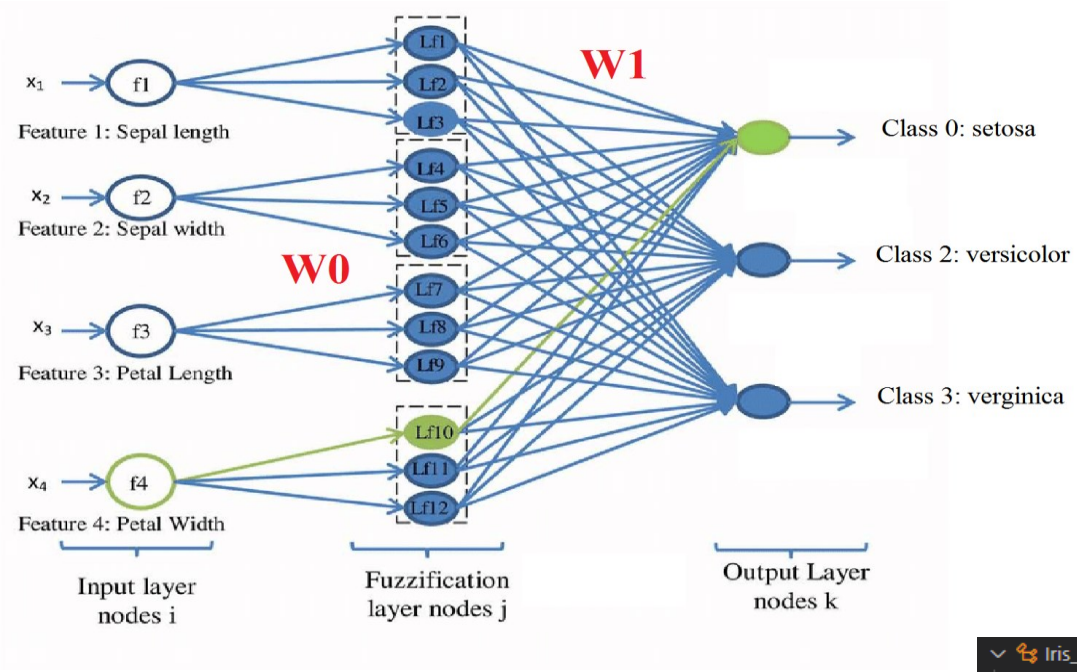
“Learning Activations in Neural Networks”

Nagarkoti Mohit Singh
7310628352
mohitnagarkoti1996@gmail.com
June 2020

Abstract— The choice of Activation Functions (AF) has proven to be an important factor that affects the performance of an Artificial Neural Network (ANN). Use a 1-hidden layer neural network model that adapts to the most suitable activation function according to the data-set. The ANN model can learn for itself the best AF to use by exploiting a flexible functional form, $\boldsymbol{k_0 + k_1 * x}$ parameters $\boldsymbol{k_0}$ **and** $\boldsymbol{k_1}$ being learned from multiple runs.

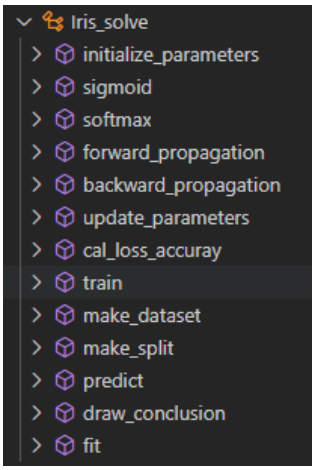
Introduction— Given a specific activation function $\boldsymbol{g(x) = k_0 + k_1}$ and categorical cross-entropy loss, A Neural Network on IRIS data has been created where the activation function parameters $\boldsymbol{k_0}$ **and** $\boldsymbol{k_1}$ has been learned by training on the IRIS data. The report consist learnable parameter values i.e final $\boldsymbol{k_0}$ **and** $\boldsymbol{k_1}$ values at the end of training, some plots describing the data, changes in $\boldsymbol{k_0}$ **and** $\boldsymbol{k_1}$ values on each epoch, a graph depicting training and test loss, train vs test accuracy and a loss function plot. In code, **class Iris_solve** is made to implement and solve the above problem.

Terminologies used in Neural Network:



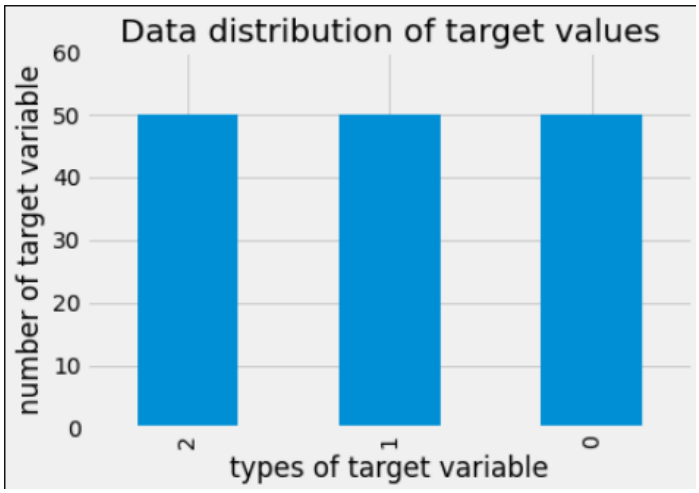
Implementation Details:

0. Code Structure:
    - Iris_solve is the class name
    - In order to fit the model to training data we Call fit method with labelled training examples
    - Training and validation set are separated outside the class.
    - Assighnment's solution are present outside the class separately.
    - **Githublink**: https://github.com/mohitnagarkotibca/Projects/tree/master/iris%20problem



1. Data Profile:
    - Data used was UCI's **IRIS dataset**
    - It was taken from Sklearn's datasets library.
    - It has 150 examples each example having 4 features.
    - The Target variable had 3 unique values which meant the data needed to be classified into 3 types

Before proceeding and for quality performance of data on model , We have to make sure it is a balanced dataset. So , following graph shows the dataset was a balanced dataset.



2. Parameter Initialization
    - *This function is implemented by initialize_parameters() method in Iris_solve.*
    - Class Iris_solve has used W_0 for K_0 and W_1 for K_1 .
    - Parameters for both layers were randomly initialized but it was made sure that their mean was 0 and the spread or standard deviation was around 0.3 to avoid the exploding gradient problem.
    - Iris_solve uses parameters variable to pack the W_0 and W_1 variables.

3. Parameter updates on epoch

```
epoch : 0  W0 mean: -0.0055  W1 mean: 0.0004236
epoch : 1000  W0 mean: -0.02182  W1 mean: 0.0004235
epoch : 2000  W0 mean: -0.114  W1 mean: 0.0004233
epoch : 3000  W0 mean: -0.14309  W1 mean: 0.0004232
epoch : 4000  W0 mean: -0.15078  W1 mean: 0.0004231
Fitting Done !
```

In the above screenshot, we can see how on every 1000th epoch, the mean of W0's layer parameters and W1's layer parameters mean is changing.

4. Final Parameter values at the end of training

```
parameters
```

```
{'W0': array([[-0.21937178,  0.64736599, -0.22956516, -0.03535584, -0.28901175,
         -0.19833516,  0.61725332,  0.23545543],
        [ 0.32814223, -1.08067928,  0.31826403,  0.29589938,  0.34013169,
          0.40646707, -0.84561112, -0.11657158],
        [-0.44832271,  1.26765191, -1.09576132, -1.22906276, -0.94117306,
         -1.10370251,  1.08895352,  0.55357139],
        [-0.46554168,  1.2643646 , -1.32156918, -1.74285595, -1.23030517,
         -1.44615628,  1.21648144,  0.63412213]]),
 'layer1_biases': array([[1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 'W1': array([[ 0.74835839, -0.27607857, -0.47797234],
        [-2.93492206,  0.87977919,  2.05116472],
        [ 1.15303267,  0.50338279, -1.66789969],
        [ 1.57351585,  0.89842093, -2.50561206],
        [ 1.06432985,  0.3636181 , -1.4396097 ],
        [ 1.35849543,  0.547108  , -1.87802032],
        [-2.37148917,  0.53445265,  1.83644382],
        [-0.62245278, -0.36724628,  1.03935466]]),
 'layer2_biases': array([[1.],
        [1.],
        [1.]])}
```

5. Train vs test _loss

```
Train loss : 7.547093415731551
Test loss : 3.7534671000691144
```

6. Train vs Test accuracy

```
Train accuracy : 97.32142857142857
Test acccuracy : 0.9473684210526315
```

7. Classification report

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	40
versicolor	0.97	0.95	0.96	37
virginica	0.94	0.97	0.96	35
accuracy			0.97	112
macro avg	0.97	0.97	0.97	112
weighted avg	0.97	0.97	0.97	112

8. **F1 score**

```
F1 Score: 0.9732194537085527
```

9. Plot of cost function vs number of epochs