Infer2Train: leveraging inference for better training of deep networks

Elad Hoffer[1], **Berry Weinstein** [2,3], **Itay Hubara**[2], **Sergei Gofman**[2], **Daniel Soudry**[1]
{elad.hoffer, daniel.soudry}@gmail.com
{bweinstein, ihubara, sgofman}@habana.ai

(1) Technion - Israel Institute of Technology, Haifa, Israel
(2) Habana Labs, Caesarea, Israel
(3) Interdisciplinary Center, Herzliya, Israel

Abstract

Training large scale Deep Neural Networks (DNNs) requires ever growing computational resources. This growth is usually based on larger and faster training devices. However, a new category of inference-only accelerators is emerging, allowing fast and energy efficient forward pass using low precision operations.

In this study, we explore how to leverage such inference-only accelerators for improving training performance. We examine several alternatives and show preliminary results with improved test accuracy on visual-classification tasks such as training ResNet model on the ImageNet and Cifar datasets.

1 Introduction

The recent success of Deep neural networks (DNNs) stems not only from better model architectures and optimization algorithms but also from computational power enhancements. For the past few years, GPUs are the most prominent and frequently used hardware components for training neural networks for tasks such as computer vision [19, 10], speech recognition [11], text translation [24] and generative modeling [23]. The increasing amount of data as well as the increase in model size necessitates the use of distributed algorithms on multiple training devices [9, 21]. Today more than ever, practitioners are willing to assign extreme amount of training resources to gain time-to-convergence speed-ups and model accuracy improvements. To allow the use of these resources, a data-parallelism approach is often adopted, where more training samples are fed in parallel to multiple devices [18]. However, increasing the number of GPUs eventually reaches a point where there is no further improvement neither in training time nor in the final model quality [3].

Recently, specialized hardware devices for inference-only usage of neural network have emerged [16, 1]. These devices allow fast and efficient inference using low-precision arithmetic operations. Since the advantage of adding more training workers is limited, we suggest considering adding a low precision inference hardware instead, as a resource to improve training efficiency, since they usually require much lower overhead of cost and power, .

1.1 Contributions

In this work, we explore novel methods in which an inference low-precision device can improve the training process done on a different full-precision machine. At each iteration, we leverage a fast inference engine to extract a large set of prediction which we use for auxiliary loss. Based on a data selection algorithm we choose out of those predictions a subset that we feed the training worker.

For example, one variant we explored is using the inference device as a hard-negative-miner. In this approach, the samples with the lowest current classification score will be prioritized as the next

mini-batch to the trainer. The inference model can predict faster than the trainer; hence it will be able to select the most useful subset out of a number of examples larger than the training batch size.

We describe and empirically evaluate various methods, and show that a simple variant, named "NM-augment", can leverage common data-augmentation practices and yields improvements in the performance of the final model. We believe that future exploration of co-design approach may prove useful for improving model quality and convergence time, while maintaining low cost of the overall system.

2 Previous works

Over the last couple of years, deep neural networks has drawn a huge amount of attention showing state-of-the-art results on larger more challenging datasets often by using larger and more complex models which in return require more time to converge. Many researchers suggested different methods to improve convergence and reduce training time. Anil et al. [3] suggested to use an online version of neural network distillation [12] by utilizing the output of multiple networks to benefit one another. In their method, each network is augmented with an additional similar network on a separate set of examples. During training, some of the inputs are fed to additional models as to extract their input and construct an additional auxiliary loss consisting of the Kullback-Leibler divergence (KLD) between the outputs.

Another widely used technique for training neural networks is data augmentation [19], in which a random transformation is applied for each input, effectively, allowing for a much larger data set. These augmentation transforms usually use properties of the task domain, such as translations, scaling and rotations for natural images. Lately, Anonymous [4] suggested using large batches composed of multiple instances of the same samples. Thus, they claim that improved model performance can be attained given an under-utilized setting – when batch size can be increased without incurring a time penalty.

A different approach is simply to accelerate the mathematical operations (e.g., matrix multiplication) that are vastly used in neural networks, by using low-precision arithmetic, i.e, quantized neural networks. Neural networks are known to work well for inference, when weights and activations are quantized and operations are performed in low precision arithmetic[15, 24]. Although these models are usually optimized in full precision, and quantized post-training, they were lately used in low-precision even within the training procedure [14, 20, 5]. Low precision quantization allows a significant increase in computational efficiency and performance, allowing both faster inference time as well as lower power requirements. Additionally, usage of 8-bit quantized operations at inference time was noted to incur a negligible decrease in model accuracy. Due to these merits, new hardware platforms were created to use low-precision inference on a large scale. In this work, we are interested in leveraging these platforms to improve model training when using a hybrid inference-training system.

3 Infer2Train ratio

We are interested in a setting, where for a trained model $F(\mathbf{x}; \mathbf{w})$, we possess a faster rate oracle model $\tilde{F}(\mathbf{x}; \mathbf{w})$, serves as inference only version of the model. $\tilde{F}$ may be a noisy version of the original model, such as computing in low precision arithmetic and being time-delayed compared to the current version (taken from previous training step).

We will denote γ as the ratio between the oracle $\tilde{F}$ prediction time and the model training time over the same input. For example, for a standard current device, a training iteration takes roughly $\times 3$ more time than a feed-forward only one (T_{FF}), as it requires, in addition, a gradient computation with respect to the input (T_{GI}) and with respect to the parameters (T_{GW}) which are comparable in size. Thus, given one device for training and an identical one for inference only, we will expect a γ of

$$\gamma = \frac{T_{train}}{T_{infer}} = \frac{T_{FF} + T_{GI} + T_{GW}}{T_{FF}} \approx 3$$

As a clarifying example, let us consider the case of the ResNet-50 model [10] training over the ImageNet dataset [6].

A modern DGX station with 4 Tesla V100 GPU by NVIDIA allows training with the ResNet50 topology at about $3,000$ images-per-second (IPS) [2]. Using a single Tesla V100 card in low-precision inference mode yields approximately $5,000$ IPS – allowing an infer-to-train ratio of $\gamma = 1.67$, which may not be useful in our setting. However, the recently introduced Goya platform [1] allows inference using this model at $15,000$ images-per-second which amounts to $\gamma = 5$ using only a single card.

In this work, we will focus on the setting where $\gamma = 10$ to ease exploration in a realistic environment and to allow future comparison to our results. Following our example, we can attain this ratio by using a hybrid system composed of a DGX station with two additional Goya cards.

We suggest three different methods for inference-to-training acceleration:

- Negative-mining by sample selection
- Negative-mining by augmentation selection
- Self-distillation by augmentation

3.1 Negative-mining using an Oracle

Our primary efforts were focused on using our inference-to-training ratio by selecting the most "helpful" samples for each iteration. This selection procedure is reminiscent of the classical methods for "hard-negative-mining" used by machine-learning practitioners over the years [22, 25]. We will use the cross-entropy loss as a proxy for the utility of a training example, where we assume that a sample with the most positive impact on our model is the one measured to have the highest loss by our oracle $\tilde{F}$.

We will perform negative-mining by sample selection by the following rule. Given a set of inputs $\{x_1, ..., x_\gamma\}$ where γ is the infer-to-train ratio, $\mathcal{L}$ is the loss function and t is the target corresponding for x. We will select the sample with the highest loss value such that

$$\hat{x} = \arg\max_i \left\{ \mathcal{L}\left(\tilde{F}(x_i), t\right) \right\}_{i=1}^{\gamma}$$

We can then proceed the training process using the $\hat{x}$ sample only, as we consider it to be the best representative of the whole $\{x_1, ..., x_\gamma\}$ set. Will call this method negative-mining by sample selection or "NM-sample".

Instead, we can choose to select across the image transformation space. As modern convolutional networks are usually trained using "data augmentations" – random transforms over the image space (crops, scales, etc.), we can use our Oracle to select the most informative instance of each sample. Using the same notation as before, but now introducing T_i – a random transform over the image x, we can select using the rule:

$$\hat{T} = \arg\max_i \left\{ \mathcal{L}\left(\tilde{F}\left(T_i(x)\right), t\right) \right\}_{i=1}^{\gamma}$$

where now our preferred input to our training network is the transformed image $\hat{x} = \hat{T}(x)$. Will call this method negative-mining by augmentation selection or "NM-augment".

Algorithm 1 gives a rough description of our negative-mining approach, where the selection batch $\mathcal{B}_\gamma(i)$ can operate over sample space (NM-sample) or different augmentation transforms (NM-augment).

3.2 Self-distillation by augmentation

As an alternative to the negative-mining approach, we wish to follow the ideas presented by Anil et al. [3] on the co-distillation of neural networks. In their work, Anil et al. [3] trained separately two instances of the same network, each on its separate split of the training data. The training loss was then added with a "co-distillation" loss penalty – the amount of agreement between the two networks, when given the same input. This penalty was conjectured to allow the networks to "share" information regarding the data they used, accelerating convergence time and improving the final result. Since for each network, the additional "mirror" model serves only as a surrogate loss; it is evaluated in inference mode, without gradient computation. This method of operation fits within our framework, allowing for each model γ "mirror" models for distillation.

Algorithm 1:

Require: Inputs-target pairs $\{(x,t)\}_{n=1}^{N}$
$F(x;w)$ – Trained model , $\tilde{F}(x;w)$ – Oracle
$\mathcal{L}(y,t)$ – loss function , $\mathcal{B}_\gamma(i)$ – batch of size γ at step i
T_{update} - number of iterations between oracle updates
repeat
  if $i \mod T_{update} == 0$ **then**
    update oracle: $\tilde{F}(\cdot;w) \leftarrow F(\cdot;w)$
  end if
  $\hat{x},\hat{t} \leftarrow \arg\max_{x,t} \mathcal{L}\left(\tilde{F}(x;w),t\right),\ (x,t) \in \mathcal{B}_\gamma(i)$
  Optimize w according to $\mathcal{L}\left(F(\hat{x};w),\hat{t}\right)$
  $i \leftarrow i+1$
until training is finished

In this work, we aim to improve the training of a single model, rather than the multi-model approach of Anil et al. [3]. Therefore, we are interested to find a viable way to distill information using multiple *input* instances, rather than models. We suggest the use of multiple transforms $T(x)$ of each image, where one example is trained on, and γ different instances are fed in inference-only mode, with the outputs used as auxiliary distillation loss. Our updated loss for model F, sample x, and original loss $\mathcal{L}$ is:

$$\hat{\mathcal{L}}(x,t) = \mathcal{L}(F(x),t) + \lambda \cdot \frac{1}{\gamma}\sum_{i=1}^{\gamma} \mathrm{KL}\left(F(x) \| \tilde{F}(T_i(x))\right) \tag{1}$$

where λ is the weight used for our Kullback-Leibler (KL) divergence penalty.

4 Accelerating inference by low-precision computation

We evaluate the possible use of custom hardware named Goya HL-1000 [1] which accelerates the inference phase of neural networks. The Goya processor is a ground-up design for neural network processing that incorporates a fully programmable Tensor Processing Core (TPC) along with matrix multiplication engine (MME) and software stack named SynapseAI.

To accelerate the neural network general matrix multiplication (GEMM) and non-linear function computations, Goya utilizes TPC and MME computation in low precision mode, using 8 and 16-bit quantization on the network weights and intermediate results. We follow a quantization procedure where the trained model is measured for dynamic range in activations and weights, and then quantized and deployed for use with Goya. This measurement and model update must be performed every 10 to 100s of steps to prevent an oracle with a stale state compared to the trained model.

We are interested in performing b-bit quantization for both activations and weights. We follow the scheme used by Jacob et al. [15] in a signed integer setting, where the range of possible values is $[Q_{min}, Q_{max}] \in \mathbb{Z}$, with

$$Q_{max} = 2^{b-1} - 1 \ , \ Q_{min} = -2^{b-1}$$

Given a tensor $\mathbf{x} \in \mathbb{R}^N$, we obtain the per layer scale S and zero point Z by measuring the dynamic range of the tensor, such that:

$$S = \frac{T_{max} - T_{min}}{Q_{max} - Q_{min}} \ , \ Z = \left[Q_{min} - \frac{T_{min}}{S}\right]$$

where

$$T_{max} = \max_i\{\mathbf{x}_i\} \ , \ T_{min} = \min_i\{\mathbf{x}_i\}$$

and $[\cdot]$ denotes the integer-rounding operation.

A quantized input $\mathbf{x}$ will have the form:

$$\text{quantize}(\mathbf{x}) = \min\left\{\max\left\{\left[\frac{\mathbf{x} - T_{min}}{S}\right] + Z, Q_{min}\right\}, Q_{max}\right\}$$

For activation quantization, we use an average value of T_{min} and T_{max} across multiple samples. We additionally fold intermediate batch-norm operations to the weights of the preceding convolutional layer [15]. To fully benefit from the performance gains of Goya, we use a quantization level of $b = 8$. To allow the use as an inference oracle which mimics the current trained model, we regularly transfer and apply updated scales and zero points, in addition to the quantized weights.

5 Experiments

We evaluate the suggested methods empirically on visual classification tasks, wherein each setting the baseline results is compared with an infer-to-train regime in which $\gamma = 10$.

5.1 Cifar10

We first evaluated our methods on the simple Cifar10 dataset [17] consisting of $50,000$ training images and $10,000$ test images, each belonging to one of 10 categories. As our baseline model, we used ResNet44 [10] with its original hyper-parameters and training regime. As a baseline, we trained with a batch-size of $B = 64$ as initially used. Optimization was performed for 90 epochs, with a learning-drop at epoch 81, after which baseline accuracy of 93.07% was obtained with no apparent improvement.

We tested the three methods suggested in section 3. We first used NM-sample, wherein each step an inference batch of size $B \cdot \gamma$ was sampled *with-replacement*. After computing the loss by our oracle, B samples were chosen for training in each step. To avoid selection bias with respect to class, we limited our selection to balanced subsets of samples with the highest loss for each category. Additionally, we raised the ratio of negative-mined examples with a fixed rate of $r = \min\{1, 0.01 \cdot T\}$ where T is the epoch number, so that in each batch at any given time, only $r \cdot B$ samples were mined, and the rest were randomly selected. We found it crucial to have a small rate at early steps of training, as otherwise a negative impact on performance was observed (see Appendix figure 3 for example). This use of NM-sample provided an average improvement of 0.4% in final validation accuracy.

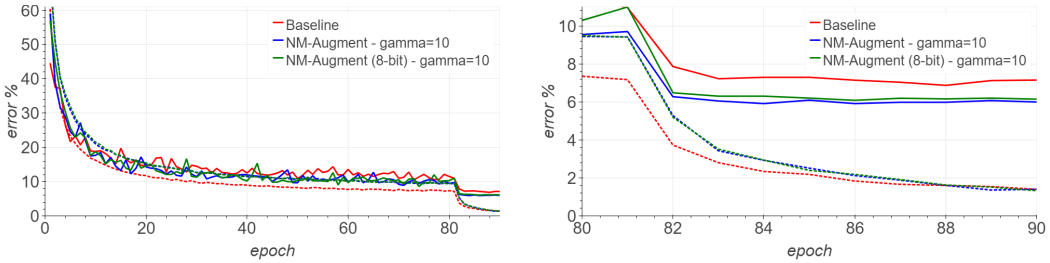
We then continued to try the *self-distillation* method described in section 3.2. For each batch B examples were chosen, each with $\gamma + 1$ different transforms applied to it from the original data augmentation scheme, with overall $B \cdot (\gamma + 1)$ images in each step. An anchor batch of B examples was fed to the training device, and another $\gamma \cdot B$ samples were fed to the inference device. We then added a distillation loss (eq. 1) for each sample to the γ alternative transformations of it. We cross-validated to find the best value of λ, and achieved a 0.3% improvement of validation accuracy for $\lambda = 1e^{-5}$.

The most successful method observed for this task was the NM-augment procedure. In each step, B examples were chosen, each with γ different transforms applied to it from the original data augmentation scheme. For each sample, only the input transform with the highest loss was used for training. Figures 1 shows the results obtained from our NM-augment method using $\gamma = 10$. We can observe an increase in training error (as more "difficult" samples are shown in each iteration) alongside *decrease* in validation error (Figure 1a). Final validation accuracy improved by more than 1%, reaching 94.1% (Figure 1b).

We were additionally interested to see the impact of a low-precision inference as our infer-to-train oracle. We performed the same experiment using a network quantized to 8-bit precision using the methods described earlier. Figure 1b shows that advantage of using NM-augment was kept even using the quantized oracle (green), with only negligible difference from our full-precision oracle (blue).

5.2 ImageNet

Given the strong results on Cifar10 using the NM-augment method, we continued to evaluate its performance on a larger scale task. We used the ResNet-50 model [10], together with the optimization,



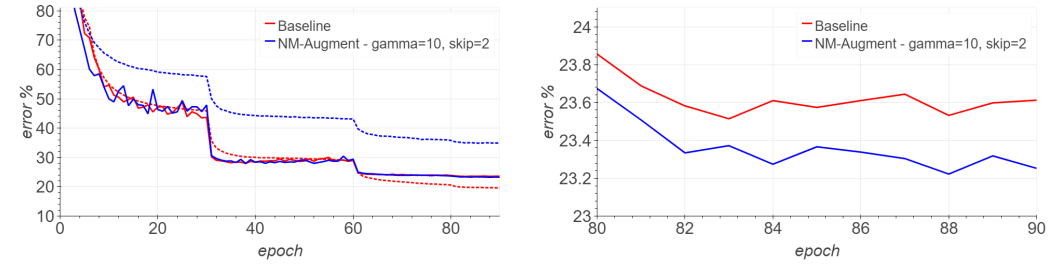
(a) Training (dotted) and validation error

(b) Final training (dotted) and validation error

Figure 1: Infer2Train by NM-augment (ResNet44, Cifar10). We compared the original (red) training regime to Infer2Train with γ = 10 (blue). We also demonstrate that the use of a low precision 8-bit inference as our Infer2Train oracle (green) has negligible impact on our results.

initialization and data augmentation techniques described by Goyal et al. [9]. We evaluated on the ImageNet [6] ILSVRC2012 dataset, composed of approximately 1.2 million images, belonging to 1000 possible classes.

We used a ratio of $\gamma = 10$ again but performed the mining procedure only every two batches. This relaxation can be implemented using half the resources and allowed us to shorten experimentation time. As we can see in figure 2a, the use of NM-augment caused the training error to be significantly higher ($> 15\%$) throughout the entire time. This large deviation may indicate that the effective augmentation regime may be too harsh and that further benefits can be attained using a different regime, or by the use of a higher capacity model. Nevertheless, we observed a benefit in final validation accuracy (figure 2b), improving results from the baseline 76.4% to approximately 76.8% (0.4% improvement).



(a) Training (dotted) and validation error

(b) Final training (dotted) and validation error

Figure 2: Infer2Train by NM-augment (ImageNet, ResNet50). We compared the original (red) training regime to Infer2Train with γ = 10 (blue)

6 Discussion and future work

This work demonstrates for the first time, as far as we know, the possibility of using inference only devices, as a means to improve the training of neural networks. We believe that with the emergence of additional inference devices, this may provide a strong argument for the design of hybrid deep learning systems. These hybrid systems will be augmented with both training and inference devices, allowing better training with little additional cost.

We can also consider additional possibilities not explored in this work, such as methods with alternating training and inference stages. For example, the Generative Adversarial framework [8] requires both a discriminator and a generator, where one part may be used in inference only mode, while the other trains. Another example is back-translation in neural-machine-translation, where monolingual data is fed into an intermediate model to serve as additional multi-lingual training data for a subsequent model. This practice was recently shown to improve performance considerably [13, 7]. The usage of additional inference devices may allow to better incorporate back-translation into the training procedure, with no adverse impact on the required time.

References

[1] Goya inference card. https://habana.ai/wp-content/uploads/2018/09/Datasheet-HL-10x-Sept-14.pdf.

[2] Nvidia tesla v100 gpu architecture. http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[3] Anil, R., Pereyra, G., Passos, A. T., et al. Large scale distributed neural network training through online distillation. In *ICLR*, 2018.

[4] Anonymous. Augment your batch: better training with larger batches. In *Submitted to International Conference on Learning Representations*, 2019. under review.

[5] Banner, R., Hubara, I., Hoffer, E., and Soudry, D. Scalable methods for 8-bit training of neural networks. *arXiv preprint arXiv:1805.11046*, 2018.

[6] Deng, J., Dong, W., Socher, R., et al. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

[7] Edunov, S., Ott, M., Auli, M., and Grangier, D. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018.

[8] Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[9] Goyal, P., Dollár, P., Girshick, R., et al. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[10] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] Hinton, G., Deng, L., Yu, D., et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29 (6):82–97, 2012.

[12] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[13] Hoang, V. C. D., Koehn, P., Haffari, G., and Cohn, T. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pp. 18–24, 2018.

[14] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[15] Jacob, B., Kligys, S., Chen, B., et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference.

[16] Jouppi, N. P., Young, C., Patil, N., et al. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pp. 1–12. IEEE, 2017.

[17] Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[18] Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[19] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[20] Ortiz, M., Cristal, A., Ayguadé, E., and Casas, M. Low-precision floating-point schemes for neural network training. *arXiv preprint arXiv:1804.05267*, 2018.

[21] Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.

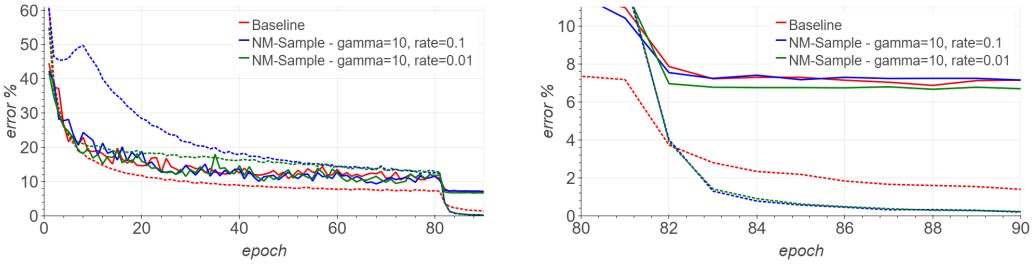
[22] Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

[23] van den Oord, A., Dieleman, S., Zen, H., et al. Wavenet: A generative model for raw audio. In *Arxiv*, 2016.

[24] Wu, Y., Schuster, M., Chen, Z., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[25] Yu, H., Zhang, Z., Qin, Z., et al. Loss rank mining: A general hard example mining method for real-time detectors. *arXiv preprint arXiv:1804.04606*, 2018.

Appendix



(a) Training (dotted) and validation error

(b) Final training (dotted) and validation error

Figure 3: Infer2Train by NM-sample (ResNet44, Cifar10). We compared the original (red) training regime to Infer2Train with $\gamma = 10$. We also varied the rate of change in which the ratio of negative-mined samples from the entire batch. We show a rate of 0.1 (blue) and 0.01 (green)