COL774 Assignment - 1.2

2022ME21336 - Divam Manchanda

2022CS51133 - Yash Bansal

Algorithm and Hyperparameter Selection

Algorithm Selection: -

From our observations -

a) Ternary Search Algorithm converged much faster than its counterparts in part a). [i.e., it took ternary search way lesser number of epochs. Ternary still took more time than its counterparts]

b) Ternary Search applied on Mini-batch had a very high tendency to overfit to the final batch it was run on since there were ~1000 features and the batch size was of comparable size.

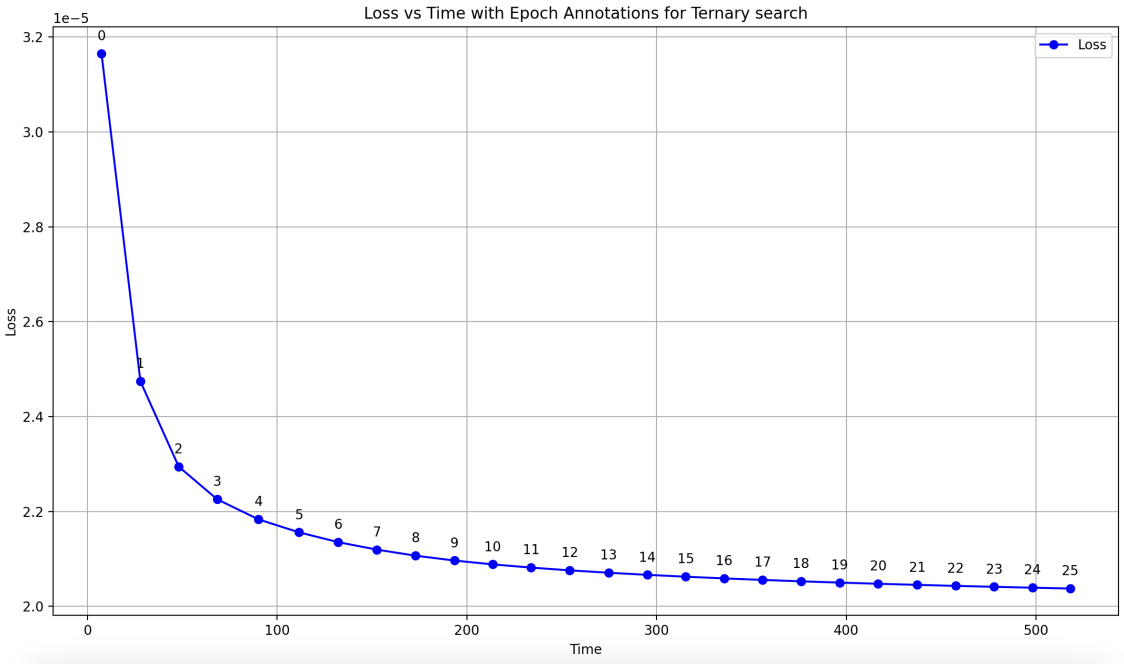
Thus, an initial choice was made to train the model using ternary search and batch gradient descent.

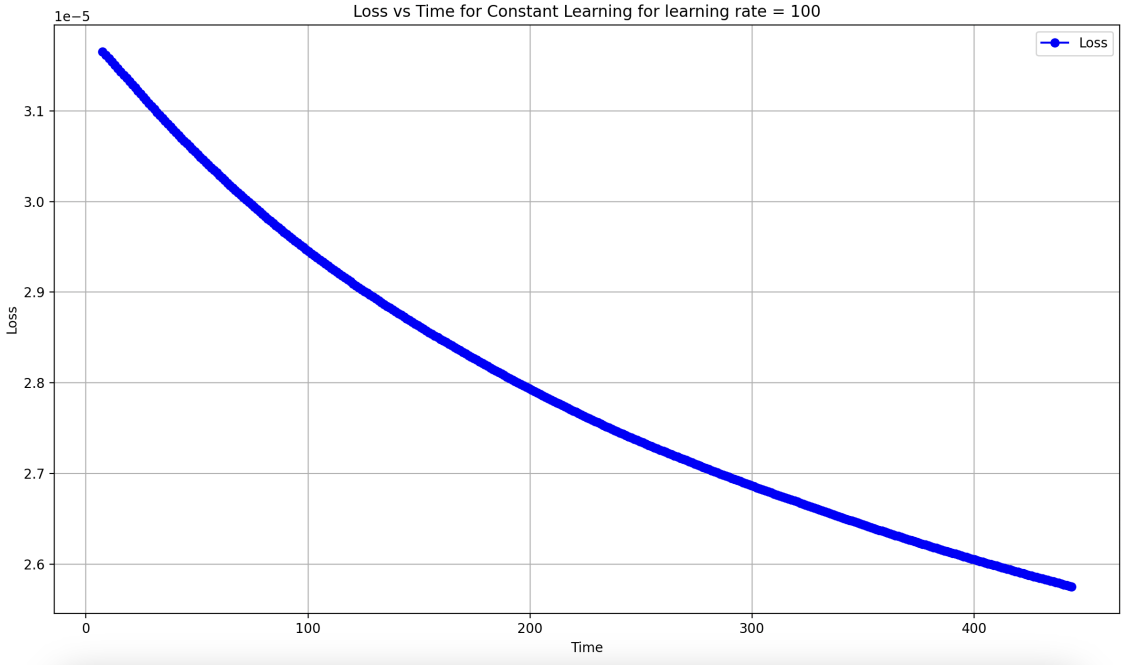
But still the associated higher time in the ternary search algorithm was a point of contention.

(pre-feature scaling, none of the algorithms were converging to the minimum error; hence, ternary was given a preference despite the high time cost,

since when all the algorithms were given full freedom, an atrocious number of epochs, and unbounded time to run, it was able to produce the lowest error)

Then, in part b, after feature scaling and running a similar number of epochs as given in part (a) test cases, results were obtained that ternary is able to provide significantly lower errors as well.





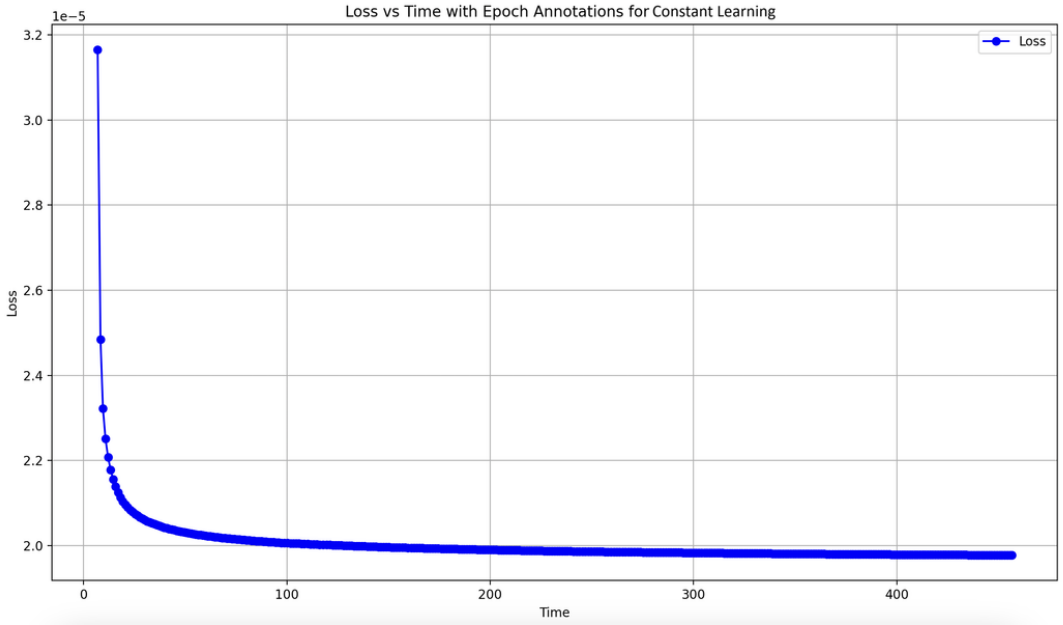
350 epochs in both the constant learning graphs in this report

In similar run time cases as well (where constant learning and adaptive learning algorithms were given 250 epochs, and ternary was given 8 epochs, total run time being ~4 minutes), ternary significantly outperformed its counterparts.

Upon noticing the near-linearity in the constant learning graph above, we inferred that the learning rate may be too low, as the gradient isn't significantly changing since the loss is near-linear.

So we saw what values of alpha were being outputted by ternary search for each epoch.

Upon noticing they were all very close to 48,000, we set alpha to 40k in the constant learning algorithm and trained the model for a similar time to get:-



Which closely resembled the graph for the ternary search algorithm, but since we weren't sure whether the learning rates for ternary search would always lie in the same near range for a general case,

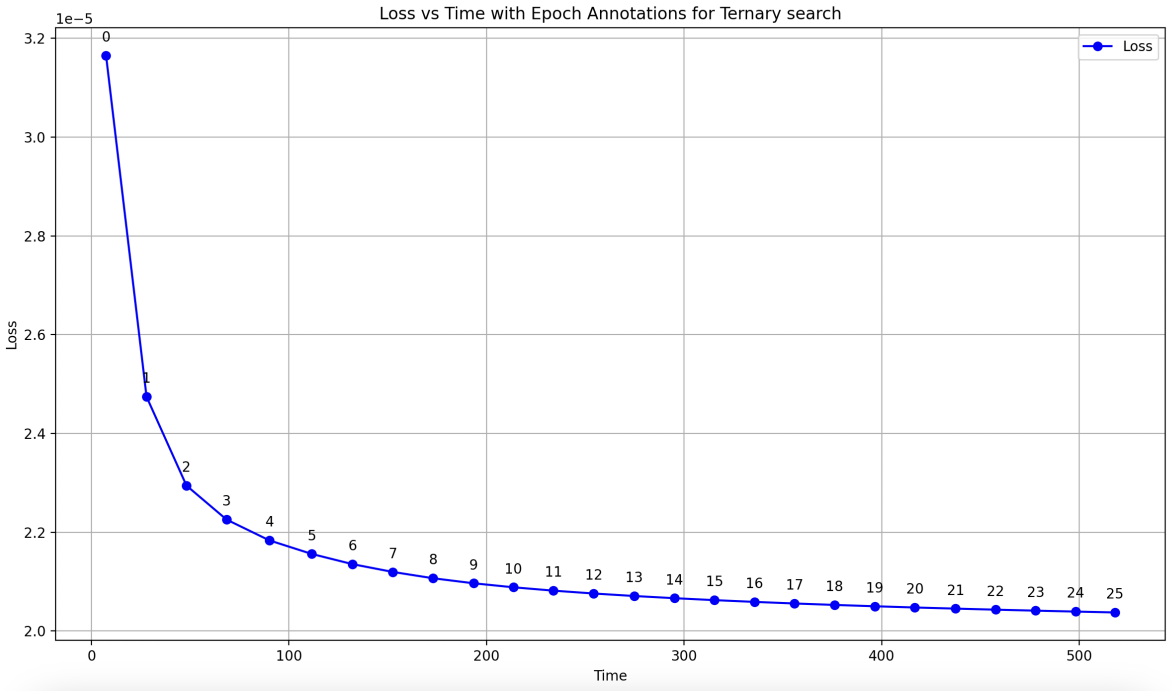
we weren't sure whether we could directly implement the constant learning algorithm with the alpha that we found from the first iteration of the ternary search algorithm. (for the risk of overshooting and non convergence)

As a result, a choice was made to implement ternary search only, as we could say with certainty that we would get near-optimal convergence in the given time.

Parameter Selection: -

3 Parameters had to be selected, namely, batch size, number of epochs,

- As stated earlier, upon reducing the batch size, the tendency to overfit increased as the number of features started to become comparable to the number of examples in a given batch. Also, as we were able to reach convergence within the time frame upon computing using the entire batch, we decided to take the batch size = size of the entire dataset.
- For Epochs, we observed that beyond 8 epochs, there was not a significant drop in loss to warrant the extra time. Furthermore, beyond epochs = 8, we observed our predictions getting worse on test data. Hence, the epoch size was taken to be 8 in our final implementation.



- n0 randomly chosen to be 1e-5. We conjectured that since n0 is only used for initializing nh, and beyond initializing, it doesn't make much of a difference since nh and nl both change; we just have to make sure of:-
  - n0 shouldn't be too small, as otherwise, the precision error in nh*gradient is too small when initializing nh. (this was happening when n0 was taken to be 1e-9)
  - P.S. since ternary converges in logarithmic times. Increasing n0 by, say, 10^4 does not significantly impact the time.
  - So, we arbitrarily set n0 to 1e-5, and convergence was obtained within the time constraints. and hence, we let it be

Feature Engineering (Part (c))

Step 1: redundant features were dropped:

Cluster 1:

Hospital Service Area, Hospital County, Facility Name, Permanent Facility ID:

- All represent, in some sense, the Location of the hospital
- Facility Name / Permanent Facility ID - the most specific indicator, encompasses any effect the Hospital Service Area and Hospital County may have.
- Facility Name arbitrarily selected in place of Permanent Facility ID. (Won't make a difference as either one would be encoded later)
- Therefore, Facility Name was kept as a feature, and the remaining features from the cluster were dropped

Zip Code, Operating Certificate number, Payment Typology 1,2,3 → seen to be lowly correlated with data and rather seen to be equivalent to noise, as removing these features improved the model predictions post-training.

Step 2: Encoding and Feature Reduction

Post removing these redundant features and one-hot encoding of every other feature, we had ~1300 features remaining, and we had to compress the available info into <1000 features.

So we simply target-encoded our features one by one to see if one-hot encoding them affected the results significantly.

Upon target encoding APR-DRG code, results went from 66.9% to 66.7%, and the number of features went under the 1000 constraint.
So we Target encoded APR-DRG code, and left the remaining features one-hot encoded.