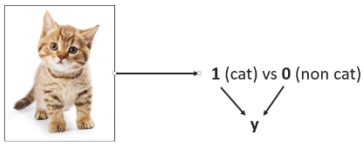
Ex No: 3 Date: 21/08/24	Deep Neural Network for Image Classification

Objective:

To build and train a Deep Neural Network (DNN) model to classify images into cat and non-cat categories.

Descriptions:

In this experiment, we aim to implement a Deep Neural Network (DNN) for binary classification. The model will take an input image X and output a prediction y, where y=1 indicates that the image contains a cat, and y=0 indicates a non-cat object. The DNN model consists of multiple layers with weights initialized randomly. The model undergoes forward propagation to compute the loss and backward propagation to update the weights using gradient descent.

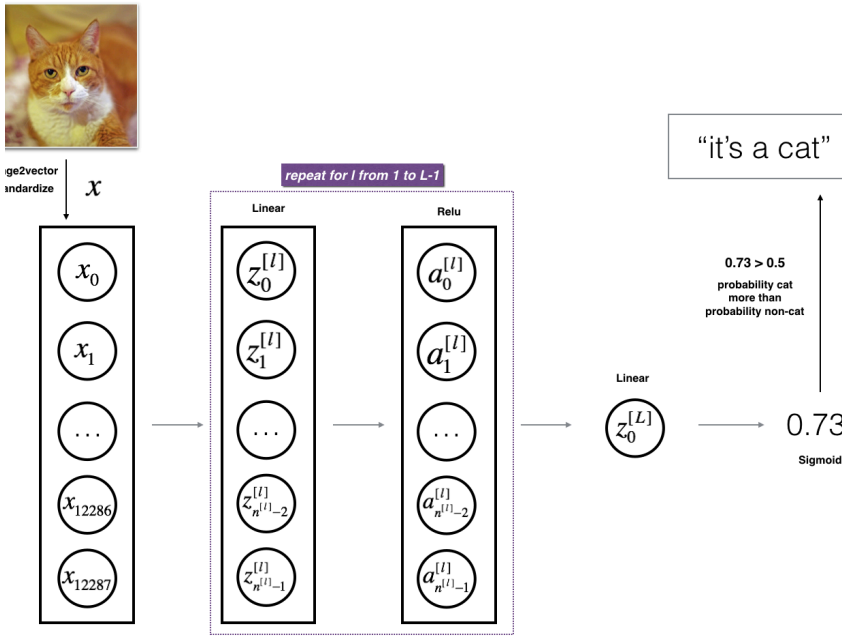


A Deep Neural Network (DNN) is a supervised learning algorithm used for complex classification tasks, such as distinguishing between cat and non-cat images. Unlike logistic regression, which has no hidden layers, a DNN consists of multiple layers of neurons that learn hierarchical representations of the input data. Each layer applies a linear transformation followed by a non-linear activation function, such as ReLU for hidden layers and sigmoid for the output layer. The initial weights are typically initialized randomly. As the model iterates through the training data, the weights are adjusted based on the gradients of the loss function, enabling the DNN to gradually learn the features necessary to accurately classify the images. Over time, the network fine-tunes these weights to minimize the error, improving its predictive performance with each iteration.

Model:

The model structure is defined with the following layers:

1. **Input Layer:** Takes in the pixel values of the image.
2. **Hidden Layers**: Multiple layers with ReLU activation.
3. **Output Layer:** A single neuron with sigmoid activation to produce a binary output (0 or 1).



Building the parts of algorithm

The steps involved in building the DNN are:

1. **Initialize Parameters:**
   Define and initialize the weights $W^{[l]}$ and biases $b^{[l]}$ for each layer *l*. The parameters are typically initialized as follows:
   $W^{[l]} = random\ values,\ b^{[l]} = 0$

2. **Forward Propagation:**
   Compute the linear transformation and activation for each layer. For layer *l*, the operations are:
   $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$

   $A^{[l]} = g(Z^{[l]})$

   where g(·) is the activation function (e.g., ReLU or sigmoid).

3. **Compute Loss:**
   Calculate the cost function based on the predictions. For binary classification, the cost function J is typically the binary cross-entropy loss:

$$J = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(\widehat{y^{(i)}}) + (1 - y^{(i)})log(1 - \widehat{y^{(i)}})$$

   where $\hat{y}^{(i)}$ is the predicted output and $y^{(i)}$ is the actual label.

4. **Backward Propagation:**
   Compute the gradients of the cost function with respect to the parameters to minimize the cost. For layer l, the gradients are:

   $$\frac{\partial J}{\partial Z^l} = A^l - Y$$

   $$\frac{\partial J}{\partial W} = \frac{1}{m} dZ^l A^{(l-1)T}$$

   $$\frac{\partial J}{\partial b^l} = \frac{1}{m} \sum_{i=1}^{m} dZ^l$$

5. **Update Parameters:**
   Adjust the weights and biases using the computed gradients with a learning rate α.

GitHub Link:

https://github.com/reethuthota/Deep_Learning/tree/main/Lab3.2