EN2550 2020: Assignment 04: 180437U

A function is predefined to pre-process input data: preproc(norm,reshape). “norm” and “reshape” are Boolean inputs. Pixel value normalization and reshaping the input data can be done by setting them to True.

**preproc(norm,reshape)**

Parameters,

* norm: Boolean value. Set to True, to normalize training and testing data sets.
* Reshape: Boolean value. Set to True, to reshape training and testing data sets.

1. Linear Classifier

Loss function and the accuracy function is predefined to reduce the complexity in coding.

* Loss function: Mean sum of squared errors with regularization.
* Accuracy: Normalized difference between the true label and the label with the highest score is used to calculate the accuracy.

I tested code with different initial learning rates. Normally 0.01 is used as the initial learning as a rule of thumb. I tested for higher learning rates and the loss got exploded at 0.016. So I’m using 0.014 as initial learning rate as safety margin. I’m using 0.999 as the learning rate decay for a faster learning as we are only training for 300 epochs.

I have defined a function to run linear classifier for a given data set.

**linclas(x\_train,y\_train,x\_test,y\_test,lr,lr\_decay,reg):**

Parameters,

* x\_train: Training data set
* y\_training: labels for training data
* x\_test: validation data set
* y\_test: labels for validation data
* lr: learning rate
* lr\_decay: learning rate decay
* reg: regularization parameter

Running the linear classifier for 300 iterations with an initial learning rate of 0.014.

iterations = 300

lr = 1.4e-2

lr\_decay= 0.999

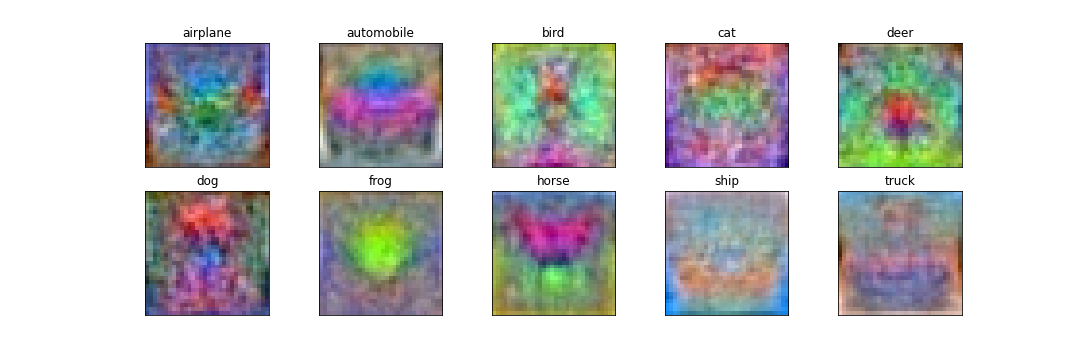
reg = 5e-6 #lamda(regularization constant for the loss function)

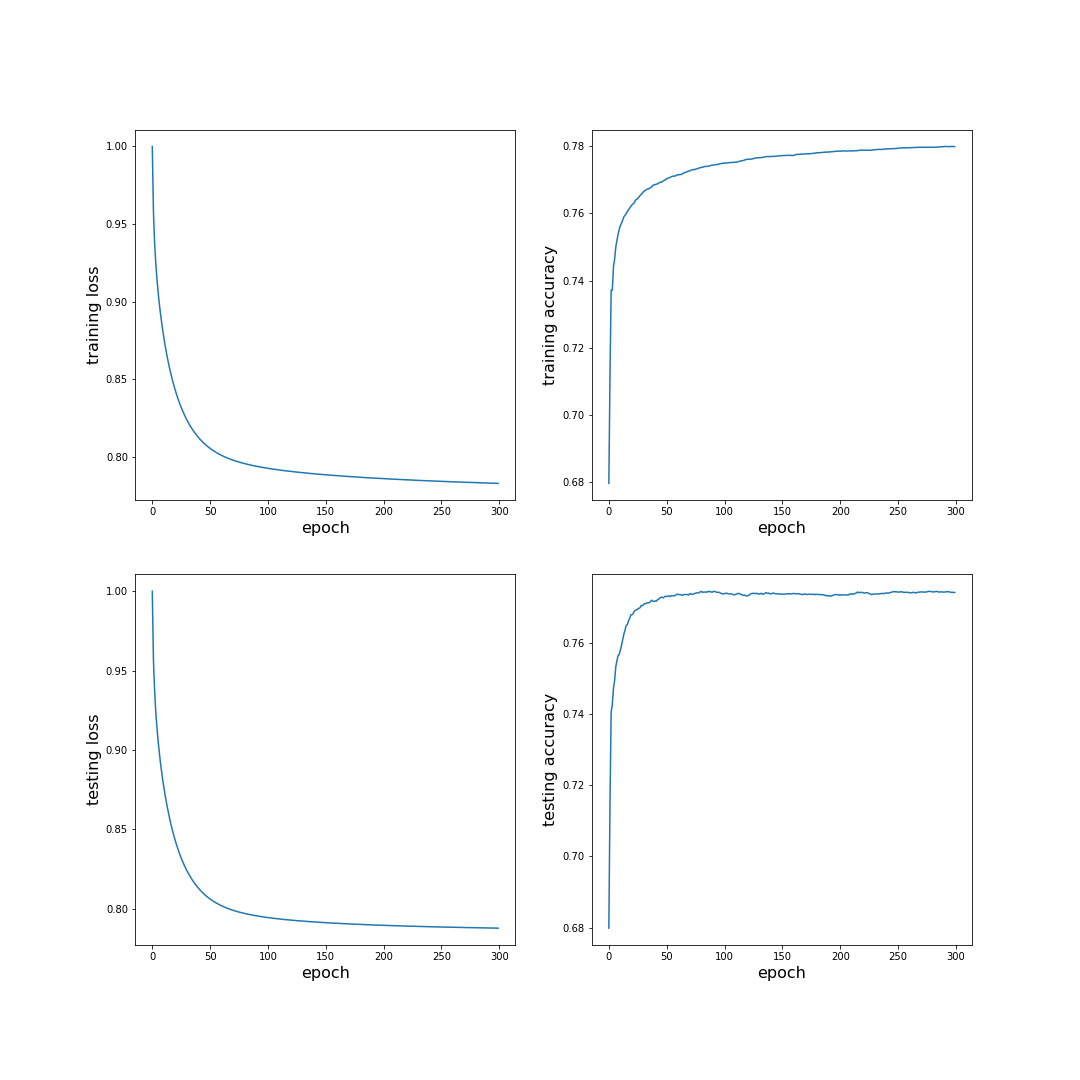
x\_train,y\_train,x\_test,y\_test=preproc(norm=True,reshape=True)

w1,b1,loss\_history,loss\_history\_test,train\_acc\_history,val\_acc\_history=linclas(x\_train,y\_train,x\_test,y\_test,lr,lr\_decay,reg)

W1 weight array is of the shape 3072 x 10. Where 3072 is the length of the flattened input images and 10 is the number of classes. So, each node(each column of W1) of the 10-node linear layer will correspond to each class in CIFAR10 data set. So, at the end of the training each node must be able to calculate a score to an image considering its’ similarity to the corresponding class.

Our linear score function is,

Since we are taking the vector product between the node and the input image, the node must be a similar image to label of its’ class. If we reshape the weight arrays back to an image, we can see this similarity. We can achieve this by reshaping each column of W1.

Let us observe the loss and accuracy of training and validation processes.

As we can observe the loss and accuracies have a large gradient at the beginning but after a while gradient has become smaller. This is due to the low number of layers and nodes. Testing data also shows similar characteristics. So, we can conclude that the linear classifier is not either underfitting or overfitting

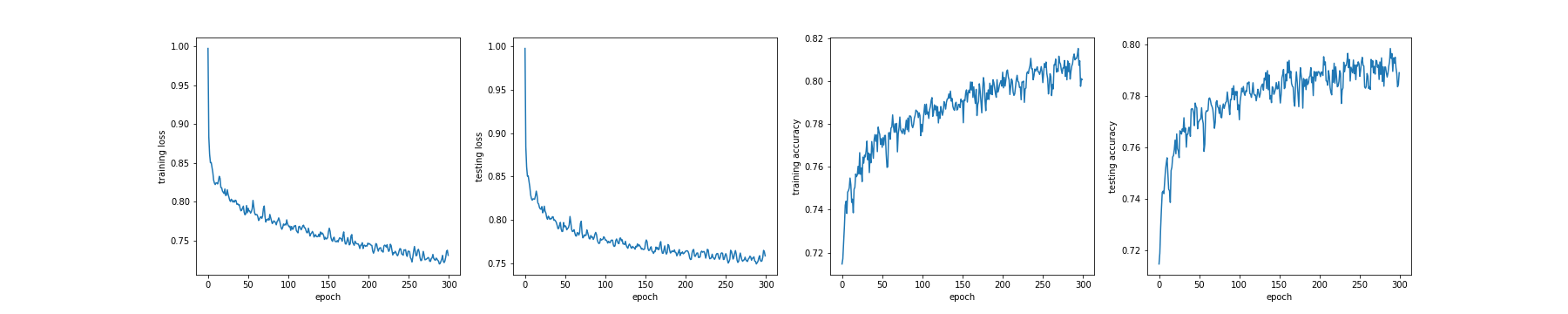
1. 2 Layer dense Network

Defining the two layer dense network

Preprocssing the input data set without pixel normalization to avoid underfitting. Because we don’t need normalization of data as we use sigmoid function to normalize the score of each node of the hidden layer.

x\_trainn,y\_trainn,x\_testn,y\_testn=preproc(norm=False,reshape=True)

Defining the parameters and running the 2 Layer dense neural network.

Training process shows oscillations be as we increase the hidden layers and nodes. But the overal accuracy has increased.

1. Stochastic gradient descent

Running the 2 Layer Dense Neural Network with mini batching of batch size 500.

batch\_size = 500

H=200

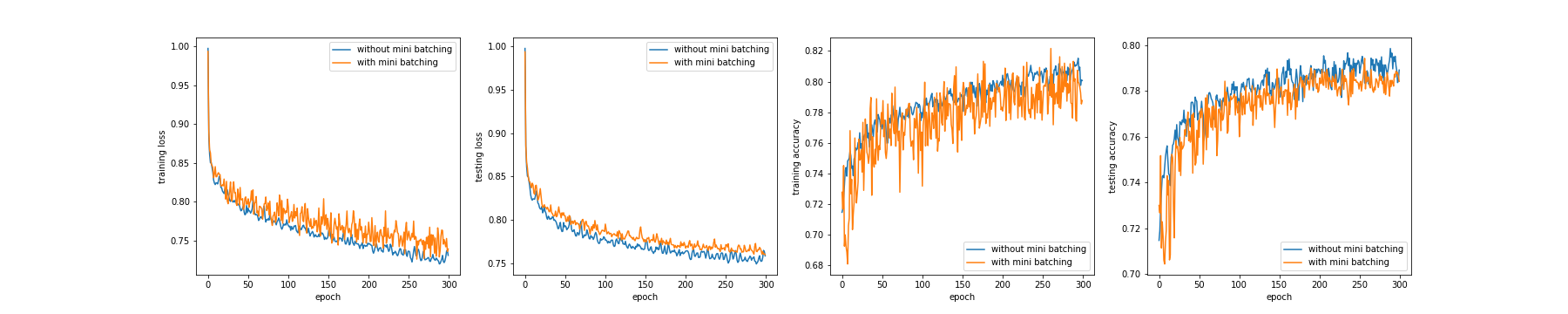
iterations = 300

lr = 1.4e-2

lr\_decay= 0.999

reg = 5e-6

w1m,b1m,w2m,b2m,loss\_historym,loss\_history\_testm,train\_acc\_historym,val\_acc\_historym=layer2(x\_trainn,y\_trainn,x\_testn,y\_testn,batch\_size,H,K,Din,lr,lr\_decay,reg)

`Comparison of gradient descent with and without mini batching.

As we can observe with mini stochastic gradient descent training process shows huge oscilations in loss and accuracy functions. This is due to random selection of data from mini batches. But the algorithm becomes fast. Eventhough the training process is oscillating, the overall is sufficiently similar to the gradient discent without minibatching. So it is a good tradeoff.

1. CNN

Preprocessing data without normalization of pixels and without reshaping the images.

x\_trainc,y\_trainc,x\_testc,y\_testc=preproc(norm=False,reshape=False)

CNN coded using Keras.models.Sequential.

We can get the summary of the CNN by using model.summary(). According to the output CNN has,

* Total parameters: 73418
* Learnable parameters: 73418
* Non Learnable parameters: 0