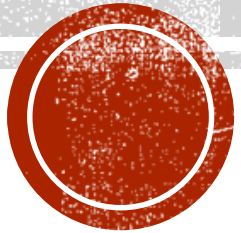
DEEP NEURAL NETWORKS ON DISTRIBUTED ENVIRONMENT

Clothing Image Classification with Deep Neural Networks in Big Data Environment



PROJECT DESCRIPTION

Objective:

- To create a big data environment for deep learning in computer vision tasks
- To analyze the time-based performances of training process

Motivation:

- Images are also big data
- Mostly not the number of images, but the dimensionality of images

Why use fashion datasets?

- On my research project, I am currently working on deep learning in fashion domain
- The model has too much parameters.
- We explore the big data environment to train a neural network to improve the training process.

ENVIRONMENTAL SETUP

Programming language:

- Python

Programming frameworks:

- Hadoop (as a file system)
- Spark (as a distributed computing)
- Keras (as a neural network computing)

Hortonworks Sandbox, used for Hadoop container.

DATASET

- FashionMNIST:
  - contains 60.000 gray-scaled fashion images
  - with 10 category labels
  - Dimensionality (h, w, c): 28x28x1

- DeepFashion:
  - contains appr. 800.000 RGB fashion images
  - For classification task, there are roughly 290.000 images
  - with 46 category labels
  - Dimensionality (h, w, c): 128x128x3
  - Gray-scaled & PCA applied (h, w, c): 28x28x1

STAGES

- Installing the environment
  - Hortonworks Sandbox
  - Keras integration
- Loading the datasets into HDFS
- Preprocessing the dataset (Pixel-wise normalizing etc.)
- Creating a deep neural network model with Keras
- Training the model with different Trainer algorithms for calculating the gradients in distributed environment
- Analyzing the time-based performances with different batch sizes

TRAINER ALGORITHMS

The trainer algorithms used in this project as follows:

1. Single Trainer
2. Downpour SGD
3. Elastic Averaging SGD (EASGD)
4. Asynchronous Distributed Adaptive SGD (ADAG)

SINGLE TRAINER

- An optimizer which will train a network on a single machine
- following the traditional scheme of training a neural network model
- using sequential gradient updates to optimize the parameters by executing the training procedure on a single Spark executor

WHY WE NEED NEW TRANIER ALGORITHM?

- The traditional formulation of SGD is inherently sequential

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$$

where θ is parameters, α is learning rate.

- Impractical to apply to very large data sets
- when the data is distributed.

DOWNPOUR

- An asynchronous stochastic gradient descent procedure which uses multiple replicas of a single model
- The basic approach is as follows:
  - dividing the data into a number of subsets
  - running a copy of the model on each subset on multiple replicas
  - maintaining the communication between models through centralized parameter server (master), which keeps the meta-information of the model split into replicas
- This approach is asynchronous since:
  - the model replicas run independently of each other
  - the parameter server shards also run independently of one another

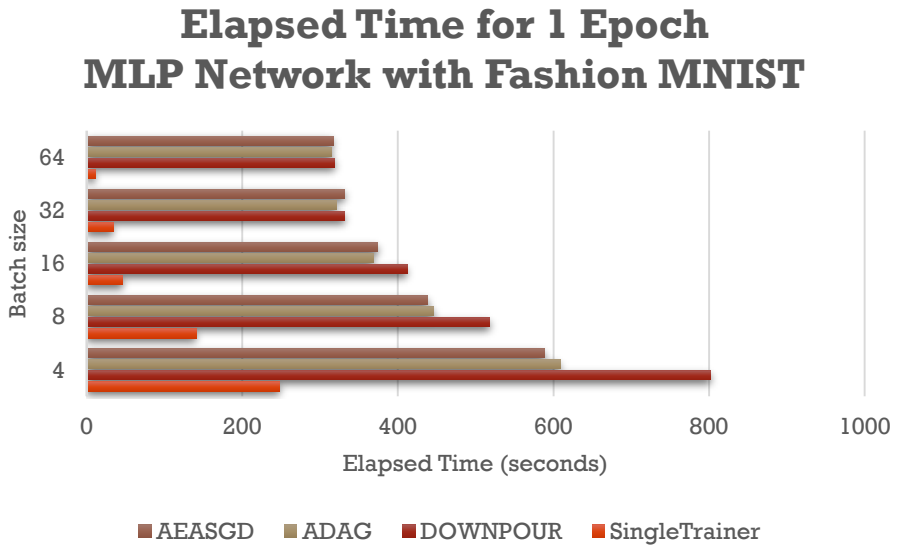
EASGD

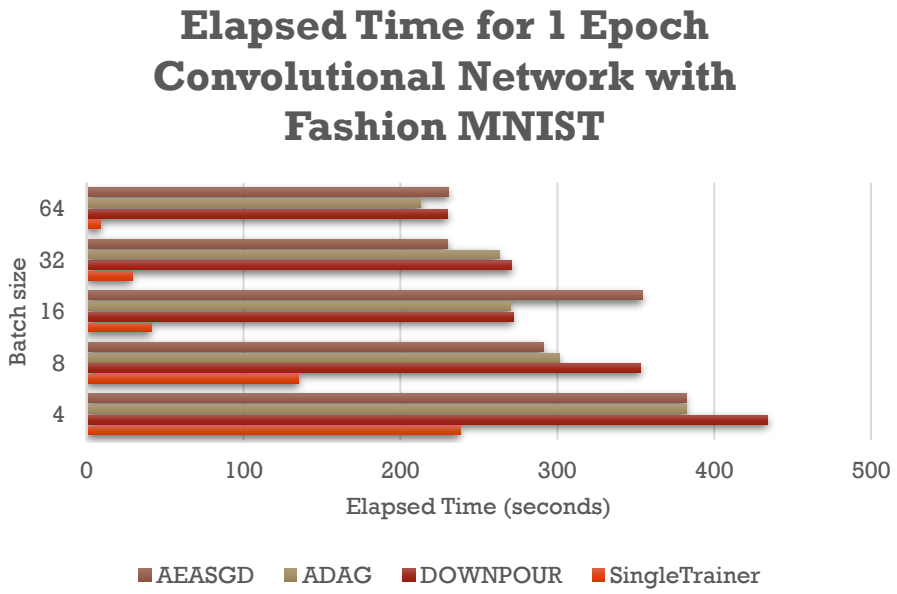
- Allowing each worker maintain its own local parameter
- Elastic force: Links the parameters of the local workers with a center variable stored by master
- It acts like a pivot for the communication and coordination among the local workers.
- The center variable is updated as a moving average where the average is taken in time and also in space over the parameters computed by local workers.

- Asynchronous variant has been used in this project.

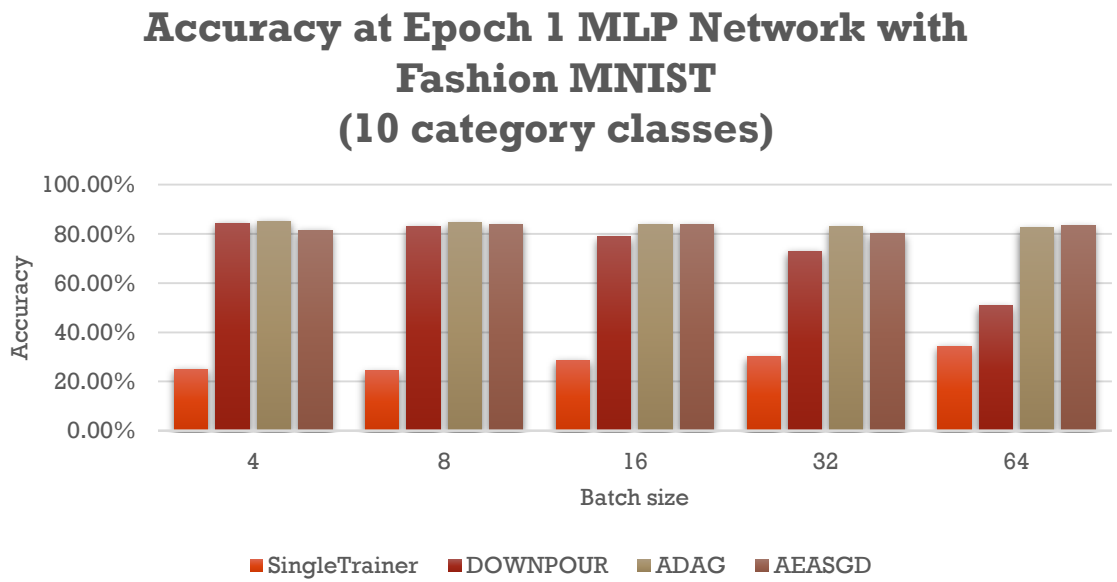
ADAG

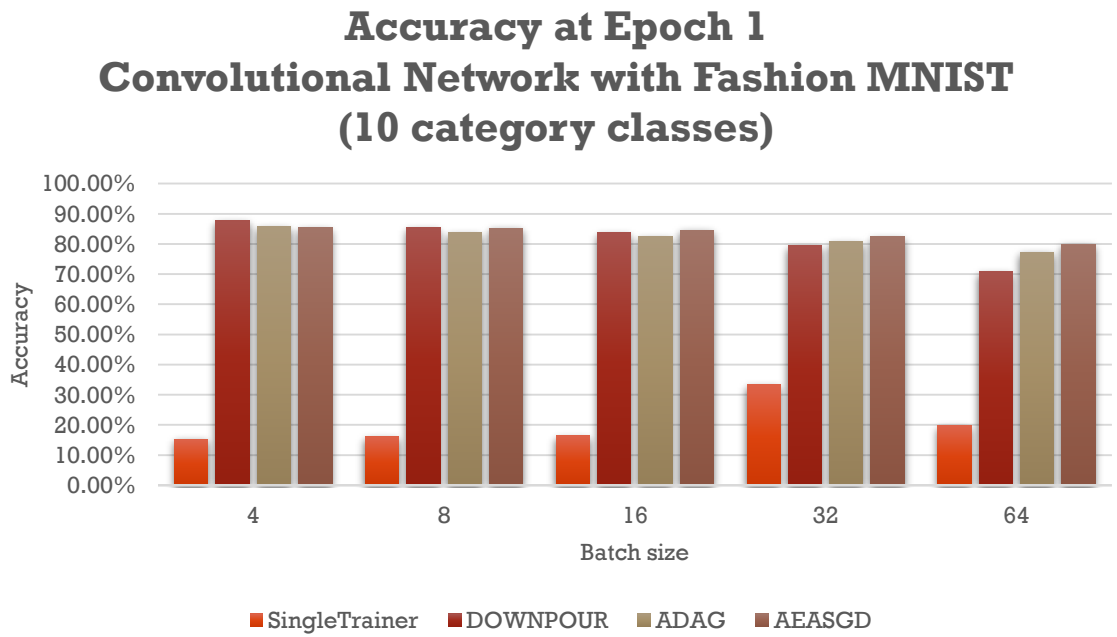
- Asynchronous!
- A set of workers –individually– contribute updates the gradients asynchronously to a master node.
- Staleness: The delay of the update time at t.
- The problem:
  - Some workers updates the gradients that are possibly from the previous parametrization
  - The staleness leads to lose some necessary information or to gain some unnecessary information during updating the gradients
- In this algorithm, the formula of traditional SGD has re-written in the context of Lagrangian mechanics with new terms such a "proxy" and "gradient energy matching" to ensure the stability of asynchronous SGD in distributed environments.





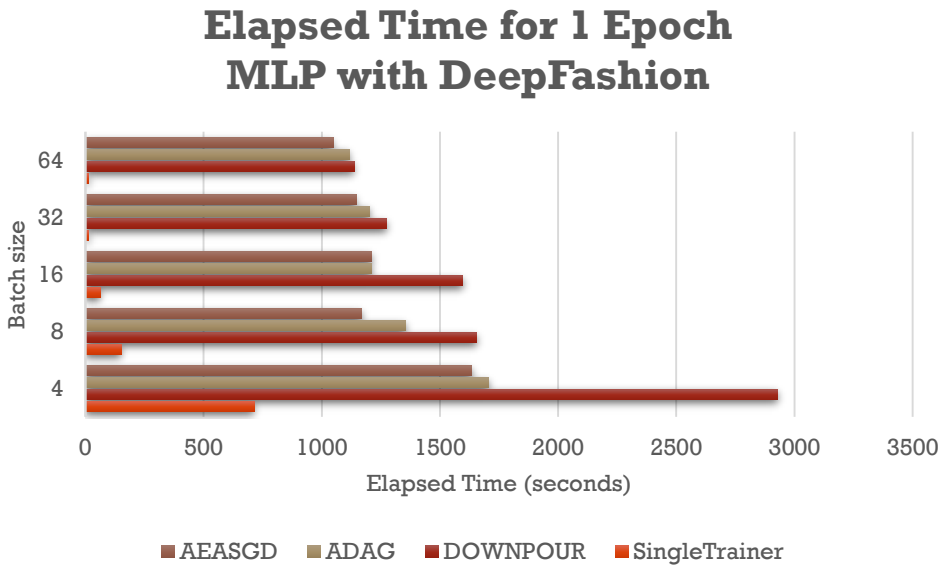
RESULTS FOR FASHION MNIST (ELAPSED TIME)

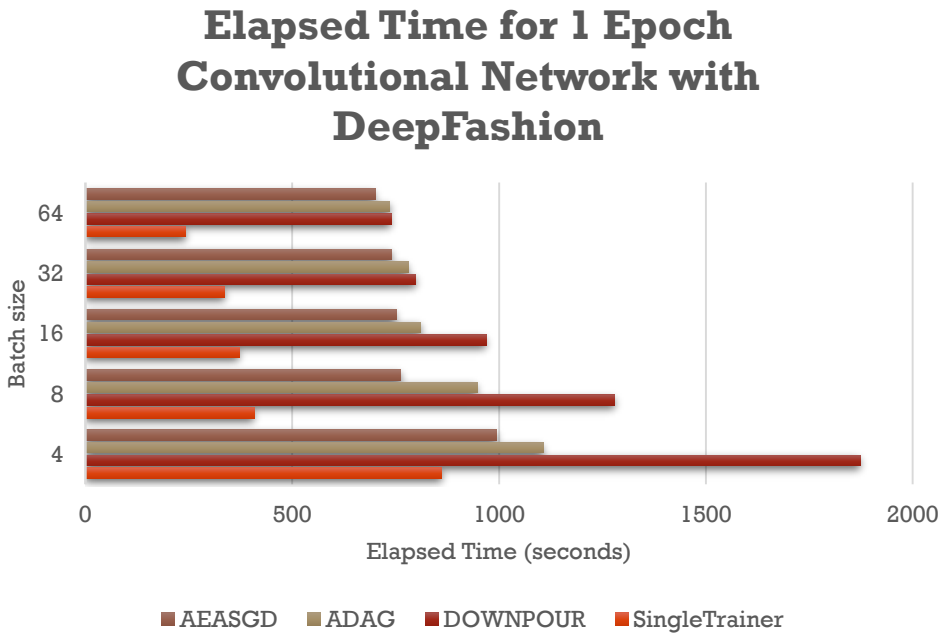


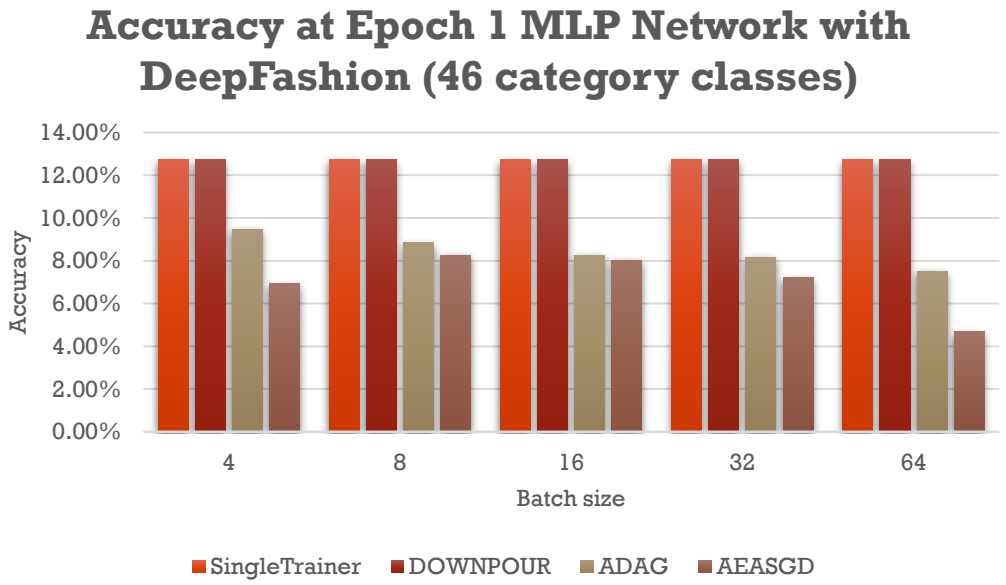


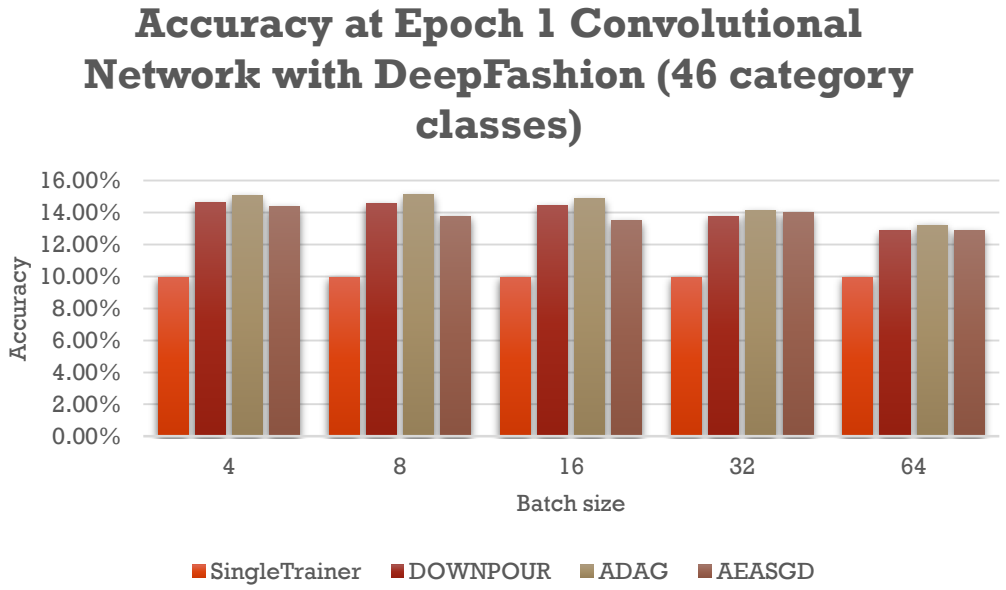
RESULTS FOR FASHION MNIST (ACCURACY)

RESULTS FOR DEEPFASHION (ELAPSED TIME)









RESULTS FOR DEEPFASHION (ACCURACY)

OBSERVATIONS

- SingleTrainer does not work stable.
- It is a trade-off between training time and accuracy where the batch size changes.
- ADAG is robust to different parametrization.
- DOWNPOUR is slow and less accurate by comparison with ADAG and AEASGD.
- In general, the results (elapsed time && accuracy) are worse than our results in the local environment.
  - Accuracy is very close
  - Elapsed time is not close

REFERENCES

1. Hortonworks Sandbox Hadoop Tutorial Getting Started with HDP https://hortonworks.com/tutorial/hadoop-tutorial-getting-started-with-hdp/
2. Distributed Keras (dist-keras) https://github.com/cerndb/dist-keras
3. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms", H. Xiao, K. Rasul, R. Vollgraf, 2017
4. "DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations", Z. Liu, P. Luo, S. Qiu, X. Wang, X. Tang, 2016
5. "An overview of gradient descent optimization algorithms", S. Ruder, 2017.
6. "Large Scale Distributed Deep Networks", J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A. Y. Ng, 2012
7. "Deep learning with Elastic Averaging SGD", S. Zhang, A. Choromanska, Y. LeCun, 2015.
8. "Heterogeneity-aware Distributed Parameter Servers ", J. Jiang, B. Cui, C. Zhang, L. Yu, 2017.
9. "Gradient Energy Matching for Distributed Asynchronous Gradient Descent", J. R. Hermans, G. Louppe, 2018.

THANK YOU!

Osman Furkan Kınlı – S002969