National University of Singapore
School of Computing
IT5005 Artificial Intelligence

Deep Neural Networks

1. **Dot Product-Activate, Forward Propagation**

   In this question, 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. FYI: $\text{ReLU}(x) = \max(0, x)$.

   Here's the basic structure:

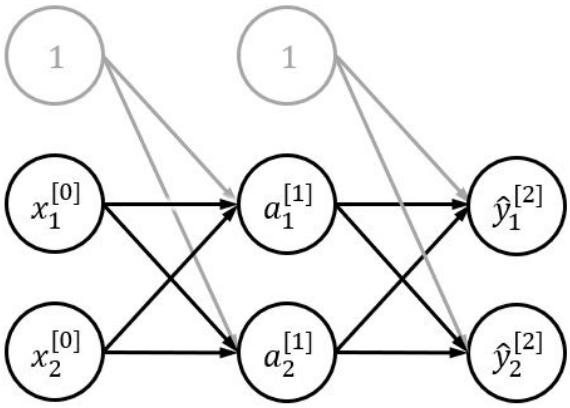

Figure 1: A simple neural network with one hidden layer

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

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 $L(\hat{\mathbf{y}}^{[2]}, \mathbf{y})$. Here, we use MSE (mean squared error) loss function.

2. **Let's Activate!**

   We can define a neural network as follows:

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

   where $\mathbf{W^{[l]\in\{1,\cdots,L\}}}$ 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!*

   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 shows 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?

3. **Working with Dimensions**

   You're building a self-driving car program that takes in grayscale images of size 32×32 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 Perceptron composed of three linear layers of the form $y = W^T x$, where $x \in \mathcal{R}^d$ is the input vector, W is the weight matrix, and y is the network output.

   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	_____ $\times 1$	_____ $\times$ _____	512×1
Linear layer 2	512×1	_____ $\times 128$	_____ $\times 1$
Linear layer 3	128×1	_____ $\times 4$	_____ $\times 1$

4. Consider a two-layer neural network with single perceptron in each layer and assume that bias term $b = 0$. The weights for the first and second layer are initialized as 2 and 3, respectively, i.e., $w^{[1]} = 2$ and $w^{[2]} = 3$. Furthermore, an exponential linear unit (ELU) is used as the activation function at each perceptron. The ELU is defined as:

$$g(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1), & \text{else} \end{cases} \tag{1}$$

The derivative of ELU is defined as:

$$g'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ \alpha(\exp(x)), & \text{else} \end{cases} \tag{2}$$

Assume that the parameter $\alpha = 1$. The input to the neural network is $x = -1$ and the expected output is $y = -1.4255$.

(a) Calculate $a^{[1]}$ and $a^{[2]}$.

(b) Calculate the mean square error, i.e., $(y - \hat{y})^2$, where $\hat{y}$ is the output of neural network.

(c) Calculate $\frac{\partial L}{\partial w^{[2]}}$ and $\frac{\partial L}{\partial w^{[1]}}$.

5. **Potential Issues with Training Deep Neural Networks**

Gradient Norm: The gradient norm of a weight matrix $W^{[l]} \in \mathbb{R}^{m \times n}$ in layer l is defined as

$$\left\| \frac{\partial L}{\partial \mathbf{W}^{(l)}} \right\|_2 = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \left(\frac{\partial L}{\partial w_{i,j}^{[l]}} \right)^2} \tag{3}$$

where L is the loss function, $w_{i,j}^{[l]}$ is the element at the i-th row and j-th column of the matrix $W^{[l]}$.

Gradient norms provide a measure of size of weight update across layers. Low gradient norm means small gradient values (consequently small weight updates). Gradient norm would be zero if gradients with respect to all weights at a layer are zero (i.e., no weight updates or no learning). Therefore, we can use gradient norm to track the issue of vanishing gradient problem in deep neural networks. Tracking gradient norms across training iterations can provide insights into the stability and efficiency of the learning process, helping identify issues early and adjust hyperparameters accordingly.

In the accompanying Python notebook, a code snippet is provided to calculate the gradient norm across all layers of a neural network.

Suppose we use σ (the sigmoid function) as our activation function in a neural network with L hidden layers, as per the code in the accompanying Python notebook.

(a) Play around with the code. Vary the value of L. Notice that when performing back propagation, the gradient magnitudes of the first few layers are extremely small. What do you think causes this problem?

(b) Based on what we have learnt thus far, how can we **mitigate** this problem? Test out your solution by modifying the code and checking the gradient magnitudes.
Hint: Explore different activation functions, weight initialization strategies