On Efficient Constructions of Checkpoints

Yu Chen, Zhenming Liu, Bin Ren and Xin Jin

Checkpoint for ML applications

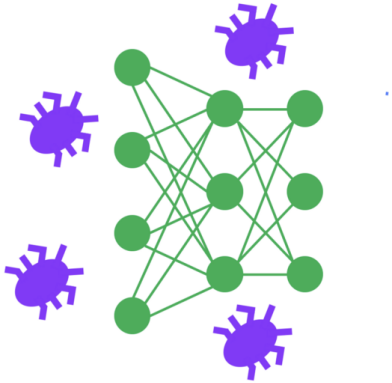
```
def train_and_checkpoint(net, manager):
  ckpt.restore(manager.latest_checkpoint)
  if manager.latest_checkpoint:
    print("Restored from {}".format(manager.latest_checkpoint))
  else:
    print("Initializing from scratch.")

  for _ in range(50):
    example = next(iterator)
    loss = train_step(net, example, opt)
    ckpt.step.assign_add(1)
    if int(ckpt.step) % 10 == 0:
      save_path = manager.save()
      print("Saved checkpoint for step {}: {}".format(int(ckpt.step), save_path))
      print("loss {:1.2f}".format(loss.numpy()))
```

Recovery from checkpoint

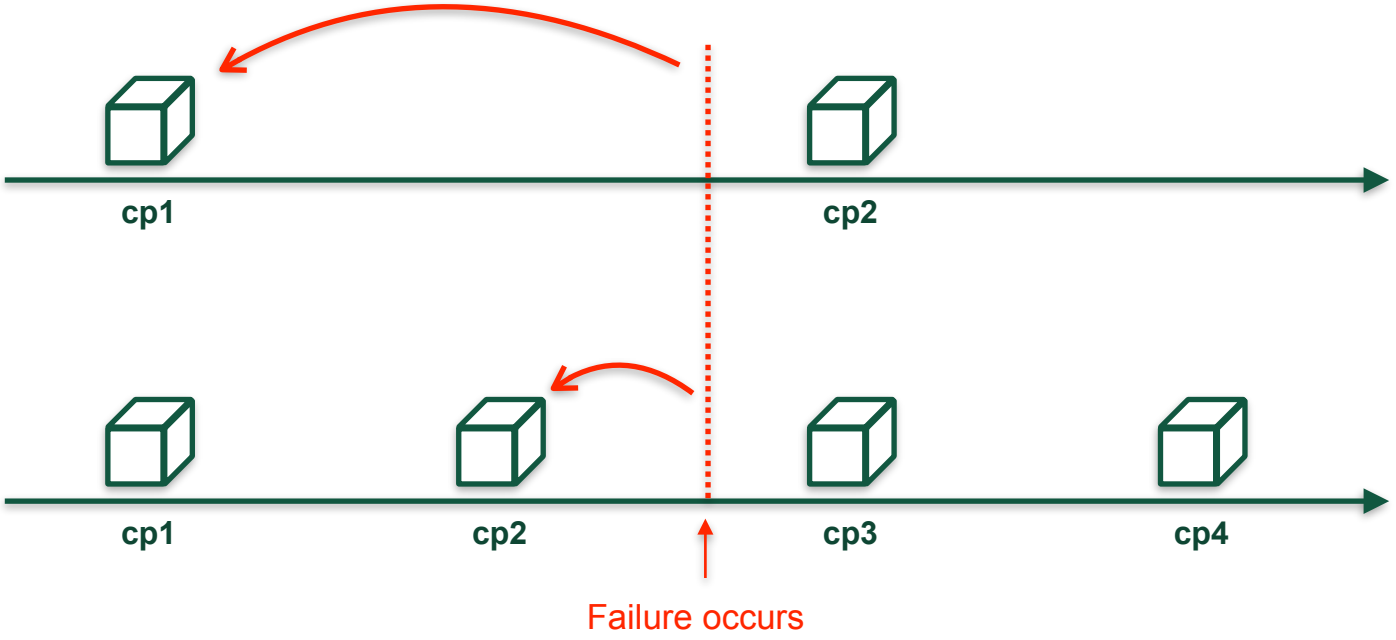
Save model's state for recovery

Checkpoint for ML applications



- Application errors
  - Divide by zero
  - Gradient explosion
  - Dead activation
- System failures
  - Power outages
  - Unstable network
  - Unhealthy disks
- Cloud computing
  - Spot instance
  - Container rescheduling

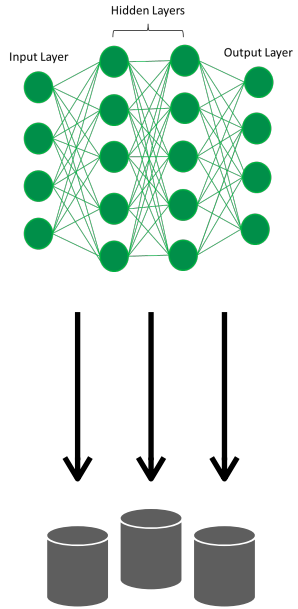
Checkpoint for ML applications



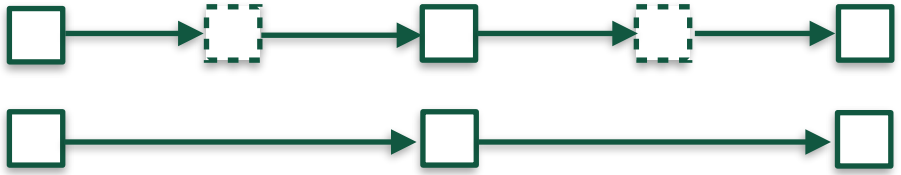
Frequent checkpoint has less recovery cost

Checkpoint for ML applications

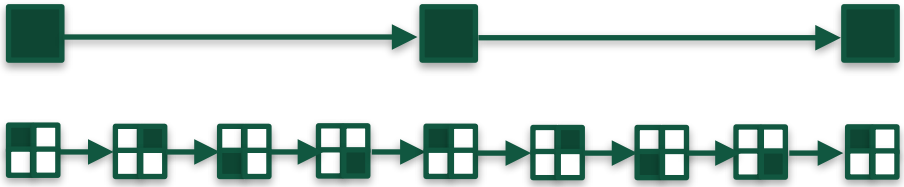
Frequent checkpointing is costly for IO and storage



System: **Decrease checkpoint frequency**



ML & System: **Partial checkpoint**

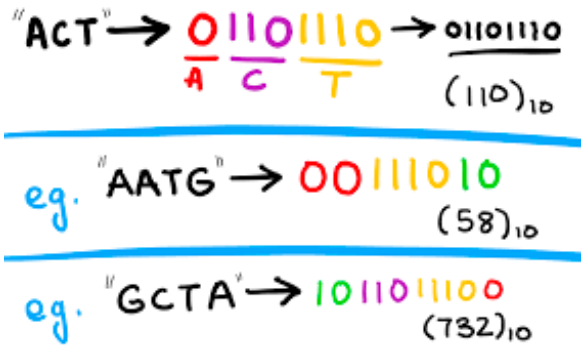


ML & System & Information theory

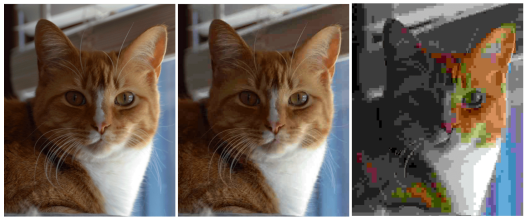
How can we compress the model checkpoint?

Compression

- Lossless compression

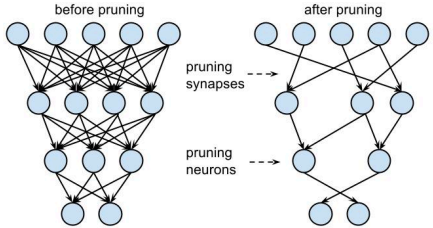


- Lossy compression
  - l_2 distance-based compression



Three levels of JPG compression. The left-most image is the original. The middle image offers a medium compression, which may not be immediately obvious to the naked eye without closer inspection. The right-most image is maximally compressed.

  - Model compression



How to find the redundant information?

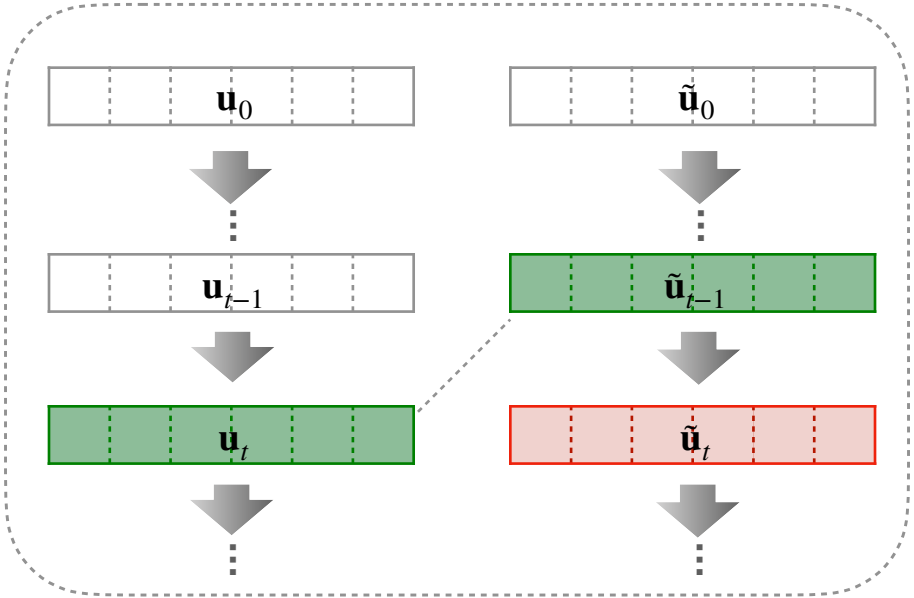
How to design a suitable scheme?

Design

- Design principles
  - Minimize irritation to SGD
  - Maximize redundancies in residual information

- Two key components
  - Approximate tracking by delta-coding.
  - Quantization and Huffman coding.

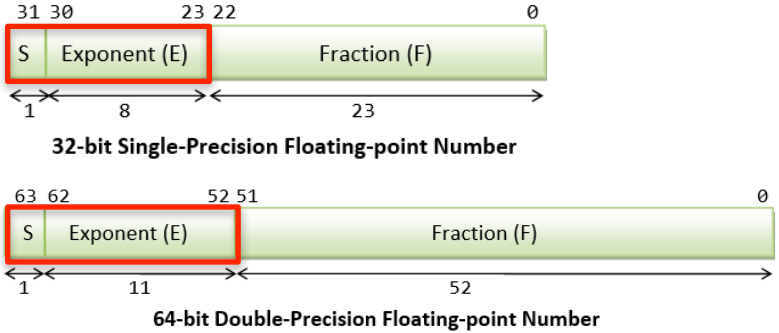
Approximate tracking by delta-coding.

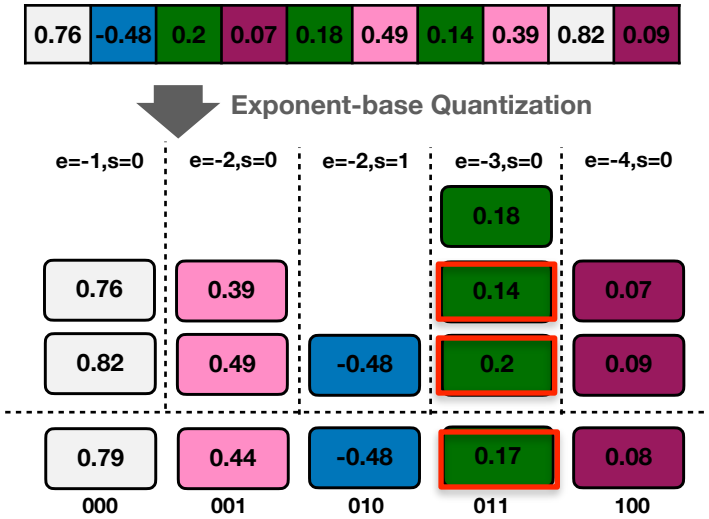


$$\begin{cases} \tilde{\mathbf{u}}_t = \mathbf{u}_0 + \sum_{i \le t} \tilde{\delta}_i \\ \delta_t = \mathbf{u}_t - \tilde{\mathbf{u}}_{t-1} \\ \tilde{\delta}_t = f(\delta_t) \end{cases}$$

Quantization and Huffman coding

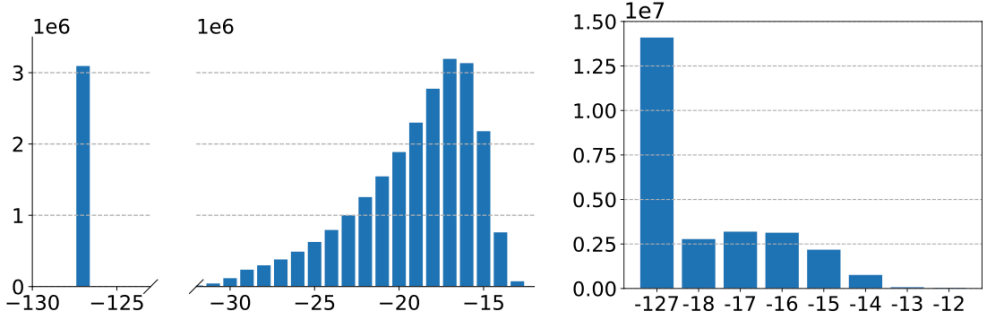
- Two stage quantization
  - Exponent-based quantization
  - Priority Promotion
- Huffman coding



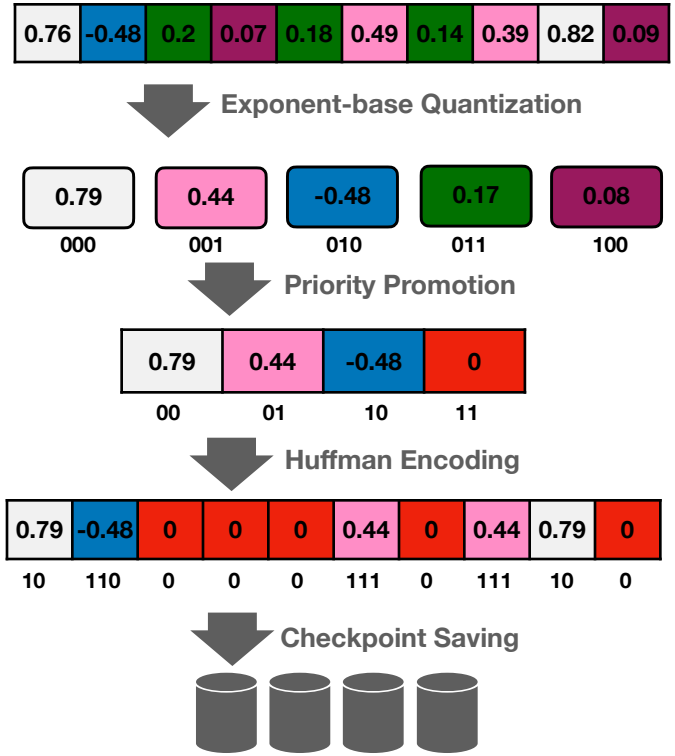


Quantization and Huffman coding

- Two stage quantization
  - Exponent-based quantization
  - Priority Promotion
- Huffman coding



(a) Exponent distribution of δ. (b) Exponent distribution of $\tilde{\delta}$ (3-bit promotion)



Design

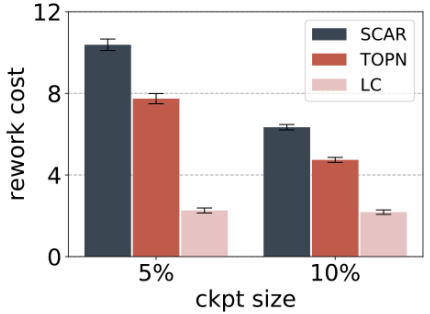
- System optimization
  - Asynchronous execution
  - Checkpoint merging
  - Huffman code table caching

Evaluation

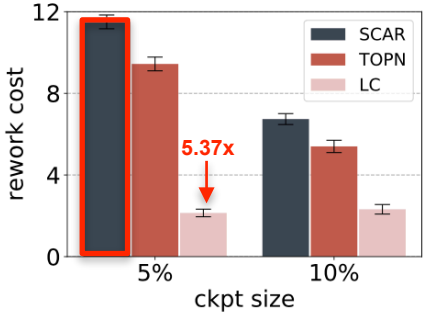
- Models
  - Logistic Regression
  - LeNet-5
  - AlexNet
  - Matrix Factorization
- Dataset
  - MNIST
  - Fashion-MNIST
  - Jester
  - MoiveLens10M
- Objective
  - Comparing the recover cost with previous works
  - Evaluating the compression benefit brought by different approaches
  - Validating the effectiveness of priority promotion
  - Confirming the low overhead

Recovery cost comparison

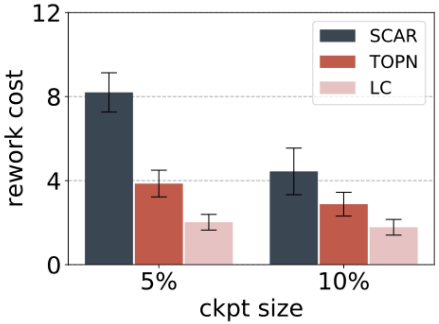
- Outperforming SCAR by 2.88x-5.77x, and TOPN by 2.17x-4.06x at 5% checkpoint size
- Outperforming SCAR by 1.9x-4.82x, and TOPN by 1.52x-2.17x at 10% checkpoint size
- LC-checkpoint has more stable rework cost as the checkpoint size decreasing



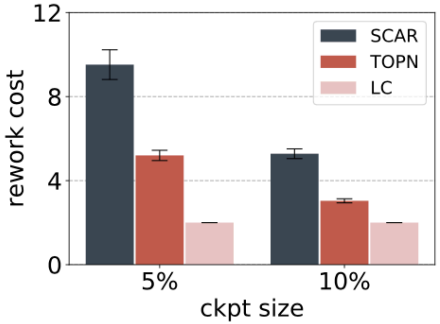
(a) MLR on MNIST.



(b) LeNet on MNIST.



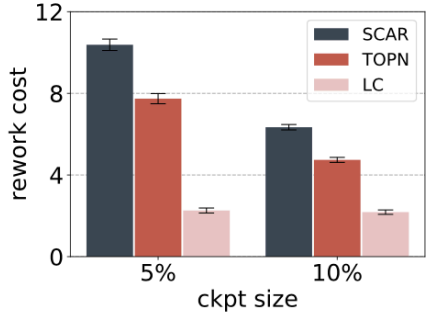
(c) AlexNet on MNIST.



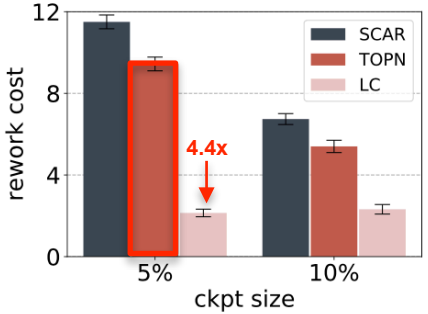
(d) MF on MovieLens.

Recovery cost comparison

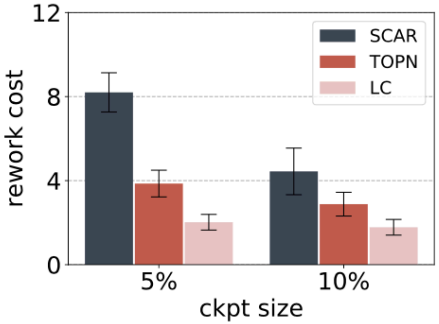
- Outperforming SCAR by 2.88x-5.77x, and TOPN by 2.17x-4.06x at 5% checkpoint size
- Outperforming SCAR by 1.9x-4.82x, and TOPN by 1.52x-2.17x at 10% checkpoint size
- LC-checkpoint has more stable rework cost as the checkpoint size decreasing



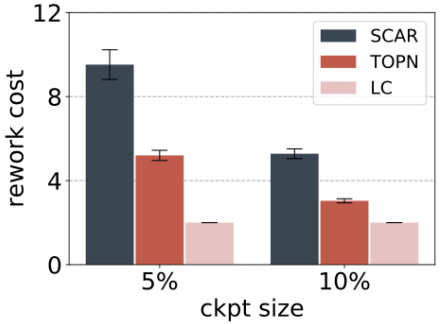
(a) MLR on MNIST.



(b) LeNet on MNIST.



(c) AlexNet on MNIST.

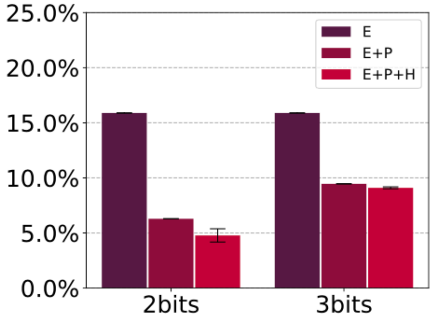


(d) MF on MovieLens.

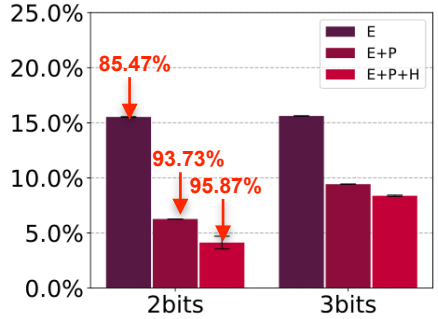
Compression effect breakdown

- Exponent-based quantization
- Priority promotion
- Huffman coding

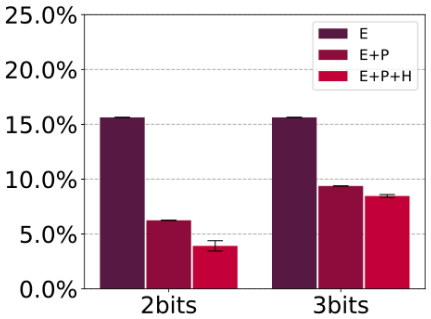
- E yields a compression ratio of 85% on average
- P brings 9.26% extra compression ratio on average for 2-bit and 6.23% for 3-bit
- H brings 2% extra compression ratio with 2-bits priority promotion, and 1.6% with 3-bits one
- P with smaller bits yields more benefits for H



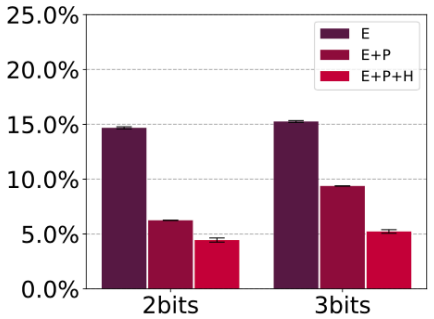
(a) MLR on MNIST.



(b) LeNet on MNIST.



(c) AlexNet on MNIST.



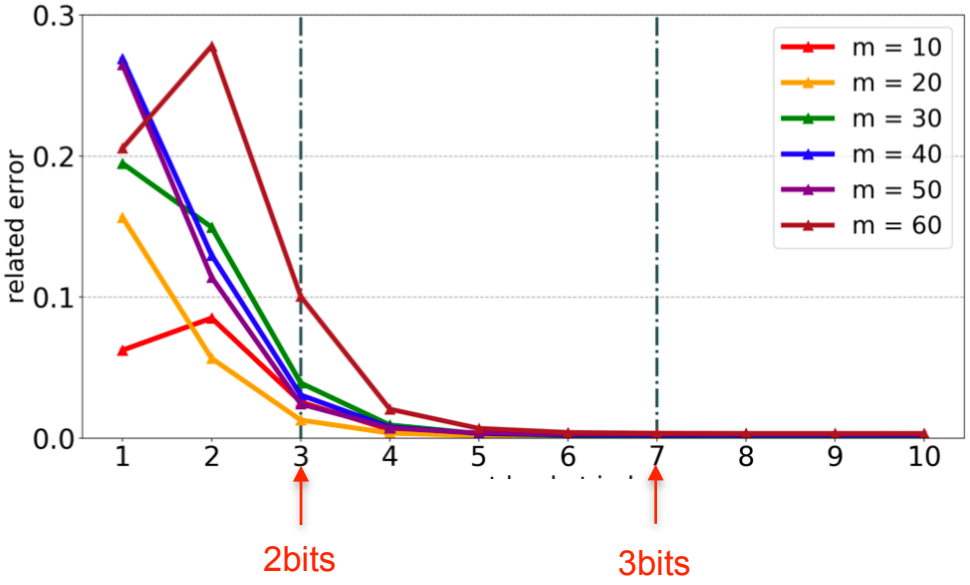
(d) MF on MovieLens.

The effectiveness of priority promotion

Rebuild the $\mathbf{u}_{t+m}$ by $\mathbf{u}_t + \delta_m$

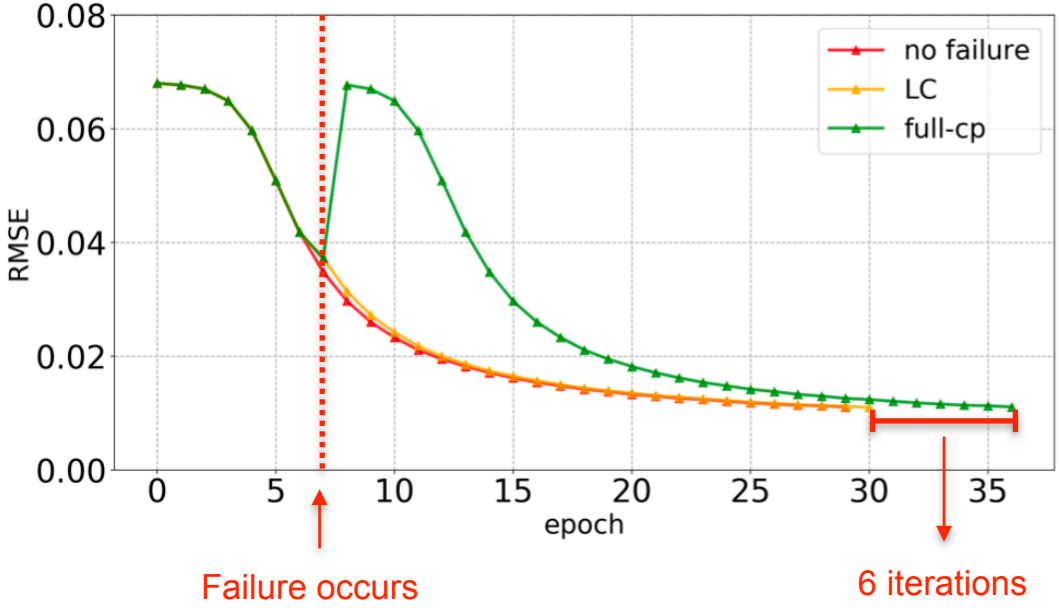
- X-axis: The exponent bucket id which to be removed from δ_m
- Y-axis: Related error calculated by loss function, lower is better.

- Smaller exponent buckets have negligible impact to model state
- 3 buckets (2bits) and 7 buckets(3bits) can hold most of significant bits.



Overhead

- Each iteration costs 91 seconds on average
- A failure occurs at 7th iteration
- LC checkpoint saves 6 iterations (546 seconds)
- LC checkpoint has less than 4 seconds overhead



Conclusion

- Propose an important research question: how to compress checkpoint
- Characterize a family of compression schemes for tracking learning process
- Design a lossy coding scheme to compress checkpoint
- Optimize the training systems to achieve low overhead checkpoint
- Achieve the compression rate up to **28x** and recovery speedup up to **5.77x** over the state-of-the-art algorithms

Thank you for your attention!

ychen39@email.wm.edu

Checkpoint for ML applications

- Classic checkpoint mechanism
  - Save model state periodically
  - Partially save model state for faster recovery
- Key technical challenge
  - Frequent checkpointing is costly for IO and storage
- How can we compress model checkpoint?
  - Maximize the compression rate
  - The scheme needs to be optimized for ML application

Delta encoding scheme with lossy compression