CSC830 Proposal: Towards Faster Deep Learning by Leveraging Anti-patterns

By: Rahul Yedida, under the supervision of Dr. Tim Menzies

Deep learning literature has long yielded that while powerful, the computational demands of the approach can be excessive, and at times beyond the means of an organization that wishes to implement it. On the extreme end are massive models such as GPT-2 and reinforcement learning approaches for hyper-parameter optimization that use 2,500+ GPU hours to train, only to yield a set of hyper-parameters for yet another deep learner.

In this independent study, I wish to explore this issue and suggest potential methods to start alleviating this problem. Literature has suggested several approaches towards faster convergence—these may broadly be categorized into theoretical explorations that argue for cyclical or non-monotonic learning rate policies (such as 1cycle/LipschitzLR), better initialization schemes (such as He initialization), optimization algorithms (such as AdaSecant), or novel architectural components (such as Batch Normalization). Over time, these have been used together for obtaining state-of-the-art results on various tasks. Yet, the focus has largely been towards obtaining better results, not *faster* convergence.

This raises several questions: is it possible to make deep learning faster? And are the suggestions from literature the best approach for this? Can going against the recommendations proposed in existing literature actually have a *positive* effect on the training time?

The independent study will explore the above in detail. Specifically, the aim is to answer the following questions: (a) can we use larger learning rates to converge faster? (b) can we use *less* data to train deep learners with no statistically significant impact on performance? (c) knowing that commonly used loss functions optimize for accuracy rather than precision or recall, can oversampling help optimize these instead? (d) can we get better results with non-monotonic activation functions, thus showing empirically that the results of Cybenko (1989) are true even in a broader sense? Experiments will be run on software engineering data sets and compared to the reported results from other papers on the same data sets. In addition, the runtime (where time is the wall time, not CPU time) will also be monitored.