Coded-InvNet for Resilient Prediction Serving Systems

Supplementary Material

Tuan Dinh [1] **Kangwook Lee** [2]

In (A), we present additional experiment results, including a concrete example of the illustration in Section 1.1 (A.1), additional results of degraded accuracy for different k values and architectures (A.2), full results of multi-task classification (A.3), and more end-to-end latency evaluations (A.4).

In (B), we present the detail of architectures, choices of loss, training parameters, and observations. We also present a learning curve of encoder training.

Furthermore, we provide a more detailed discussion on decoding overhead and online decoding in (C).

A. Additional Experiment Results

A.1. Linear Functions on Synthesis Dataset

To complete the story of illustration in Section 1.1, we synthesize a 2D-dataset with a rotation function. Here, we use the setting $n = k + 1$ with k inputs and n parallel workers. The inference function f_θ is the rotation function with angle $\theta = \frac{\pi}{3}$, which has rotation matrix as $\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}$. The inverse function of f has the rotation matrix as $\begin{bmatrix}\cos\theta & \sin\theta \\ -\sin\theta & \cos\theta\end{bmatrix}$. Input distribution $\mathcal{D}$ is the mixture Gaussian $\frac{1}{2}\mathcal{N}(\mu_1, \Sigma) + \frac{1}{2}\mathcal{N}(\mu_2, \Sigma)$, where $\mu_1 = \begin{bmatrix}1\\0\end{bmatrix}, \mu_2 = \begin{bmatrix}0\\1\end{bmatrix}, \Sigma = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$.

For evaluation, we randomly draw a set of k inputs $\{x_j\}_1^k$ from the input distribution $\mathcal{D}$. We randomly select an input x_a as the missing target to recover and remove it from the input set. We recover x_a from the f values of the remaining inputs $\{x_j\}_{j\neq a}$, as

$$\hat{x_a} = kf(\frac{\sum_{j\neq a} x_j}{k}) - \sum_{j\neq a} f(x_j)$$

[1]Department of Computer Sciences, University of Wisconsin-Madison, Madison, USA [2]Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, USA. Correspondence to: Tuan Dinh <tuan.dinh@wisc.edu>, Kangwook Lee <kangwook.lee@wisc.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

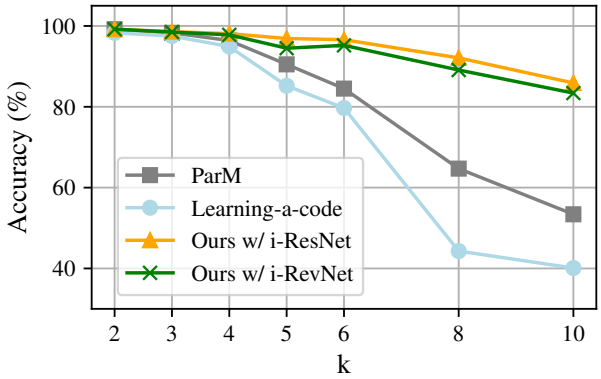

Figure 1. **Degraded Accuracy measured on MNIST.** *Comparison the accuracy of two baselines, ParM (grey), Learning-a-code(blue), with Coded-InvNet built on top of i-ResNet (orange) and i-RevNet (green). Our Coded-InvNet models outperforms both baselines on every k, especially when k is large.*

We measure the reconstruction error as $\|x_a - \hat{x_a}\|_2$. We repeat this process 50K for each value of $k \in [2, 100]$, and calculate the mean of reconstruction errors.

We found that the reconstruction errors are almost negligible for all k values (from 10^{-15} to 10^{-14}). These errors are probably caused by floating-point errors in computing. This result shows that our framework exactly recovers the missing inputs.

A.2. Additional Degraded Accuracy on More Values of k, and An Additional Invertible Architecture.

Fig. 1 presents the degraded accuracy measured on MNIST with $k = 2, 3, 4, 6, 8, 10$. We add another invertible architecture, i-RevNet (Jacobsen et al., 2018) as an alternate to i-ResNet in our Coded-InvNet framework. The result confirms our findings that Coded-InvNet outperforms the baselines, and the gap becomes larger as k increases.

A.3. Applicability to Multi-task Serving

Table 1 presents results of 2-task classification with $k = 2, 4, 10$. The two tasks are image classification on FINE (10-class) and COARSE (2-super-class) sets using the shared embedding learned by Coded-InvNet.

Table 1. **Illustration of Coded-InvNet on multi-task learning.** *Normal accuracy (Normal) and degraded mode accuracy ($k = 2, 4, 10$) on 10-class* FINE *image classification and 2-super-class* COARSE *image classification. The two tasks use the same embedding learned by Coded-InvNet. While maintaining the high degraded mode accuracy on the* FINE *task, Coded-InvNet achieves high accuracy on* COARSE *tasks with a modest overhead of computing (just adding the linear classifiers).*

Task	Normal	$k = 2$	$k = 4$	$k = 10$
Fine	86.2%	74.7%	43.4%	31.2%
Coarse	98.2%	93.5%	86.6%	71.4%

We note that the normal accuracy drops in the fine classification. Indeed, we do not need to retrain the embedding layer $f(\cdot)$ since the coarse classification task can be viewed as a sub-task of the fine classification task. However, to mimic scenarios where we do not have such a hierarchical relationship between tasks, we retrain the embedding layer as well jointly with two task-specific classifiers $g_1(\cdot)$ and $g_2(\cdot)$, accounting for the drops in fine classification accuracy.

A.4. Additional Results of End-to-end Latency

Shown in 2 the measurements on end-to-end latency with more values of $k = 2, 3, 4, 10$, on ParM, Coded-InvNet, and inference models without stragglers. For each k, we use $k + 2$ instances for ParM and Coded-InvNet (an extra redundancy worker). For the measurement of inference models without stragglers, we use $k + 1$ instances (without redundancy). For all values of k, Coded-InvNet shows negligible overhead compared to ParM and inference models.

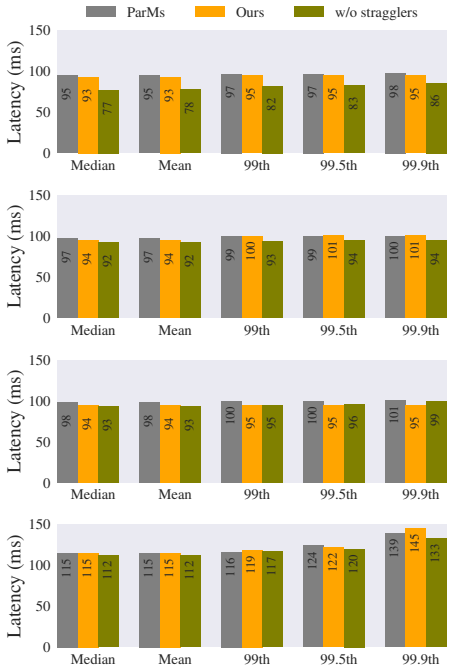

Figure 2. **End-to-end latency on AWS cluster with $k = 2, 3, 4, 10$ (top to bottom).** *Latencies are measured on ParM (grey), our Coded-InvNet (orange) and inference models without stragglers (olive). Coded-InvNet shows negligible overhead compared to ParM and inference models. For instance, in case $k = 10$, the 99^{th}-tail latency of Coded-InvNet is only 3ms higher than ParM, accounting for less than 3% inference latency. Note that, compared to the setup on the inference models, we use an extra redundancy worker on ParM and Coded-InvNet ($k + 2$ instances), our extra latencies (and ParM) probably come from the communication to the redundancy worker.*

B. Details on Architectures and Training

Encoding Training Curves Fig. 3 shows the training curve of our encoding function, with $(n, k) = (5, 4)$. We select the best encoding model based on the valuation loss.

i-ResNet Architecture We use 7, 9 and 9 convolutional i-ResNet blocks for MNIST, Fashion-MNIST and CIFAR10 respectively. Note that 7 and 9 i-ResNet blocks correspond to ResNet-64 and ResNet-82, respectively. For i-ResNet, we remove the injective padding module that introduces zero paddings to increase spatial dimensions of images for classification performance improvement. This removal results in a slight classification accuracy decrease, but significantly improves the invertibility of i-ResNet, especially for off-manifold embedding vectors.

Pix2Pix Architecture We use the U-Net architecture for generators and the PixelGAN model (Makhzani & Frey, 2017) for discriminators (instead of PatchGAN in (Isola et al., 2017)), with the recommended architectures (Isola et al., 2017). We maintain the low encoding overhead by designing a sufficiently small architecture for the encoder. Furthermore, one may minimize the trained encoder's inference time by applying compression techniques to the larger encoder models.

Choice of Loss We have tried different loss functions for the encoder training, including various GAN losses, regression loss, knowledge distillation loss, and their combinations. Regression losses (L_1 and L_2 losses) do not capture well the semantic, so a small regression loss does not necessarily imply a small error in the embedding space. In our experiments, encoder training failed when L_1 or L_2 losses were used without GAN losses, except when trained on MNIST with $k = 2$. Knowledge distillation (KD) loss is observed to work better than regression losses and sometimes even better than GAN loss in terms of degraded accuracies, as KD loss directly utilizes the soft labels. However, as KD loss is specific to the classification, and it is not clear how one can use the KD loss for encoder training when different

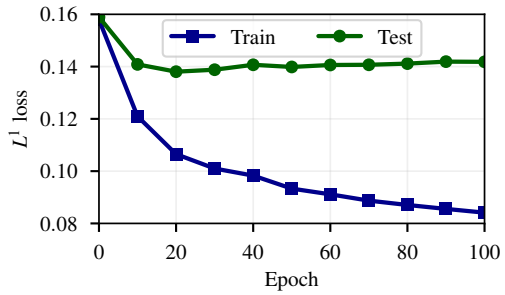

Figure 3. **Encoder training curve.** *We show the encoder training curve. Here, we train an encoder on MNIST with $k = 4$. The loss function is a combination of GAN loss and L_1 loss. Observe that the L_1 (or L^1) loss on train data keeps decreasing while the test loss saturates around epoch 20. We choose the best-performing epoch based on the validation loss.*

types of downstream tasks are given. The combination of GAN loss and L_1 loss worked the best for most cases, but we also observed several failure cases. When k is large, the ideal encoded inputs lose most of their structural patterns and semantic representation, making GAN loss less useful. The design of an efficient loss function for encoder training is an interesting open problem.

Further Training Details We train the classifier with Manifold Mixup. Specifically, we apply Manifold Mixup on random layers (including the input layer) with the mixup coefficient (α_{mixup}) being 1. Each classifier is trained for 200, 400, 600 epochs on MNIST, Fashion-MNIST, and CIFAR10, respectively. For Imagette2 dataset, we transfer the i-RevNet classifier (Jacobsen et al., 2018) for ImageNet to ImagetNett2 and fine-tune with Manifold Mixup in 100 epochs. For *the classifier training*, we use Adam optimizer (Kingma & Ba, 2014) with $\beta_1 = 0.5, \beta_2 = 0.999$. We set the learning rate as 0.1 with 40 warming-up epochs, and decay the learning rate by 0.2 every 60 epochs. The batch size is 128. For *the encoder training*, we train 100, 200, and 500 epochs for $k = 2, 4, 10$ respectively, as it becomes harder to learn when k increases. We also use Adam optimizer with $\beta_1 = 0., \beta_2 = 0.9$ and lambda schedulers for both generators and discriminators. Learning rates are set to be $2e-4$ for both optimizers. We train 5 iterations of the discriminator per each iteration of the generator. The batch size is 64. For *implementation*, we use the PyTorch framework. For the small-scale datasets, we use a single compute node consisting of a 12-GB NVIDIA TITAN Xp GPU, 128-GB of DRAM, and 40 Intel Xeon E5-2660 CPUs. For the large-scale dataset (ImageNette2), we use a 48-GB RTX8000 GPU.

C. Decoding Overhead and Online Decoding

The decoding overhead is minimal. When $n = k + 1$, the decoding procedure simply requires one scalar-vector multiplication and $k-1$ subtractions. To see this, recall that $\widehat{f(x_1)} := kf(x_{k+1}) - \sum_{i=2}^{k} f(x_k)$.

The decoding time can be further reduced by performing online decoding. This is possible because, in practice, not all of the k tasks will complete exactly at the same time. Instead, their task results will be available to the decoder one by one. Therefore, the decoder can continuously update the best-effort estimates of $\widehat{f(x_i)}$'s while receiving task results one by one. More specifically, the decoder can run the following update algorithm at the time of task j ($1 \le j \le k+1$) completion:

$$\widehat{f(x_i)} = \begin{cases} f(x_j) & \text{if } i = j \\ \widehat{f(x_i)} - f(x_j) & \text{if } 1 \le i \le k, i \ne j, \\ \widehat{f(x_i)} + kf(x_j) & \text{if } i = k+1, \end{cases} \quad (1)$$

for all $1 \le i \le k$. Note that one does not have to continuously update $\widehat{f(x_i)}$ after receiving $f(x_i)$. This online algorithm hides all the decoding overhead but one operation, minimizing the decoding overhead by a factor of k, i.e., the decoding overhead does not scale with k.

References

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Jacobsen, J.-H., Smeulders, A., and Oyallon, E. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Makhzani, A. and Frey, B. J. Pixelgan autoencoders. In *Advances in Neural Information Processing Systems*, pp. 1975–1985, 2017.