SIMPLE FEED - FORWARD NEURAL NETWORK

Overview :

Architecture	2 layered neural-network
No. of hidden neurons	26
No. of output neurons	26
Weight initialization	Uniform Xavier
Activation function of hidden layer	RelU / Tanh
Activation function of output layer	Softmax
Loss function	Cross - Entropy
Optimizer	Adam

Preliminary tasks : First of all, the dataset was loaded as a panda dataframe and then the target feature-column was moved to the end of the same. This was completely a personal preference to deal with the dataset in general. Labels were then organised. Observing it the task at hand to be a classification problem, the categorical classes were converted to numeric counterparts using *Label Encoder*, and then the target feature was converted to vectors with the help of *One-Hot Encoder*. Total possible classes are 26, hence *n_outputs :* total number of neurons in the output layer and the length of the vectored targets were kept 26. Next, in accordance with standard theory, the dataset was then split into training and testing sets with ratio 80-20. All chunks of datasets were then converted to numpy-arrays to facilitate easy matrix operations involved.

Weight Initialization : As instructed, the *Xavier weight initialisation* procedure was implemented. The used method, *Uniform Xavier initialization* involves drawing weights from a random uniform distribution in range $[-x, x]$ where x is given by the formula:

$$x = \sqrt{\frac{6}{n\_inputs + n\_outputs}}.$$

Optimization Method : *Adam optimizer* was built from scratch and used to minimize the loss. The pseudo-code for the same goes by:

```
momentum1 = 0
momentum2 = 0

for i in range(epochs):
    dw = compute_gradient(w)
    momentum1 = decay1 * momentum1 + (1- decay1) * dw
    momentum2 = decay2 * momentum2 + (1- decay2) * dw * dw
    w -= learn_rate * momentum1 / (np.sqrt(momentum2) + 1e-7)
```

Loss Function : *Cross-Entropy loss function* was used which goes by the formula:

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N}\sum_{i}^{N} y \log \hat{y} = -\frac{1}{N}\sum_{i}^{N} [y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)].$$

Hyper-parameters : In standard texts, hyper-parameters are those parameters which are not taken input and neither are dependant on any procedure or flow of data in the entire task. The significant hyper parameters involved are as follows:

- **Number of neurons in the hidden layers :** In code it is *n_hidden*. At present there are two major schools of thought which use their respective rule-of-thumb for the number of neurons in the hidden layer. One says that the number of hidden neurons should be between the size of the input layer and the size of the output layer. Also, another says that the number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer. In the present context, the later say the number should be 28. Hence, also considering the former reasoning, the number here is kept 26.

- **Decay rates for Adam optimizer :** Here, apparently the decay rates used in the Adam optimizer are also hyper-parameters. Intuitively, the parameters were kept 0.9 and 0.88. But due to unsatisfactory results they were tweaked to 0.8 and 0.88.
- **Constant to support momentum :** Well in the last line of the pseudo-code given for Adam optimizer, there is a constant which supports the momentum. In the following code line it is the last constant. Here $1e^{-7}$ is used where as $1e^{-8}$ has also evidently seen to be used.

```
w -= learn_rate * momentum1 / (np.sqrt(momentum2) + 1e-7})
```

Methodology : The significant elements of the methodology has been put as follows. Basically, we analyse the the inputs/hyper-parameters whose combinations has been used to get an optimum result.

- **Epochs :** Epochs basically means the total iterations for which the dataset has undergone through the network. Well, for epochs = 500, 3500, 5000, the accuracy obtained on the training dataset was not satisfactory. But for epochs 10000, 15000, 20000, the same where pretty decent.
- **Learning rate :** Learning rates are intuitively a measure of how quickly the parameters/ weights get the optimized state. For learning rate 0.05, results obtained on the training dataset were decent. When 0.065 was used as the learning rate, results obtained were not better than what was expected, whereas good results were seen for learning rates 0.035, 0.02 and 0.01. Well, for a quick remark, the dataset has way too less data points for a standard neural network and hence for this low learning rates are working.

Results : The results obtained on the test data with corresponding combinations of parameters are as follows: 2

***	Ep=10000	Ep=15000	Ep=20000
Lr=0.05	41.10	19.22	86.10
Lr=0.035	87.67	81.87	88.52
Lr=0.02	89.52	89.72	90.17
Lr=0.01	89.00	88.62	90.30

Tabelle 1: Accuracy with RelU function with respective Epochs and Learning Rates

***	Ep=10000	Ep=15000	Ep=20000
Lr=0.05	85.70	81.02	83.92
Lr=0.035	85.95	86.65	85.10
Lr=0.02	87.72	90.10	85.5
Lr=0.01	89.67	88.97	90.30

Tabelle 2: Accuracy with Tanh function with respective Epochs and Learning Rates

Best configuration :

Learning Rate	Epochs	Activation Function	Accuracy (in %)
0.01	20000	Tanh*	90.30

*Even the relU algorithm at the same parameters gave the same accuracy, but the loss value for the Tanh counterpart is less.