Impact Hardware Predictive Uncertainty

How does choice of hardware impact predictive uncertainty?

Collaborators: Haojun Xia (University of Sydney), Allen Hui (University of Sydney), Donglin Zhuang (University of Sydney), Xingyao Zhang (UW Seattle), , Shuaiwen Leon Song (University of Sydney), Sara Hooker (Google Brain, shooker@google.com)

Thanks to colleagues for input so far: cliffyoung@, eriche@, schmit@, hugolarochelle@, Aaron Courville. Balaji Lakshminarayanan

[[note, collaborator list and order not finalized, subject to change based upon contributions]]

1. What problem are we trying to solve?

Recent work[1][2][3] finds that training is dominated by the non-determinism from GPUs, rather than by the initialisation of the weights and biases of the network or by the sequence of minibatches given. For example, in the experiment below fixed batch and fixed initialization there is still a high degree of predictive divergence between runs for GPUs. The y-axis is the % of predictions that don't overlap between ensembles.

[1] Deep Ensembles: A Loss Landscape Perspective [Link]
[2] Non-Determinism in TensorFlow ResNets [Link]https://arxiv.org/pdf/1912.02757.pdf
[3] A Workaround for Non-Determinism in TensorFlow [Link]
4. Problems and Opportunities in Training Deep-Learning Software Systems: An Analysis of Variance, ASE [Link]

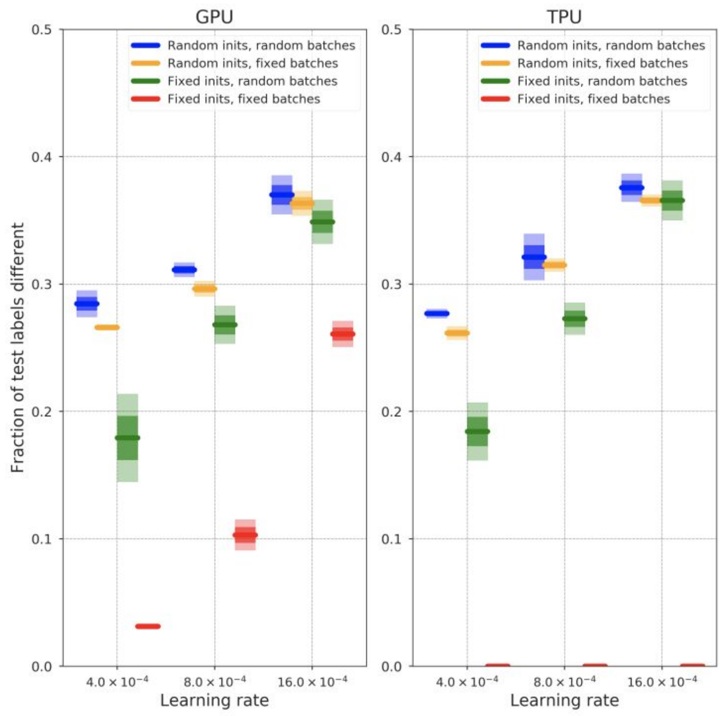


Figure S4: The effect of random initializations and random training batches on the diversity of predictions. For runs on a GPU, the same initialization and the same training batches (red) do not lead to the exact same predictions. On a TPU, such runs always learn the same function and have therefore 0 diversity of predictions.

This introduction of stochasticity is invisible to the machine learning researcher, and prompts the question of how much it impacts the end performance of the algorithm. In this project we embark on answering the following questions:

1) What causes the introduction of stochasticity in GPUs?
2) What is the impact of this stochasticity -- is model performance on certain subsets of the data distribution disproportionately impacted?
3) For the previous two questions, we measure the cause and impact of stochasticity introduced by hardware in the absence of algorithmic stochasticity (by controlling for batch and random seed). However, it is also useful to consider the interaction between stochasticity introduced by hardware versus algorithm? Is this stochasticity additive, or impacting model behavior in different ways?

Implications of this work: GPUs are ubiquitous hardware in the field of machine learning, widely used for training networks that are ultimately deployed at scale. Understanding the implications of hardware design choices that impact end performance is critical from a perspective of trustworthy AI.

Early hypotheses about GPU behavior (thanks in part to conversations with Cliff, Erich, Herman):

- Non-deterministic ops such as atomic floating point addition (TPUs don't use atomics)
- Threaded behavior -- the OS has a bunch of cyclic dependencies, and its state is not precisely the same at the start of execution. Therefore, different threads can get activated at different times, run to run (TPUs are single thread)
- Order-of-evaluation differences in floating-point libraries, where the library writer has a choice between doing the faster thing or the deterministic thing.
- GPUs also have caches, which means that the faster core might vary run-to-run.
- Forward passes are usually deterministic unless you use (un)sorted_segment_sum (so it in the backward pass)
- CUDNN has multiple ways of computing convolutions. Tensorflow autotunes over these algorithms to pick the fastest one. Some of the convolution techniques also use atomics (in the backward pass) which introduce this slight non-determinism.
- This blog post would suggest that the stochasticity is produced using a combination of software and hardware choices.

2. Key Experiments

Whether or not you care about the stochasticity introduced by GPUs hinges on the nature of that stochasticity, and how it interplays with stochasticity that arises naturally from training.

From a trustworthy ML perspective, this stochasticity is concerning if it systematically impacts a subset of the distribution, or amplifies existing algorithmic bias towards that subset. To measure this, we will need to train multiple networks on a GPU, and isolate examples with high predictive divergence between members of the ensemble. We will both visually inspect these images, as well as measure average test set accuracy for this subset across the ensemble.

We will also need to benchmark GPU performance on a dataset with known spurious correlations, such as the CelebA dataset. Here, I would suggest we conduct a similar set of experiments to this work.

3. What are our success criteria and metrics?

- Isolate the operations motivating stochasticity in GPUs
- Quantify and characterize the impact of this stochasticity (both alone and when combined with the stochasticity introduced by training)
- Ideally suggest guidelines as to where stochasticity can be introduced to reduce harm.

3. Collaboration Dynamics

- Biweekly group meeting
- Presentation of intermediate results
- Discussion of additional steps.
- Slide deck from Donglin

Assumption 1:(Xingyao, Leon)

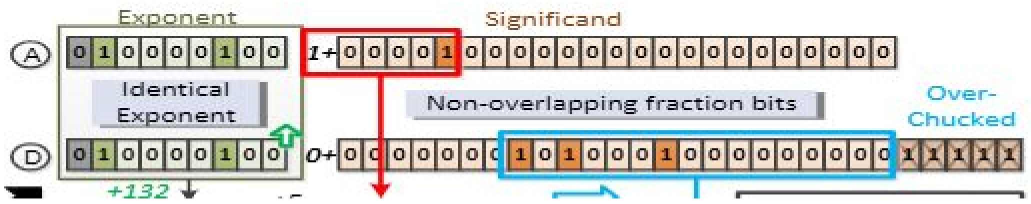
hypotheses 2,4,7 make sense to me.

For example, there are three variables, one is large positive valued one is large negative value and one is near zero data.

if the large positive value first subtract the large positive value and generate near zero data, then add up with another near zero data, then the precision can be promised

If the large positive value first adds up with the near zero data, floating add will match the exponent bits, usually matching the one with larger value, so the near zero data's precision could be lost.

(A+B)+C != A+(B+C) for floating points operations on GPU



W_new = W_old + \delta W * lr or other optimizer
The value difference will be enlarged by larger learning rate

Blue and Yellow: initial difference
Green: loss addition order

Software design suggestions:

Hardware design suggestions: floating point add ordering.

Short-term plan:

After ISCA deadline next week, we will meet again offline and discuss what to do next. Students, if you are not involved in ISCA deadline, please read about our discussion and the newest A100 GPU specs. Thank you !

Experimental Methodology

To begin, we consider volatility on small scale datasets:

Cifar-10 and Cifar-100 (dataset link) example of cifar-10 codebase

For cifar-10 you should achieve a top-1 accuracy of ~90+%, for cifar-100 you should achieve a top-1 accuracy of ~78+%.

Network - ResNet-18, Wide ResNet or ResNet-20.

Code - TensorFlow (we can also experiment with Pytorch as a secondary code base to see if any of the uncertainty is code specific)

Hyperparameters - as a first step, we are attempting to replicate Fig S4 in this paper. So, it is a good starting point to evaluate the hyperparameters described in this paper.

First deliverables:

Let's start with Cifar-10 (and then proceed to Cifar-100).

- Train a baseline where you achieve the required accuracy on Cifar-10, Cifar-100. Plot both top-1 and top-5 accuracy over the course of training. Add those plots to this working doc (along with the specification of what hardware was tested on). We will reproduce these plots for every hardware considered.
- Train 3 different models (an ensemble). Using these 3 different checkpoints, replicate the figure S4 in this paper. The y-axis is the percentage of test labels that are different, that means that for each of the test-examples, you compare the 3 predictions of the trained ensemble, if the labels differ between these 3 models you create a binary variable to indicate that the test labels differ. The y-axis plots the fraction of all test-set labels that has this binary label.

[WIP] Experiment

Training on Cifar10

ResNet18 training curve on V100, dataset: Cifar-10, learning rate: 4e-4, batch size: 128, optimizer: Adam

Test Accuracy: 93%(85% w/o data augmentation)

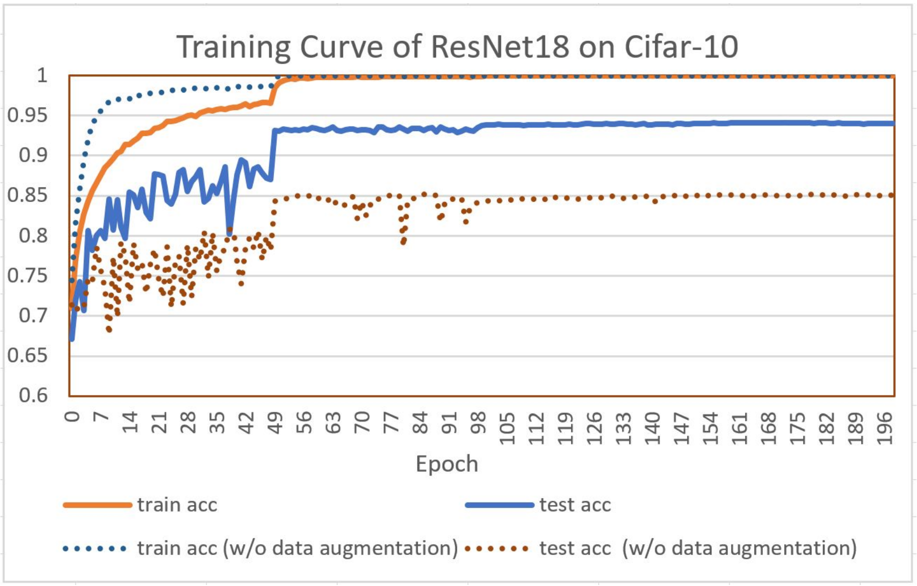
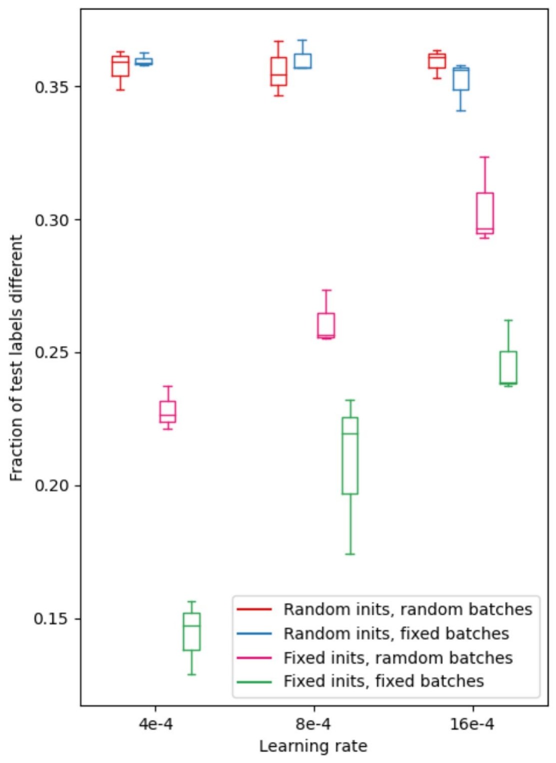


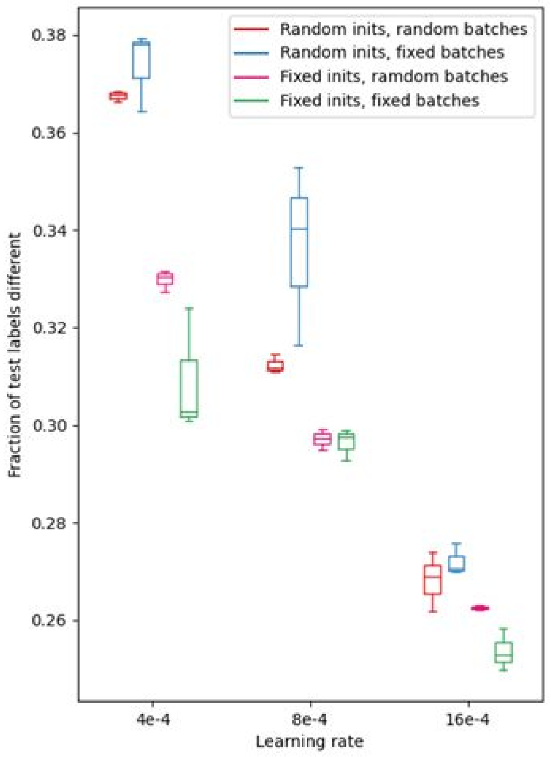
Figure S4 reproduce:

Hints to reproduce the experiment:

1. Do not use data augmentation.
2. Do not use Adam optimizer, use SGD instead. I have two guesses on why Adam optimizer failed to reproduce this experiment, but which one is the main reason still needs further investigations.
    a. Parameters in adam optimizer will incur additional noise.
    b. Default optimizer hyperparameters settings (i.e. beta1=0.9 and beta2=0.999) might alleviate the algorithmic stochasticity.
3. Set momentum=0 in SGD: experiments show that momentum in SGD will (to some extent) counteract the algorithmic stochasticity. (please see below side by side comparison)

Reproduce Figure S4 experiment with Resnet18 on Cifar10 dataset. Left: reproduce use SGD optimizer without momentum, right: reproduce use SGD optimizer with momentum=0.9





Fixed inits means fixed random seed for model weight initialization, fixed batches means fixed random seed for shuffle training data.The y axis represent the fraction of test labels that are different, that means that for each of the test-examples. Under fixed inits and fixed batches constraints, we expect we can get rid of algorithmic stochasticity.

Some interesting points:

1. Without algorithmic stochasticity, we still observe models in the same ensemble diverged on prediction results. The divergence further exacerbated with learning rate increase.
2. Fraction of different labels of Random inits+random batches/random inits+fixed batches in the lefr figure does not increase with learning rate. Could the fraction of different labels somehow be saturated at ~0.35 since 0.35 is already a pretty large fraction. This needs further investigation.

TODOs:

- Remaining experiment under experimental methodology section.
- Different from inner layers

Questions and Concerns:

We have some questions regarding the experimental settings (probably needs to ask the authors) to further understand the story behind figure s4.

1. Since we target exploiting hardware level stochasticity, it is essential to make sure there won't be any software level stochasticity. To our best knowledge, the Tensorflow backend computation can be considered as deterministic if we abandon the data augmentation. In other words, Tensorflow provides deterministic mathematical calculations. Can you please help double confirm this hypothesis?

2. We can hardly reproduce the exact same results as the Figure S4 and even get the contrary results when implementing momentum based optimizer. To better understand the configuration impact on the non-deterministic, can you please help provide the hyperparameters/configurations that were adopted in figure s4's experiment (e.g., model, dataset, training epoch, learning rate decay, model test accuracy, optimizer and ect..)?

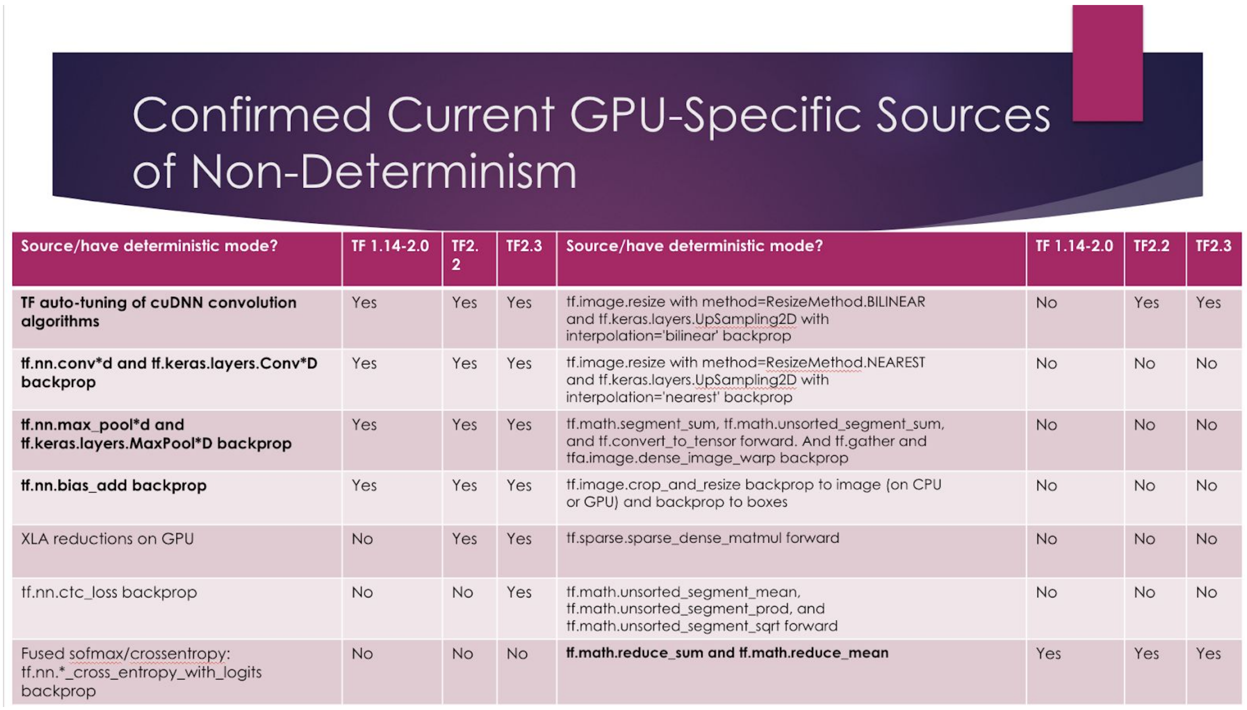
3. In Figure S4, we notice that the fraction of differences is around 0.3 for most experiments, indicating significant accuracy impacts. Can you please provide us the information about whether the model that evaluated is converged in each case and what are the final accuracies? That could be yaa great help for analyzing our reproduced cases.

Next steps we agreed upon:

- Now we have a working baseline -- let's understand how much this baseline changes depending on hardware type -- what different gpu types does it make sense to benchmark here Shuaiwen, Xingyao? To start with, we can take the code variant which matches the fig s4 and deploy on multiple different types of hardware.
    - **We should run two variants for each:** 1) with code as is, 2) one with TF_DETERMINISTIC_OPS and TF_CUDNN_DETERMINISTIC set to true. It will be important to measure the difference in cost between these two variants -- both in terms of final difference in overlapping predictions (fig S4), but also in terms of latency -- here several metrics could be interesting -- overall training time, latency for a full backprop update. What else should we be tracking?
- While the baseline we have chosen contains a subset of tensorflow ops that are not deterministic on GPUs, it would be helpful to understand the range of noise introduced between different types of operations in the chart below. To understand this better, it would

be good to have better insight into the underlying GPU operation that causes the noise for each operation -- for example we have multiple image transformations that introduce noise -- why is this the case -- is it the same underlying GPU operation each time?

-



Confirmed Current GPU-Specific Sources of Non-Determinism

Source/have deterministic mode?	TF 1.14-2.0	TF2.2	TF2.3	Source/have deterministic mode?	TF 1.14-2.0	TF2.2	TF2.3
TF auto-tuning of cuDNN convolution algorithms	Yes	Yes	Yes	tf.image.resize with method=ResizeMethod.BILINEAR and tf.keras.layers.UpSampling2D with interpolation='bilinear' backprop	No	Yes	Yes
tf.nn.conv*d and tf.keras.layers.Conv*D backprop	Yes	Yes	Yes	tf.image.resize with method=ResizeMethod.NEAREST and tf.keras.layers.UpSampling2D with interpolation='nearest' backprop	No	No	No
tf.nn.max_pool*d and tf.keras.layers.MaxPool*D backprop	Yes	Yes	Yes	tf.math.segment_sum, tf.math.unsorted_segment_sum, and tf.convert_to_tensor forward. And tf.gather and tfa.image.dense_image_warp backprop	No	No	No
tf.nn.bias_add backprop	Yes	Yes	Yes	tf.image.crop_and_resize backprop to image (on CPU or GPU) and backprop to boxes	No	No	No
XLA reductions on GPU	No	Yes	Yes	tf.sparse.sparse_dense_matmul forward	No	No	No
tf.nn.ctc_loss backprop	No	No	Yes	tf.math.unsorted_segment_mean, tf.math.unsorted_segment_prod, and tf.math.unsorted_segment_sqrt forward	No	No	No
Fused sofmax/crossentropy: tf.nn.*_cross_entropy_with_logits backprop	No	No	No	**tf.math.reduce_sum and tf.math.reduce_mean**	Yes	Yes	Yes