Assignment 1 INFO 7374 Multilayer Perceptron on CIRAF-10 dataset

Vignesh Murali (NUID:001886775)

Abstract

The CIFAR-10 Dataset consists of 60,000 images belonging to 10 classes. We are attempting to build a feedforward neural network or multilayer perceptron to perform this task. 11 models are built and various parameters such as activation function, number of layers, neurons, learning rate and number of epochs are tweaked. Based on a result-driven approach, we subsequently attempt to build more efficient models. We finally identify a model which yields 0.559 accuracy and 1.29 loss which is relatively the best performance obtained among all my experiments.

Introduction

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

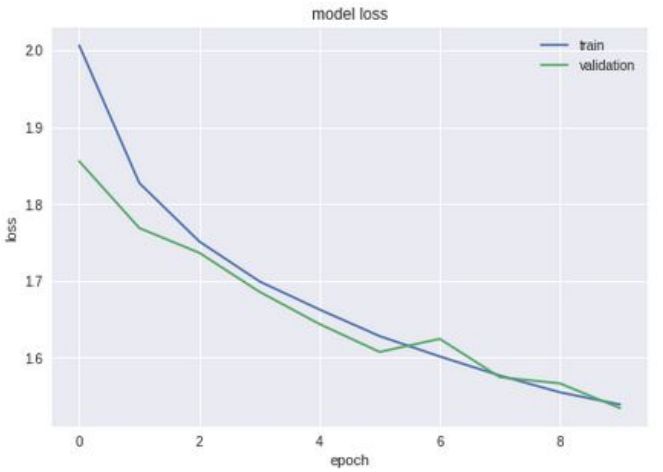
Approach

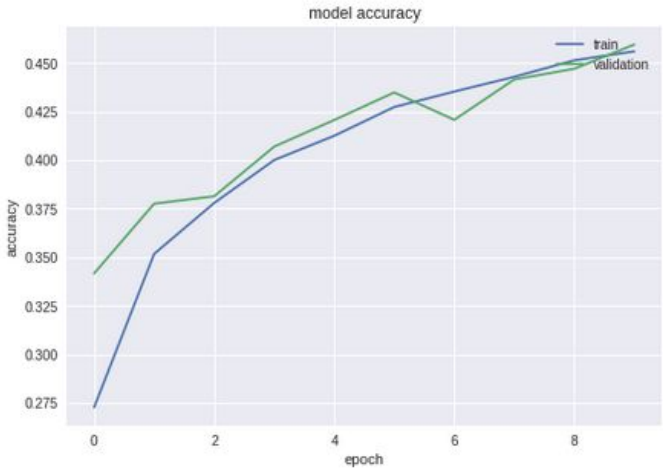
We will use a result-based approach where we tune multiple parameters and document the results and based on these results we build subsequent models to obtain optimal accuracy and low loss values. I have tuned the following parameters - 1. No of epochs 2. Batch size 3. Network configuration a. Number of neurons in a layer b. Number of layers 4. Learning rate 5. Activation functions 6. Dropout rates

Observations

- Model 1(Base Model)

Input layer = 512 neurons, ‘ReLu’ activation,1 Hidden layer = 512 neurons, ‘ReLu’ activation,Dropout 0.2,Output layer = ‘SoftMax’ Activation,SGD Optimizer with default learning rate,10 epochs,Batch size = 64





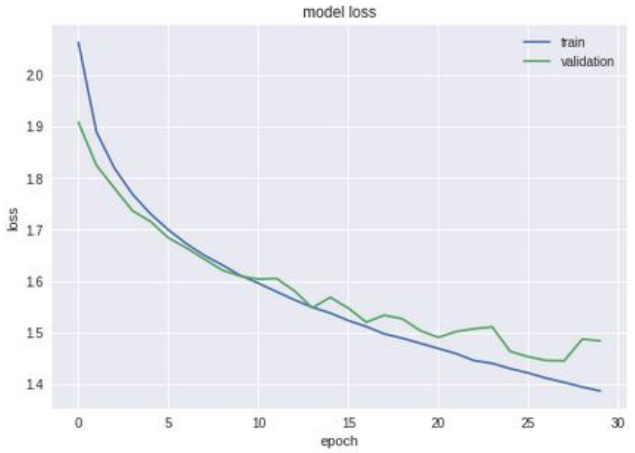
Training Accuracy: 0.4561 Training Loss: 1.5393
Validation Accuracy: 0.4596 Validation Loss: 1.5345
Test Accuracy: 0.4667 Test Loss: 1.50277

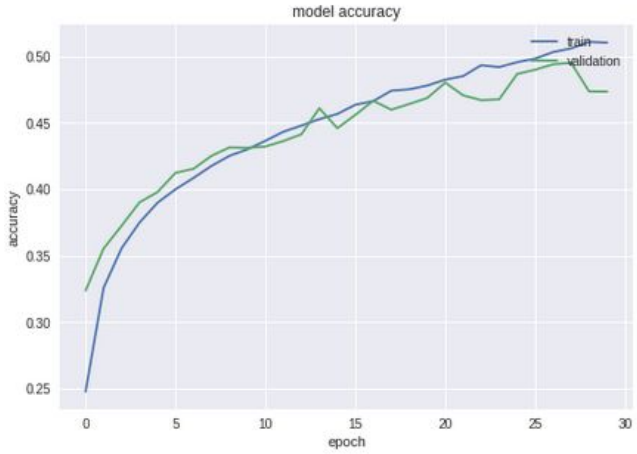
Observations:

- Our first model potentially underfits because of the small number of epochs.
- Training time was high because of small batch size.

- Model 2 (Adjusting Batch Size & Increasing epochs)

Input layer = 512 neurons, ‘ReLu’ activation,1 Hidden layer = 512 neurons, ‘ReLu’ activation,Dropout 0.2,Output layer = ‘SoftMax’ Activation,SGD Optimizer with default learning rate,**30 epochs,Batch size = 128**





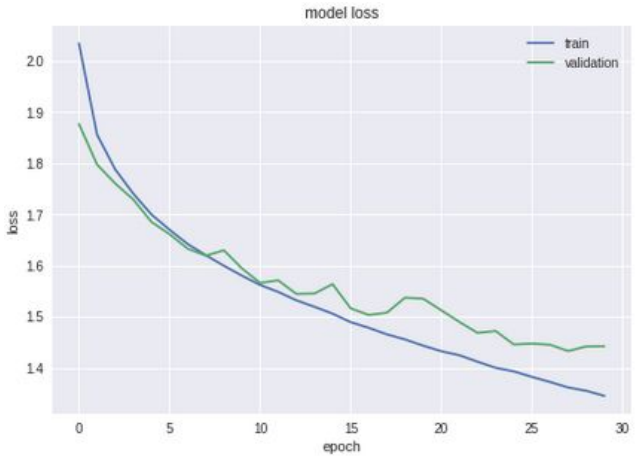
Training Accuracy: 0.5105 Training Loss: 1.3870
Validation Accuracy: 0.4736 Validation Loss: 1.4843
Test Accuracy: 0.4771 Test Loss: 1.4635

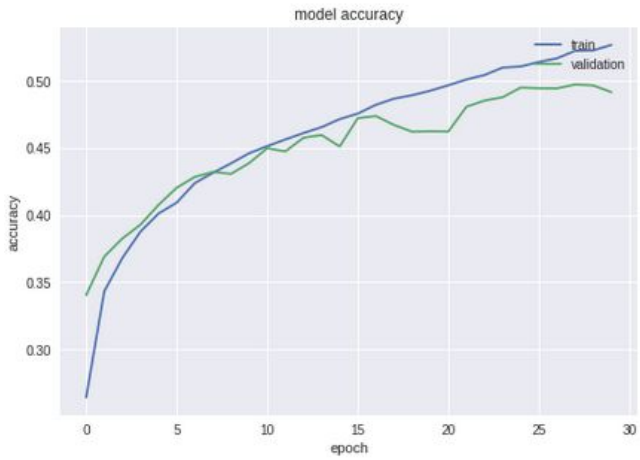
Observations:

- Computation was a lot faster because we increased batch size.
- The model slightly overfits after the 20th epoch.

- Model 3 (Adding more neurons to layers)

Input layer = 1024 neurons, 'ReLu' activation,1 Hidden layer = 1024 neurons, 'ReLu' activation,Dropout 0.2,Output layer = 'SoftMax' Activation,SGD Optimizer with default learning rate,30 epochs,Batch size = 128



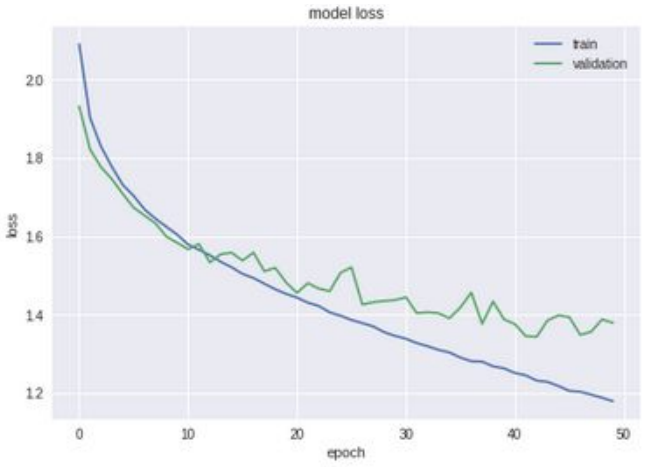


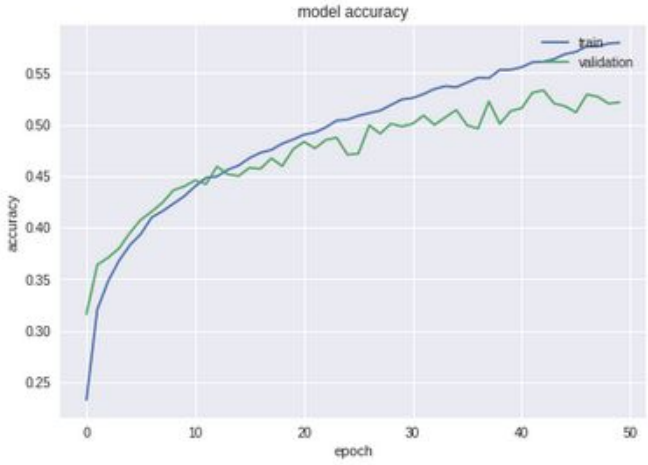
Training Accuracy: 0.5270 Training Loss: 1.3450
Validation Accuracy: 0.4918 Validation Loss: 1.4420
Test Accuracy: 0.4963 Test Loss: 1.41530

Observations:
- Slight increase in accuracy.
- Overfits sooner than previous experiments.

- Model 4 (Adding more layers)

Input layer = 1024 neurons, 'ReLu' activation**,2 Hidden layers = 1024 neurons 'ReLu' activation & 512 neurons 'ReLu activation'**,Dropout 0.2,Output layer = 'SoftMax' Activation,SGD Optimizer with default learning rate,50 epochs,Batch size = 128



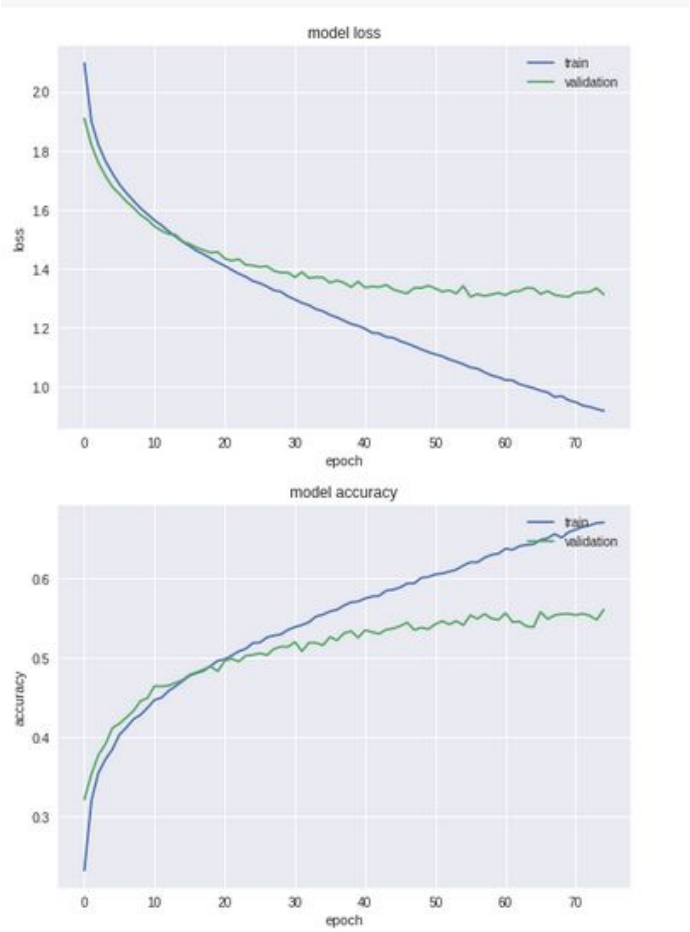


Training Accuracy: 0.5794 Training Loss: 1.1786
Validation Accuracy: 0.5216 Validation Loss: 1.3788
Test Accuracy: 0.5199 Test Loss: 1.35288

Observation:
- Lower loss, higher accuracy.
- Overfits after 40th epoch
- It is clear that on this dataset adding an extra hidden layer has been beneficial.

- Model 5 (Change learning rate)

Input layer = 1024 neurons, 'ReLu' activation,2 Hidden layers = 1024 neurons 'ReLu' activation & 512 neurons 'ReLu activation',Dropout 0.2,Output layer = 'SoftMax' Activation,**SGD Optimizer with 0.001 learning rate**,75 epochs,Batch size = 128

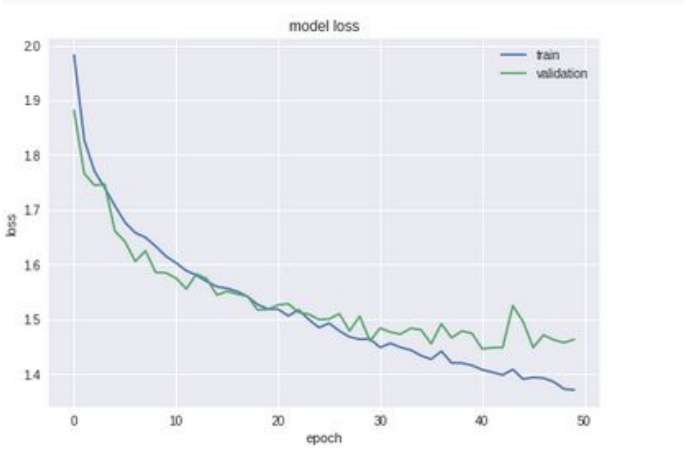


Training Accuracy: 0.5794 Training Loss: 1.1786
Validation Accuracy: 0.5216 Validation Loss: 1.3788
Test Accuracy: 0.559 Test Loss: 1.2958

Observations:
- Model overfits after the 20th - 25th epoch, yet performs relatively well.
- Reducing the learning rate has yielded a lower loss and high accuracy. Thereby, improving performance.
- Further lowering learning rate makes the model perform worse so we will stick with this learning rate for future experiments.

- Model 6 (Different Optimization Technique)

Input layer = 1024 neurons, 'ReLu' activation,2 Hidden layers = 1024 neurons 'ReLu' activation & 512 neurons 'ReLu activation',Dropout 0.2,Output layer = 'SoftMax' Activation, **Adam Optimizer with 0.001 learning rate**,50 epochs,Batch size = 128

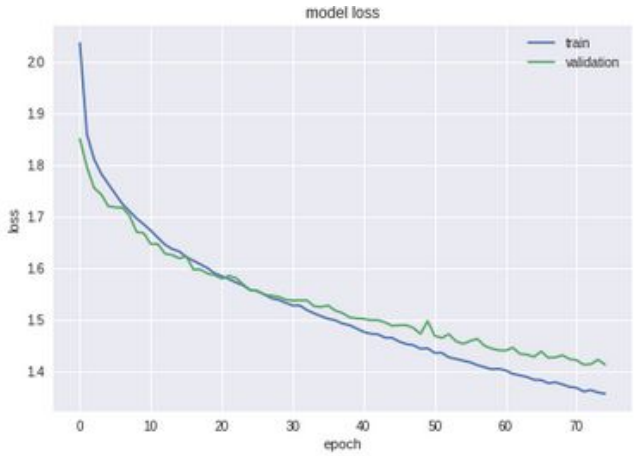


Training Accuracy: 0.5063 Training Loss: 1.3711
Validation Accuracy: 0.4869 Validation Loss: 1.4631
Test Accuracy: 0.4918 Test Loss: 1.4385

Observations:
- Overfits after the 40th epoch.
- Since SGD yielded better results than Adam, we will continue with SGD.

- Model 7 (Change activation functions)

Input layer = 1024 neurons, 'tanh' activation,2 Hidden layers = 1024 neurons 'tanh' activation & 512 neurons 'tanh activation',Dropout 0.2,Output layer = 'SoftMax' Activation, SGD Optimizer with 0.001 learning rate,75 epochs,Batch size = 128

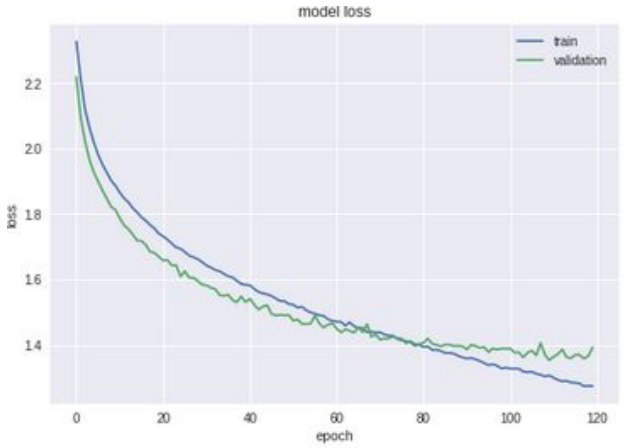


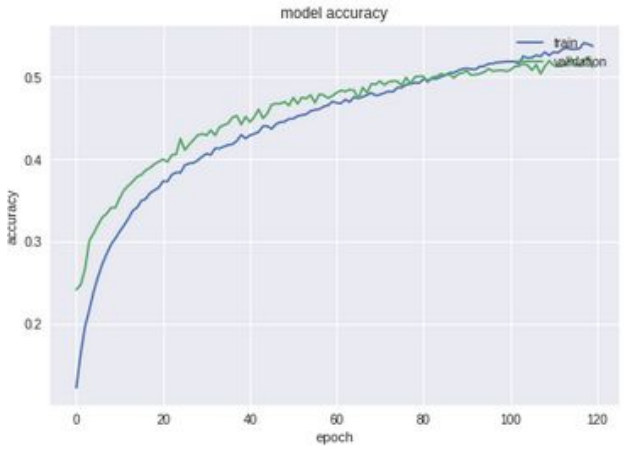
Training Accuracy: 0.5199 Training Loss: 1.3567
Validation Accuracy: 0.5003 Validation Loss: 1.4131
Test Accuracy: 0.5083 Test Loss: 1.388

Observations:
- The model overfits after the 45th epoch.
- The model performed better with relu activations.

- Model 8 (Changing dropout rates)

Input layer = 1024 neurons, 'ReLu' activation,2 Hidden layers = 1024 neurons 'ReLu' activation & 512 neurons 'ReLu activation',**Dropout 0.3**,Output layer = 'SoftMax' Activation, SGD Optimizer with 0.001 learning rate,120 epochs,Batch size = 128



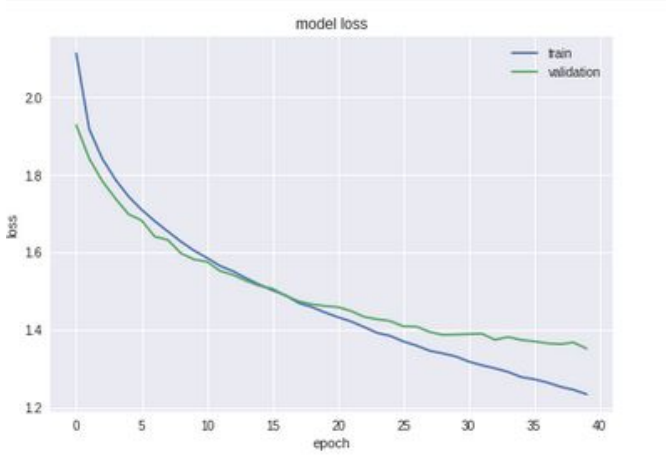


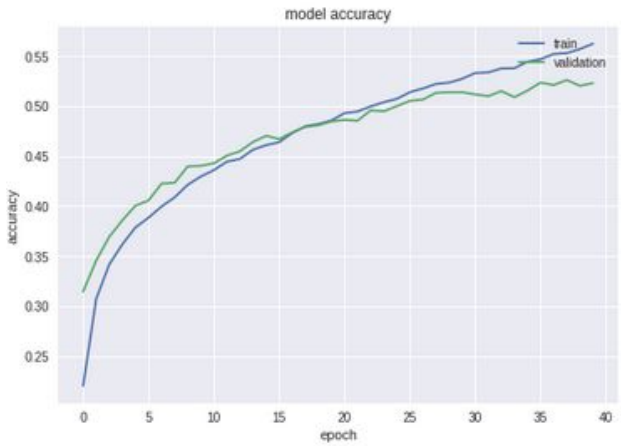
Training Accuracy: 0.5376 Training Loss: 1.2746
Validation Accuracy: 0.5119 Validation Loss: 1.3930
Test Accuracy: 0.5084 Test Loss: 1.318

Observations:
- Slightly overfits after the 100th epoch.
- Good performance, low loss.

- Model 9

Input layer = 1024 neurons, ‘ReLu’ activation,2 Hidden layers = 512 neurons ‘ReLu’ activation & 512 neurons ‘ReLu activation’,Dropout 0.2,Output layer = ‘SoftMax’ Activation, SGD Optimizer with 0.001 learning rate,40 epochs,Batch size = 128





Training Accuracy: 0.5622 Training Loss: 1.2346
Validation Accuracy: 0.5229 Validation Loss: 1.3522
Test Accuracy: 0.5323 Test Loss: 1.3267

Observations:
- Model overfits after 30th epoch.
- Relatively high accuracy and low loss compared to previous experiments.

Results

We can hereby conclude that Model 5 has yielded the best results on the CIFAR-10 dataset though it slightly overfits. I would recommend this model for classification.

The parameters of this model are as follows:
- Input Layer (1024 neurons, 'ReLu' Activation)
- 2 Hidden Layers (Layer #1 = 1024 neurons, 'ReLu' Activation
Layer #2 = 512 neurons, 'ReLu' Activation)
- Dropout rate is 0.2
- Optimization function SGD (with learning rate 0.001)
- Batch Size: 128

- Epochs: 75

Why is this best recommended model?

- Adding a 2nd hidden layer and increasing number of neurons increases performance on this dataset. Adding more layers adds dimensionality and for a complex problem such as image classification, it has proven to be beneficial.
- Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. Reducing the learning rate to 0.001 has lead to better performance. The lower the value, the slower we travel along the downward slope.

How to improve the model?

- Increasing the complexity of the model (by adding more hidden layers and neurons) could possibly improve the model's performance.

References

https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py
https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/
https://keras.io/getting-started/sequential-model-guide/