Universidad del Valle de Guatemala
Deep Learning
Sec. 20

Javier Chavez 21016
Andres Quezada 21085
28/07/24

# Laboratorio #1

https://github.com/andresquez/DeLe-Lab1

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

Si hay diferencia, inicializar los parámetros en cero puede llevar a problemas como causar simetría entre las neuronas lo que impide que la red aprenda correctamente. En cambio sí inicializamos los parámetros con números aleatorios rompemos esta simetría desde el inicio, permitiendo a las neuronas aprender características diferentes.

Resultados inicializando en 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.]])}
{'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
```

Resultados inicializando aleatoriamente:

```
Cost after iteration# 0: 1.052558
Cost after iteration# 100: 0.695402
Cost after iteration# 200: 0.693668
Cost after iteration# 300: 0.693206
Cost after iteration# 400: 0.692966
Cost after iteration# 500: 0.692779
Cost after iteration# 600: 0.692587
Cost after iteration# 700: 0.692352
Cost after iteration# 800: 0.692030
Cost after iteration# 900: 0.691539
Cost after iteration# 1000: 0.690679
{'W1': array([[-0.59247105, -0.47282144],
       [-2.06763357, -0.23592616]]), 'W2': array([[-0.43995116, -0.16049007]]), 'b1': array([[-1.50109455],
       [-1.77729809]]), 'b2': array([[-0.56827845]])}
Neural Network prediction for example (1, 1) is 1
```

Como se puede observar, los resultados concuerdan con la teoría, la función de costo nos muestra que en el caso de la inicialización en cero, es constante, se mantiene en 0.693147 durante todas las iteraciones, lo que nos indica que el modelo no está aprendiendo. Esto se debe a que todas las neuronas reciben las mismas actualizaciones durante el proceso de retro propagación, lo que hace que aprendan de manera idéntica. Como resultado, no se produce una reducción significativa en la función de costo, y los pesos y sesgos permanecen en cero. Mientras que al inicializarlos de manera aleatoria, la función de costo comienza en un valor más alto (1.052558) y disminuye gradualmente a 0.690679 a lo largo de 1000 iteraciones. Esto muestra que el modelo está aprendiendo de los datos y ajustando los parámetros para minimizar la función de costo. Y también los pesos y sesgos se han ajustado a valores no nulos, lo que indica que la red ha aprendido diferentes características de los datos.

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?

El learning rate es importante para la convergencia general de la función de costo y de los parámetros, ya que nos ayuda a determinar que tanto avanza el algoritmo de optimización para actualizar los pesos y sesgos en cada iteración. Ajustarlo para que no se salte el mínimo de la función de costo y tampoco sea muy tardado o se pierda en mínimo local.

Resultados con diferentes learning rates:

- 0.01

```
Training with learning rate: 0.01
Cost after iteration# 0: 1.052558
Cost after iteration# 100: 0.948307
Cost after iteration# 200: 0.864690
Cost after iteration# 300: 0.803459
Cost after iteration# 400: 0.765007
Cost after iteration# 500: 0.742498
Cost after iteration# 600: 0.729047
Cost after iteration# 700: 0.720556
Cost after iteration# 800: 0.714882
Cost after iteration# 900: 0.710898
Cost after iteration# 1000: 0.707985
Neural Network prediction for example (1, 1) is 1
```

- 0.1

```
Training with learning rate: 0.1
Cost after iteration# 0: 1.013592
Cost after iteration# 100: 0.674649
Cost after iteration# 200: 0.629729
Cost after iteration# 300: 0.590321
Cost after iteration# 400: 0.561134
Cost after iteration# 500: 0.540238
Cost after iteration# 600: 0.524827
Cost after iteration# 700: 0.512317
Cost after iteration# 800: 0.498693
Cost after iteration# 900: 0.452745
Cost after iteration# 1000: 0.312239
Neural Network prediction for example (1, 1) is 0
```

- 0.5

```
Training with learning rate: 0.5
Cost after iteration# 0: 0.713866
Cost after iteration# 100: 0.637009
Cost after iteration# 200: 0.531485
Cost after iteration# 300: 0.504387
Cost after iteration# 400: 0.494727
Cost after iteration# 500: 0.489987
Cost after iteration# 600: 0.487218
Cost after iteration# 700: 0.485415
Cost after iteration# 800: 0.484154
Cost after iteration# 900: 0.483226
Cost after iteration# 1000: 0.482515
Neural Network prediction for example (1, 1) is 1
```

Podemos observar que la tasa de aprendizaje realmente si afecta significativamente la convergencia de la función de costo y la precisión de las predicciones. Con una tasa de aprendizaje baja, 0.01, la función de costo disminuye lentamente, creando una convergencia gradual y subóptima. En el segundo caso, con el valor intermedio de 0.1, la disminución de la función de costo es más rápida y eficiente, alcanzando un valor de costo menor y logrando una predicción correcta en la mayoría de los casos. Por último, con una tasa de aprendizaje alta de 0.5, aunque la función de costo disminuye rápidamente, el modelo tiende a saltar alrededor del mínimo, lo que puede llevar a una convergencia menos estable y predicciones inconsistentes. Por lo tanto, una tasa de aprendizaje intermedia, como 0.1, es la elección correcta para este modelo ya que nos da un equilibrio adecuado entre la velocidad de convergencia y la estabilidad del aprendizaje, logrando los mejores resultados.

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

Para hacer esto creamos una función de pérdida nueva, que calcula el MSE. Y se utilizan las funciones de propagación dentro del modelo.

Resultados originales:

```
Cost after iteration# 0: 1.052558
Cost after iteration# 100: 0.695402
Cost after iteration# 200: 0.693668
Cost after iteration# 300: 0.693206
Cost after iteration# 400: 0.692966
Cost after iteration# 500: 0.692779
Cost after iteration# 600: 0.692587
Cost after iteration# 700: 0.692352
Cost after iteration# 800: 0.692030
Cost after iteration# 900: 0.691539
Cost after iteration# 1000: 0.690679
{'W1': array([[-0.59247105, -0.47282144],
       [-2.06763357, -0.23592616]]), 'W2': array([[-0.43995116, -0.16049007]]), 'b1': array([[-1.50109455],
       [-1.77729809]]), 'b2': array([[-0.56827845]])}
Neural Network prediction for example (1, 1) is 1
```

Resultados usando MSE:

```
 Cost after iteration# 0: 0.357529
 Cost after iteration# 100: 0.251127
 Cost after iteration# 200: 0.250260
 Cost after iteration# 300: 0.250029
 Cost after iteration# 400: 0.249910
 Cost after iteration# 500: 0.249816
 Cost after iteration# 600: 0.249720
 Cost after iteration# 700: 0.249602
 Cost after iteration# 800: 0.249441
 Cost after iteration# 900: 0.249196
 Cost after iteration# 1000: 0.248766
 {'W1': array([[-0.59247105, -0.47282144],
        [-2.06763357, -0.23592616]]), 'W2': array([[-0.43995116, -0.16049007]]), 'b1': array([[-1.50109455],
        [-1.77729809]]), 'b2': array([[-0.56827845]])}
 Neural Network prediction for example (1, 1) is 1
```

Podemos observar varios cambios y sobre todo una mejora significativa en la reducción y el valor final de la función de costo. Como primera observación tenemos que el valor desde la iteración #0 ya era la mitad de lo que fue el valor de costo final sin usar MSE, y luego también podemos ver una disminución constante del costo a lo largo de las iteraciones, con valores más bajos y un descenso más suave. Este comportamiento indica que el modelo está ajustando sus parámetros para minimizar el error cuadrático medio, lo cual es beneficioso para el modelo. Comparado con la pérdida logística, como se discutió en clase, MSE penaliza con mayor severidad los errores grandes, lo que puede llevar a una convergencia más rápida y precisa en algunos casos.