|  |  |
| --- | --- |
| **Ex No: 3.2**  **Date: 21-08-2024** | **Using the deep neural network built in lab 3.1 to classify a given image as cat versus non-cat.** |

**Objective:**

The objective of this experiment is to implement and apply a deep neural network (DNN) for supervised learning tasks, specifically to classify images of cats and non-cats. The experiment involves building the model architecture step by step, initializing parameters, and applying forward and backward propagation to optimize the model.

**Descriptions:**

A 2-class classification neural network is designed to categorize data into one of two distinct classes. In this experiment, the neural network will be tasked with classifying images into binary categories, such as determining whether an image contains a cat or not. The neural network model will be built with a single hidden layer and will utilize non-linear activation functions to better capture complex patterns within the data.

The dataset used for this classification task consists of images, which are stored in variables

X (features) and Y (labels). The objective is to train the neural network to correctly classify these images into two categories. Unlike logistic regression, which is a linear model and might struggle with more complex patterns, the inclusion of a hidden layer in our neural network allows the model to capture non-linear relationships in the data, thereby improving classification performance.

Model Structure:

**Input Layer:** The number of neurons in the input layer corresponds to the number of features in the dataset.

**Hidden Layer:** The hidden layer introduces non-linearity into the model. The neurons in this layer use the tanh activation function, which maps input values to a range between -1 and 1, capturing complex relationships between the features.

**Output Layer**: The output layer contains a single neuron, which uses the sigmoid activation function to produce a probability that the input data belongs to one of the two classes.

**Training Process:**

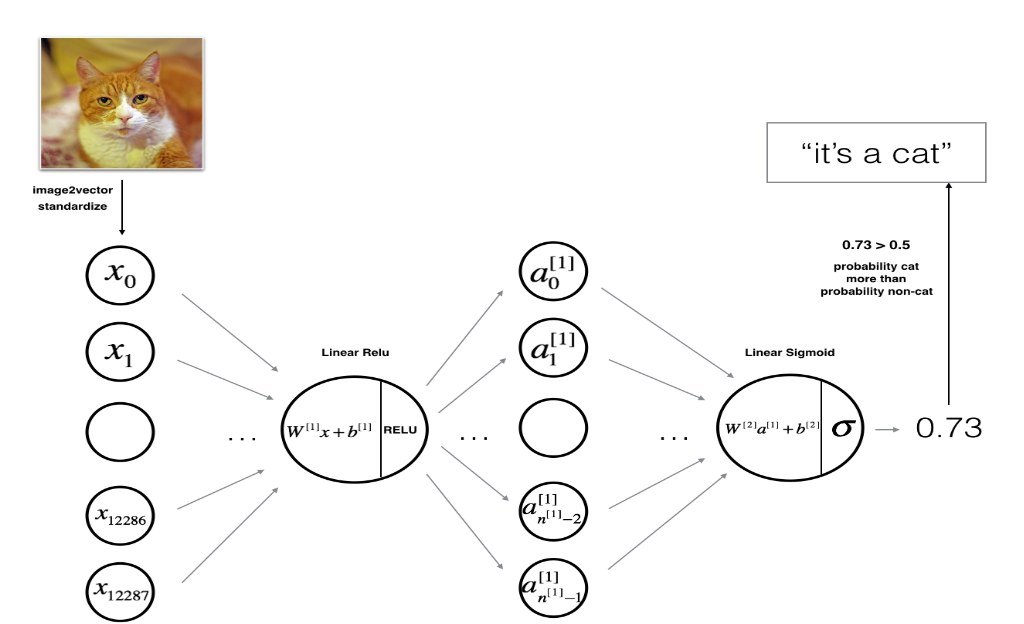
The model is trained using a loss function, specifically the cross-entropy loss. This loss function measures the difference between the predicted probability distribution and the actual labels, guiding the network to learn accurate classifications. During training, the model undergoes several iterations, where it refines its weights and biases to minimize the loss, thereby improving its classification performance.

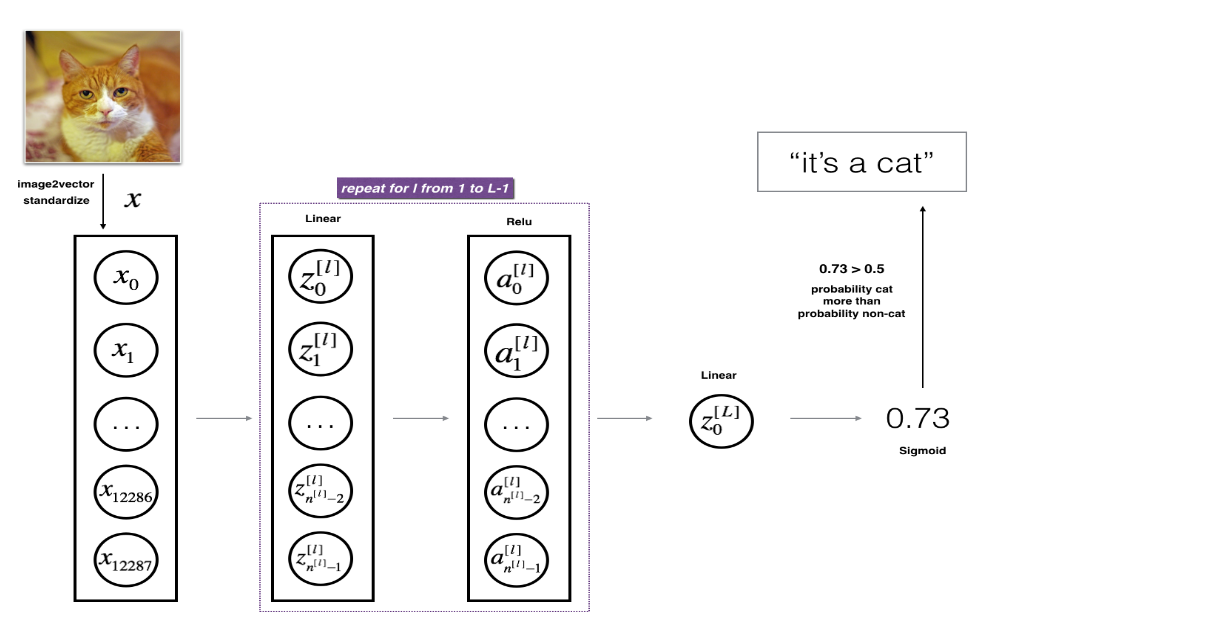
**Forward Propagation:**

In the forward propagation step, the input data is passed through the network layers to calculate the predicted output. The hidden layer applies the tanh activation function to the weighted sum of the inputs, and the output layer applies the sigmoid function to produce a final prediction.

**Backward Propagation:**

Backward propagation is the process of adjusting the model's weights to minimize the loss function. It involves calculating the gradients of the loss with respect to each weight and bias, then updating the parameters in the direction that reduces the loss. This iterative process is crucial for training the model to accurately classify the input data.

****

****

**Steps to Build the Model:**

**Define the Model Structure:**

* Specify the sizes of the input layer (nxn\_xnx​), hidden layer (nhn\_hnh​), and output layer (nyn\_yny​).
* nxn\_xnx​: Number of input features (e.g., number of pixels in an image).
* nhn\_hnh​: Number of neurons in the hidden layer.
* nyn\_yny​: Number of output classes (in this case, 1 for binary classification).

**Initialize the Model’s Parameters:**

* Initialize the weight matrices and bias vectors for the hidden and output layers.
* Weights are usually initialized randomly to break symmetry, while biases can be initialized to zeros.

**Forward Propagation:**

* Compute the linear combination of inputs and weights for the hidden layer
* Apply the tanh activation function to the hidden layer output
* Compute the linear combination for the output layer
* Apply the sigmoid activation function to the output layer to produce the final prediction

**Compute the Cost:**

* Calculate the cross-entropy loss

**Backward Propagation:**

* Compute the gradient of the loss with respect to the output layer’s parameters.
* Calculate the gradients for the weights and biases of the output layer
* Propagate the gradient to the hidden layer
* Compute the gradient of the tanh function and the corresponding weights and biases

**Update the Parameters:**

* Update the weights and biases using the gradients computed during backward propagation

**Repeat for Multiple Iterations:**

* Loop through the forward propagation, cost computation, backward propagation, and parameter update steps for a set number of epochs or until the model converges (i.e., the cost no longer decreases significantly).

**Evaluate the Model:**

* After training, test the model on a separate validation or test set to evaluate its performance.
* Analyze the effects of varying the hidden layer size and explore potential issues like overfitting by monitoring the training and validation accuracy.

**GitHub Link:**

**https://github.com/HiddenMachine3/IDL/tree/main/Lab3.2**