# COMP7606C: ASSIGNMENT #1 - QUESTION 1

Due date: 3/13/2020 23:59 HKT for Question 1 of Assignment #1. Assignment #1 comprises 3 Questions, each of involves a programming task. These questions require no written answers. Rather, for each question, student must submit code via moodle. We allow students to discuss the Assignment with other students, but each student must finish all Questions of the Assignment individually.
The following instructions apply to all 3 Questions:

- Please use `Python 3 (3.5-3.7)` as the default programming language, and `Tensorflow 2.1` as the deep learning platform. Besides, the following packages may be required:
  - (required) `Numpy >= 1.17`
  - (optional) `sklearn` and `matplotlib`
- All you need to submit are Python scripts and, in the case of Question 3, `results/prediction.txt`. Please compress all the files in `your_student_id.zip` (You may use command `zip -r your_student_id.zip *.py ./results`).

**IMPORTANT:** Strictly follow the instructions, otherwise your assignment will not be appropriately graded.

## 1. Basic Neural Network (30 points)

Given inputs $x$ we want to predict outputs $\hat{y}$ via some function $f(x)$. Let us choose a simple neural network to model that function. Consider a two layer neural network, which outputs $\hat{y}$ when inputting an example $x$ using for loops:

$$z^{[1]} = W^{[1]}\, x + b^{[1]} \tag{1.1}$$

$$a^{[1]} = g^{[1]}(z^{[1]}) \tag{1.2}$$

$$z^{[2]} = W^{[2]}\, a^{[1]} + b^{[2]} \tag{1.3}$$

$$a^{[2]} = g^{[2]}(z^{[2]}) \tag{1.4}$$

$$\hat{y} = a^{[2]} \tag{1.5}$$

where $z^{[l]}, a^{[l]}$ are the hidden units before and after passing through the activation function $g^{[l]}$, and $W^{[l]}, b^{[l]}$ are weight and bias parameters for $l$-th layer ($l = 1, 2$), respectively. In our example, $x \in \mathbb{R}^{3\times 1}, y \in \mathbb{R}^{1\times 1}, W^{[1]} \in \mathbb{R}^{3\times 3}, b^{[1]} \in \mathbb{R}^{3\times 1}, W^{[2]} \in \mathbb{R}^{1\times 3}, b^{[2]} \in \mathbb{R}^{1\times 1}$

Given $m$ training samples $X = \left[x^{(1)}; \ldots, x^{(m)}\right] \in \mathbb{R}^{3\times m}$ , $Y = \left[y^{(1)}; \ldots, y^{(m)}\right] \in \mathbb{R}^{1\times m}$, we update the parameters $W^{[l]}$ and $b^{[l]}$ using the stochastic gradient descent algorithm:

$$W^{[l]} = W^{[l]} - \alpha \cdot \frac{\partial J}{\partial W^{[l]}} \tag{1.6}$$

$$b^{[l]} = b^{[l]} - \alpha \cdot \frac{\partial J}{\partial b^{[l]}} \tag{1.7}$$

where $J$ is the cost function $J = \frac{1}{m}\sum_1^m L^{(i)}$ and $L^{(i)}$ is the loss function for a single example $(x^{(i)}, y^{(i)})$, and $\alpha$ is learning rate. Assume we use cross-entropy for the loss function:

$$L(y, \hat{y}) = -\left((1-y)\,\log(1-\hat{y}) + y\,\log(\hat{y})\right) \tag{1.8}$$

1.1. **Activation function (2 points).** Assume the activation functions $g^{[l]}, l = 1, 2$ are both the sigmoid function. Implement sigmoid function (`sigmoid`) and its gradient (`sigmoid_grad`) using pure **Numpy** in `q1_basic_nn.py`. Remember that

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.9}$$

$$\sigma'(x) = \sigma(x)\,(1-\sigma(x)) \tag{1.10}$$

1.2. **Parameter initialization (4 points).** Implement `init_layers` in `q1_basic_nn.py`. This will initialize $W^{[l]}$ and $b^{[l]}$ with standard normal distributed values.

1.3. **Feedforward propagation (8 points).** Compute $Z^{[1]}, A^{[1]}, Z^{[2]}$ and $A^{[2]}$ in the Equations 1.1-1.4. Note that vectorized computation shall be used, e.g., $Z^{[1]} = W^{[1]}X + b^{[1]}$. Implement `feedforward_propagation` in `q1_basic_nn.py`.

1.4. **Backforward propagation (12 points).** Compute the following gradients:

$$\frac{\partial J}{\partial W^{[2]}} \quad \frac{\partial J}{\partial b^{[2]}} \quad dZ^{[2]} \quad \frac{\partial J}{\partial W^{[1]}} \quad \frac{\partial J}{\partial b^{[1]}} \quad dZ^{[1]} \tag{1.11}$$

where $dZ^{[l]} = \left[\frac{\partial L^{(1)}}{\partial z^{[l](1)}}, \ldots, \frac{\partial L^{(m)}}{\partial z^{[l](m)}}\right]$, and $dZ^{[1]} \in \mathbb{R}^{3\times m}, dZ^{[2]} \in \mathbb{R}^{1\times m}$. Implement `backforward_propagation` in `q1_basic_nn.py`.

**Hints**: The gradients with respect to the parameters can be computed using chain rule. For a single example:

$$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}} = dz^{[2]}\, a^{[1]T} \tag{1.12}$$

$$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}} = dz^{[2]} \tag{1.13}$$

where

$$dz^{[2]} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \tag{1.14}$$

$$= -\frac{\partial}{\partial \hat{y}} \left((1-y)\, \log(1-\hat{y}) + y\, \log(\hat{y})\right) \frac{\partial a^{[2]}}{\partial z^{[2]}} \tag{1.15}$$

$$= \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \sigma(z^{[2]})(1-\sigma(z^{[2]})) \tag{1.16}$$

$$= \hat{y} - y \tag{1.17}$$

For $m$ examples, we obtain:

$$dZ^{[2]} = \left[dz^{[2](1)}, \ldots, dz^{[2](m)}\right] = \hat{Y} - Y \tag{1.18}$$

$$dW^{[2]} = \frac{1}{m} \sum_{1}^{m} dz^{[2](i)}\, a^{[1](i)T} = \frac{1}{m}\, dZ^{[2]}\, A^{[1]T} \tag{1.19}$$

$$db^{[2]} = \frac{1}{m} \sum_{1}^{m} dz^{[2](i)} = \frac{1}{m} \text{np.sum}(dZ^{[2]}, axis = 1, keepdims = True) \tag{1.20}$$

where $dW^{[2]} = \frac{\partial J}{\partial W^{[2]}}, db^{[2]} = \frac{\partial J}{\partial b^{[2]}}$ and `np.sum` is Numpy function to calculate the sum of array elements over a given axis. Compute the remaining gradients by yourselves.

1.5. **Parameters update (2 points).** Apply parameters update rules Equations 1.6-1.7 and implement `parameter_update` in `q1_basic_nn.py`.

1.6. **Cost function (2 points).** Implement `cost_fn` in `q1_basic_nn.py`. This function computes the cost $J$ for a batch of training data.

1.7. **Training with toy data.** A tiny dataset defined in `datasets/toy_data.py` is provided for training. After filling in all the implementations, you can train your model by running `python3 q1_basic_nn.py`. The training losses should decrease with iterations.