Universidad del Valle De Guatemala

Facultad de Ingeniería

Deep Learning


# Laboratorio 1

Javier Mombiela 20067
Roberto Ríos 20979.

Guatemala, 30 de mayo 2023

En parejas, utilicen el código publicado en NNsimple.ipynb conteste los siguientes problemas. Analice las preguntas con la teoría vista en clase y complemente con capturas de código y resultados.

1. ¿Existe diferencia entre la convergencia de los parámetros (pesos y sesgos) si estos son inicializados en 0 o como números aleatorios?
    - Iniciando con 0

```
Cost after iteration# 0: 0.693147
Cost after iteration# 100: 0.693147
Cost after iteration# 200: 0.693147
Cost after iteration# 300: 0.693147
Cost after iteration# 400: 0.693147
Cost after iteration# 500: 0.693147
Cost after iteration# 600: 0.693147
Cost after iteration# 700: 0.693147
Cost after iteration# 800: 0.693147
Cost after iteration# 900: 0.693147
Cost after iteration# 1000: 0.693147
{'W1': array([[0., 0.],
       [0., 0.]]), 'W2': array([[0., 0.]]), 'b1': array([[0.],
       [0.]]), 'b2': array([[0.]])}
Neural Network prediction for example (1, 1) is 1
```

    - Iniciando con random

```
Cost after iteration# 0: 0.761448
Cost after iteration# 100: 0.620654
Cost after iteration# 200: 0.482822
Cost after iteration# 300: 0.406094
Cost after iteration# 400: 0.378729
Cost after iteration# 500: 0.367436
Cost after iteration# 600: 0.361691
Cost after iteration# 700: 0.358308
Cost after iteration# 800: 0.356109
Cost after iteration# 900: 0.354576
Cost after iteration# 1000: 0.353452
{'W1': array([[ 3.6745176 , -1.40379209],
       [ 4.03105715,  2.76560019]]), 'W2': array([[-3.0316094 ,  3.83283969]]), 'b1': array([[ 0.76162811],
       [-0.7322501 ]]), 'b2': array([[-0.79771193]])}
Neural Network prediction for example (1, 1) is 1
```

    - Se puede mencionar que a la hora de inicializar todo como random, se puede optimizar el costo y también podemos observar como el bias y los pesos se van ajustando. Por otro lado, si se inicializa todo en 0, las neuronas están muertas, por lo que el costo se mantiene constante, ya que los cálculos no están siendo afectados. Cabe mencionar que la red proporciona predicciones erróneas con estos parámetros.

2. ¿Qué diferencia en la convergencia de la función de costo y los parámetros existe si el learning rate del código es 0.01? 0.1? 0.5?
    - 0.01:

```
Cost after iteration# 0: 0.856267
Cost after iteration# 100: 0.734628
Cost after iteration# 200: 0.681811
Cost after iteration# 300: 0.650483
Cost after iteration# 400: 0.627906
Cost after iteration# 500: 0.609697
Cost after iteration# 600: 0.594146
Cost after iteration# 700: 0.580478
Cost after iteration# 800: 0.568228
Cost after iteration# 900: 0.557060
Cost after iteration# 1000: 0.546716
{'W1': array([[ 0.16661794, -0.42561887],
       [-1.77775139,  2.18867142]]), 'W2': array([[-1.81670895, -1.14558662]]), 'b1': array([[-0.0840932 ],
       [ 0.88867515]]), 'b2': array([[0.075449]])}
Neural Network prediction for example (1, 1) is 0
```

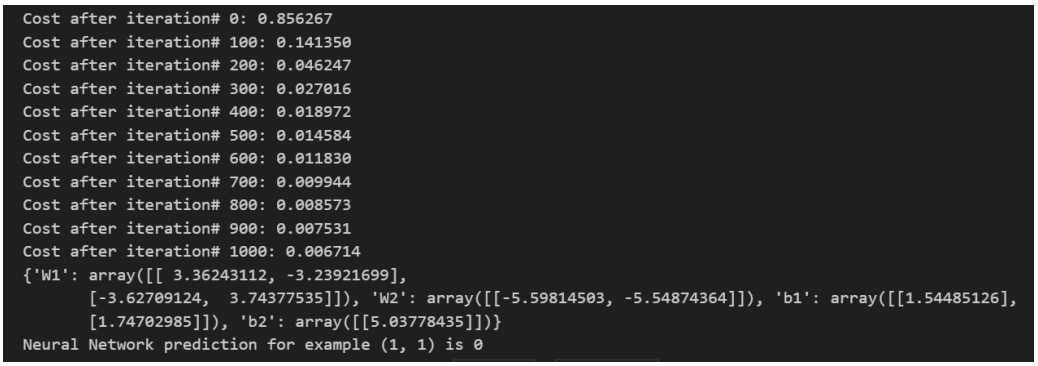
```
Cost after iteration# 0: 0.856267
Cost after iteration# 100: 0.734628
Cost after iteration# 200: 0.681811
Cost after iteration# 300: 0.650483
Cost after iteration# 400: 0.627906
Cost after iteration# 500: 0.609697
Cost after iteration# 600: 0.594146
Cost after iteration# 700: 0.580478
Cost after iteration# 800: 0.568228
Cost after iteration# 900: 0.557060
Cost after iteration# 1000: 0.546716
{'W1': array([[ 0.16661794, -0.42561887],
       [-1.77775139,  2.18867142]]), 'W2': array([[-1.81670895, -1.14558662]]), 'b1': array([[-0.0840932 ],
       [ 0.88867515]]), 'b2': array([[0.075449]])}
Neural Network prediction for example (0, 0) is 0
```

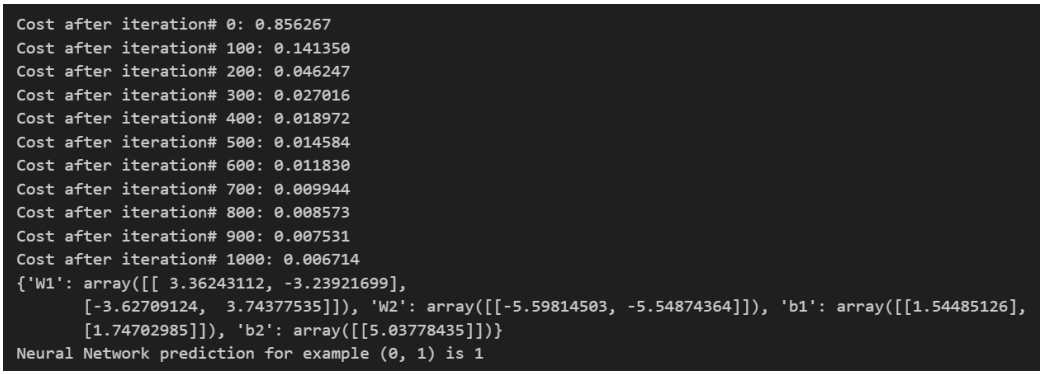
    - 0.1:

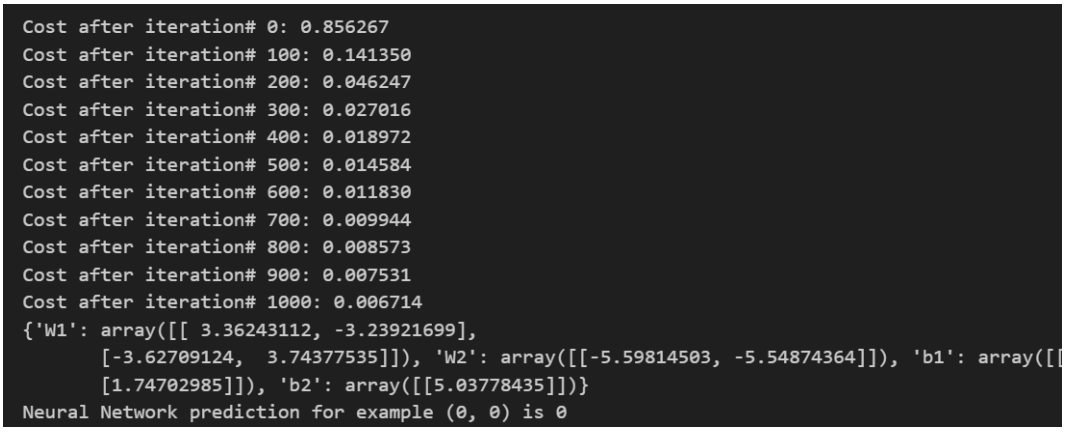
```
Cost after iteration# 0: 0.856267
Cost after iteration# 100: 0.547399
Cost after iteration# 200: 0.458590
Cost after iteration# 300: 0.344033
Cost after iteration# 400: 0.213431
Cost after iteration# 500: 0.139159
Cost after iteration# 600: 0.100705
Cost after iteration# 700: 0.078216
Cost after iteration# 800: 0.063661
Cost after iteration# 900: 0.053536
Cost after iteration# 1000: 0.046114
{'W1': array([[ 2.50712973, -2.38669949],
       [-3.1057134 ,  3.23893042]]), 'W2': array([[-3.8645041 , -3.77136637]]), 'b1': array([[1.10857647],
       [1.49874497]]), 'b2': array([[3.20605829]])}
Neural Network prediction for example (1, 1) is 0
```

```
Cost after iteration# 0: 0.856267
Cost after iteration# 100: 0.547399
Cost after iteration# 200: 0.458590
Cost after iteration# 300: 0.344033
Cost after iteration# 400: 0.213431
Cost after iteration# 500: 0.139159
Cost after iteration# 600: 0.100705
Cost after iteration# 700: 0.078216
Cost after iteration# 800: 0.063661
Cost after iteration# 900: 0.053536
Cost after iteration# 1000: 0.046114
{'W1': array([[ 2.50712973, -2.38669949],
       [-3.1057134 ,  3.23893042]]), 'W2': array([[-3.8645041 , -3.77136637]]), 'b1': array([[1.10857647],
       [1.49874497]]), 'b2': array([[3.20605829]])}
Neural Network prediction for example (0, 0) is 0
```

- 0.5:




- 


- 


0.01 es el aprendizaje más lento; sin embargo, esto asegura que se inspeccione más por cada paso, aunque este ritmo es muy lento que no llega a mejorar lo suficiente para ciertas curvas de aprendizaje. Una tasa del 0.1 podría ser más rápida, sin embargo aún no es óptima, el 0.5 es la mejor opción para este ejemplo, ya que no tiene riesgos de saltarse mínimos y empeorar a pesar de ir más rápido, por lo que solo se diferencia por aprender más rápido. En todos los casos las predicciones son correctas.

3. Implemente MSE como función de costo y propague los cambios en las funciones que lo requieran. ¿Qué cambios observa?

- Función de costo:

```python
def MSE(A2, Y):
    m = Y.shape[1]
    cost = (1/m) * np.sum(np.square(Y - A2))

    return np.squeeze(cost)
```

✓ 0.0s

- Backward propagation:

```python
def backward_propMSE(X, Y, cache, parameters):
    A1 = cache["A1"]
    A2 = cache["A2"]

    W2 = parameters["W2"]

    dZ2 = (A2 - Y) / m
    dW2 = np.dot(dZ2, A1.T)
    db2 = np.sum(dZ2, axis=1, keepdims=True)

    dZ1 = np.dot(W2.T, dZ2) * (1 - np.power(A1, 2))
    dW1 = np.dot(dZ1, X.T)
    db1 = np.sum(dZ1, axis=1, keepdims=True)

    grads = {
        "dW1": dW1,
        "db1": db1,
        "dW2": dW2,
        "db2": db2
    }

    return grads
```

- 0.01:

```
Cost after iteration# 0: 0.326659
Cost after iteration# 100: 0.270597
Cost after iteration# 200: 0.244583
Cost after iteration# 300: 0.229293
Cost after iteration# 400: 0.218226
Cost after iteration# 500: 0.209303
Cost after iteration# 600: 0.201728
Cost after iteration# 700: 0.195125
Cost after iteration# 800: 0.189265
Cost after iteration# 900: 0.183977
Cost after iteration# 1000: 0.179129
{'W1': array([[ 0.16661794, -0.42561887],
       [-1.77775139,  2.18867142]]), 'W2': array([[-1.81670895, -1.14558662]]), 'b1': array([[-0.0840932 ],
       [ 0.88867515]]), 'b2': array([[0.075449]])}
Neural Network prediction for example (1, 1) is 0
```

- 0.1:

```
Cost after iteration# 0: 0.326659
Cost after iteration# 100: 0.179437
Cost after iteration# 200: 0.138982
Cost after iteration# 300: 0.089280
Cost after iteration# 400: 0.040062
Cost after iteration# 500: 0.018369
Cost after iteration# 600: 0.009976
Cost after iteration# 700: 0.006148
Cost after iteration# 800: 0.004132
Cost after iteration# 900: 0.002952
Cost after iteration# 1000: 0.002208
{'W1': array([[ 2.50712973, -2.38669949],
       [-3.1057134 ,  3.23893042]]), 'W2': array([[-3.8645041 , -3.77136637]]), 'b1': array([[1.10857647],
       [1.49874497]]), 'b2': array([[3.20605829]])}
Neural Network prediction for example (1, 1) is 0
```

- 0.5:

```
Cost after iteration# 0: 0.326659
Cost after iteration# 100: 0.018917
Cost after iteration# 200: 0.002220
Cost after iteration# 300: 0.000774
Cost after iteration# 400: 0.000386
Cost after iteration# 500: 0.000229
Cost after iteration# 600: 0.000151
Cost after iteration# 700: 0.000107
Cost after iteration# 800: 0.000080
Cost after iteration# 900: 0.000062
Cost after iteration# 1000: 0.000049
{'W1': array([[ 3.36243112, -3.23921699],
       [-3.62709124,  3.74377535]]), 'W2': array([[-5.59814503, -5.54874364]]), 'b1': array([[1.54485126],
       [1.74702985]]), 'b2': array([[5.03778435]])}
Neural Network prediction for example (1, 1) is 0
```

- 

```
Cost after iteration# 0: 0.326659
Cost after iteration# 100: 0.018917
Cost after iteration# 200: 0.002220
Cost after iteration# 300: 0.000774
Cost after iteration# 400: 0.000386
Cost after iteration# 500: 0.000229
Cost after iteration# 600: 0.000151
Cost after iteration# 700: 0.000107
Cost after iteration# 800: 0.000080
Cost after iteration# 900: 0.000062
Cost after iteration# 1000: 0.000049
{'W1': array([[ 3.36243112, -3.23921699],
       [-3.62709124,  3.74377535]]), 'W2': array([[-5.59814503, -5.54874364]]), 'b1': array([[1.54485126],
       [1.74702985]]), 'b2': array([[5.03778435]])}
Neural Network prediction for example (0, 1) is 1
```

- 

```
Cost after iteration# 0: 0.326659
Cost after iteration# 100: 0.018917
Cost after iteration# 200: 0.002220
Cost after iteration# 300: 0.000774
Cost after iteration# 400: 0.000386
Cost after iteration# 500: 0.000229
Cost after iteration# 600: 0.000151
Cost after iteration# 700: 0.000107
Cost after iteration# 800: 0.000080
Cost after iteration# 900: 0.000062
Cost after iteration# 1000: 0.000049
{'W1': array([[ 3.36243112, -3.23921699],
       [-3.62709124,  3.74377535]]), 'W2': array([[-5.59814503, -5.54874364]]), 'b1': array([[1.54485126],
       [1.74702985]]), 'b2': array([[5.03778435]])}
Neural Network prediction for example (1, 0) is 1
```

```
Cost after iteration# 0: 0.326659
Cost after iteration# 100: 0.018917
Cost after iteration# 200: 0.002220
Cost after iteration# 300: 0.000774
Cost after iteration# 400: 0.000386
Cost after iteration# 500: 0.000229
Cost after iteration# 600: 0.000151
Cost after iteration# 700: 0.000107
Cost after iteration# 800: 0.000080
Cost after iteration# 900: 0.000062
Cost after iteration# 1000: 0.000049
{'W1': array([[ 3.36243112, -3.23921699],
       [-3.62709124,  3.74377535]]), 'W2': array([[-5.59814503, -5.54874364]]), 'b1': array([[1.54485126],
       [1.74702985]]), 'b2': array([[5.03778435]])}
Neural Network prediction for example (0, 0) is 0
```

- Lo que se puede observar es que en general el MSE es una mejor función de costo, ya que reduce el costo mucho más que la función anterior y da predicciones correctas aun teniendo un learning rate alto.