Vanilla Feedforward Neural Network

Execution Instructions

1. Extract the vanilla.zip file in a directory
2. Open Terminal and change directory to vanilla root folder (where you'll find the vanilla.py and iris_model.py file)
3. Make sure you have sklearn, numpy and matplotlib libraries installed (or follow instruction below)

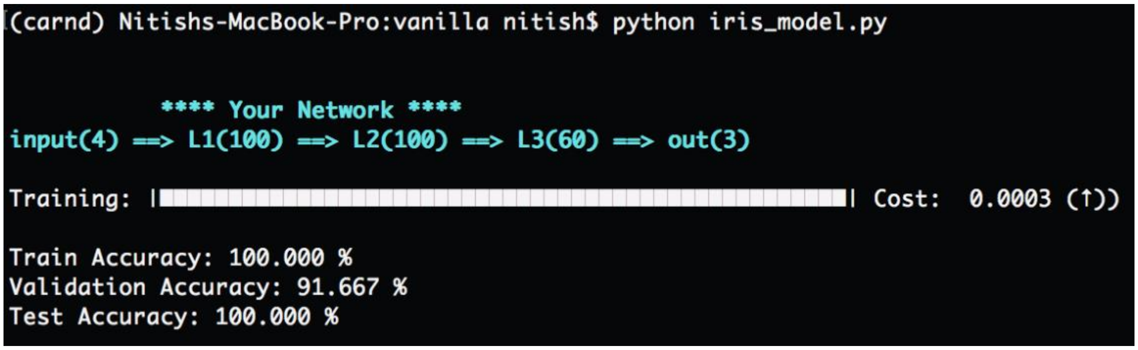
```
a.  Using Pip
    pip3 install -U scikit-learn
    python3 -m pip install numpy
    python3 -m pip install matplotlib

b.  Using Conda3
    conda install scikit-learn
    conda install -c anaconda numpy
    conda install -c conda-forge matplotlib
```

4. Run:

```
python3 iris_model.py
```

5. You will see an interface like below:



```
(carnd) Nitishs-MacBook-Pro:vanilla nitish$ python iris_model.py


           **** Your Network ****
input(4) ==> L1(100) ==> L2(100) ==> L3(60) ==> out(3)

Training: |██████████████████████████████████████████████████| Cost:  0.0003 (↑))

Train Accuracy: 100.000 %
Validation Accuracy: 91.667 %
Test Accuracy: 100.000 %
```

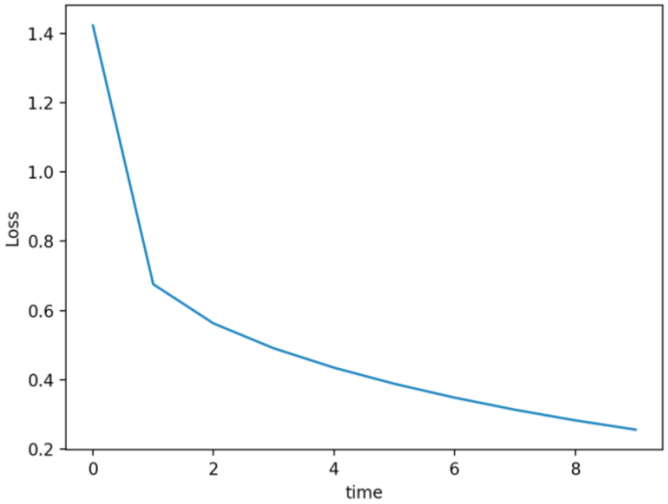
Evaluation

Network: input(4) ==> L1(100) ==> L2(100) ==> L3(60) ==> out(3)

Loss Function: Cross Entropy

Hidden Layer Activation	Train	Validation	Test
Sigmoid	98.22	97.32	97.77
Tanh	99.98	96.4	99.67
Relu	99.99	94.5	95.5

Loss Plot with Sigmoid activation for hidden layers:



Note: Vanilla can be used to train your model on different data than Iris, for instance, in *iris_model.py* you'll find commented out lines for Boston dataset. Try that out.

Functional Documentation Summary

Class: Vanilla

Method Description:

1. def add_layer(self, units, input_dim=0, activation="sigmoid")

   Description: Adds neural network layers with specified number of units and activation function

2. def compile(self, learning_rate=0.01, decay_rate = 0.001, loss="entropy")

   Description: To set Learning Rate, Learning Decay Rate and Loss Function

3. def fit(self, X, y, validation_split=0.2, shuffle=True, nb_epoch=5, steps_per_epoch=100, lambda_ = 0.01, normalize=False)

   Description:

   a. Splits data (X) into Train and Validation set
   b. Weight Initialization
   c. Draws Network art
   d. Forward and Backpropagation Model
   e. Weight Updates
   f. Loss Trend
   g. Regularization
   h. Prints Accuracy on Train and Validation set

4. def predict(self, X)

   Description: Predicts the Output labels on a trained network based on the given data (X)

5. def scores(self, y, y_pred)

Description: Returns a score dictionary comprising of several measures of a neural network performance

Following functions are auxiliary functions

6. def printProgressBar (self, cost, decrease, iteration, total, prefix = '', suffix = '', decimals = 1, length = 100, fill = '█')

   Description: Prints the Progress bar while training along with the loss after each epoch

7. def draw_network(self, layers, act)

   Description: Prints the Network layer diagram

8. def validate(self, X, y)

   Description: Validates the Input parameters such as data and label dimensions

9. def sigmoid(self, x)

   Description: Returns sigmoid of x

10. def d_sigmoid(self, x)

    Description: Returns derivative of sigmoid of x

11. def d_tanh(self, x)

    Description: Returns derivative of tanh of x

12. def softmax(self, x)

    Description: Returns derivative of sigmoid of x

13. def relu(self, x)

    Description: Returns the relu value of x

14. def d_relu(self, x)

    Description: Returns the derivative of relu value of x

15. def rate_decay(self, alpha_0, decay_rate, n)

Description: Returns the decayed value of learning rate based of the decay rate

16. def cross_entropy_loss(self, y_train, y_hat, epsilon=1e-11)

    Description: Calculates the loss based on cross entropy

17. def logistic_loss(self, y_train, y_hat, epsilon=1e-11)

    Description: Calculates the loss based on logistic log function

18. def l1_reg(self, x, lam)

    Description: Adds L1 regularization to avoid overfitting

19. def l2_reg(self, x, lam)

    Description: Adds L2 regularization to avoid overfitting

20. def forward(self, input, weight, bias, activation="sigmoid")

    Description: Maps the input to output based on the weights and biases of given layer

21. def backward(self, inp, out, w_out, dz_out, activation="sigmoid", output_layer=False)

    Description: Propagates the gradient of the previous layer to the given layer

22. def normalize(self, x)

    Description: Normalizes the data (x)

23. def confusion_matrix(self, y, y_pred)

    Description: Computes the confusion matrix based on the predictions

References

[1] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *“Deep Learning”*, 2016

[2] Machine Learning by Andrew Ng, Coursera

[3] CS231n: Convolutional Neural Networks for Visual Recognition

[4] http://ozzmaker.com/add-colour-to-text-in-python/