Deep Learning Specialization by Andrew Ng
A Review

Keivan Mokhtarpour

April 2021

1. *PART I: Neural Networks and Deep Learning*

1. Logistic regression doesn't have a hidden layer. If you initialize the weights to zeros, the first example x fed in the logistic regression will output zero but the derivatives of the logistic regression depend on the input x (because there's no hidden layer) which is not zero. So in the second iteration, the weights values follow x's distribution and are different from each other if x is not a constant vector.

2. The general methodology to build a neural network is to:

   - Define the neural network structure (# of input units, # of hidden layers, etc.)
   - Initialize the model's parameters
   - Loop:
     - Implement forward propagation
     - Compute loss
     - Implement backward propagation to get the gradients
     - Update parameters (gradient descent)

3. There are many ways to implement the cross-entropy function:

$$-\mathbf{\Sigma_{i=0}^{m} y^{(i)} log(a^{[2](i)})} \tag{1.1}$$

For the implementation:

- logprobs = np.multiply(np.log(A2),Y)
- cost = -np.sum(logprobs)
  - we can use either np.multiply() and then np.sum() or directly np.dot(). However, if we use np.multiply() followed by np.sum(), the final result will be of type float, whereas if we use np.dot(), the result will be a 2D numpy array. We can use np.squeeze() to remove redundant dimensions (in the case of single float, this will be reduced to a zero-dimension array). We can cast the array as a type float using float().

4. Summary of gradient descent (element-wise):

$$\mathbf{dz^{[2]}} = \mathbf{a^{[2]}} - \mathbf{y} \tag{1.2}$$

$$\mathbf{dW^{[2]}} = \mathbf{dz^{[2]} a^{[1]T}} \tag{1.3}$$

$$\mathbf{db^{[2]}} = \mathbf{dz^{[2]}} \tag{1.4}$$

$$\mathbf{dz^{[1]}} = \mathbf{W^{[2]T} dz^{[2]}} * \mathbf{g^{[1]'}(z^{[1]})} \tag{1.5}$$

$$\mathbf{dW^{[1]}} = \mathbf{dz^{[1]} X^{T}} \tag{1.6}$$

$$\mathbf{db^{[1]}} = \mathbf{dz^{[1]}} \tag{1.7}$$

Implementation (Vectorized):

$$\mathbf{dZ^{[2]} = A^{[2]} - Y} \tag{1.8}$$

$$\mathbf{dW^{[2]} = (1/m) dz^{[2]} a^{[1]T}} \tag{1.9}$$

$$\mathbf{db^{[2]} = (1/m) np.sum(dZ^{[2]}, axis = 1, keepdims = True)} \tag{1.10}$$

$$\mathbf{dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})} \tag{1.11}$$

$$\mathbf{dW^{[1]} = (1/m) dZ^{[1]} X^T} \tag{1.12}$$

$$\mathbf{db^{[1]} = (1/m) np.sum(dZ^{[1]}, axis = 1, keepdims = True)} \tag{1.13}$$

- If we choose a tanh activation function $g^{[1]'}(z) = 1 - (g^{[1]}(z))^2 - 1 - a^2$.

5. Question: Is it possible to implement vectorization instead of the for loop when we have a n-layer NN and need a recurrent algorithm?

6. Building blocks of deep neural networks:

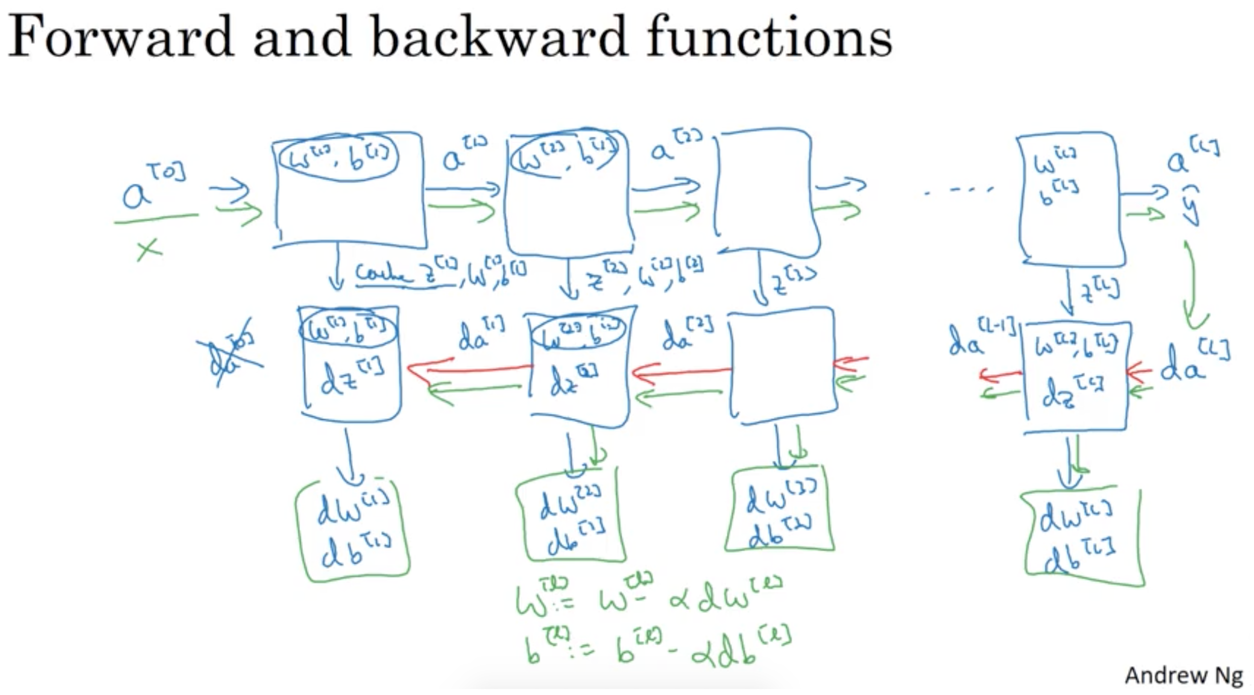


Figure 1.1:

- As you can see above, there's a forward propagation step and a corresponding backward propagation step which use a cache to pass information between each other.

7. Backward propagation for layer l: Input: $da^{[l]}$ Output: $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$\mathbf{dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})} \tag{1.14}$$

$$\mathbf{dW^{[l]} = dz^{[l]} * a^{[l-1]}} \tag{1.15}$$

$$\mathbf{db^{[l]} = dz^{[l]}} \tag{1.16}$$

$$\mathbf{da^{[l-1]} = W^{[l]T} dz^{[l]}} \tag{1.17}$$

$$\mathbf{da^{[l]} = W^{[l+1]T} dz^{[l+1]}} \tag{1.18}$$

$$\mathbf{dz^{[l]} = W^{[l+1]T}dz^{[l+1]} * g^{[l]'}(z^{[l]})} \tag{1.19}$$

To implement:

$$\mathbf{dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})} \tag{1.20}$$

$$\mathbf{dW^{[l]} = (1/m)dZ^{[l]}A^{[l-1]T}} \tag{1.21}$$

$$\mathbf{db^{[l]} = (1/m)np.sum(dZ^{[l]}, axis = 1, keepdims = True)} \tag{1.22}$$

$$\mathbf{dA^{[l-1]} = W^{[l]T}dZ^{[l]}} \tag{1.23}$$

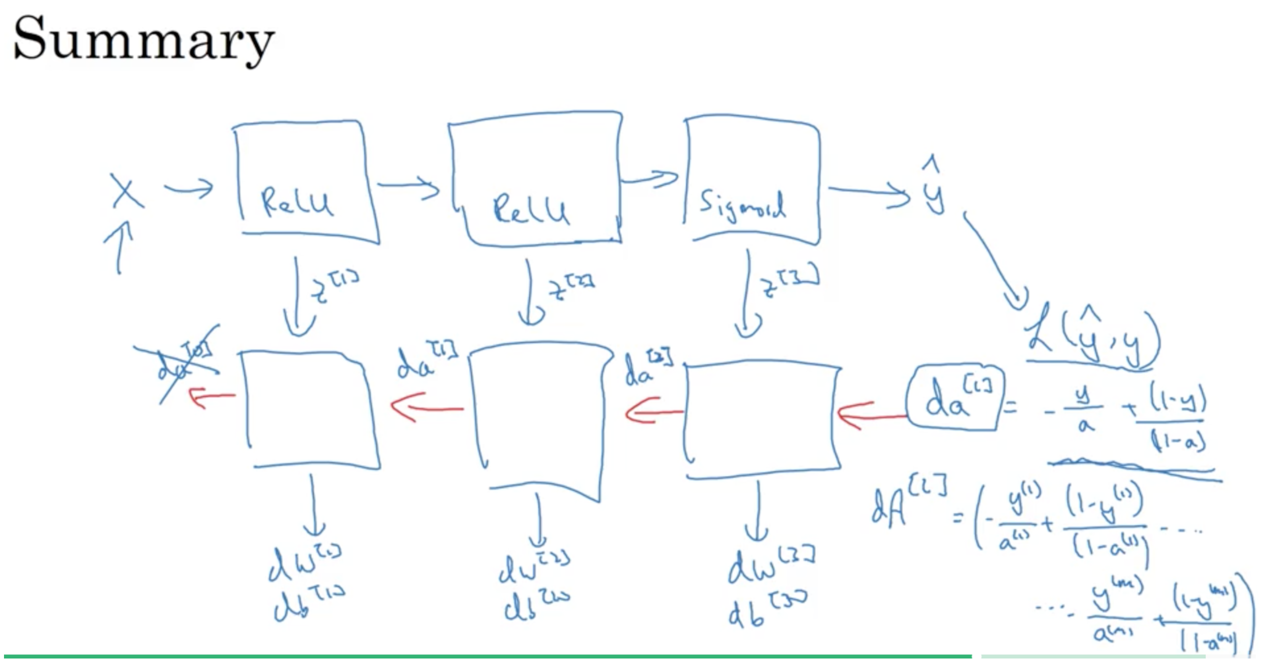

Summary
X
ReLU
ReLU
Sigmoid
ŷ
z[1]
z[2]
z[3]
L(ŷ, y)
da[0]
da[1]
da[2]
da[L] = -y/a + (1-y)/(1-a)
dA[L] = (-y(1)/a(1) + (1-y(1))/(1-a(1)) ... ... -y(m)/a(m) + (1-y(m))/(1-a(m)))
dW[1]
db[1]
dW[2]
db[2]
dW[3]
db[3]

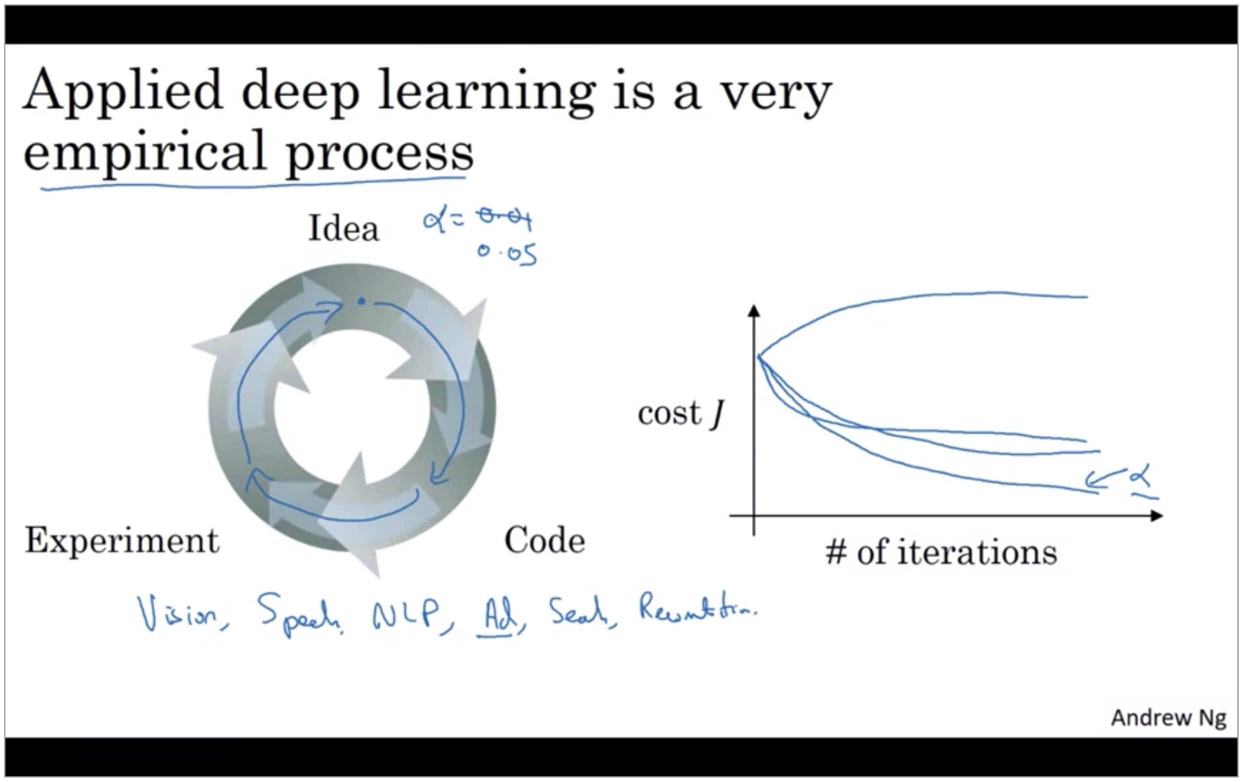
Figure 1.2:

8. Parameters and Hyperparameters: Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, ...$ Hyperparameters:

- learning rate α
- # iterations
- # hidden layers l
- # hidden units $n^{[1]}, n^{[2]}, ...$
- choice of activation functions

Hyperparameters control the ultimate parameters $(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, ...)$ and that's why we call them hyperparameters. LAter we'll see other hyperparameters as: Momentum, mini-batch size, regularization terms, ...

9. Applied deep learning is a very empirical process.



Applied deep learning is a very empirical process

Figure 1.3:

When starting on a new application, it can be difficult to know in advance exactly what is the best value of the hyperparameters. So what often happens is you just have to try many values and go around this cycle and iterate.

Applications of deep learning:
Computer Vision, Speech Recognition, NLP, web search, product recommendation.
Since CPU, GPU are advancing, it's good to keep trying different values of hyperparameters, evaluate them on a hold-on cross validation set and pick the value that works for your problem.