Solver: Hans Albert Lianto

Tips for scoring on CZ4042:

- The CZ4042 exam should be renamed to 'how well can you optimize using your calculator fast enough to crunch all the numbers' exam, in addition to understanding the fundamental mathematics behind neural networks. So, learn how to use your calculator to crunch numbers; some tutorial questions (that don't involve coding) help you do this.
- Practice doing previous papers and time yourself at least once. Be very thorough in crunching the numbers (i.e. at least know what you're doing, don't blindly do stuff); one small mistake and it will cascade a **lot** (and it's very easy to make mistakes crunching these matrices). Because of this, I apologize in advance if I made any small mistakes. They're almost inevitable.
- Get at least above average in the projects. Otherwise you're screwed to begin with.
- Do still understand the intuitions of concepts well (i.e. GD learning, Adam optimizer, etc.) so you don't need to look at your lecture notes in the open-book exam so you can do things faster. This helped in doing the true-false questions as checking the notes for each question is a drag and could be a waste of time.
- $\log x$ in the lecture notes is $\ln x$. This is a horrible thing that isn't emphasized enough.
- Don't panic! 😊

1)

a)

i) **FALSE**. GD learning (and even SGD learning) could converge to the **local minimum** of the cost function if it is trapped in a feature space 'trough' (i.e. a dip).

ii) **TRUE**. SGD learning is said to consider the randomness of individual data (GD learning could cancel out or 'suppress' some of this randomness) and is more likely to find a more optimal cost than GD learning, and hence can find better weights and biases than GD learning.

iii) **FALSE**. Functions learned by linear neurons are all **hyperplanes**, and due to the absence of the exponent-involved sigmoid function, cannot learn non-linear mapping.

iv) **TRUE**. A discrete perceptron, which uses a sigmoid activation function $\sigma(z)$, where z is the pre-synaptic input, returns 1 if $\sigma > 0.5$ and 0 if $\sigma \leq 0.5$. This is equivalent to 1 of $z > 0$ and 0 if $z \leq 0$. Since the pre-synaptic input z to a discrete perceptron is a hyperplane (i.e. $z = \vec{w}.\vec{x} + b$), a discrete perceptron does learn a hyperplane as the decision boundary in feature space.

v) **FALSE**. Who are you to decide it could be done only with GD learning? Use SGD learning if you want, do what you please.

vi) **TRUE**. This should be done to prevent gradient descent to be slow due to low gradients of both ends of the sigmoid function. Weights are initialized so that learning can proceed faster when z is within the linear region of the sigmoid activation function (gradient is larger, so

learning is faster)

vii) **FALSE**. Not always, once the batch size is increased to a certain value, we are bottlenecked with the parallelism capacity of your processor/GPU and multiple (and hence more) gradient descent operations need to be done for each mini batch. So, your updating weight time increases again when the batch size increases further.

b)

i)

$$W = \begin{pmatrix} 2 & 1 & 0 \\ -3 & -1 & -2 \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

ii) Refer to the Week 1 lecture notes. We refer the pre-synaptic outputs to the three neurons forming the softmax layer as z_1, z_2 and z_3.

$$z_1 = 2x_1 - 3x_2 + 1$$

$$z_2 = x_1 - x_2 - 1$$

$$z_3 = -2x_2 + 1$$

The decision boundary separating Class 1 and Class 2 is:

$$z_1 = z_2$$

$$2x_1 - 3x_2 + 1 = x_1 - x_2 - 1$$

$$\boxed{x_2 = 0.5x_1 + 1}$$

The decision boundary separating Class 1 and Class 3 is:

$$z_1 = z_3$$

$$2x_1 - 3x_2 + 1 = -2x_2 + 1$$

$$\boxed{x_2 = 2x_1}$$

The decision boundary separating Class 2 and Class 3 is:

$$z_2 = z_3$$

$$x_1 - x_2 - 1 = -2x_2 + 1$$

$$\boxed{x_2 = -x_1 + 2}$$

iii) Output is Class 1 if $z_1 > z_2$ ($x_2 < 0.5x_1 + 1$) and $z_1 > z_3$ ($x_2 < 2x_1$).
Output is Class 2 if $z_2 > z_1$ ($x_2 > 0.5x_1 + 1$) and $z_2 > z_3$ ($x_2 > -x_1 + 2$).
Output is Class 3 if $z_3 > z_1$ ($x_2 < 2x_1$) and $z_3 > z_2$ ($x_2 < -x_1 + 2$).

Let us plot these three lines on Desmos in the next page.

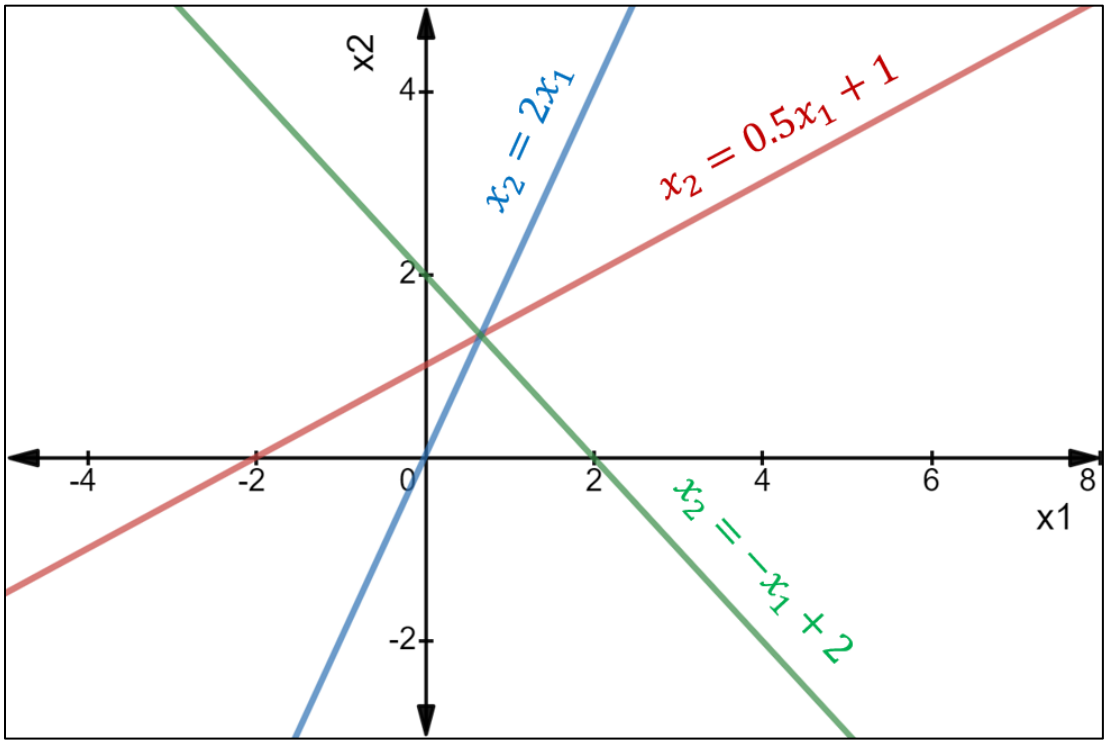

Figure 1: Decision boundaries

Now figure out the regions corresponding to Class 1, 2 and 3:

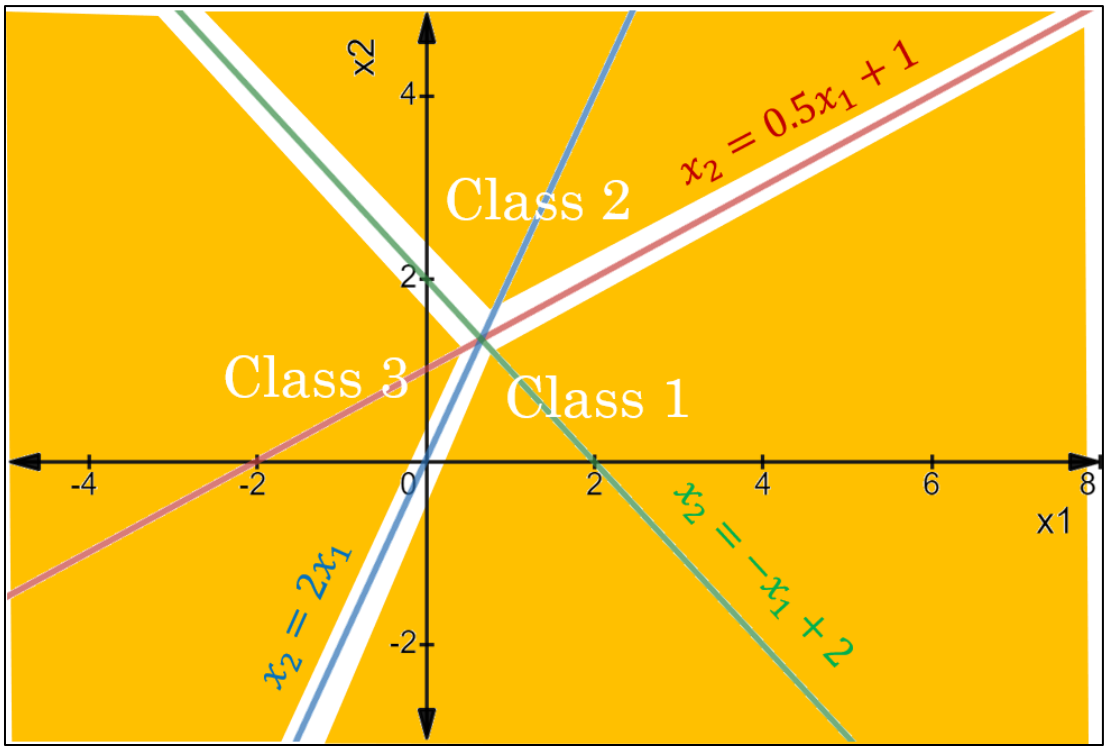

Figure 2: Decision boundaries with regions assigned to each class

Simplify/prune/cut the graph:

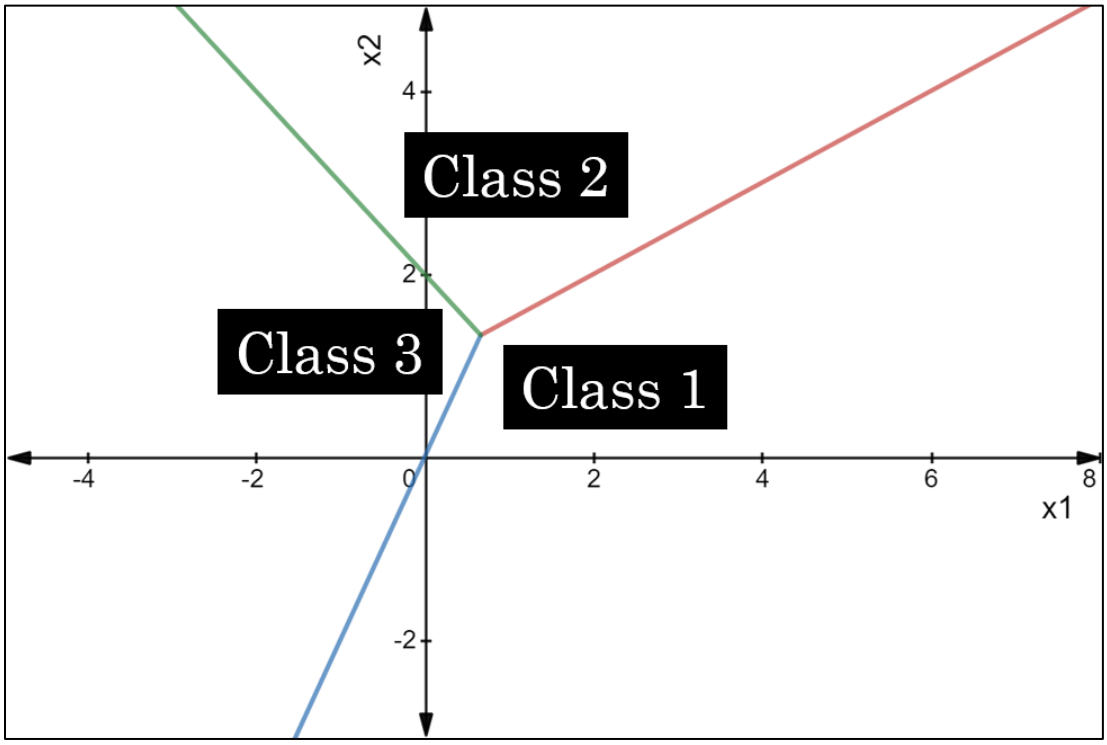

Figure 3: Simplified decision boundaries with class labels

iv) The input that yields equal class probabilities is the input (x_1, x_2) where the lines in the graph in 1(b)(iii) all intersect. That is, $(x_1, x_2) = (\frac{2}{3}, \frac{4}{3})$. To get this value, simply derive the intersection between any two of the three decision boundaries in 1(b)(ii):

$$x_2 = 0.5x_1 + 1 \; and \; x_2 = 2x_1$$

$$0.5x_1 + 1 = 2x_1$$

$$x_1 = \frac{2}{3}$$

$$x_2 = 2x_1 = \frac{4}{3}$$

$$\boxed{\therefore (x_1, x_2) = \left(\frac{2}{3}, \frac{4}{3}\right)}$$

2) The question requires to give answers up to two decimal places. So, in intermediate computations, it's important to take note of the values up to three or four decimal places for better precision. It's even better to keep your intermediate values saved in calculator memory. If your calculator has

matrix computation capabilities, you can input the matrices and get your answers immediately, but I believe you can't save those values in memory. In the computations below, we use 3 decimal places.

a)

$$W = \begin{pmatrix} 1 & 0 & -2 \\ 2 & 1 & -1 \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \qquad V = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \qquad \vec{c} = (1)$$

b)

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad D = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

c)

$$Z = XW + B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & -2 \\ 2 & 1 & -1 \end{pmatrix} + \begin{pmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & -2 \end{pmatrix}$$

$$H = \frac{1}{1+e^{-Z}} = \begin{pmatrix} 0.731 & 0.881 & 0.269 \\ 0.500 & 0.731 & 0.119 \end{pmatrix} \approx \begin{pmatrix} 0.73 & 0.88 & 0.27 \\ 0.50 & 0.73 & 0.12 \end{pmatrix}$$

$$U = HV + C = \begin{pmatrix} 0.731 & 0.881 & 0.269 \\ 0.500 & 0.731 & 0.119 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.119 \\ 0.888 \end{pmatrix} \approx \begin{pmatrix} 1.12 \\ 0.89 \end{pmatrix}$$

$$Y = 1(\sigma(U) > 0.5) = 1\left(\begin{pmatrix} 0.754 \\ 0.708 \end{pmatrix} > \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}\right) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

d)

$$J = J_1 + J_2 = -\ln(0.754) - \ln(1 - 0.708) = 1.51$$

$$classification\ error = \frac{1}{2} \times 100\% = 50\%$$

e)

$$\nabla_U J = -\big(D - \sigma(U)\big) = -\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.754 \\ 0.708 \end{pmatrix}\right) = \begin{pmatrix} -0.246 \\ 0.708 \end{pmatrix}$$

$$\nabla_Z J = \nabla_U J(V^T) \cdot \sigma'(Z) = \nabla_U J(V^T) \cdot H(1-H) = \begin{pmatrix} -0.246 \\ 0.708 \end{pmatrix}(1 \quad -1 \quad 1) \cdot \begin{pmatrix} 0.197 & 0.105 & 0.197 \\ 0.250 & 0.197 & 0.105 \end{pmatrix}$$
$$= \begin{pmatrix} -0.048 & 0.026 & -0.048 \\ 0.177 & -0.139 & 0.074 \end{pmatrix}$$

f)

$$\nabla_V J = H^T \nabla_U J = \begin{pmatrix} 0.731 & 0.500 \\ 0.881 & 0.731 \\ 0.269 & 0.119 \end{pmatrix} \begin{pmatrix} -0.246 \\ 0.708 \end{pmatrix} = \begin{pmatrix} 0.174 \\ 0.301 \\ 0.018 \end{pmatrix} \approx \begin{pmatrix} 0.17 \\ 0.30 \\ 0.02 \end{pmatrix}$$

$$\nabla_{\vec{c}} J = (\nabla_U J)^T 1_P = sum\ of\ elements\ in\ every\ column\ of\ \nabla_U J = (0.462) \approx (0.46)$$

$$\nabla_W J = X^T \nabla_Z J = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -0.048 & 0.026 & -0.048 \\ 0.177 & -0.139 & 0.074 \end{pmatrix} = \begin{pmatrix} 0.177 & -0.139 & 0.074 \\ -0.048 & 0.026 & -0.048 \end{pmatrix}$$
$$\approx \begin{pmatrix} 0.18 & -0.14 & 0.07 \\ -0.05 & 0.03 & -0.05 \end{pmatrix}$$

$$\nabla_{\vec{b}} J = (\nabla_Z J)^T 1_P = sum\ of\ elements\ in\ every\ column\ of\ \nabla_Z J = \begin{pmatrix} 0.129 \\ -0.113 \\ 0.026 \end{pmatrix} \approx \begin{pmatrix} 0.13 \\ -0.11 \\ 0.03 \end{pmatrix}$$

g)

$$V_{new} = V_{curr} - \alpha \nabla_V J = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} - 0.4 \begin{pmatrix} 0.174 \\ 0.301 \\ 0.018 \end{pmatrix} = \begin{pmatrix} 0.930 \\ -1.120 \\ 0.993 \end{pmatrix} \approx \begin{pmatrix} 0.93 \\ -1.12 \\ 0.99 \end{pmatrix}$$

$$\vec{c}_{new} = \vec{c}_{curr} - \alpha \nabla_{\vec{c}} J = (1) - 0.4(0.462) = (0.815) \approx (0.82)$$

$$W_{new} = W_{curr} - \alpha \nabla_W J = \begin{pmatrix} 1 & 0 & -2 \\ 2 & 1 & -1 \end{pmatrix} - 0.4 \begin{pmatrix} 0.177 & -0.139 & 0.074 \\ -0.048 & 0.026 & -0.048 \end{pmatrix}$$
$$= \begin{pmatrix} 0.929 & 0.056 & -2.030 \\ 2.019 & 0.990 & -0.981 \end{pmatrix} \approx \begin{pmatrix} 0.93 & 0.06 & -2.03 \\ 2.02 & 0.99 & -0.98 \end{pmatrix}$$

$$\vec{b}_{new} = \vec{b}_{curr} - \alpha \nabla_{\vec{b}} J = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} - 0.4 \begin{pmatrix} 0.129 \\ -0.113 \\ 0.026 \end{pmatrix} = \begin{pmatrix} -1.052 \\ 1.045 \\ -0.010 \end{pmatrix} \approx \begin{pmatrix} -1.05 \\ 1.05 \\ -0.01 \end{pmatrix}$$

3)

a) Output of ReLU function and **feature map at first convolution layer**:

$$ReLU\left(\begin{pmatrix} -0.09 & 0.74 & -0.70 & -0.03 \\ -0.24 & 1.04 & 1.11 & -0.10 \end{pmatrix}\right) = \begin{pmatrix} 0 & 0.74 & 0 & 0 \\ 0 & 1.04 & 1.11 & 0 \end{pmatrix}$$

Feature map at average pooling layer:

$$\begin{pmatrix} \frac{0 + 0.74 + 0 + 1.04}{4} & \frac{0 + 0 + 1.11 + 0}{4} \end{pmatrix} = (0.445 \quad 0.2775) \approx (0.45 \quad 0.28)$$

b) This is the **Dying Neuron** phenomenon, wherein the gradient of the ReLU function is zero because the input to the ReLU function is less than zero. Hence training stops for this neuron if the input continues being zero. One way to prevent this is to use a **Leaky ReLU neuron**, which

has a small but positive gradient on the negative side to prevent training from stopping

c) No of pixels = $(1+2+2+2+2+2)^2 = 121$ pixels. Use the *formula for receptive field*.

d) Yes, allocating only 7 marks to a question with 2×4 by 4×3 matrix computations. Bravo, SCSE.
i)

$$H = \varphi(XW+B) = \varphi\left(\begin{pmatrix}1 & 1 & 0 & 0\\ 0 & 1 & 1 & 1\end{pmatrix}\begin{pmatrix}-2.0 & 1.2 & -0.8\\ 0.5 & 1.5 & 2.2\\ -2.2 & 3.2 & -1.2\\ 2.0 & -1.0 & -3.6\end{pmatrix} + \begin{pmatrix}0.2 & -0.5 & 0.0\\ 0.2 & -0.5 & 0.0\end{pmatrix}\right)$$

$$= \varphi\left(\begin{pmatrix}-1.3 & 2.2 & 1.4\\ 0.5 & 3.2 & -2.6\end{pmatrix}\right) = \begin{pmatrix}-0.862 & 0.976 & 0.886\\ 0.462 & 0.997 & -0.989\end{pmatrix} \approx \begin{pmatrix}-0.86 & 0.98 & 0.89\\ 0.46 & 1.00 & -0.99\end{pmatrix}$$

ii) At this point, I assumed that these autoencoders were representing bits, so, since the activation function $\sigma(u)$ ranges from -1 to 1, the final output for the neurons in the output layer is 1 if $\sigma(u) > 0$ and 0 otherwise. I assumed that reconstruction errors are how many bits were the final outputs different from the inputs.

$$Y = \sigma(HW^T + C)$$

$$= \sigma\left(\begin{pmatrix}-0.862 & 0.976 & 0.886\\ 0.462 & 0.997 & -0.989\end{pmatrix}\begin{pmatrix}-2.0 & 0.5 & -2.2 & 2.0\\ 1.2 & 1.5 & 3.2 & -1.0\\ -0.8 & 2.2 & -1.2 & -3.6\end{pmatrix} + \begin{pmatrix}0.5 & 1.0 & -0.6 & 2\\ 0.5 & 1.0 & -0.6 & 2\end{pmatrix}\right)$$

$$= \sigma\left(\begin{pmatrix}2.686 & 3.982 & 3.356 & -3.890\\ 1.564 & 0.551 & 2.761 & 5.487\end{pmatrix}\right) = \begin{pmatrix}0.872 & 0.963 & 0.933 & -0.960\\ 0.654 & 0.269 & 0.881 & 0.992\end{pmatrix}$$

$$O = (\sigma(Y) > 0) = \left(\begin{pmatrix}0.872 & 0.963 & 0.933 & -0.960\\ 0.654 & 0.269 & 0.881 & 0.992\end{pmatrix} > \begin{pmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & 1 & 1 & 0\\ 1 & 1 & 1 & 1\end{pmatrix}$$

$Reconstruction\ errors = 2$ (1 per input pattern on average)

e) **Undercomplete autoencoders** represent input patterns with *less* number of features than the number of features/dimensions of the input patterns themselves, and as a result of this, these autoencoders capture the *most salient* features from these input patterns and hence, hidden structures or patterns from the input patterns can be inferred by these autoencoders.

4)

a) The question states that the output layer neuron uses a logistic activation function. I assume that this is the sigmoid (σ) activation function, not ($\sigma > 0.5$).

$$\vec{h}(1) = \tanh\left(U^T\vec{x}(1) + W^T\vec{h}(0) + \overrightarrow{b_1}\right)$$
$$= \tanh\left(\begin{pmatrix}-0.1 & 0.2\\ 1.5 & 1.0\\ 2.0 & -1.0\end{pmatrix}\begin{pmatrix}0.5\\ 1.5\end{pmatrix} + \begin{pmatrix}-1.0 & 1.5 & -0.9\\ -0.6 & 1.7 & 2.0\\ -0.2 & 1.5 & 0.3\end{pmatrix}\begin{pmatrix}0\\ 0\\ 0\end{pmatrix} + \begin{pmatrix}0.2\\ 0.2\\ 0.2\end{pmatrix}\right)$$
$$= \tanh\left(\begin{pmatrix}0.45\\ 2.45\\ -0.30\end{pmatrix}\right) = \begin{pmatrix}0.422\\ 0.985\\ -0.291\end{pmatrix}$$

$$y(1) = \sigma\left(V^T\vec{h}(1) + \overrightarrow{b_2}\right) = \sigma\left((0.3 \quad 1.0 \quad 0.4)\begin{pmatrix}0.422\\ 0.985\\ -0.291\end{pmatrix} + (0.2)\right) = \sigma(1.195) \approx 0.77$$

$$\vec{h}(2) = \tanh\left(U^T\vec{x}(2) + W^T\vec{h}(1) + \overrightarrow{b_1}\right)$$
$$= \tanh\left(\begin{pmatrix}-0.1 & 0.2\\ 1.5 & 1.0\\ 2.0 & -1.0\end{pmatrix}\begin{pmatrix}1.0\\ -1.0\end{pmatrix} + \begin{pmatrix}-1.0 & 1.5 & -0.9\\ -0.6 & 1.7 & 2.0\\ -0.2 & 1.5 & 0.3\end{pmatrix}\begin{pmatrix}0.422\\ 0.985\\ -0.291\end{pmatrix} + \begin{pmatrix}0.2\\ 0.2\\ 0.2\end{pmatrix}\right)$$
$$= \tanh\left(\begin{pmatrix}1.217\\ 1.539\\ 4.506\end{pmatrix}\right) = \begin{pmatrix}0.839\\ 0.912\\ 1.000\end{pmatrix}$$

$$y(2) = \sigma\left(V^T\vec{h}(2) + \overrightarrow{b_2}\right) = \sigma\left((0.3 \quad 1.0 \quad 0.4)\begin{pmatrix}0.839\\ 0.912\\ 1.000\end{pmatrix} + (0.2)\right) = \sigma(1.764) \approx 0.85$$

$$\vec{h}(3) = \tanh\left(U^T\vec{x}(3) + W^T\vec{h}(2) + \overrightarrow{b_1}\right)$$
$$= \tanh\left(\begin{pmatrix}-0.1 & 0.2\\ 1.5 & 1.0\\ 2.0 & -1.0\end{pmatrix}\begin{pmatrix}2.5\\ -2.0\end{pmatrix} + \begin{pmatrix}-1.0 & 1.5 & -0.9\\ -0.6 & 1.7 & 2.0\\ -0.2 & 1.5 & 0.3\end{pmatrix}\begin{pmatrix}0.839\\ 0.912\\ 1.000\end{pmatrix} + \begin{pmatrix}0.2\\ 0.2\\ 0.2\end{pmatrix}\right)$$
$$= \tanh\left(\begin{pmatrix}-0.821\\ 4.997\\ 8.700\end{pmatrix}\right) = \begin{pmatrix}-0.676\\ 1.000\\ 1.000\end{pmatrix}$$

$$y(3) = \sigma\left(V^T\vec{h}(3) + \overrightarrow{b_2}\right) = \sigma\left((0.3 \quad 1.0 \quad 0.4)\begin{pmatrix}-0.676\\ 1.000\\ 1.000\end{pmatrix} + (0.2)\right) = \sigma(1.397) \approx 0.80$$

Hence the output of the network is $(0.77, 0.85, 0.80)$.

b)

i) The **generator** and the **discriminator**. The generator's goal is to maximize the cost function and hence generate images that can fool the discriminator while the discriminator's goal is to minimize the cost function and classify as accurately as possible which images are real and which images are fake.

ii) Mode collapse is when the images generated by the **generator** network only fall under a restricted or limited set of categories instead of all of them (hence collapsing into a certain

mode of images). To overcome this method, generate images and classify images in batches. The discriminator network should have extra layers or parameters to compute whether the batch is 'diverse' enough or not. Hence, if mode collapse occurs and the images generated aren't diverse enough, the discriminator will classify the images as fake. This forces the generator to generate diverse images.

-- End of Answers --