**DCSL Assignment report**

**Brief explanation of the problem**

The goal of this exercise is to build distributed ML systems that are robust against Byzantine failures. The paper being referred to is - <http://proceedings.mlr.press/v80/yin18a.html>. The paper mentions the use of median and trimmed-mean operations for aggregation while performing distributed gradient descent. The paper proves that the algorithms with these operations achieve order-optimal performance for convex loss functions. The paper also elucidates an algorithm that uses median-based aggregations and needs only one round of communication (not covered in this report).

**My work**

**Algorithm setup:**

I have created a setup borrowing pointers from the setup in the paper and from the instructions provided in this document: <https://docs.google.com/document/d/1s_2vlrtPujewogSsvaLaX-BkWGt-l35QxNN9YINE9GY/edit?usp=sharing>

1. Define a function **“build\_model”** to build a CNN classifier model as instructed in the document. TensorFlow is used to define the model.
2. Define a function **“step”** that takes input – batch of 32 training data, training labels, model – and returns the gradient calculated for each parameter by carrying out a feedforward cycle using the training data, calculating the los and finding the gradients of each parameter with respect to the loss. TensorFlow Gradient Tape is used to retrieve the gradients from the model.
3. Initialize Batch size and training rate as instructed in the document.
4. Load the MNIST data and preprocess the data (Normalize the data, Convert the labels to categorical data, divide the data to training and test set).
5. **Sharding:** Split the training data among n machines (n is 10 in my results)
6. Create a model with the “build\_model” function and define a Standard Gradient Descent(SGD) optimizer with the initialized learning rate.
7. **The** **training**: Iterate the following until error reduces below a threshold (In my experiment it iterates 1000 times). In each iteration:

* For each machine, get 32 training data items randomly and create the training batch.
* Apply label switching attack on the labels of this batch if you want to configure any attacker
* Calculate gradients for your model parameters for a step of training using “step” function
* Do mean/median/trimmed-mean aggregation of the 10 machines’ gradients.
* Apply this aggregated gradient to take one step in gradient descent.

1. Loss and accuracy are calculated at every few iterations and saved in a file for later use in analysis of results.

**Algorithm hyperparameters and results:**

For all the experiments:

* Batch size is 32
* Number of machines is 10 (m)
* Number of training examples per machine is 6000 (n)
* B for trimmed-mean is 0.2
* The model is a CNN based classifier as defined in the document given above.

1. 2 Attackers (a=0.2), Learning Rate: 0.01

Chart, histogram

Description automatically generated

Insights:

* On the long run, trimmed-mean aggregation leads to better performance
* Median based method initially performs well but tends to get trapped on a local minimum. This is probably because of large attackers-to-machines ratio (0.2 in this experiment).
* This seems intuitive as the median calculated may turn out to be the component of the attacker and it sways the weights in a wrong direction. This sway is greater than in mean or trimmed-mean as they are averaged over all machines.
* This large step may cause it to get stuck in a sub-optimal minimum.

To Do:

* Try larger number of machines and lesser attackers-to-machine ratio to see the performance of median based aggregator.
* Lower learning rate to lower the sway.
* Regularization!?

1. 1 Attacker (a=0.1), Learning Rate: 0.01

Chart, histogram

Description automatically generated

Insights:

* Since there is only one attacker, mean and trimmed-mean aggregators seem to give similar low loss
* Median based algorithm still reaches a sub-optimal solution due to large learning rate.

1. 2 Attackers (a=0.2), Learning Rate: 0.005

Chart, histogram

Description automatically generated

Insights:

* Trimmed-mean based algorithm performs better in face of greater number of attackers.
* Median-based algorithm does not reach a sub-optimal minimum, but it can benefit from further reducing the learning rate or using regularization.

1. 1 Attacker (a=0.1), Learning rate: 0.005

Chart, histogram

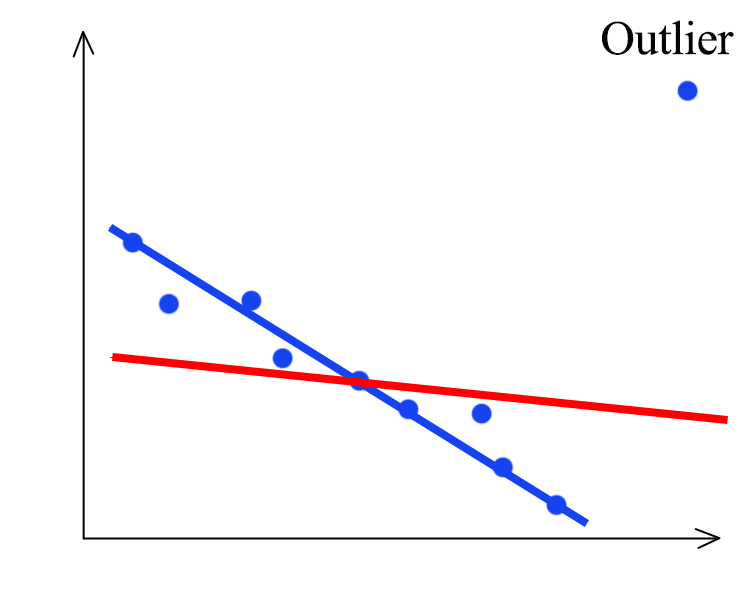
Description automatically generated

Insights:

* Trimmed-mean based algorithm performs better in face of greater number of attackers.
* Median-based algorithm can benefit from a lesser number of attackers or lower/decaying learning rate.

**Improvements / Future research:**

1. Regularization! – To decrease the sway by median and trimmed-mean.



* Regularization sets a penalty on the weights so the weights can’t have a large value. This helps reduce over-fitting and reduces the effects of outliers.
* Median and trimmed-mean based algorithms try to do exactly this. They try to ignore the outliers and select the gradients or mean of gradients from the non-attacker machines.
* Adding regularization to SGD can reduce the effect when an attacker’s gradient is chosen as median or when attacker’s gradient is not trimmed out.

1. Quantify the effects of complexity (Number of parameters to find median / trimmed mean between) on the performance:

* As the complexity increases, the number of parameters to find median and trimmed-mean for increases. This would mean there are more chances of attacker median being chosen and the weights swayed in the wrong direction.

1. Determine the type of problem median-based aggregator is suited for:

* Median-based aggregator is useful when the gradients calculated tend to be outliers. In label switching, they may not be outliers as its switching labels and calculating loss on it.
* Research security problems in Distributed ML systems and determine where median-based algorithm fits better.