# 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)
   1. Using Pip  
      pip3 install -U scikit-learn  
      python3 -m pip install numpy  
      python3 -m pip install matplotlib
   2. Using Conda3  
      conda install scikit-learn  
      conda install -c anaconda numpy  
      conda install -c conda-forge matplotlib
4. Run:

python3 iris\_model.py

1. You will see an interface like below:



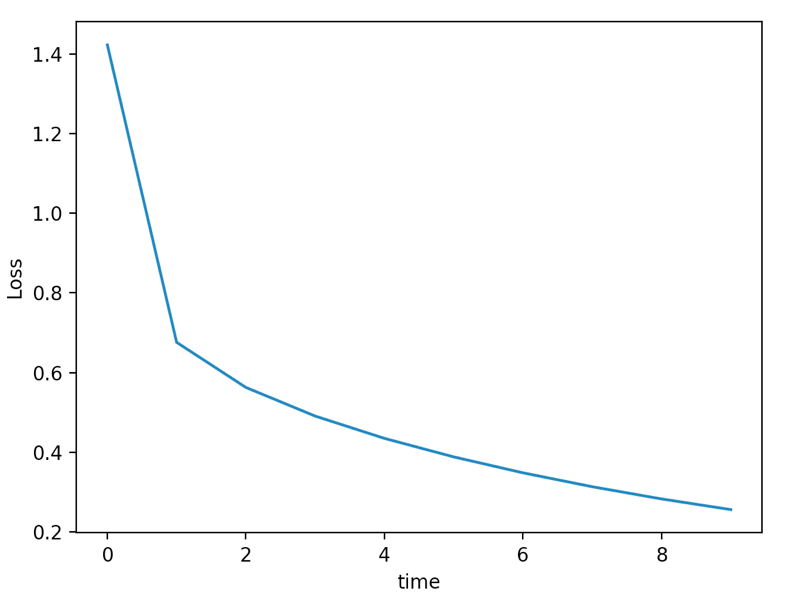
## Evaluation

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

This architecture is inspired by Keras which makes it elegant to add / remove layers and to set various associated parameters with minimum programming interface required by user.

**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

1. def compile(self, learning\_rate=0.01, decay\_rate = 0.001, loss="entropy")

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

1. def fit(self, X, y, validation\_split=0.2, shuffle=True, nb\_epoch=5, steps\_per\_epoch=100, lambda\_ = 0.01, normalize=False)

Description:

* 1. Splits data (X) into Train and Validation set
  2. Weight Initialization
  3. Draws Network art
  4. Forward and Backpropagation Model
  5. Weight Updates
  6. Loss Trend
  7. Regularization
  8. Prints Accuracy on Train and Validation set

1. def predict(self, X)

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

1. 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*

1. 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

1. def draw\_network(self, layers, act)

Description: Prints the Network layer diagram

1. def validate(self, X, y)

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

1. def sigmoid(self, x)

Description: Returns sigmoid of x

1. def d\_sigmoid(self, x)

Description: Returns derivative of sigmoid of x

1. def d\_tanh(self, x)

Description: Returns derivative of tanh of x

1. def softmax(self, x)

Description: Returns derivative of sigmoid of x

1. def relu(self, x)

Description: Returns the relu value of x

1. def d\_relu(self, x)

Description: Returns the derivative of relu value of x

1. def rate\_decay(self, alpha\_0, decay\_rate, n)

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

1. def cross\_entropy\_loss(self, y\_train, y\_hat, epsilon=1e-11)

Description: Calculates the loss based on cross entropy

1. def logistic\_loss(self, y\_train, y\_hat, epsilon=1e-11)

Description: Calculates the loss based on logistic log function

1. def l1\_reg(self, x, lam)

Description: Adds L1 regularization to avoid overfitting

1. def l2\_reg(self, x, lam)

Description: Adds L2 regularization to avoid overfitting

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

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

1. 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

1. def normalize(self, x)

Description: Normalizes the data (x)

1. 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/