Report Assignment 1.2

Name: Shubham Sarda **Entry Number: 2018TT10958**

Part C

Note: Cpu is the number of cpu used in HPC (High Performance Computing at IITD).

The following experiments have been done with hyperparameters:

We have have split the train_large into train and test of 1400000 and 200000 respectively. Then the following graphs haven been plotted:

Hyperparameters of batch size and lr have been varied manually by seeing the slopes of graph. The one which minimizes best under small gflops/runtime is chosen. Initially only batch size was varied to find the optimal value. Then for the optimal batch size, other parameters were found.

By doing training on train_large.csv as well as train.csv, it was realized that hyperparameters also depend on size of training sample, thus these tuned parameters may not work during final training. To take care of this, backtracking has been used in final evaluation with alpha =0.5 and beta=.9. Initial lr can be chosen as large as 1.5 since it'll be automatically reduced to the optimal lr. Only drawback is additional time taken for backtracking which is minimized by not setting lr to intial large value after every iteration and continually decreasing. This is deemed correct since as number of epochs increase, optimal lr is found to be lower than the previous one.

(Last part of the code in Part C of Assignment1.1.ipynb deals with graph plotting)

```
plt.clf()

plt.plot([gflop*(i+1) for i in range(len(L)-1)],L[:-1],color='r', label='train_loss')

plt.plot([gflop*(i+1) for i in range(len(L)-1)],L_val[:-1],color='g', label='val_loss')

plt.xlabel('GFLOP')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

plt.plot(runtime[:-1],L[:-1],color='r', label='train_loss')

plt.plot(runtime[:-1],L_val[:-1],color='g', label='val_loss')

plt.legend()

plt.xlabel('Runtime')

plt.ylabel('Loss')

plt.show()
```

Gflop per iteration is estimated using:

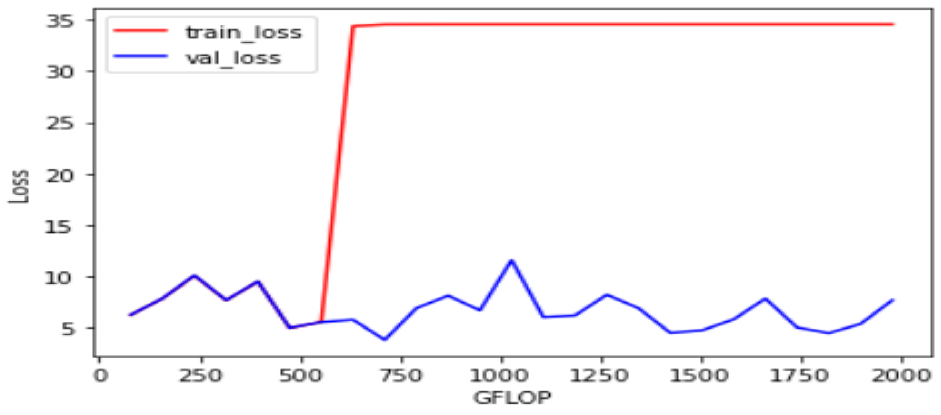
gflop=1400000*(33*X_train.shape[1]+16*X_train.shape[1]/bs+1624)/10**9

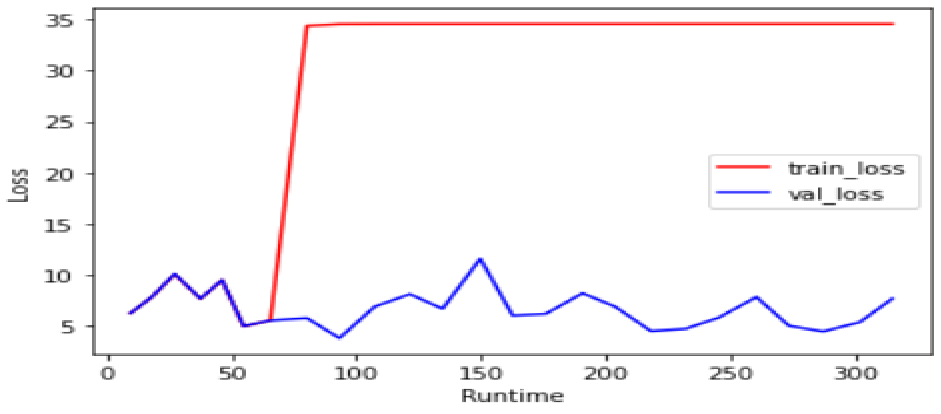
This is based on assumption that exp and log take 100 flops each. The other constants appear by analyzing the exact gradient descent algorithm.

Early stopping has been implemented to break the loop whenever train or test error explodes

1. Mini Batch gradient descent: learning rate =10, batch size=1000, number of epochs=500, cpu=12

Early stopping at Loss: 3.799780430074323 Iteration: 24

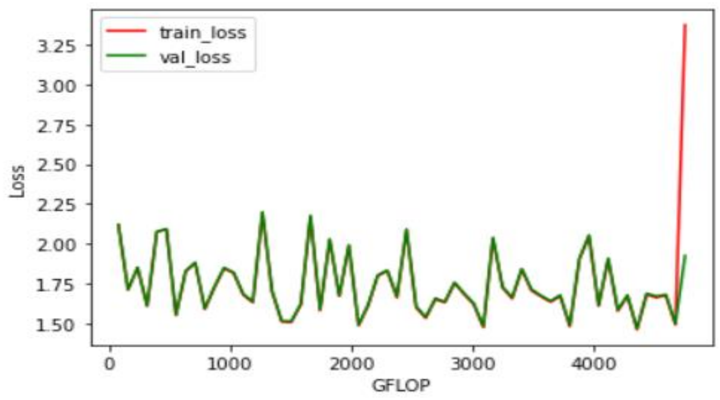


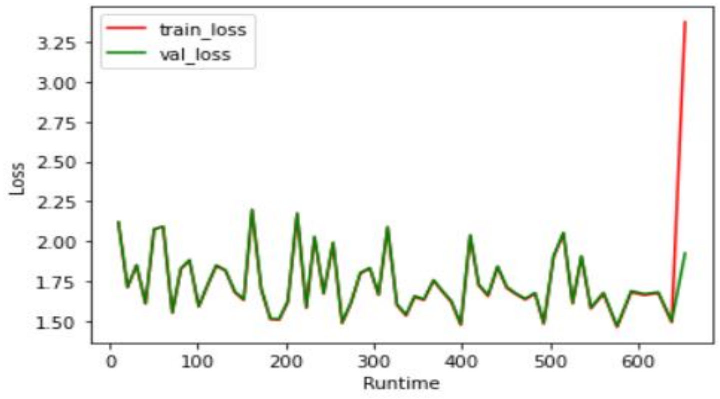


2. Mini Batch gradient descent: learning rate = 3, batch size=1000, number of epochs=500, cpu=12

12cpu

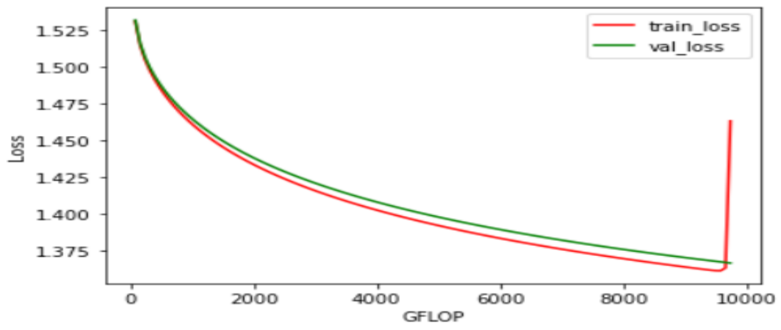
```
Early stopping at Loss: 1.4302008695971087 Iteration: 60
```

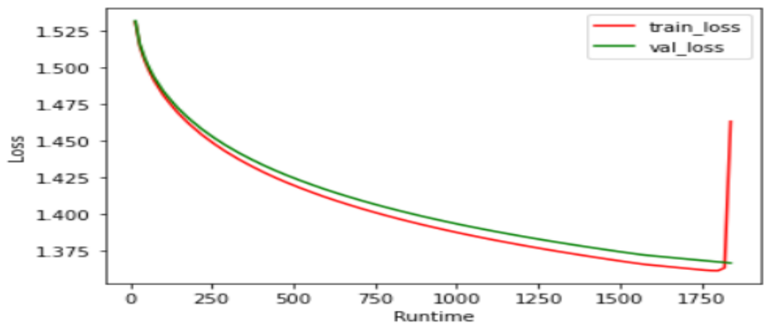




3. Mini Batch gradient descent: learning rate = 1.5, batch size=1000, number of epochs=500, cpu=10

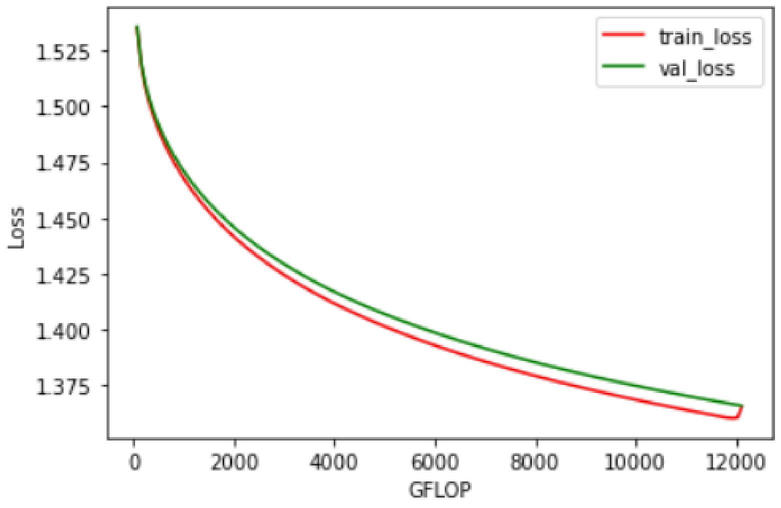
```
Early stopping at Loss: 1.366736640098581 Iteration: 124
```

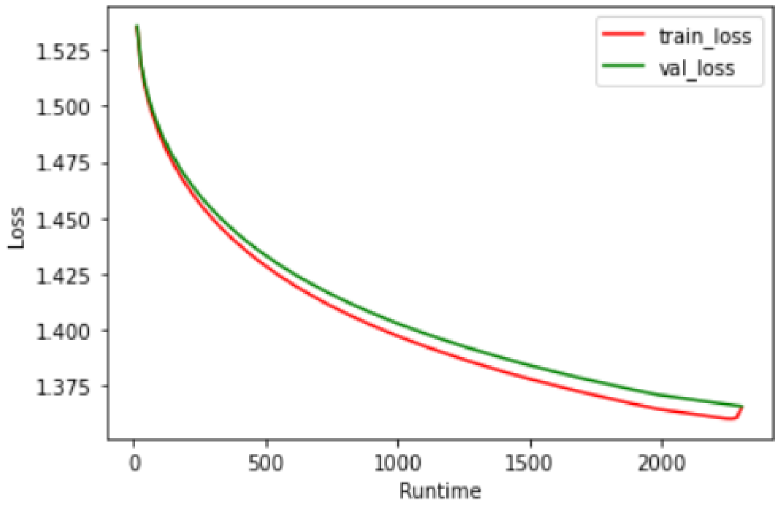




4. Mini Batch gradient descent: learning rate = 1.2, batch size=1000, number of epochs=500, cpu=10

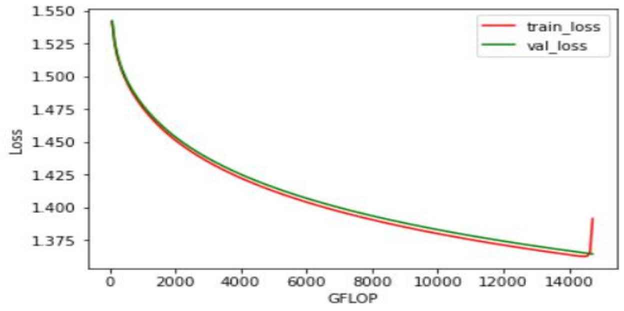
```
Early stopping at Loss: 1.3657857023209377 Iteration: 154
```

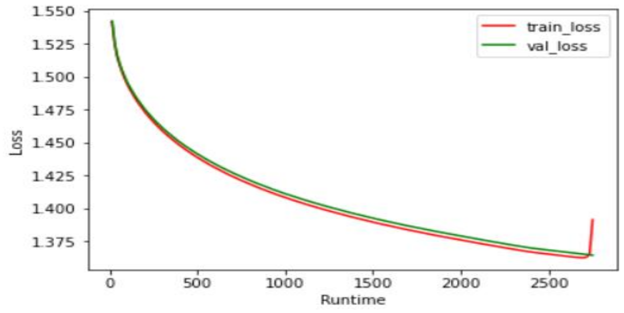




5. Mini Batch gradient descent: learning rate = 1, batch size=1000, number of epochs=500, cpu=10

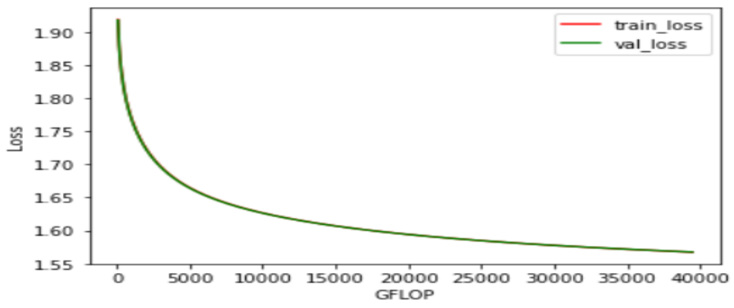
```
Early stopping at Loss: 1.364558014115096 Iteration: 185
```

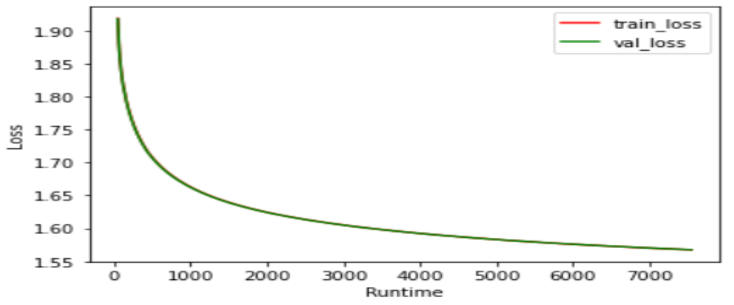




6. Batch gradient descent: learning rate = 1, batch size=1400000, number of epochs=500, cpu=10

```
Early stopping at Loss: 1.5674736496669261 Iteration: 500
```



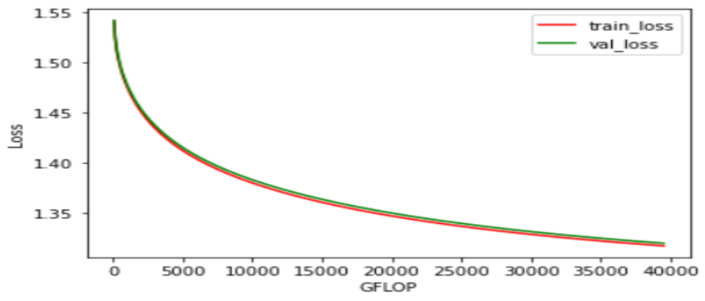


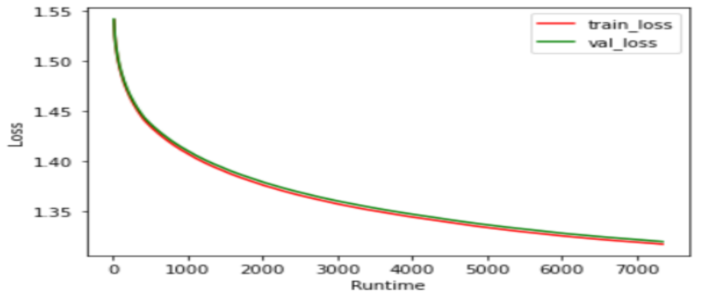
7. Batch gradient descent: learning rate = 1, batch size=1000, number of epochs=500, cpu=4

Note: We removed '-max(a)' from softmax to prevent explosion of values due to np.exp()

```
Early stopping at Loss: 1.3194173108032579 Iteration: 500
```

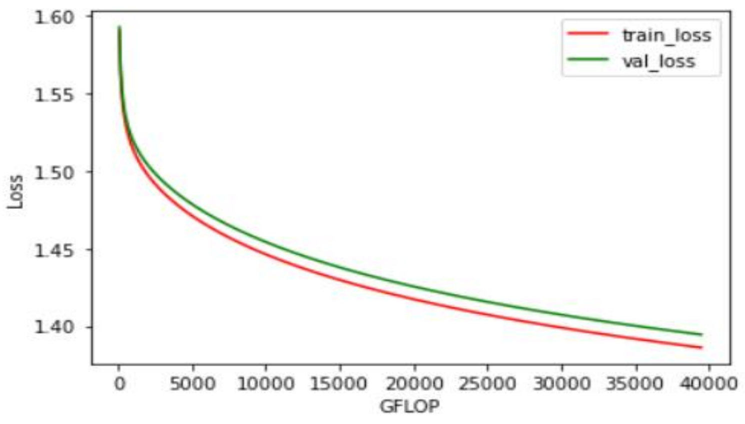
```
500 1.3194173108032579
```

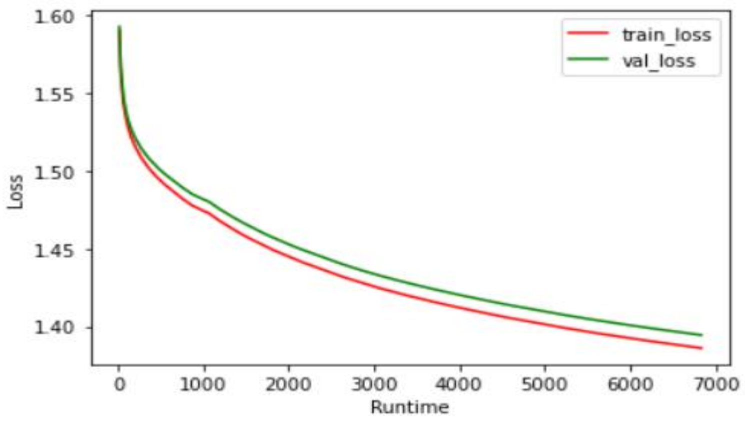




8. Batch gradient descent: learning rate = 1, batch size=5000, number of epochs=500, cpu=4

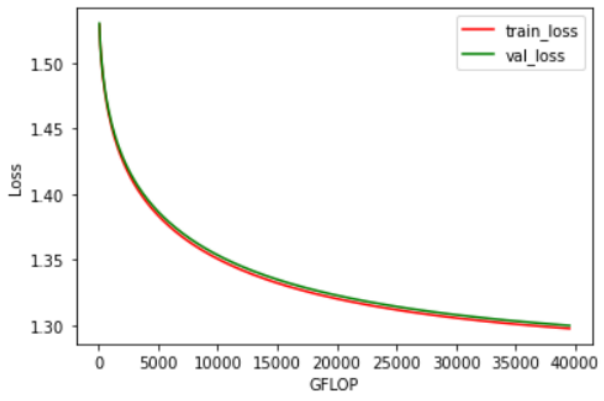
```
Early stopping at Loss: 1.3945415442206954 Iteration: 500
```

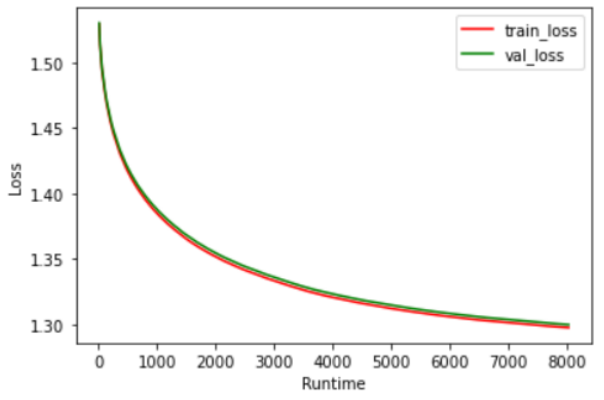




9. Batch gradient descent: learning rate = 1, batch size=500, number of epochs=500, cpu=4

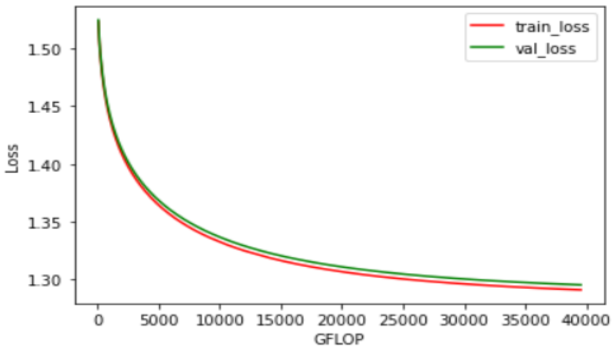
```
Early stopping at Loss: 1.2997953790310315 Iteration: 500
```

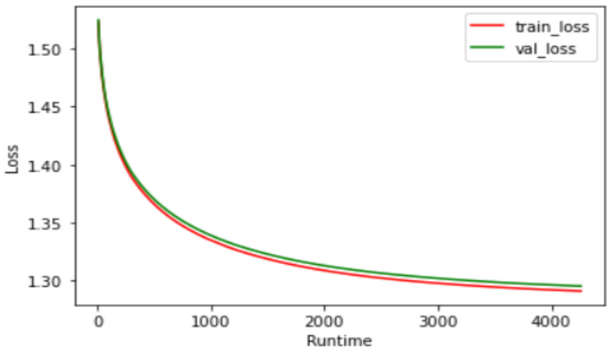




10. Batch gradient descent: learning rate = 1, batch size=300, number of epochs=500, cpu=4

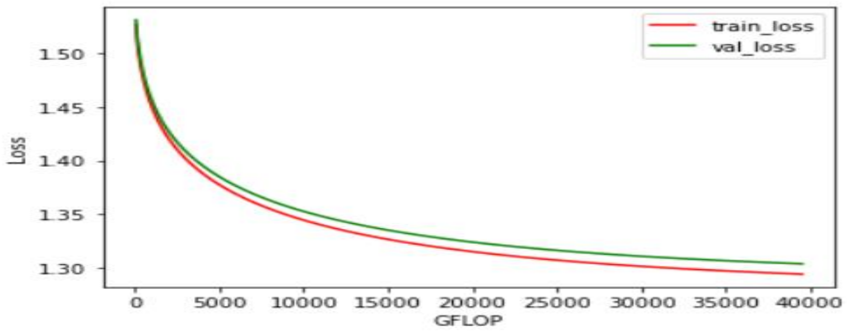
```
Early stopping at Loss: 1.2951035750532445 Iteration: 500
```

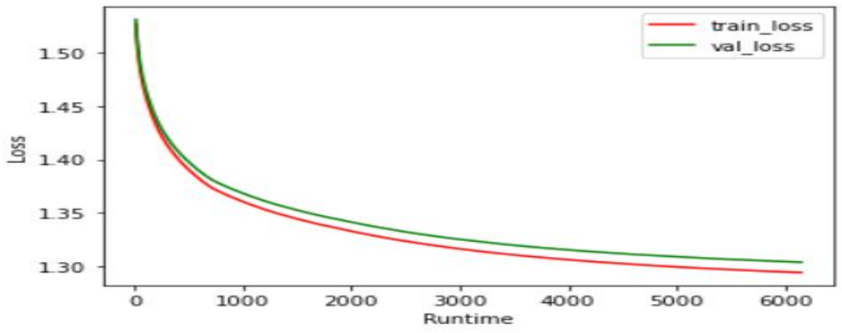




11. Batch gradient descent: learning rate = 1.15, batch size=500, number of epochs=500, cpu=4

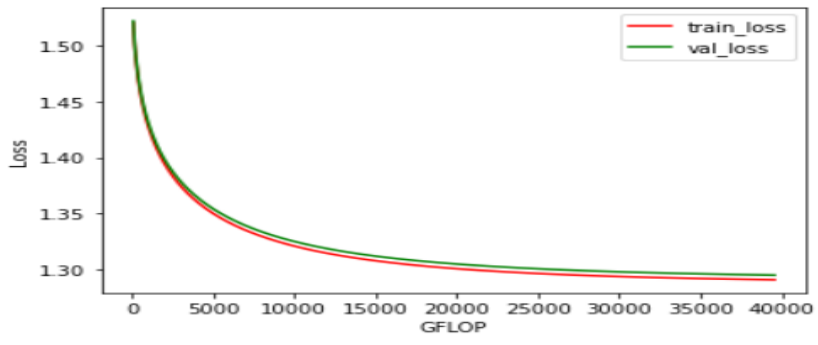
```
Early stopping at Loss: 1.303994599036309 Iteration: 500
```

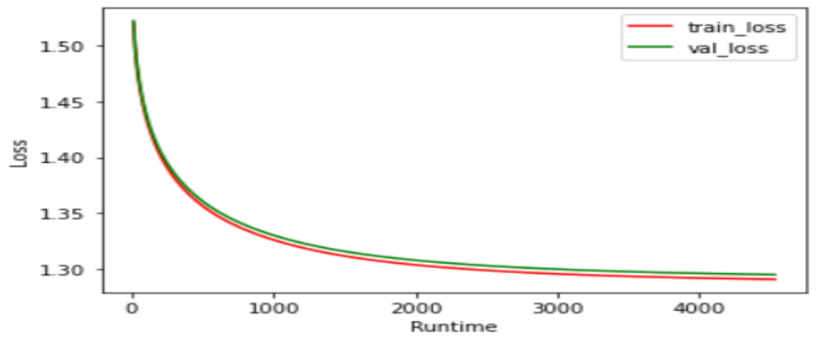




12. Batch gradient descent: learning rate = 1, batch size=200, number of epochs=500, cpu=4

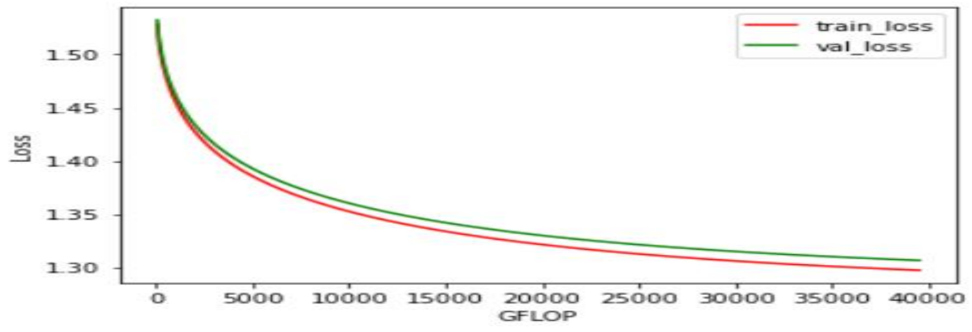
```
Early stopping at Loss: 1.2952663052527207 Iteration: 500
```

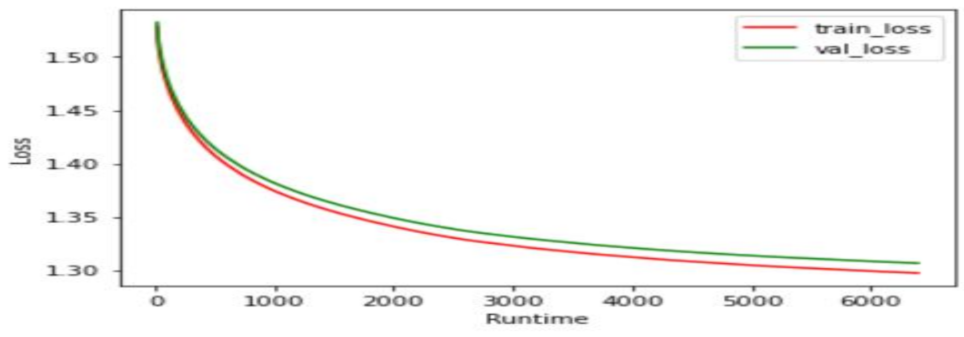




13. Batch gradient descent: learning rate = 0.9, batch size=200, number of epochs=500, cpu=4

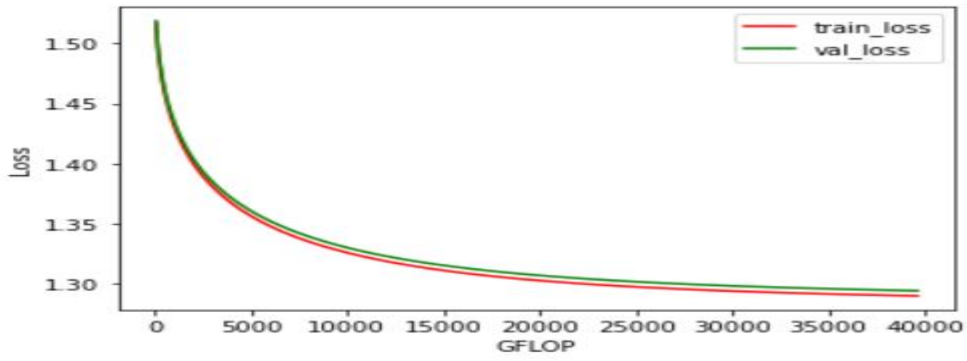
```
Early stopping at Loss: 1.306520311642382 Iteration: 500
```

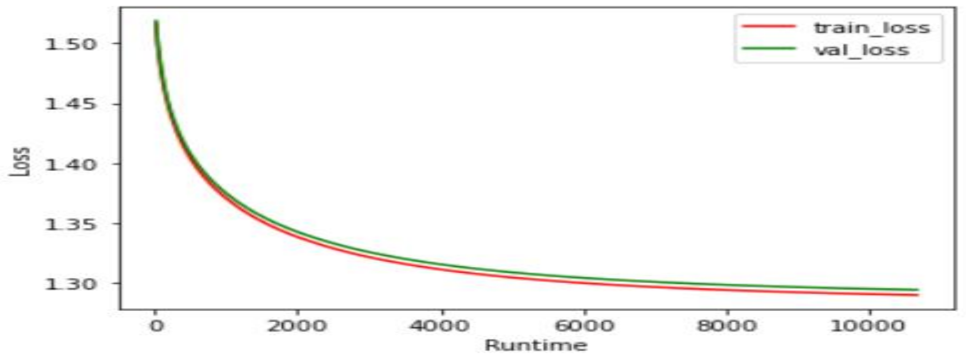




14. backtrack Batch gradient descent: learning rate = 1.5, batch size=300, number of epochs=500, cpu=4, alpha=0.5, beta =.9, final lr=1.215 (after reduction)

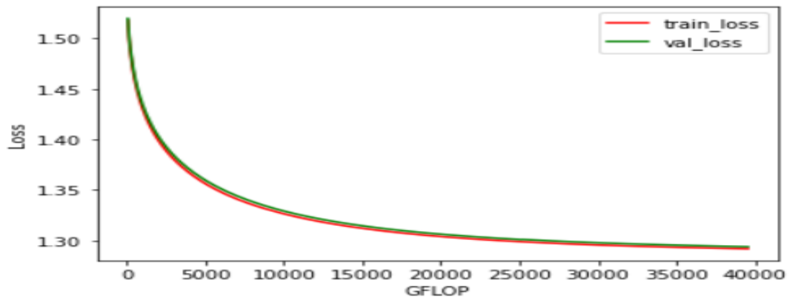
```
Early stopping at Loss: 1.2939749394813205 Iteration: 500
```

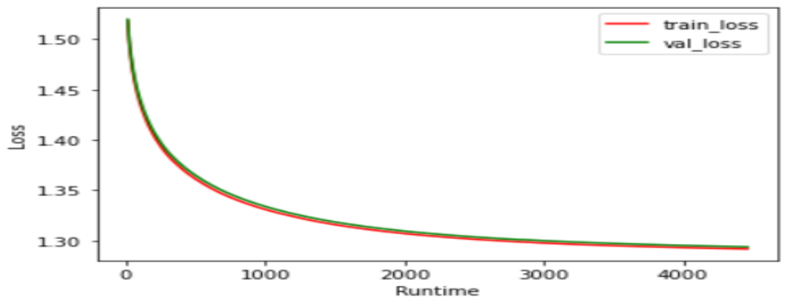




15. Batch gradient descent: learning rate = 1.225, batch size=300, number of epochs=500, cpu=4

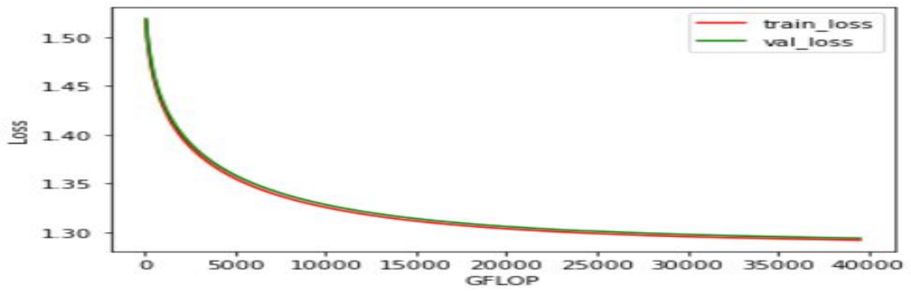
```
Early stopping at Loss: 1.2938657940493845 Iteration: 500
```

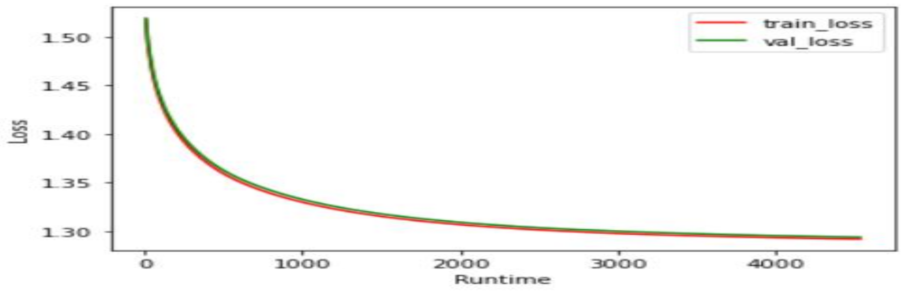




16. Batch gradient descent: learning rate = 1.265, batch size=300, number of epochs=500, cpu=4

```
Early stopping at Loss: 1.293691758381836 Iteration: 500
```





Best errors are given by 14th, 15th and 16th entries in the above graphs. They are very close

For mini batch grad descent:

Lr=1.265 is best value with a batch size of 300

But backtracking gives optimal value of learning rate during every iteration therefore that has been preferred.

Final Selected Paramters: bs=300, alpha=0.5, beta =0.9, lr=2, Max Epochs=1000

Higher learning rate is chosen as it automatically reduces to optimal value and since it was also observed that optimal lr increases when number of training examples increases (when moving from train.csv to train_large.csv). Thus to compensate further increase in size of final training set, initial lr has been increased.

PART D

In this part, we first use ANOVA to find the most predictive features from the original dataset. For further feature creation, we use the only the important features and neglect the rest.

We have **combined several categorical features** to create new categorical features.

Eg: *train_df['Age Group'].astype('str') + "_" + train_df['APR Severity of Illness Code'].astype('str')*

New Combined features along original features have then been one hot encoded. To ensure consistency of OHE, train and test were concatenated together along rows and then encoding was done.

Further feature creation has been done by adding a column of mean values of total costs for different categorical variables:

Eg: function agg_cat(df,cat_col,num_col) has been used to do the same

Further np.square and np.exp of total costs has been added (**Transformations of numerical Features**)

Different **Categorical encodings** (like target encodings, james stein encoding, helmert encoding, BaseNEncoding etc.) and polynomial features have also been experimented

For different kinds of categorical encoding:

categorical_encoders library has been used (https://contrib.scikit-learn.org/category_encoders/)

NOTE: **categorical_encoders** library needs to be installed in the system to run **Part C** of **Assignment1.2.ipynb**

Total features Created:2087

Selected: 500

For Feature selection, ANOVA has been used to find top 500 features after feature creation using sklearn

Scoring function f_classif has been used with the function SelectKBest function from sklearn

Top 500 indexes were saved to be directly used in logistic.py instead of performing feature selection again.

Code for feature Selection:

```
selector = SelectKBest(score_func=f_classif, k='all')

selector.fit(X_train,np.argmax(y_train,axis=1)+1)

X_train = selector.transform(X_train)

selected_features=list(np.argsort(selector.scores_))[::-1][:500]

X_train=X_train[:,selected_features]
```

Hyperparameters have been tuned for alpha beta backtracking,

Final parameters: batch size: 350, alpha=0.5, beta=0.91, lr=3, Max Epochs=750

Higher learning rate is chosen as it automatically reduces to optimal value and since it was also observed that optimal lr increases when number of training examples increases (when moving from train.csv to train_large.csv). Thus, to compensate further increase in size of final training set, initial lr has been increased.

It was noticed that once lr decreases, for upcoming iterations the optimal lr is never greater than the previous lr, hence lr was continually decreased by multiplying with beta and wasn’t set to the original value of 3 after every iteration. This helped save time.

Accuracy on training set: 0.47