y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| NOT | |
|---|---|
| x | y |
| 0 | 1 |
| 1 | 0 |

Figure 1: Logic Gates

1. Determine a function that can be used to model the decision boundaries of the logical NOT, OR, and AND gates. For this question, assume that AND, and OR functions take $2$ inputs while the NOT function takes a single input. What are the weights and bias for each perceptron respectively? The steps are given below:
   - Initialise all your weights to 0 (including bias term)
   - Positive class is 1 and negative class is 0
   - Use the Perceptron Update Rule and the learning rate from Lecture 9 ($\eta = 0.1$)

- Update the model for each misclassified instance in the **exact order** of data samples as **given** in the question.
- Let the activation function for this question be the Heaviside function (a modification of the sign function). The Heaviside function is defined as $\mathrm{H}(z) = 1$ if $z \geq 0$ else $0$

**Solution:**

We will use a linear function to model the decision boundary. The linear function is given by the following equation:

$$z = b + w_1 x_1 + w_2 x_2$$

*Using the Perceptron Update Rule, we will get the following weights:*

$$\begin{aligned} \text{AND Gate: } & w_1 = 0.2, w_2 = 0.1, b = -0.3 \\ \text{OR Gate: } & w_1 = 0.1, w_2 = 0.1, b = -0.1 \\ \text{NOT Gate: } & w_1 = -0.1, b = 0. \end{aligned}$$

Please refer to the jupyter notebook for the code.

2. Is it possible to model the XOR function using a single Perceptron? Refer to Figure 2 for the truth table of the XOR gate. Comment on your answer.

| XOR | | |
|---|---|---|
| x1 | x2 | y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Figure 2: XOR Gate

Hint : Try out an analogous problem at TensorFlow Playground.

Remember that the TensorFlow representation is not the exact same as the given question but is similar.

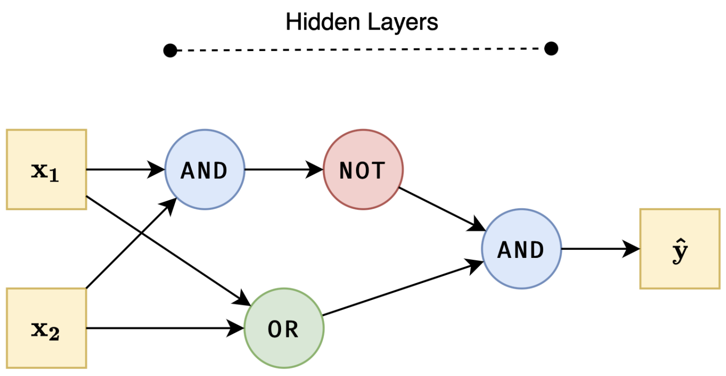
**Solution:**

Unlike the AND and OR gate, an XOR gate requires an intermediate hidden layer for preliminary transformation in order to achieve the logic of an XOR gate. This is because the XOR gate is not linearly separable.

3. Model XOR function (takes 2 inputs) using a number of perceptrons that implement AND, OR, and NOT functions. Show the diagram of the final Perceptron network. What can you conclude now that you have intermediate functions acting on the inputs and not just a single perceptron unit?

**Solution:**

$XOR(x_1, x_2) = AND(NOT(AND(x_1, x_2)), OR(x_1, x_2))$. *We resort to using these intermediate "layers" that help with extra computation that is not possible with a single perceptron. This shows us how stacking perceptrons enables us to model complex behaviors in data. Additionally, Layers are important to better generalize complex data.*



4. If we change the ordering of data samples in Perceptron Update Rule, will the model converge to a different model weight for the AND operator? What can you conclude from the observation? (Hint: Use your solution from (a), and try switching the row-orderings of the input data from $(0, 1, 2, 3)$ to $(0, 2, 3, 1)$ and $(0, 2, 1, 3)$ respectively)

**Solution:**

For ordering $(0, 2, 3, 1)$, the weight differs from the original ordering after two steps and takes more steps to converge to the same optimum model $(-0.3, 0.2, 0.1)$ at the end.

For ordering $(0, 2, 1, 3)$, it follows the same initial update steps as the original ordering but converges to a optimum model $(-0.3, 0.1, 0.2)$.

From the above observations, we can conclude that:

1. Reordering data points could help the model converge much faster.
2. Manipulating the ordering could direct the model to a different weight. There is no guarantee to converge to the same model even for such a simple gate function.

5. With regards to the AND gate in Figure 1, does your proposed model have high bias? Does it have high variance? Justify your answer.

**Solution:**

The proposed model has low bias and low variance. All these models perform as expected to the AND gate, and the model is powerful enough to learn the function. The linear model is the simplest model to perform the AND gate, and these different models are similar to each others.

## B Single Layer Neural Network vs Multi Layer Neural Network

Suppose you have a dataset of 500 product sales, each with three input features (TV, radio, and newspaper advertising expenditure) and a continuous label indicating the sales.

You decide to use a single-layer neural network and a multi-layer neural network to make predictions. After training both networks, you obtain a mean squared error of 1000 on the training set and a mean squared error of 2000 on the validation set for the single-layer neural network, and a mean squared error of 800 on the training set and a mean squared error of 1200 on the validation set for the multi-layer neural network.

1. What might be the reasons for the difference in performance between the single-layer neural network and the multi-layer neural network?

   **Solution:**

   The difference in performance between the single-layer neural network and the multi-layer neural network can be attributed to the fact that the single-layer neural network is a linear classifier, while the multi-layer neural network can learn non-linear decision boundaries. In this case, the multi-layer neural network is able to capture more complex relationships between the input features and the output label, leading to better performance on the validation set.

2. How might you modify the single-layer neural network to improve its performance, and what are the advantages and disadvantages of doing so?

   **Solution:**

   To improve the performance of the single-layer neural network, you could add more features, such as polynomial or interaction terms, to capture non-linear relationships between the input features and the output labels. However, this approach can quickly become computationally expensive and may lead to overfitting if the number of features is too high. Note that the final decision is still based on a linear combination of the transformed features.

3. What techniques could you use to improve the performance of the multi-layer neural network?

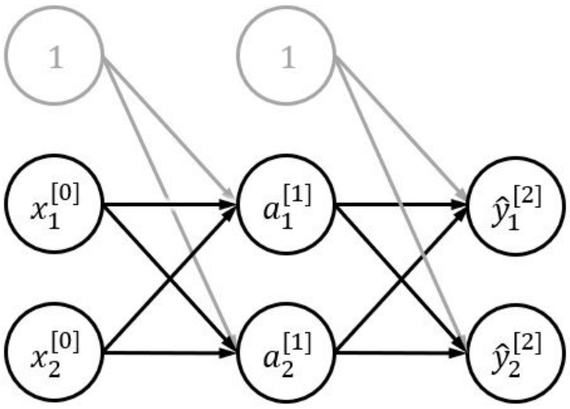
   **Solution:**

   You could try adding more hidden layers (increasing the depth of the network) or increasing the number of neurons in the hidden layer (increasing the width). However, the complexity of the computed function grows exponentially with the depth of an architecture, and increases more slowly with the width. You can find out more in the paper by Eldan and Shamir (2016) on the expressive power of deep neural networks. For further details, see this link. Hence, increasing the depth is generally more efficient. Additionally, while both approaches lead to the multi-layer neural network increasing its capacity to learn more complex patterns, it can also increase the risk of overfitting.

## C Forward Propagation

In this section, we are going to use a neural network with **a 2-D input, one hidden layer with two neurons, and two output neurons**. Additionally, the hidden neurons and the input will **include a bias**. We use **ReLU function** as the non-linear activation function. Remember: $\text{ReLU}(x) = \max(0, x)$.

Here's the basic structure:



*Disclaimer: You may use `NumPy` to compute the following!*

1. Suppose there is a data input $\mathbf{x} = (2, 3)^\top$ and the actual output label is $\mathbf{y} = (0.1, 0.9)^\top$. The weights for the network are

$$\boldsymbol{W}^{[1]} = \begin{bmatrix} 0.1 & 0.1 \\ -0.1 & 0.2 \\ 0.3 & -0.4 \end{bmatrix}, \boldsymbol{W}^{[2]} = \begin{bmatrix} 0.1 & 0.1 \\ 0.5 & -0.6 \\ 0.7 & -0.8 \end{bmatrix},$$

Of course, we also include biases of value $b = 1$ for both hidden and output layers. Calculate the following values after the forward propagation:
$\mathbf{a}^{[1]}$, $\hat{\mathbf{y}}^{[2]}$ and $L(\hat{\mathbf{y}}^{[2]}, \mathbf{y})$. Use the MSE (mean squared error) loss function.

**Solution:**

$\mathbf{a}^{[1]} = ReLU((\mathbf{W}^{[1]})^\top \mathbf{X})$

$$\begin{aligned} a_1^{[1]} &= ReLU(x_0 \times W_{01}^{[1]} + x_1 \times W_{11}^{[1]} + x_2 \times W_{21}^{[1]}) \\ &= ReLU(0.1 + 2 \times (-0.1) + 3 \times 0.3) \\ &= ReLU(0.8) \\ &= 0.8 \end{aligned}$$

$$\begin{aligned} a_2^{[1]} &= ReLU(x_0 \times W_{02}^{[1]} + x_1 \times W_{12}^{[1]} + x_2 \times W_{22}^{[1]}) \\ &= ReLU(0.1 + 2 \times 0.2 + 3 \times (-0.4)) \\ &= ReLU(-0.7) \\ &= 0 \end{aligned}$$

$\hat{\mathbf{y}}^{[2]} = ReLU((\mathbf{W}^{[2]})^\top \mathbf{a}^{[1]})$

$$\begin{aligned}
\hat{y}_1^{[2]} &= ReLU(a_0^{[1]} \times W_{01}^{[2]} + a_1^{[1]} \times W_{11}^{[2]} + a_2^{[1]} \times W_{21}^{[2]}) \\
&= ReLU(0.1 + 0.8 \times 0.5 + 0 \times 0.7) \\
&= ReLU(0.5) \\
&= 0.5
\end{aligned}$$

$$\begin{aligned}
\hat{y}_2^{[2]} &= ReLU(a_1^{[0]} \times W_{02}^{[2]} + a_1^{[1]} \times W_{12}^{[2]} + a_2^{[1]} \times W_{22}^{[2]}) \\
&= ReLU(0.1 + 0.8 \times (-0.6) + 0 \times (-0.8)) \\
&= ReLU(-0.38) \\
&= 0
\end{aligned}$$

$$\begin{aligned}
L(\hat{\mathbf{y}}^{[2]}, \mathbf{y}) &= \frac{1}{2} \times ((\hat{y}_1^{[2]} - y_1)^2 + (\hat{y}_2^{[2]} - y_2)^2) \\
&= \frac{1}{2} \times ((0.5 - 0.1)^2 + (0 - 0.9)^2) \\
&= \frac{1}{2} \times (0.16 + 0.81) \\
&= 0.485
\end{aligned}$$

## D Non-linear Activation Functions

We can define a neural network as follows:

$$\hat{y} = g(\mathbf{W^{[L]}}^{\mathbf{T}} \dots g(\mathbf{W^{[2]}}^{\mathbf{T}} \cdot g(\mathbf{W^{[1]}}^{\mathbf{T}} x)))$$

where $\mathbf{W}^{[\mathbf{l}] \in \{\mathbf{1}, \cdots, \mathbf{L}\}}$ is a weight matrix. You're given the following weight matrices:

$$\mathbf{W^{[3]}} = \begin{bmatrix} 1.2 & -2.2 \\ 1.2 & 1.3 \end{bmatrix}, \mathbf{W^{[2]}} = \begin{bmatrix} 2.1 & -0.5 \\ 0.7 & 1.9 \end{bmatrix}, \mathbf{W^{[1]}} = \begin{bmatrix} 1.4 & 0.6 \\ 0.8 & 0.6 \end{bmatrix}$$

Furthermore, you are given $g(z) = \text{SiLU}(z) = \frac{z}{1+e^{-z}}$ between all layers *except the last layer.*

*Disclaimer: Feel free to use `NumPy` to help with the following question!*

1. Is it possible to replace the whole neural network with just one matrix in both cases **with** and **without** non-linear activations $g(z)$? For both cases, either show that it is possible by providing such a matrix or prove that there exists no such matrix. What does this signify about the importance of the non-linear activation?

**Solution:**

Without SiLU$(z)$, we can show that we can replace the neural network with the following matrix $\mathbf{M^T}$:

$$\begin{aligned} M^T &= \begin{bmatrix} 1.2 & -2.2 \\ 1.2 & 1.3 \end{bmatrix}^T \begin{bmatrix} 2.1 & -0.5 \\ 0.7 & 1.9 \end{bmatrix}^T \begin{bmatrix} 1.4 & 0.6 \\ 0.8 & 0.6 \end{bmatrix}^T \\ &= \begin{bmatrix} 4.56 & 3.408 \\ -6.82 & -3.658 \end{bmatrix} \end{aligned}$$

With SiLU$(z)$, assume there exists such a matrix $\mathbf{M^T}$. Consider 2 inputs, $x_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}$ and $x_2 = \begin{bmatrix} 2 & 0 \end{bmatrix}$:

$$\begin{aligned} \hat{y_1} &= \begin{bmatrix} 1.2 & -2.2 \\ 1.2 & 1.3 \end{bmatrix}^T g\left( \begin{bmatrix} 2.1 & -0.5 \\ 0.7 & 1.9 \end{bmatrix}^T g\left( \begin{bmatrix} 1.4 & 0.6 \\ 0.8 & 0.6 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)\right) \\ &= \begin{bmatrix} 3.0571 \\ -5.2727 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \hat{y_2} &= \begin{bmatrix} 1.2 & -2.2 \\ 1.2 & 1.3 \end{bmatrix}^T g\left( \begin{bmatrix} 2.1 & -0.5 \\ 0.7 & 1.9 \end{bmatrix}^T g\left( \begin{bmatrix} 1.4 & 0.6 \\ 0.8 & 0.6 \end{bmatrix}^T \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right)\right) \\ &= \begin{bmatrix} 7.7257 \\ -13.2458 \end{bmatrix} \end{aligned}$$

Since $x_2 = 2x_1$, $\mathbf{M^T}x_2 = 2\mathbf{M^T}x_1 \implies \hat{y_2} = 2\hat{y_1}$ by linearity of $\mathbf{M^T}$. But by the computation we did above, $\hat{y_2} \neq 2\hat{y_1}$, thus there exist no such $\mathbf{M^T}$.

*Without non-linear activations, the entire network* ***collapses to a simple linear model****. It's as good as using a network with one layer; we can no longer capitalise on this "depth" of the original neural network (i.e., $L$ layers deep).*

$$\begin{aligned}\hat{y} &= \mathbf{W^{[L]T}} \dots \mathbf{W^{[2]T}} \mathbf{W^{[1]T}} x \\ &= \mathbf{A}x, \quad \text{where } \mathbf{A} = \mathbf{W^{[L]T}} \dots \mathbf{W^{[2]T}} \mathbf{W^{[1]T}} \text{ by matrix multiplication}\end{aligned}$$

*For non-linear data, these non-linear activation functions let the network model those relationships. Without non-linear activations, all the parameters in the network behave the same way, lowering the representation power or "learning capacity" the neural network.*

## E Working with Dimensions

You're building a self-driving car program that takes in grayscale images of size $32 \times 32$ where 32 is the image height and width. There are 4 classes your simplified program has to classify: $\{\text{car}, \text{person}, \text{traffic light}, \text{stop sign}\}$. You start off experimenting with a multi-layer neural network composed of three linear layers of the form $y = W^T x$, where $x \in \mathbb{R}^d$ is the input vector, $W$ is the weight matrix, and $y$ is the network output.

1. What are the dimensions of the input vector, the weight matrix, and the output vector of the three linear layers, given the following details? Assume the batch size is 1.

| layer | Input dim | Weight Matrix dim | Output dim |
|---|---|---|---|
| Linear layer 1 | ____ $\times 1$ | ____ $\times$ ____ | $512 \times 1$ |
| Linear layer 2 | $512 \times 1$ | ____ $\times 128$ | ____ $\times 1$ |
| Linear layer 3 | $128 \times 1$ | ____ $\times$ ____ | ____ $\times 1$ |

**Solution:**

| layer | Input dim | Weight Matrix dim | Output dim |
|---|---|---|---|
| Linear layer 1 | $1024 \times 1$ | $1024 \times 512$ | $512 \times 1$ |
| Linear layer 2 | $512 \times 1$ | $512 \times 128$ | $128 \times 1$ |
| Linear layer 3 | $128 \times 1$ | $128 \times 4$ | $4 \times 1$ |

Students should note the multi-layer neural networks are sequences of matrix multiplication operations applied one after the other. The weight matrix dimensions "flow" into one another i.e., they commute well. Mostly, only the input and output dimensions are of importance, the hidden layer dimensions can easily be figured out.