Programming Assignment – Neural Network

1. 목표: 숫자를 구별하는 3 Layer-based Neural Network 코드 작성 (Classification)

2. 학습 데이터: 20x20 pixel 기반의 Gray 숫자 영상 → 400x1 vector로 표현됨

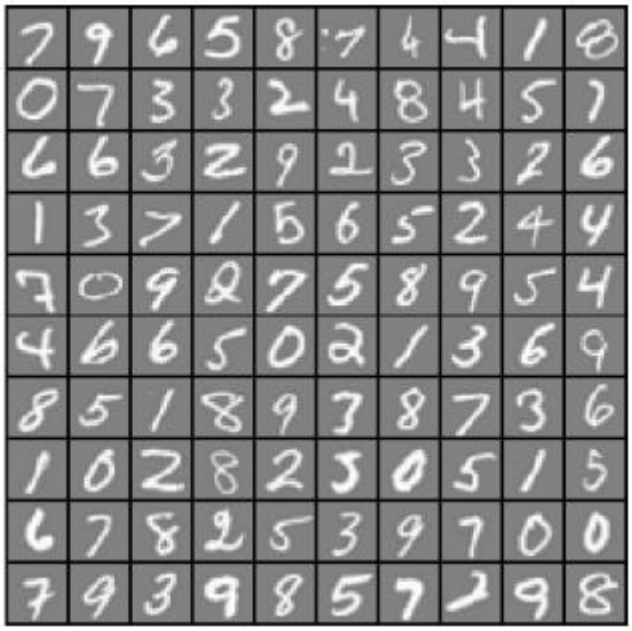

Figure 1. 숫자 그림 예제

총 m=5,000개의 학습 Sample이 제공됨. 최종 학습 Data는 5,000 x 200 크기의 X matrix임

$$X = \begin{bmatrix} — \ (x^{(1)})^T \ — \\ (x^{(2)})^T \\ \vdots \\ — \ (x^{(m)})^T \ — \end{bmatrix}$$

3. 최종 결과: 0~9로 표현 되나 여기서는 편의를 위해서 1,2,3,..,9,0 으로 표현할 예정임. 즉, 아래 y label은 one hot vector 형태 이고 각각 1,2,..,9,0을 뜻함

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

4. 설계 Network 모델: 3 Layer 기반 Neural Network로 Classification Task임

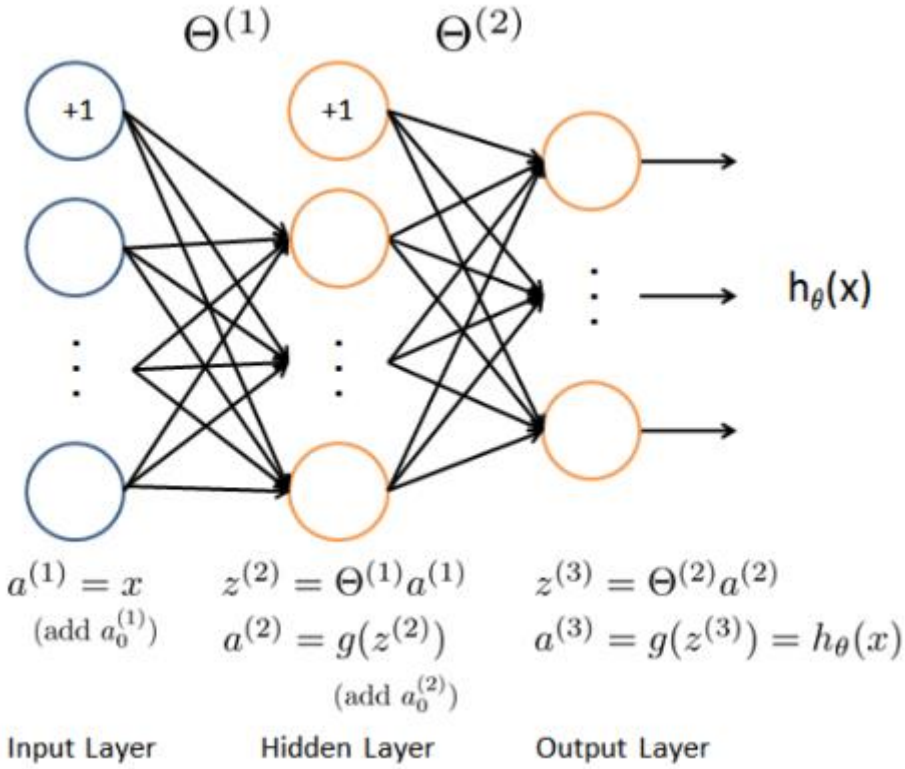

Figure 2. 숙제에 사용될 Neural Network 구조

- 상기 네트워크는 중간에 Hidden Layer로 총 25개의 Unit이 존재하고 Bias가 존재

- 입력 벡터 ($x \in \Re^{400}$)에 대해서 각 Layer별 파라미터는 $\Theta^{(1)} \in \Re^{25\times 401}$, $\Theta^{(2)} \in \Re^{10\times 26}$임

5. Forward Propagation 제작 (ForwardPropagation.m) [Part 3/4]

- 각 Unit (노란색)의 Activation Function은 Sigmoid Function으로 제작됨

- Regularization이 없는 Cost function은 다음과 같음

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right]$$

여기서 m은 학습 데이터 개수, K는 Class의 수이기 때문에 본 과제에서는 10이 됨

- 해당 코드의 최종 결과 J는 0.287629 임.

* Regularization을 추가 해야함

- 앞선 수식에서 Regularization을 집어넣으면 아래 수식과 같이 됨

- Regularization에서 Bias 값은 추가되지 않는 것을 참고할 것

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

- 최종 결과물은 lamda가 1일 때 J값은 0.383770 임

6. Backpropagation 제작 (Backpropagation.m) [Part 5/6/7]

- Sigmoid 미분 함수 제작 (sigmoidGradient.m) [Part 5]

- 본 과제에서 Weight의 Random Initialization 값은 기존 코드를 그대로 사용함

- 본 과제에서 gradient checking은 사용하지 않음

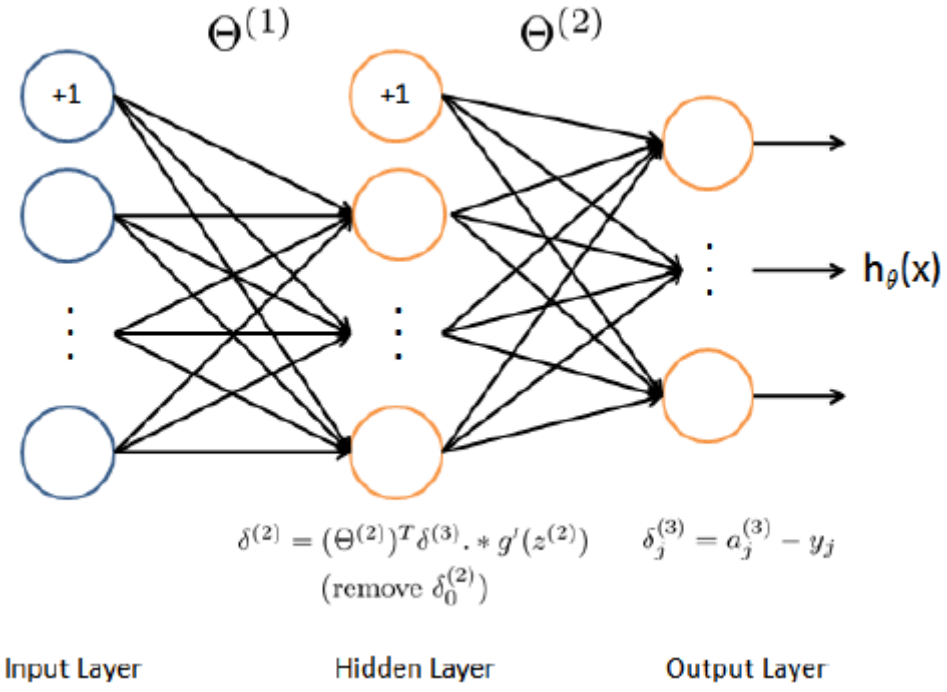

Figure 3. 본 과제 네트워크의 Backpropagation

- Backpropagation되는 delta값은 위 그림과 같음

- 실제 Backpropagation되는 방식은 아래와 같음

1. Set the input layer's values ($a^{(1)}$) to the t-th training example $x^{(t)}$. Perform a feedforward pass (Figure 2), computing the activations ($z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)}$) for layers 2 and 3. Note that you need to add a `+1` term to ensure that the vectors of activations for layers $a^{(1)}$ and $a^{(2)}$ also include the bias unit. In Octave/MATLAB, if `a_1` is a column vector, adding one corresponds to `a_1 = [1 ; a_1]`.

2. For each output unit k in layer 3 (the output layer), set

$$\delta_k^{(3)} = (a_k^{(3)} - y_k),$$

where $y_k \in \{0, 1\}$ indicates whether the current training example belongs to class k ($y_k = 1$), or if it belongs to a different class ($y_k = 0$). You may find logical arrays helpful for this task (explained in the previous programming exercise).

3. For the hidden layer $l = 2$, set

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$$

4. Accumulate the gradient from this example using the following formula. Note that you should skip or remove $\delta_0^{(2)}$. In Octave/MATLAB, removing $\delta_0^{(2)}$ corresponds to `delta_2 = delta_2(2:end)`.

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

- Backpropagation에서 Regularization은 다음과 같음

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

- nnCostFunction.m에는 FowardPropagation과 BackPropagation 함수를 호출해서 J값과 Gradient 값을 출력해 주고 있음

- 해당 함수를 호출하면 lambda가 3일 경우 최종 출력 J값은 0.576051 임

7. Gradient Descent 학습 [part 8]

- 총 50번 Iteration을 통해서 학습을 수행

- 50번 Iteration 동안 Cost값은 감소 해야함

8. 학습 Data에 대한 성능 평가 (prediction.m) [part 10]

- 최종 학습 Data에 대한 성능은 약 95%가 나옴

- Learning Rate, Regularization Parameter, Iteration 수를 조정해서 학습 Data에 Overfitting이 가능함. 또한 해당 Parameter들의 변화에 따른 다양한 현상을 확인해 볼 수 있음