Universidad Simón Bolívar

Departamento de Electrónica y Circuitos

EC5724: Redes Neuronales y Aprendizaje Profundo

Adrián González 14-10433 Anderson Contreras 16-11350

Actividad 1

Para esta actividad, se procedió a implementar una clase en Python que emulara el funcionamiento de una neurona de una red neuronal o perceptrón. Dicha clase está definida en el archivo *perceptron.py* y consta de cinco métodos:

- **Init**: constructor de la variable de la clase, donde se inicializan los pesos y el bias con valores entre -1 y 1 con distribución uniforme.
- **Feed**: función que realiza los cálculos correspondientes al hacer pasar la data a través del perceptrón y de la función de costo.
- **Grad_desc**: función que calcula el gradiente de los pesos y del bias.
- **Back_prop**: función donde se reajustan los pesos y el bias mediante retropropagación.
- **Print_metrics**: función que calcula e imprime las métricas de clasificación (actividad 1.2).

Actividad 1.1 – Compuerta AND

Se creó un dataset que contiene las cuatro combinaciones posibles de las dos entradas de la compuerta y se realizó el entrenamiento de la red. Variando manualmente la cantidad de épocas y el learning rate, se obtienen resultados óptimos para **400 épocas con un learning rate de 1, llegando a minimizar la función de costo hasta 0.04292.**

La estimación del perceptrón ante las posibles entradas se muestra a continuación.

Entradas		**Salida**	
A	**B**	**Y**	**Ŷ**
0	0	0	1.95E-04
0	1	0	4.89E-02
1	0	0	4.89E-02
1	1	1	9.31E-01

Tabla 1. Tabla de la verdad y estimación del perceptrón.

La siguiente imagen muestra la evolución de la función de costo en función de las épocas.

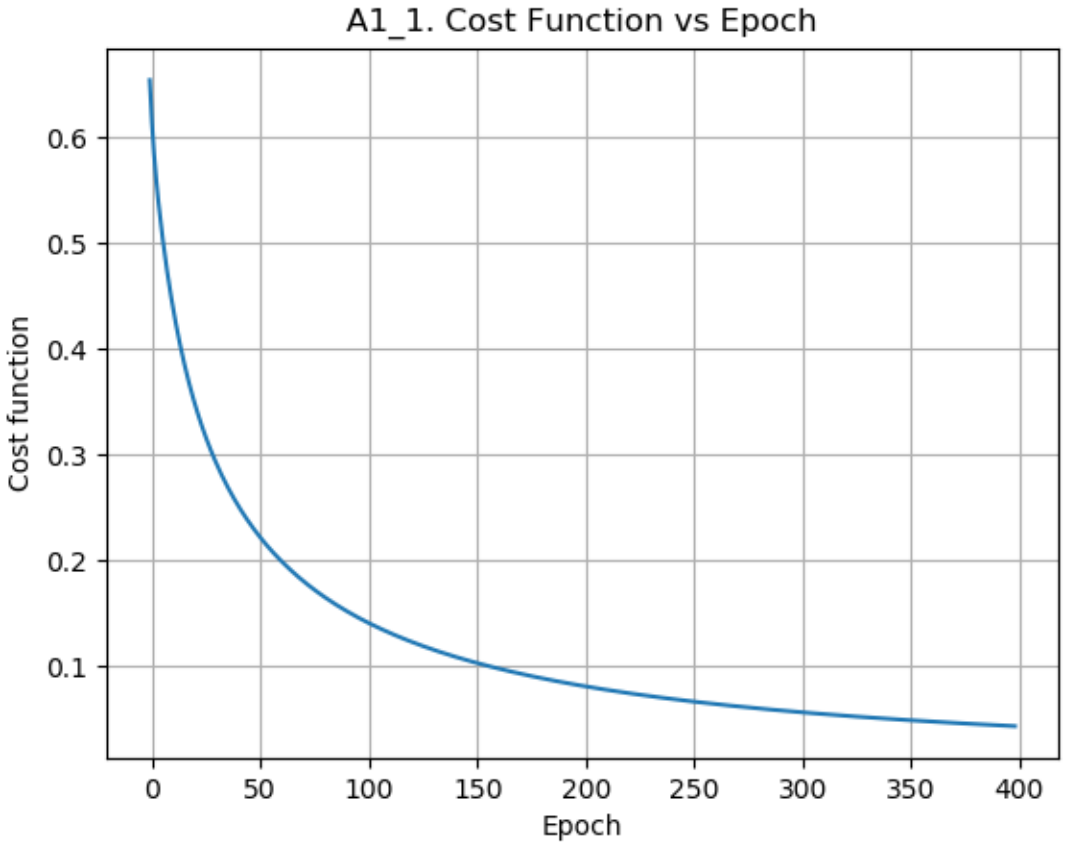

Figura 1. Variación de la función de costo en función del número de épocas.

Actividad 1.2 – Clasificación de imágenes

El dataset suministrado puede ser interpretado como un diccionario de tres claves (keys):

- **train_set_x**: set de 209 imágenes de 64x64 pixeles con los 3 canales RGB. Para pasar el set por el perceptrón, se redimensiona cada imagen en forma de vector a fin de que cada pixel sea una entrada, es decir, el set queda de dimensiones (209,12288). Además, se ha de normalizar, dividiendo entre 255, para evitar que se sature la función de activación.
- **train_set_y**: vector compuesto de 0 y 1 para comparar con la salida estimada del perceptrón durante su entrenamiento.
- **list_classes**: vector que indica el significado del 0 (non-cat) y del 1 (cat)

Iterando manualmente, se obtuvieron resultados satisfactorios para **1500 épocas y un learning rate de 0,3**. La matriz de confusión queda:

		Predicción	
	M = 209	**Gato**	**No Gato**
Real	**Gato**	**TP** 72	**FN** 0
	No Gato	**FP** 0	**TN** 137

Tabla 2. Matriz de confusión

Además, se tiene que **la exactitud es del 100%, la precisión del 34,4%, el recall 100% y la puntuación F1: 51,2%, y el valor de la función de costo fue 0.0212 en la última época**. La evolución de la función de costo se muestra en la siguiente imagen.

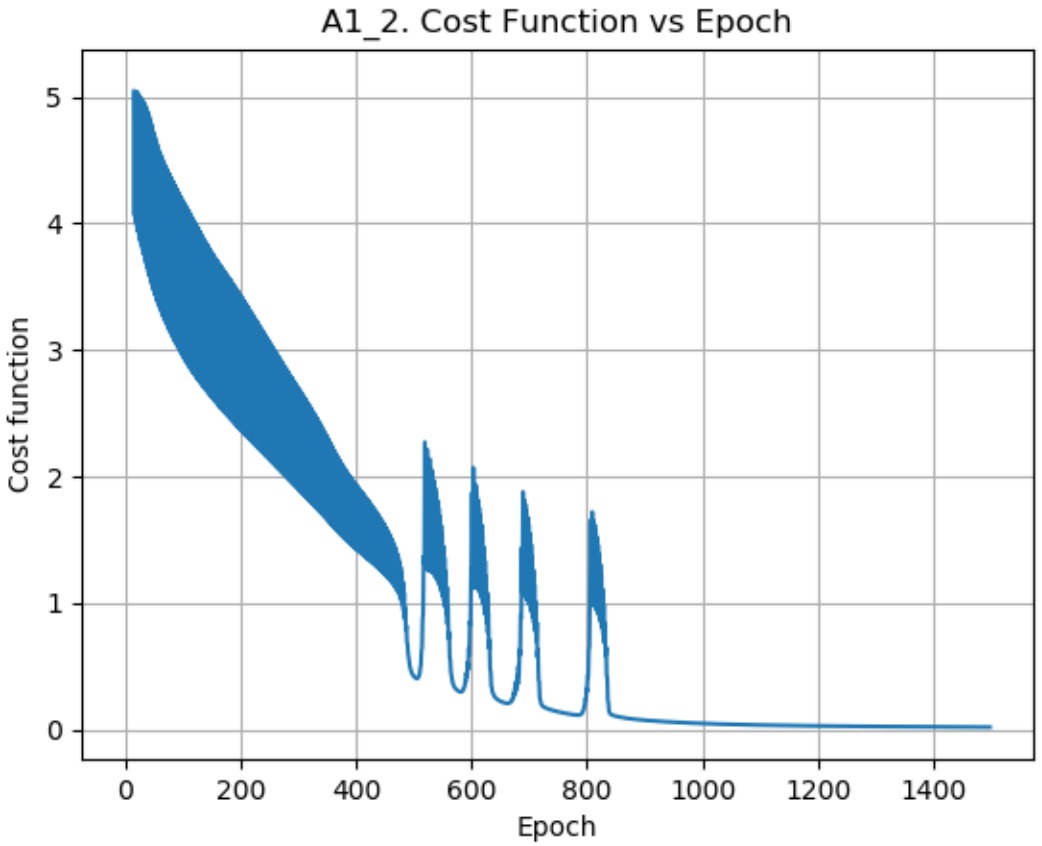

Figura 2. Variación de la función de costo en función de la época.

Por último, luego del entrenamiento, se procede a seleccionar aleatoriamente y graficar dos imágenes que hayan resultado como TP y dos como TN.

Figura 3. Resultados aleatorios de muestra.