# Notations and Dimensions

## Dimensions

A summary for the dimensions for major variables for a generalized neural network:
$W^{[L]} \quad (n^{[L]}, n^{[L-1]})$
$b^{[L]} \quad (n^{[L]}, 1))$
$Z^{[L]} \quad (n^{[L]}, m)$
$A^{[L]} \quad (n^{[L]}, m)$
$X = A^{[0]} \quad (n^{[0]}, m)$

- For scalar cost function, the dimensions of $dW, db$ have the same dimensions of $W$ and $b$.
- The first dimension of all the matrices is $n^{[L]}$. This is like treating each layer in a visualized neural network as a column vector in a matrix. With this in mind, it is easy to remember all these dimensions.
- From $z^{(i)} = w^T x^{(i)} + b$ to understand $W^{[L]} \quad (n^{[L]}, n^{[L-1]})$. The second index of $w^T$ and first index of $x^{(i)}$ must be same. However, the first index of $x^{(i)}$ is just the first index of the previous layer relative to $W^{[L]}$.

## Notations

- Superscript $[l]$ denotes a quantity associated with the $l^{th}$ layer.
  - Example: $a^{[L]}$ is the $L^{th}$ layer activation. $W^{[L]}$ and $b^{[L]}$ are the $L^{th}$ layer parameters.
- Superscript $(i)$ denotes a quantity associated with the $i^{th}$ example.
  - Example: $x^{(i)}$ is the $i^{th}$ training example.
- Lowerscript $i$ denotes the $i^{th}$ entry of a vector.
  - Example: $a_i^{[l]}$ denotes the $i^{th}$ entry of the $l^{th}$ layer's activations).

# Single layer and single unit neural network

Take the activation function of the unit as sigmoid for example. In other words, this is just logistic regression.

For one example $x^{(i)}$:

$$z^{(i)} = w^T x^{(i)} + b$$
$$\hat{y}^{(i)} = a^{(i)} = \text{sigmoid}(z^{(i)})$$
$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$
$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})$$
$$\frac{\partial J}{\partial w} = \frac{1}{m} X (A - Y)^T$$
$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) = \frac{1}{m} \text{np. sum}(A - Y)$$

# Two layer multiple units neural network

The model is: three units input + four units first layer + one unit output layer. The activation function in the first layer can be tanh or any other function. The activation function for output layer is sigmoid. Normally the activation function for middle layers uses Relu or other simpler function rather than the complicated sigmoid. So the derivation is generally simpler.

**Forward propagation**

Consider the layer 1 (The input layer is conventionally taken as layer 0) in the above model, we have

$$z_k^{[1]} = (w_k^{[1]})^T x + b_k^{[1]}, \quad k = 1, 2, 3, 4$$

where $k$ denotes $k\text{th}$ unit in layer 1. For each $k$, $z_k^{[1]}$ and $b_k^{[1]}$ are scalar, while $w_k^{[1]}$ is a $3 \times 1$ column vector. The vectorized form of above equation is

$$z^{[1]} = w^{[1]} x + b^{[1]}$$

In this example, $z^{[1]}$ and $b^{[1]}$ are both $4 \times 1$ matrices, and $w^{[1]}$ is a $4 \times 3$ matrix. **Be careful that each row of $w^{[1]}$ is the original $(w_k^{[1]})^T$. So in the vectorized equation, the transpose sign $T$ is gone.**

To include different samples, we further introduce the index $i$. The element-wise and vectorized forms are respectively,

$$z^{[1](i)} = w^{[1]} x^{(i)} + b^{[1]}$$
$$Z^{[1]} = W^{[1]} X + b^{[1]},$$

where the dimensions of $Z^{[1]}$ becomes $(4, m)$, The dimensions of $W^{[1]}$ and $b^{[1]}$ are same as before because they have nothing to do specific sample.

Similarly, we do the following vectorization,

$$a^{[1](i)} = g(z^{[1](i)}) = \tanh(z^{[1](i)}), \quad A^{[1]} = g(Z^{[1]}) = \tanh(Z^{[1]})$$

$$z^{[2](i)} = w^{[2]} a^{[1](i)} + b^{[2]}, \quad Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)}), \quad A^{[2]} = \sigma(Z^{[2]}),$$

where the $b^{[2](i)}$ has been changed to $b^{[2]}$. The cost function and its vectorized version are

$$J \equiv \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y^{(i)}, a^{[2](i)}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{[2](i)}) + (1 - y^{(i)}) \log(1 - a^{[2](i)})$$

$$J = -\frac{1}{m} \text{sum} \left( Y \log(A^{[2]}) + (1 - Y) \log(1 - A^{[2]}) \right),$$

where the sum is numpy function which summing over all the elements within a vector.

**Backward propagation**

$$dz_k^{[2]} = \frac{\partial \mathcal{L}}{\partial z_k^{[2]}} = \frac{\partial \mathcal{L}}{\partial a_k^{[2]}} \frac{\partial a_k^{[2]}}{\partial z_k^{[2]}} = \left( \frac{-y_k}{a_k^{[2]}} + \frac{1 - y_k}{1 - a_k^{[2]}} \right) a_k^{[2]} (1 - a_k^{[2]}) = (a_k^{[2]} - y_k),$$

where we assume layer 2 also have multiple nodes for generality, even it has one in the model above. The vectorized version is thus

$$dz^{[2])} = a^{[2]} - y, \tag{1}$$

where the vector is actually a scalar because layer 2 has only one node.

$$dw_k^{[2]} \equiv \frac{\partial \mathcal{L}}{\partial w_k^{[2]}} = \frac{\partial \mathcal{L}}{\partial a_k^{[2]}} \frac{\partial a_k^{[2]}}{\partial z_k^{[2]}} \frac{\partial z_k^{[2]}}{\partial w_k^{[2]}} = dz_k^{[2]} a_k^{[1]}$$

The matrix form is:

$$dw^{[2]} = dz^{[2]} a^{[1]T}, \tag{2}$$

where the transpose is necessary to achieve the correct dimensions: $dw^{[2]} : (1, 4)$, $dz^{[2]} : (1, 1)$, $a^{[1]} : (4, 1)$. Note $w^{[2]}$ is just a row of the final $W^{[2]}$ (including different samples), which has a dimension of $W^{[L]} \quad (n^{[L]}, n^{[L-1]})$. For $w^{[2]}$, its dimension is $(1, n^{[2-1]} = 4)$.

$$db_k^{[2]} \equiv \frac{\partial \mathcal{L}}{\partial b_k^{[2]}} = \frac{\partial \mathcal{L}}{\partial a_k^{[2]}} \frac{\partial a_k^{[2]}}{\partial z_k^{[2]}} \frac{\partial z_k^{[2]}}{\partial b_k^{[2]}} = dz_k^{[2]}$$

The vectorized version is

$$db^{[2]} = dz^{[2]} \tag{3}$$

$$dz_k^{[1]} = \frac{\partial \mathcal{L}}{\partial z_k^{[1]}} = \frac{\partial \mathcal{L}}{\partial a_k^{[2]}} \frac{\partial a_k^{[2]}}{\partial z_k^{[2]}} \frac{\partial z_k^{[2]}}{\partial a_k^{[1]}} \frac{\partial a_k^{[1]}}{\partial z_k^{[1]}} = dz_k^{[2]} w_k^{[2]} g^{[1]'}(z_k^{[1]})$$

The vectorized form is

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]}) \tag{4}$$

The transpose sign and ordering or symbols are used to achieve the correct dimensions. The dimensions of $dz_k^{[1]}, w^{[2]}, dz^{[2]}, g^{[1]'}(z^{[1]})$ are respectively $(4,1), (1,4), (1,1), (4,1)$. The $*$ denote element-wise multiplication.
Following the similar procedure, we can obtain the following two vectorized derivatives

$$dw^{[1]} = dz^{[1]} x^T$$
$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} \text{np. sum}(dZ^{[2]}, \text{axis} = 1, \text{keepdims} = \text{True})$$
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} \text{np. sum}(dZ^{[1]}, \text{axis} = 1, \text{keepdims} = \text{True})$$

Check the link below for examples of using keepdims = True/False. [https://stackoverflow.com/questions/39441517/in-numpy-sum-there-is-parameter-called-keepdims-what-does-it-do (https://stackoverflow.com/questions/39441517/in-numpy-sum-there-is-parameter-called-keepdims-what-does-it-do)](https://stackoverflow.com/questions/39441517/in-numpy-sum-there-is-parameter-called-keepdims-what-does-it-do)
When axis is not specified, then np.sum will sum all the elements.

# Multiple layer multiple units neural network

The model is: input + L-1 layers with relu activation function + final output layer with sigmoid activation function.

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}, A^{[0]} = X$$

From input to $L-1$th layer, we have

$$A^{[l]} = RELU(Z^{[l]}) = max(0, Z^{[l]})$$

The final output layer has

$$A^{[L]} = \sigma(Z^{[L]}) = \sigma(W^{[L]} A^{[L-1]} + b^{[L]})$$

The cross-entropy cost function is given by

$$J \equiv \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(y^{(i)}, a^{[L](i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(a^{[L](i)}) + (1-y^{(i)})\log(1-a^{[L](i)})$$

$$J = -\frac{1}{m}\text{sum}\left(Y\log(A^{[L]}) + (1-Y)\log(1-A^{[L]})\right)$$

Note the back propagation process is similar to the forward propagation process. If $g()$ is the activation function, which can be `sigmoid_backward` or `relu_backward` , then we have

$$dZ^{[l]} = dA^{[l]} * A'(Z^{[l]}) = dA^{[l]} * g'(Z^{[l]})$$

The above equation is obvious as $A^{[l]}$ is function of $Z^{[l]}$. See specific examples for a two layer model derived earlier. Moreover, because $Z^{[l]}$ is a function of $W^{[l]}$, $b^{[l]}$ and $A^{[l-1]}$, using chain rule it is easy to obtain the following equations

$$dW^{[l]} = \frac{\partial\mathcal{L}}{\partial W^{[l]}} = dZ^{[l]}\frac{\partial Z^{[l]}}{\partial W^{[l]}} = \frac{1}{m}dZ^{[l]}A^{[l-1]T}$$

$$db^{[l]} = \frac{\partial\mathcal{L}}{\partial b^{[l]}} = dZ^{[l]}\frac{\partial Z^{[l]}}{\partial b^{[l]}} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial\mathcal{L}}{\partial A^{[l-1]}} = dZ^{[l]}\frac{\partial Z^{[l]}}{\partial A^{[l-1]}} = W^{[l]T}dZ^{[l]}$$

For the final output layer $L$, the activation function is sigmoid, and thus

$$dA^{[L]} = \frac{\partial\mathcal{L}}{\partial A^{[L]}} = \frac{-Y}{A^{[L]}} + \frac{1-Y}{1-A^{[L]}}$$

See similar derivations for the two-layer model introduced earlier.

$$W^{[l]} = W^{[l]} - \alpha\, dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha\, db^{[l]}$$

During the forward propagation, we have calculated $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$. The $(W^{[l]}, A^{[l-1]}, \text{ and } b^{[l]})$ are stored as linear cache. $Z^{[l]}$ is stored as linear activation cache. They are cached because they will be used in calculating the gradients in backward propagation, as shown in the equation above.

$$y^{(i)}_{prediction} = \begin{cases} 1 & \text{if } a^{[2](i)} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

# $l_2$ Regularization

The standard way to avoid overfitting is called $l_2$ *regularization**. It consists of appropriately modifying your cost function, from:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)$$

To:

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{l2 regularization cost}}$$

Implement `compute_cost_with_regularization()` with np.square and np.sum which calculate $\sum_{k}\sum_{j}W_{k,j}^{[l]2}$ as

```
np.sum(np.square(Wl))
```

np.square returns element-wise `x*x` , of the same shape and dtype as x. This is a scalar if x is a scalar. So we can calculate the square of any data structure including matrices.

Because the cost has been changed, backward propagation needs change as well with respect to this new cost. Here the changes only concern $dW_1, dW_2$ and $dW_3$. For each, add the regularization term's gradient ($\frac{d}{dW}(\frac{1}{2}\frac{\lambda}{m}W^2) = \frac{\lambda}{m}W$).