

**UNIVERSIDAD DEL VALLE DE GUATEMALA**

CC3092 - Deep Learning y Sistemas Inteligentes

Sección 21

Ing. Luis Alberto Suriano



*Excelencia que trasciende*

**DEL VALLE**  
GRUPO EDUCATIVO

## Laboratorio No. 1

José Pablo Orellana      21970

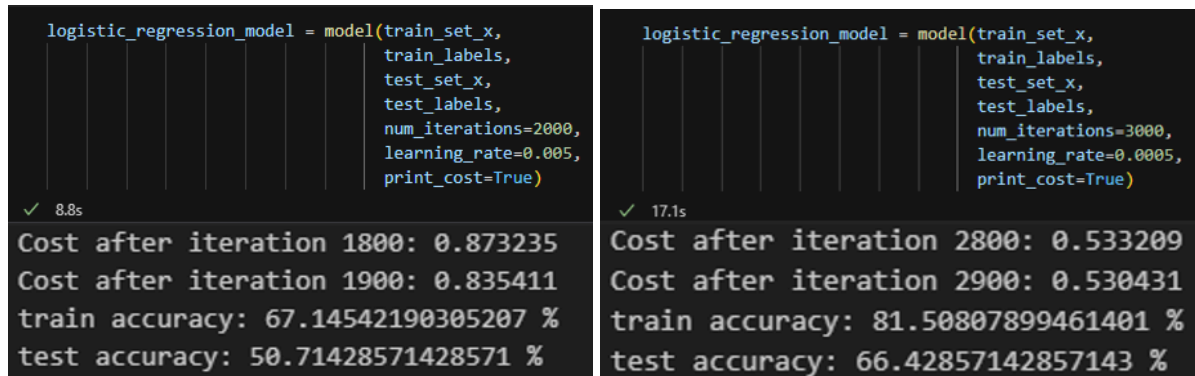
Diego Alberto Leiva      21752

**GUATEMALA, 19 de julio del 2024**

Repositorio Github: [https://github.com/JPOrellana/Laboratorio1\\_DL-SI.git](https://github.com/JPOrellana/Laboratorio1_DL-SI.git)

## Parte 1 - Regresión Logística como red neuronal

1. ¿Qué se podría hacer para mejorar el rendimiento de esta red?

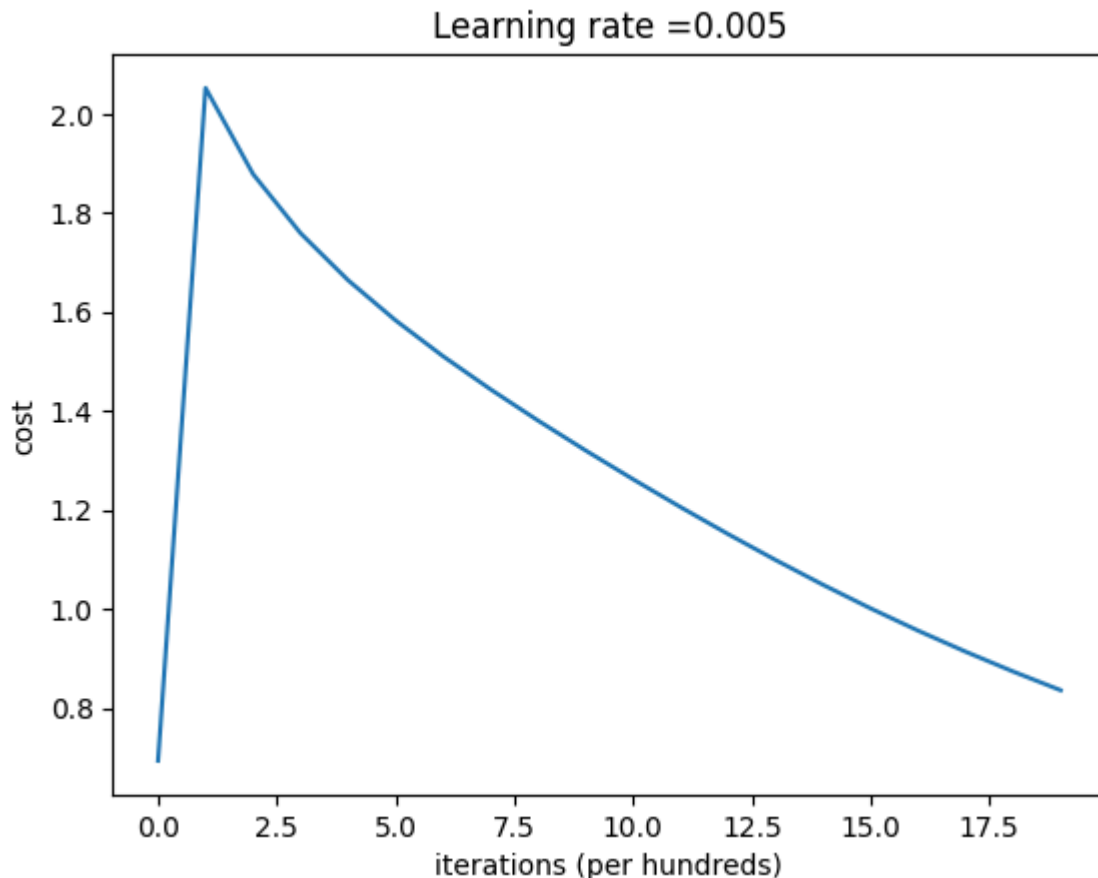


The image shows two terminal windows side-by-side, comparing the performance of a logistic regression model with different numbers of iterations. Both windows show the same code: `logistic_regression_model = model(train_set_x, train_labels, test_set_x, test_labels, num_iterations=2000, learning_rate=0.005, print_cost=True)` for the left and `num_iterations=3000` for the right. The left window shows results for 2000 iterations, while the right window shows results for 3000 iterations.

Metric	2000 Iterations	3000 Iterations
Execution Time	8.8s	17.1s
Cost after iteration 1800	0.873235	-
Cost after iteration 1900	0.835411	-
Cost after iteration 2800	-	0.533209
Cost after iteration 2900	-	0.530431
train accuracy	67.14542190305207 %	81.50807899461401 %
test accuracy	50.71428571428571 %	66.42857142857143 %

Dentro de las posibles mejoras que se le pueden dar al modelo está el modificar la cantidad de iteraciones y/o tasa de aprendizaje. Ya que al darle una mayor cantidad de iteraciones el modelo puede permitir que el algoritmo de optimización tenga más espacio para encontrar mejores parámetros, asimismo, el modificar la tasa de aprendizaje tiene un impacto en la convergencia del modelo. Mientras más pequeña sea la tasa puede darse el caso de que se quede estancado o si es muy grande no llegue a converger. En las capturas de arriba se evidencia como esa pequeña modificación incrementó significativamente la precisión durante el entrenamiento y también mostró un aumento de más de 10% con los datos de testeo. Sin embargo, incrementar más las iteraciones o disminuir más la tasa de aprendizaje puede provocar un overfitting ya que con los valores con los que se hizo la prueba evidencian que se está llegando a un overfitting debido a que el aumento en entrenamiento fue significativo pero en test no lo fue tanto.

Otra estrategia que se puede realizar es algún tipo de data augmentation, ya que con ello se tiene la posibilidad de aumentar la cantidad y diversidad de datos de entrenamiento sin necesidad de buscar más imágenes, esto puede ayudar al modelo a generalizar aún más las features de cada especie. Y otra posible estrategia es aumentar la complejidad del modelo, actualmente el modelo es una única regresión logística, es bastante simple por lo que el agregar “capas” puede conseguir que el modelo capture mejor las características de cada especie y generalice mejor los datos.



## 2. Interprete la gráfica de arriba

El gráfico nos muestra como el costo cambia a lo largo de las iteraciones durante el entrenamiento del modelo de regresión logística. En el eje 'X' se encuentra la cantidad de iteraciones en cientos, y el eje 'Y' representa el costo o pérdida de cada iteración. Para este modelo se utilizó una tasa de aprendizaje de 0.005.

Al inicio se muestra que el costo es bajo, y en las siguientes iteraciones, aproximadamente por la iteración 100, este se incrementa significativamente formando un pico en el gráfico, esto se debe a que al comienzo del entrenamiento los pesos están inicializados en 0 y el modelo aún no ha aprendido nada. Posteriormente se genera una disminución relativamente rápida por las siguientes cientos de iteraciones, lo que indica que el modelo ha aprendido rápidamente algunas características, seguido sigue una disminución gradual pues a medida que se avanza por las iteraciones la tasa de disminución del costo se vuelve más lenta, indicando que el modelo está convergiendo.

## Parte 2 - Red Neuronal Simple con PyTorch

### ¿En qué consiste “optim.SGD”?

- optim.SGD Es un optimizador en PyTorch que implementa el algoritmo de Gradiente Descendente Estocástico. Este algoritmo es uno de los métodos más básicos y populares para minimizar la función de pérdida en modelos de aprendizaje profundo. La idea principal detrás de SGD es actualizar los parámetros del modelo utilizando el gradiente de la función de pérdida respecto a los parámetros. La actualización se realiza en pequeños pasos, llamados "tasa de aprendizaje" o "learning rate" (Goodfellow et al., 2016).

### ¿En qué consiste “nn.NLLLoss”?

- nn.NLLLoss Es una función de pérdida en PyTorch que se utiliza comúnmente en problemas de clasificación multi-clase. Esta función mide la discrepancia entre la distribución de probabilidades predicha por el modelo y la distribución verdadera. Específicamente, se usa en combinación con la función de activación Softmax en la última capa de la red neuronal (Goodfellow et al., 2016).

### ¿Qué podría hacer para mejorar la red neuronal, y si no hay mejoras, por qué?

- Para mejorar la red neuronal, se pueden tomar en cuenta otras estrategias que puedan abarcar diferentes aspectos del modelo y del entrenamiento. En primer lugar, el ajuste de hiperparámetros es crucial. Se pueden probar con diferentes tasas de aprendizaje, ajustar el número de épocas y modificar el tamaño de los mini-lotes. Además, la arquitectura de la red es otro factor importante. Se podría aumentar el número de capas ocultas o el número de unidades por capa y cambiar la arquitectura a una red convolucional (CNN). La regularización también juega un papel esencial en la prevención del sobreajuste. Implementar Dropout puede ayudar a mejorar la generalización del modelo, y utilizar técnicas como Batch Normalization puede estabilizar y acelerar el entrenamiento.

## **Referencias**

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.