Assignment 2
Machine Learning with Large Datasets(2018)

Vikram Bhatt [1]

Abstract

In this assignment, we have performed multiclass classification on mulitlabel dataset(DBPedia) on local machine and various distributed settings.We have used Tensorflow(Abadi et al., 2015) with GPU support on local machine for logistic regression.We have scaled up our model on Turing cluster by using TensorFlow distributed framework with asynchronous,synchronous and bounded stale synchronous models.We have successfully accelerated speed of training and compared against local setting.

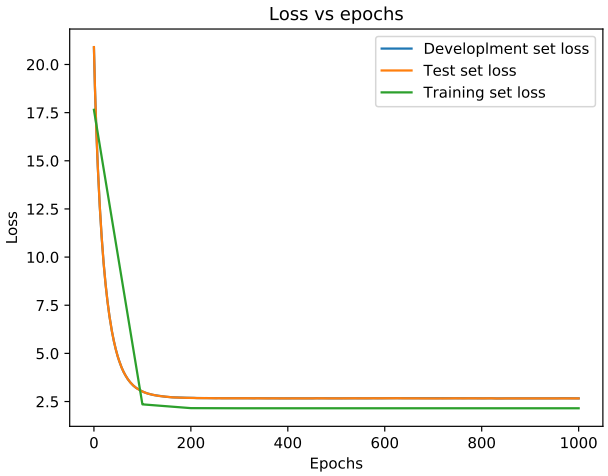

Figure 1. Train,test and validation loss with respect to epochs for learning rate=0.1,and regularization=0.02 on local setting(with tensorflow GPU support)

1. Model and Feature Selection

We have extracted features from the DBPedia dataset using Gensim Doc2Vec(Le & Mikolov, 2014) embeddings for document representation.We have used a shallow neural network with streaming corpus training for the large dataset.These extracted features are stored in .npy format for further ready made processing down the pipeline. We have trained our logistic regression model with Tensorflow(supported by GPU NVidia GeForce GTX 105) with the following loss function with regularization on weights.

$$E(X,\theta) = \sum_{i=1}^{N}\sum_{k=1}^{C} t_{ik}\log_e y_{ik} + \lambda||\theta||^2$$

where $Y = X\theta$ is the logit matrix where each row represented posterior probabilities for each training instance,t is one-hot encoded vector for each target label and θ is the learnable weight parameters,C is number of categories,N total number of training instances. In our case of multilabel instances we have treated the each document as separated entities with distinct labels for gradient descent updates(one vs rest model).

Table 1. Classification accuracy for test,development set

DATA SET	ACCURACY	PRECISION	RECALL
DBPEDIA.FULL TEST SET	0.31	0.16	0.18
DBPEDIA.FULL DEV SET	0.30	0.15	0.17

2. Training in local setting

For DBpedia.full dataset our training ran for around 4 minutes and DBPedia.small took 1 minute(on GPU), we stopped training manually after noticing no significant loss change in Tensorboard. We have tuned our hyper-parameter son development set for learning rate, regularization and size of Doc2Vec embedding for maximum categorical F1(macro)

[*]Equal contribution [1]Department of Computational and Data Sciences. Correspondence to: Vikram Bhatt <vikrambhatt@iisc.ac.in>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

accuracy.We have chosen our learning rate as 0.1, regularization as 0.02 and Doc2Vec embedding size as 100 dimensional and fixed for the rest of model.

2.1. Adaptive Learning Rate

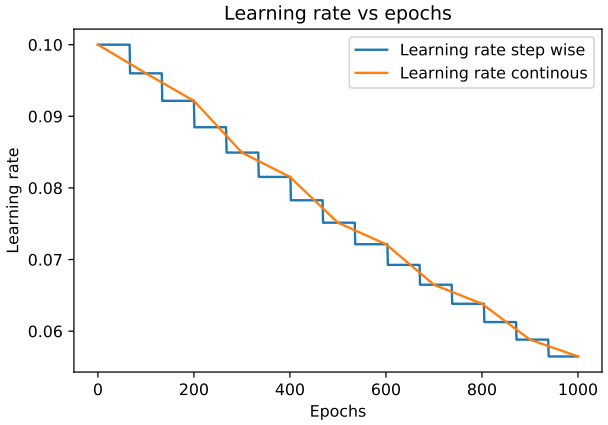

Figure 2. Step wise and continuous **DECREASING** learning rate with epochs

Adaptive learning rate does tend to converge faster than constant learning rates. Fig.2 shows pattern of learning rate

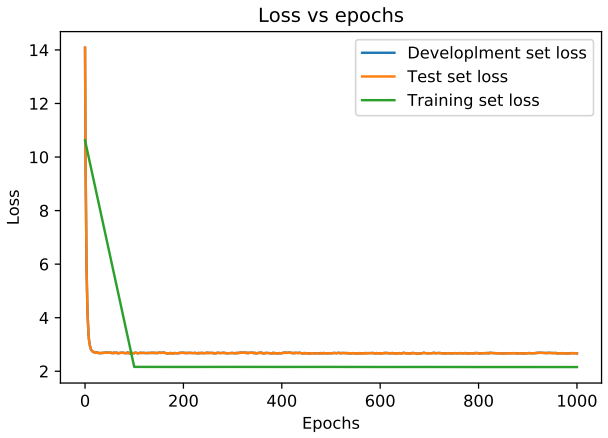

Figure 3. Training with step wise and continuous decreasing learning rates on test and development set

with every epoch for step wise decreasing and continously deceasing learning rate.We have set decay rate of 0.96 after every 100 epochs. Fig.3 shows the corresponding training loss which decays quickly due to adaptive learning rate and converged well due to decrease in step size after reaching the local minima point.

Fig.6 shows the corresponding learning rate which we increase by a factor of 1.04 after every 100 epochs. Fig.7 shows training loss intially goes down due to steps towards minima but with increase in step size,we notice oscillations happening, because gradients update jumped out of local minima by increasing our learning rate.

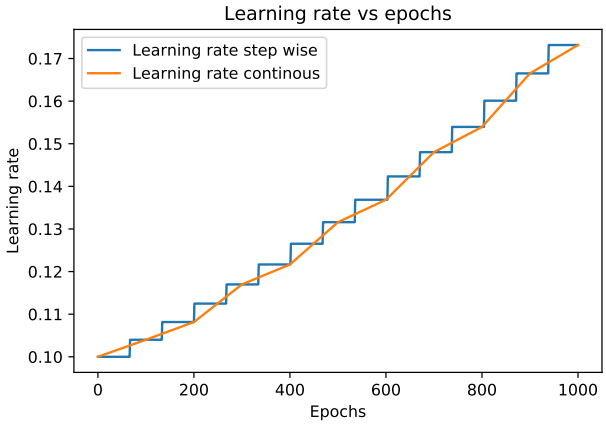

Figure 4. Step wise and continuous **INCREASING** learning rate with epochs

3. Training on Distributed Setting

We have used Tensorflow framework for the rest of the assignment as it provides simple API's for cluster specification and clients for within-graph replication,manages asynchronous data communication between nodes, and supports flexible consistency models, elastic scalability.In this within-graph approach, there is a separate client for each task,each client builds a similar graph containing the parameters pinned to parameter server and a single copy of the compute-intensive part of the model, pinned to the local task in worker. In this framework we have asynchronous-SGD,synchronous-SGD and bounded asynchronous model.

3.1. Asynchronous Model

With asynchronous training,whenever a replica finished computing the gradients,it immediately uses them to update model parameters.The worker sends the gradients for each variable to the appropriate PS task, and applies the gradients to their respective variable, using an update rule that is determined by the optimization algorithm (e.g. SGD, SGD with Momentum, Adagrad, Adam, etc.). The update rules typically use (approximately) commutative operations, so they may be applied independently on the updates from each worker, and the state of each variable will be a running aggregate of the sequence of updates received.There is no aggregation and no synchronization. Replicas just work independently of the other replicas. Since there is no wait-

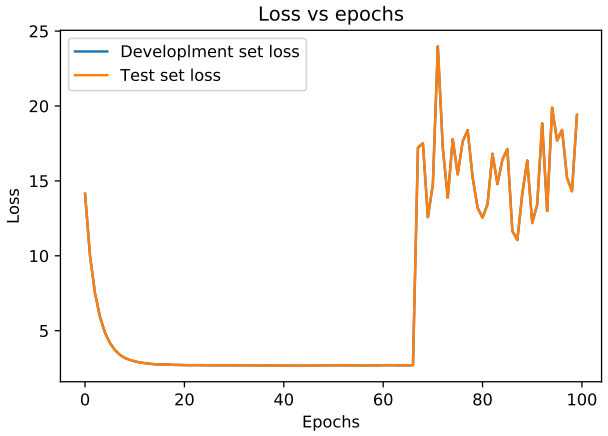

Figure 5. Training with step wise and continuous increasing learning rates on test and development set.Plot shows divergence and oscillations of test loss due to large steps.

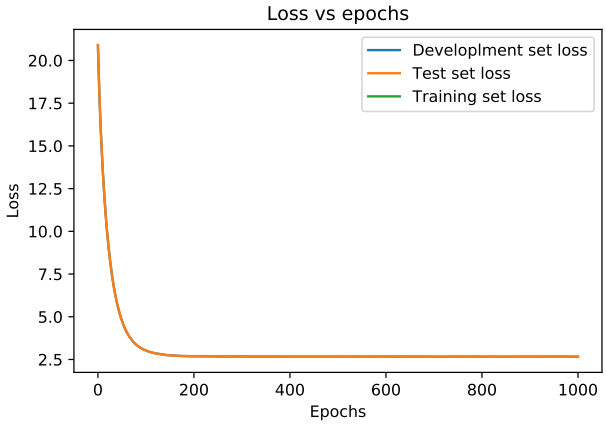

Figure 6. Training loss with constant learning rate(0.01) on test and development sets.

ing for the other replicas, this approach runs more training steps per minute.The worker reads all of the shared model parameters in parallel from the PS task(s), and copies them to the worker task. These reads are uncoordinated with any concurrent writes, and no locks are acquired. Moreover, although the parameters still need to be copied to every device at every step, this happens at different times for each replica so the risk of bandwidth saturation is reduced.

Fig.7 shows the training loss in this setting,we can observe worker0 runs ahead initially and produce updates and although the convergence is not smooth(shows oscillations but smoothed out in figure) as in bulk synchronous model. Fig.8 shows the delays between workers as a functions of wall clock time on individual training workers at each epoch.We can observe both worker gather their updates from parameter server,without any synchronization.It is also possible that each worker might see partial updates from rest of the replicas.

3.2. Bulk Synchronous Model

In synchronous training, we avoid stale gradients by collecting gradients from all replicas, averaging them, then applying them to the variables in one shot, after which replicas can fetch the new variables and continue.For this purpose we have used Tensorflow's predefined optimizer API called *SyncReplicasOptimizer*.The following objects are created.

- N gradient accumulators, one per variable to train. Gradients are pushed to them and the chief worker will wait until enough gradients are collected and then average them before applying to variables. The accumulator will drop all stale gradients.
- 1 token queue where the optimizer pushes the new global_step value after all variables are update

The optimizer add nodes to the graph and wait to collect all the gradients and all the trainers will be paused untill all the variables are updated in one shot. For parameter server job:(Ten)

- An accumulator is created for each variable, and each replica pushes the gradients into the accumulators instead of directly applying them to the variables.
- Each accumulator averages once enough gradients have been accumulated.
- Apply the averaged gradients to the variables.
- Only after all variables have been updated, increment the global step.
- Only after above, pushes global step in the token queue, once for each worker replica. The workers can now fetch the global step, use it to update its local step variable and start the next batch.

For the replicas:(Ten)

- Start a step: fetch variables and compute gradients
- Once the gradients have been computed, push them into gradient accumulators. Each accumulator will check the staleness and drop the stale.
- After pushing all the gradients, dequeue an updated value of global step from the token queue and record

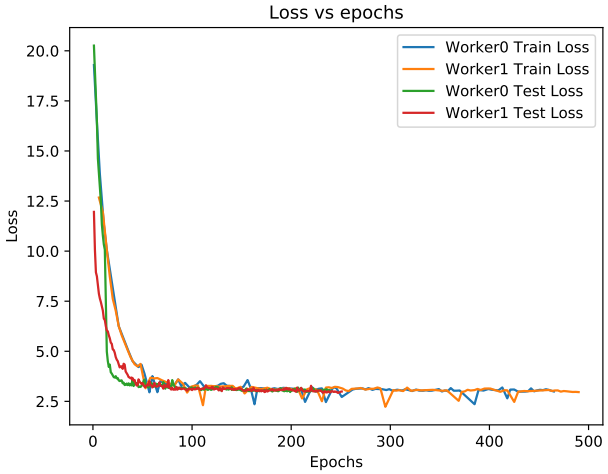

Figure 7. Asynchronous training,each worker calculate gradients locally and push updates into parameter servers.

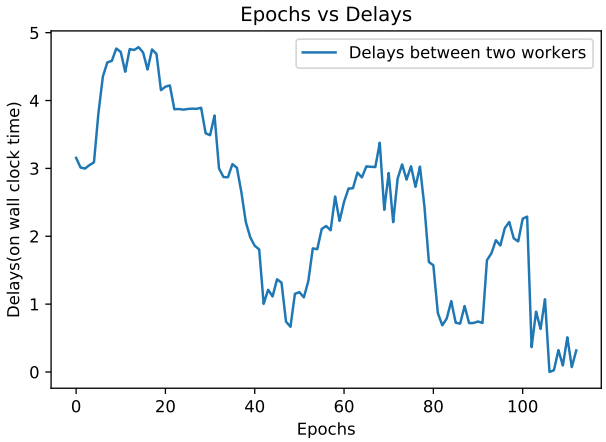

Figure 8. We can notice the race and lags between the two workers,on y-axis we have wall-clock time.

Table 2. Classification accuracy for test,development set in Bulk Synchronous Training

DATA SET	ACCURACY	PRECISION	RECALL
DBPEDIA.FULL TEST SET	0.25	0.13	0.12
DBPEDIA.FULL DEV SET	0.26	0.10	0.12

Table 3. Classification accuracy for test,development set in Asynchronous Training

DATA SET	ACCURACY	PRECISION	RECALL
DBPEDIA.FULL TEST SET	0.25	0.12	0.11
DBPEDIA.FULL DEV SET	0.26	0.10	0.11

that step to its local step variable. Note that this is effectively a barrier.

- Start the next batch. Fig.9 shows much more smoother convergence than asynchronous model for each worker on train and test set. Fig.10 shows the delays between two workers with respect wall clock time.Note that initial delay has been subtracted to make it center around zero.

3.3. Stale Synchronous Model(SSP)

In SSP briding model, compared to bulk synchronous model, worker machines may advance ahead of each other upto s iterations apart(where s is called stalennes threshold).Workers that get too far ahead are forced to stop untill slower workers catch up.Like Asynchronous model, information about model parameters is exchanged asynchronously and continously between workers, without need for synchronization barriers.The advantage of SSP model is that it behaves like Asyncrhonous model most of the time, yer SSP can also stop workers as needed to ensure correct updation of parameters.(Xing et al., 2015)

Acknowledgement

All the codes and data have been uploaded to GitHub.I have given directions to Mr.Kapil Pathak with coding and other concepts and vice versa taken help from him. I have also taken extensive help from Tensorflow documentation and various stack exchange forums(which are not cited here)for various technical details.

References

TensorFlow. URL https://www.tensorflow.org/api{_}docs/python/tf/train/SyncReplicasOptimizer.

Distributed-TensorFlow-Guide. URL https://github.com/tmulc18/Distributed-TensorFlow-Guide.

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner,

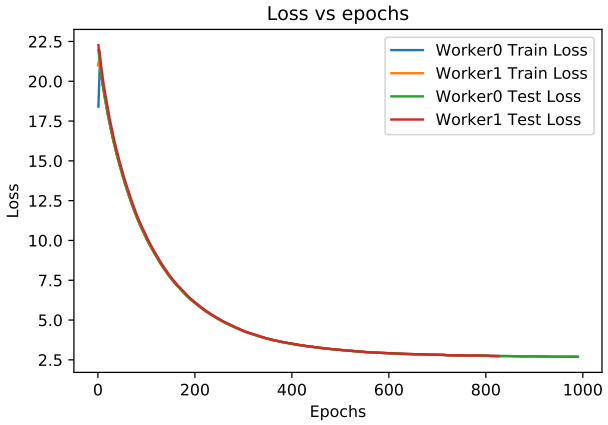

Figure 9. Synchronous training,each worker calculate gradients and paused until enough samples accumulated into parameter server.

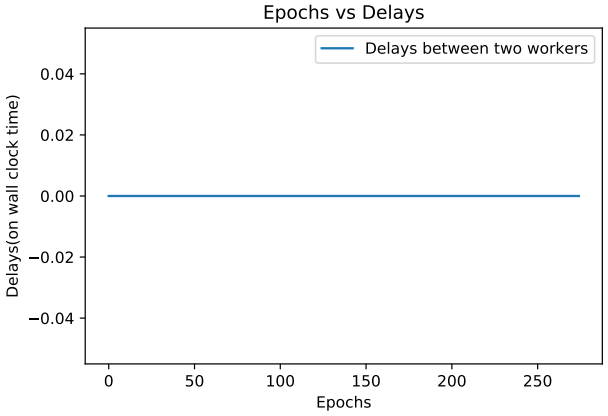

Figure 10. Delay between workers after every worker(starting delay subtracted),on y-axis we have wall-clock time.

Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Le, Quoc and Mikolov, Tomas. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. II–1188–II–1196. JMLR.org, 2014. URL http://dl.acm.org/citation.cfm?id=3044805.3045025.

Xing, Eric P., Ho, Qirong, Xie, Pengtao, and Dai, Wei. Strategies and Principles of Distributed Machine Learning on Big Data. dec 2015. URL http://arxiv.org/abs/1512.09295.