SLO-NNs: Service Level Objective-Aware Neural Networks

Daniel Mendoza and Caroline Trippel
Stanford University
{dmendo,trippel}@stanford.edu

Abstract

Machine learning (ML) inference is a real-time workload that must comply with strict Service Level Objectives (SLOs), including latency and accuracy targets. Unfortunately, ensuring that SLOs are not violated in inference-serving systems is challenging due to inherent model accuracy-latency tradeoffs, SLO diversity across and within application domains, evolution of SLOs over time, unpredictable query patterns, and co-location interference. In this paper, we observe that neural networks exhibit high degrees of per-input activation sparsity during inference. . Thus, we propose *SLO-Aware Neural Networks* (SLO-NNs) which dynamically drop out nodes *per-inference query*, thereby tuning the amount of computation performed, according to specified SLO optimization targets and machine utilization. SLO-NNs achieve average speedups of $1.3 - 56.7\times$ with little to no accuracy loss (less than 0.3%). When accuracy constrained, SLO-NNs are able to serve a range of accuracy targets at low latency with the *same trained model*. When latency constrained, SLO-NNs can proactively alleviate latency degradation from co-location interference while maintaining high accuracy to meet latency constraints.

1 Introduction

Machine Learning (ML) inference supports many important application domains such as ranking and recommendation [15], finance [14], analytics [25, 42], computer vision [12, 21], healthcare [24], computer security [34], natural language processing [13], and more. Thus, ML inference is at the heart of modern web services. For example, at Amazon Web Services (AWS), machine learning inference accounts for more than 90% of infrastructure costs [1]. At Facebook, more than 200 trillion predictions and over 6 billion languages translations are made each day [28].

Unlike training which can be done offline, ML inference is a real-time workload that must comply with strict Service Level Objectives (SLOs), such as latency and accuracy targets. Unfortunately, ensuring that SLOs are not violated, complicates the design of inference-serving systems for a few key reasons outlined below.

Latency-accuracy tradeoff: First, highly accurate models often exhibit longer inference latencies than moderately accurate models, indicating a challenge in selecting a model which meets both accuracy and latency targets.

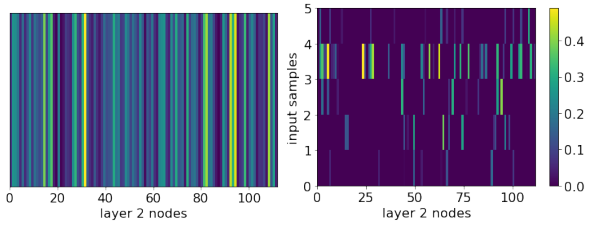

Figure 1. Left: Average per-node activation magnitudes for a 112-node hidden layer over 10,000 FMNIST input samples. Right: Per-node activations for five random inputs for the same hidden layer. SLO-NNs exploit the extreme sparsity of per-node activations on behalf of individual inputs compared to the average.

Application SLO diversity: Second, SLOs vary widely *across* and *within* application domains [20, 36], requiring many *model-variants* to satisfy diverse requirements.

Evolution of SLOs over time: Third, service providers may change SLOs over time. Suitable model-variants today may fail to satisfy SLOs in the future when combined with new compute infrastructure or deployed in a new execution environment [3].

Co-location interference: Fourth, inference models are typically co-located on *worker* machines to improve resource utilization and reduce operating costs [29, 33, 36, 40]. Unfortunately, model co-location introduces the opportunity for model interference, which can degrade inference latency and cause SLO violations.

Volatile query patterns: Fifth, query arrival patterns are difficult to predict, and query rates fluctuate over time [17]. Query load impacts queuing times as well as intermittent co-location latency degradation.

Given the challenges above, modern model serving systems [9, 36] are burdened by training and managing many models to meet diverse SLOs under varying query loads, where switching models is prohibitively time consuming.

This paper presents *SLO-Aware Neural Networks* (SLO-NNs)—neural networks which are able to dynamically adapt inference computation on a *per-input* basis to meet SLO optimization targets, even in the presence of co-location interference. Our insight, as suggested by prior work, is that per-input node activations in neural networks with ReLU activations are often sparse [2, 5, 6, 37]. Thus, full-network

inference accuracy can in theory be achieved at lower latency by using only a *subset* of the network's nodes. Fig. 1 illustrates this observation using a neuron pruned [4, 30] model trained on the FMNIST data set [39]. The figure shows that individual inputs exhibit extreme sparsity in the nodes they activate (left), despite node activations appearing dense when averaged across 10,000 input samples (right).

SLO-NNs leverage the above insight to optimize inference for SLOs by selectively dropping out nodes at inference time on a per-input basis, *avoiding computations for these nodes altogether*. Given a trained neural network (SLO-NNs place no restrictions on model training or architecture), SLO-NNs deploy a *Node Activator* at each layer (see Fig. 2) that dynamically selects which node activations to compute for a given inference query.

SLO-NN Node Activators *learn* the relative importance of nodes for groups of similar inputs. Given node importance information along with SLO optimization targets, machine utilization information, and query data features, Node activators selectively drop out nodes from inference computations. In this way, SLO-NNs can simultaneously serve a variety of SLOs with just a single model. We summarize our contributions as follows:

SLO-NNs for SLO- and interference-aware inference: We propose SLO-NNs, which to our knowledge, represent the first generic framework for dynamic dropout at inference with no restrictions on the model architecture or model training.

SLO-NN case study: We demonstrate the efficacy of SLO-NNs across five neural network architectures and datasets: FMNIST [39], FMA [10], Wiki10 [43], AmazonCat-13K [32], and Delicious-200K [38]. SLO-NNs achieve average speedups ranging $1.3 - 56.7\times$ with zero or negligible accuracy difference (less than 0.3%) compared to the original neural network.

2 SLO-Aware Neural Networks

Fig. 2 illustrates the SLO-NN architecture. SLO-NNs dynamically optimize neural network inference computations on a per-query basis given (1) an SLO optimization target and (2) information about the machine utilization of the worker machine on which it is running. Our implementation of SLO-NNs supports two SLO optimization targets: *Accuracy-Constrained Latency-Optimized* (ACLO) and *Latency-Constrained Accuracy-Optimized* (LCAO). ACLO can be used to maximize query throughput and minimize co-location interference on behalf of a particular model. On the other hand, LCAO can leverage information about current machine utilization, to adapt to changing co-location interference or query loads without violating latency SLOs.

2.1 General Framework

In this section, we present some terminology which we use to define *SLO-Aware Neural Networks* and further describe the mechanics of the ACLO and LCAO optimization targets.

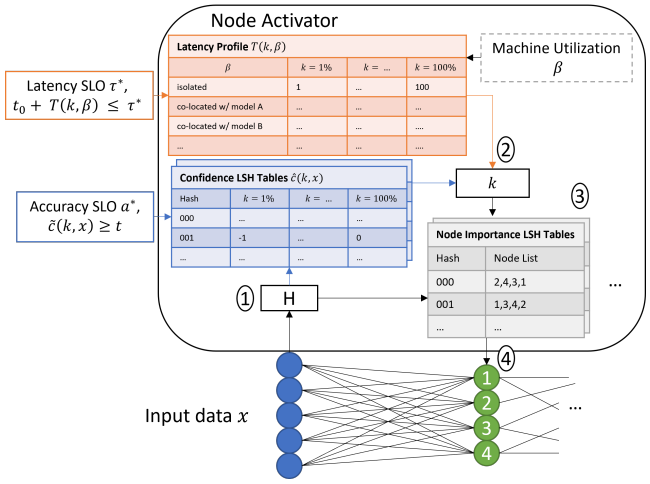

Figure 2. SLO-NN Architecture. (1) SLO-NN inputs are hashed (2) A particular percentage of nodes, k, is determined based on an SLO optimization target (e.g., ACLO or LCAO) and worker machine utilization information. (3) Extract a list of nodes, sorted by importance from per-layer Node Importance LSH Tables. (4) The top k% nodes in the sorted list are computed per layer.

To begin with, an inference *query* consists of: (1) an accuracy target a^*, (2) a latency target τ^*, and (3) input features x. For each query, SLO-NNs dynamically tune computation according to the network's *confidence* as well as supplied *accuracy* and *latency* constraints. We describe these three parameters as follows.

Confidence: Let $c(k,x)$ represent a neural network's confidence when performing inference on data input x with the top k% of nodes *at each layer* computed (not dropped out). In SLO-NNs, the top k% nodes are selected with respect to per-layer lists of nodes that are ranked by importance, where importance corresponds to the expected activation magnitude. §3.2 describes how these ranked lists are constructed. For a given input, we quantify confidence as the negative distance between the prediction of the full neural network, $\hat{y}$, and the prediction of the neural network with the top k% of nodes computed, $\hat{y}_k$.

$$c(k,x) = -distance(\hat{y}, \hat{y}_k) \quad (1)$$

The *distance* function for computing $c(k,x)$ is selected based on the prediction task. For instance, we chose the *distance* function to be cross-entropy for classification tasks.

Accuracy: Let a_t be the measured accuracy on a held-out set where a neural network predicted every data input x with confidence $c(k,x) \geq t$; t is some confidence threshold.

Latency: Let $T(k,\beta)$ denote the latency of the neural network when k% of the nodes are activated given the state of the execution environment β. β represents the machine utilization on behalf of co-located workloads which may cause interference and increase inference latency. Further,

let t_0 be the total time spent processing the query outside of inference including queuing delays and feature extraction. Thus $t_0 + T(k, \beta)$ denotes the total time spent processing the query. Note that t_0 may vary from query to query (e.g. due to varying queuing delays).

Definition 1 (SLO-Aware Neural Network)**.** *Given accuracy target a^*, latency constraint τ^*, an SLO-NN chooses k such that: $a_{c(k,x)} \geq a^*$ and $t_0 + T(k, \beta) \leq \tau^*$. If these constraints cannot be met, then the neural network cannot fulfill the SLOs.*

Note that there may be a range of k which may satisfy the SLOs. This range represents the degree of freedom for which the SLO-NN can adapt to each query. In this paper, we consider SLO-NNs which optimize for one SLO (accuracy or latency), while constrained by another—the ACLO and LCAO optimization targets.

2.2 Accuracy-Constrained Latency-Optimized

SLO-NNs can be used to minimize inference latency $T(k, \beta)$ for each input query while satisfying an accuracy target a^*. $t_0 + T(k, \beta)$ monotonically decreases with k when t_0 is held constant since decreasing k can only decrease or not affect $T(k, \beta)$. Therefore, minimizing latency $t_0 + T(k, \beta)$ is equivalent to minimizing k. Thus the optimization problem, corresponding to our ACLO SLO optimization target, is expressed as follows:

$$\begin{aligned} \min_{k} \quad & k \\ \text{s.t.} \quad & a_{c(k,x)} \geq a^* \end{aligned} \tag{2}$$

Note that confidence depends on x, indicating that for "easy" inputs SLO-NNs can drop out significantly more computation. Thus, a large high-accuracy model can adapt to serve a range of accuracy constraints where more lenient accuracy targets likely correspond to significantly lower inference time.

2.3 Latency-Constrained Accuracy-Optimized

SLO-NNs can also be used to optimize inference accuracy $a_{c(k,x)}$ per input query while satisfying a latency target τ^*. In all of our experiments (§5), we observe that as k increases, $a_{c(k,x)}$ either monotonically increases or approaches the accuracy of the full neural network.

Since the percentage of computed nodes, k, is an indicator of both latency (lower k implies lower latency) and accuracy (higher k implies accuracy closer to the full neural network), we can leverage a cost function based on k to optimize inference accuracy. Namely, we can maximize k, and thus inference accuracy $a_{c(k,x)}$, such that the latency constraint τ^* is satisfied just-in-time. Therefore, the optimization problem, corresponding to our LCAO optimization target, is expressed as follows:

$$\begin{aligned} \max_{k} \quad & k \\ \text{s.t.} \quad & t_0 + T(k, \beta) \leq \tau^* \end{aligned} \tag{3}$$

Dataset	Train size	Test size	Feature dim	Label dim	Architecture
FMNIST	60,000	10,000	782	10	112-112
FMA	84,353	22,221	518	161	64
Wiki10	14,146	6,616	101,938	30,938	128
AmazonCat13k	1,186,239	306,782	203,883	13,330	128
Delicious200k	196,606	100,095	782,585	196,606	128

Table 1. SLO-NN-evaluated datasets and model architectures.

One benefit of SLO-NNs which deploy the LCAO optimization is that they are able to serve a wide range of latency SLO targets with a single model. Furthermore, LCAO SLO-NNs can adapt to intermittent co-location interference or bursty and varying query loads by using a latency profile to predict inference latency and dynamically adjusting neural network compute accordingly. In doing so, LCAO SLO-NNs can avoid latency SLO violations where a standard neural network would not be able to satisfy.

3 SLO-NN Node Activators

In this section, we describe SLO-NN *Node Activators*, which select nodes to be dropped out for a given inference request and SLO optimization target (ALCO versus LCAO). In this work, the Node Activator is based on Locality Sensitive hashing (LSH) due to its low overhead. In future work we plan to investigate other ranking schemes.

3.1 Locality Sensitive Hashing

LSH was originally proposed as a sub-linear time approximate nearest neighbors search strategy [23]. The technique features a family of hash functions with the property that similar input objects have a higher probability of colliding (post-hash) than non-similar ones given some similarity measure. In particular, a sufficient condition for a family of hash functions $\mathcal{H}$ to be considered an *LSH family* is that for $h \in \mathcal{H}$, the post-hash collision probability $Pr_{\mathcal{H}}(h(x) = h(y))$ monotonically increases with the similarity of x and y.

The classic (K, L) LSH algorithm has two phases [23]. In the *pre-processing phase*, L hash tables are constructed. For a given table, keys are computed by concatenating the outputs of K LSH hash functions. Data elements are then stored into buckets of the L hash tables according to their computed keys. In the *query phase*, given some input query, keys for each hash table are computed and used to fetch all data elements from each of the L corresponding buckets (one bucket per table). SLO-NNs leverage LSH to efficiently identify similar data inputs and further associate with them *node importance* and *confidence* information.

3.2 Node Activator Training

LSH provides a low overhead mechanism for associating similar data samples with each other—similar inputs collide in LSH hash tables. SLO-NNs further require associating each

group of similar inputs with (1) a ranked list of nodes according to their importance for making accurate predictions (i.e., as close to the full neural network as possible) and (2) a confidence score which encodes their "hardness". As illustrated in Fig. 2, the Node Activator leverages two types of LSH hash tables for storing each association type—the *Node Importance* (gray) and *Confidence* (blue) tables, respectively. The hash tables which make up the Node Activator are populated with the help of an unsupervised training step, which can be performed pre- or post-deployment of the SLO-NN.

Node Importance LSH Tables In an SLO-NN, a set of Node Importance LSH tables (gray tables in Fig. 2) are placed at each layer. Node Importance tables map a set of similar inputs (which collide in the same table entry) to a list of nodes, ranked according to their importance in facilitating accurate inference. For a given input, ranking nodes in a specific layer according to their importance corresponds to ranking them according to their activation magnitude.

Algorithm 1 describes the unsupervised training procedure for a set of L Node Importance LSH tables at some layer l in a SLO-NN. Inputs to the training procedure include an input set of data features, $Input_l$, and LSH parameters, L (number of tables) and K (key size). The dataset $Input_l$ is representative of the data which is supplied as input to layer l of the neural network during inference. Training is initialized by first generating $K \times L$ hash functions (i.e, K hash functions per table), according to some LSH hash family of choice (§3.4). Next, for each of the L tables, the algorithm computes the K corresponding hash functions over all inputs $x \in Input_l$, thereby mapping each input x to a particular bucket in each table. For inputs which map to the same bucket in some table, their per-node activations (at layer l where the table is positioned) are summed; the result is an activation sum associated with each node. Finally, nodes are sorted according to activation sums (highest to lowest).

Confidence LSH Tables As discussed in §2.1, $c(k, x)$ is a measure of confidence the neural network exhibits on data input x when the top k% of nodes are activated at each layer. Intuitively, a given neural network will exhibit a similar level of prediction confidence when supplied with similar input features. Thus, SLO-NNs leverage a set of Confidence LSH tables (blue tables in Fig. 2) to associate groups of similar inputs with a confidence score.

Let $\hat{c}(k, x)$ be an estimate of confidence $c(k, x)$ such that

$$\hat{c}(k, x) = aggregation(\{c(k, x') | x' \in LSH(x)\}) \quad (4)$$

Where *aggregation* is a function which aggregates the confidences $c(k, x')$ of the data inputs that are hashed to the same bucket during the training procedure. In our evaluation (§5), the *aggregation* function is the arithmetic mean, which relies on the intuition that nearby data inputs are likely to exhibit similar confidence on average. To associate a confidence threshold t with an accuracy metric a_t, we test on the held-out validation set where we predict every data point x with confidence $\hat{c}(k, x) \geq t$.

Algorithm 1 Training Node List LSH at layer l

Input: $Input_l$, L, K
Output: LSH

```
h := GenerateHashFamily(K,L)
NodeScore := Map
Rank := Map
for x ∈ Input_l do
    for b = 1 : L do
        key := (h_1^b(x), h_2^b(x), ..., h_K^b(x))
        Activation = ForwardProp_l(x)
        NodeScore[b][key] += Activation
    end for
end for
for b = 1 : L do
    for key ∈ NodeScore[b] do
        Rank[b][key] := argsort(NodeScore[b][key])
    end for
end for
LSH := (Rank,h)
```

Interference-Aware Latency Estimation For a given SLO-NN, inference latency $T(k, \beta)$ may be profiled or predicted apriori for varying co-location scenarios β and varying values of k. In our current experiments (§5), we leverage latency profiles while in future work we plan to additionally train latency predictors which can be subsequently used to predict a full latency co-location profile for a given workload configuration [11, 33].

The Node Activator uses the estimated latencies when serving the model to anticipate the inference latency associated with a particular degree of dropout (i.e., a particular value of k) and co-location interference.

3.3 SLO-Aware Forward Pass

Fig. 2 illustrates the forward pass of SLO-NNs. Node Confidence LSH tables and the Latency Profile table are queried *once per inference request* to select the percentage of nodes, k, to activate for a given SLO optimization target. For ACLO, only the Node Confidence LSH tables are queried; for LCAO, only the Latency Profile table is accessed. Node Importance LSH tables are queried *once per layer* to obtain sorted lists of nodes from which the top k can be selected.

3.4 FreeHash: A Novel LSH Hash Function

We observe that deploying LSH in the context of neural networks gives us access to a unique LSH hash family *for free*. Specifically, neural network weights represent vectors that have been *trained* to preserve the similarity between data inputs. Thus, SLO-NNs derive hash keys by computing dot products between input data and a sub-sample of the neural

network weights. We call this LSH hash family FreeHash; for a given slo-nn layer l, we define FreeHash as follows.

Definition 2 (FreeHash). *Let w_i and b_i correspond to the weights and bias of some randomly selected node i in layer l of the slo-nn. We hash an input x to layer l as:*

$$FreeHash_i(x) = sign(w_i^T x + b_i) \tag{5}$$

For ReLU Layers, free hash satisfies the LSH family hash condition of §3.1.

When using FreeHash to construct hash functions for an slo-nn LSH table, a set of $K \times L$ (K keys per L tables) nodes (and their corresponding weights and biases) must be selected. Theoretically, these nodes could be selected at random from the relevant layer. However, this approach may result in disimilar data inputs, which produce sparse activations for a given neural network layer, being misclassified as similar. To address this issue, slo-nns sample node weights and biases for FreeHash with probability proportional to the variance of the nodes' activations across the training set for the LSH. FreeHash leverages computations that are already required to perform full neural network inference. Thus, in the worst case, where all nodes in an slo-nn are computed, no extra computation is required compared to the full neural network. Furthermore, the Node Activator is a lightweight data structure as it stores sparse tables with lists of node references. In our evaluation, Node Activator storage accounted for less than 10% of the neural network for all models.

4 Methodology

Model Architectures and Datasets We evaluate slo-nns on five datasets [10, 32, 38, 39, 43], summarized in Table 1. FMNIST is a multi-classification dataset of fashion products. Kitsune is an anomaly detection dataset for detecting network attacks via packet statistics. FMA is a music analysis dataset containing 106,574 tracks from 16,341 artists and 14,854 albums, arranged in a hierarchical taxonomy of 161 genres. Wiki10 dataset is a collection of Wikipedia articles with associated user-defined tags formed from over 2 million Wikipedia articles. AmazonCat-13K is a product-to-product recommendation dataset. Delicious-200K dataset is generated from a vast corpus of almost 150 million bookmarks from Social Bookmarking Systems. Amazon-670K dataset is a product to product recommendation dataset.

Model Pruning We statically neuron prune the baseline models architectures to ensure that each is reasonably sized for its corresponding dataset [4, 30]. For the FMNIST and FMA models, we applied neuron model pruning [4, 30] prior to transforming them into slo-nns. slo-nns for Wiki10, Delicious200k, and Amazoncat13k, feature a Node Activator at the output layer only; these models are not pruned since pruning cannot impact the output layer.

Inference Platform Our evaluation is conducted on a server equipped with two 64-core Intel Xeon Gold 6226R CPUs. Most inference serving systems employ server/edge CPUs due to their abundance and cost-efficiency in comparison to GPUs [20, 35]. We plan to evaluate on GPUs in future work.

slo-nn Implementation Our implementation of slo-nns use NumPy 1.19.5 [19]. Numba 0.53.1 [27] is used to compile into fast machine code. Fig. 3 compares the run times of the activating the entire neural network with PyTorch and the slo-nn. All bars represent full forward pass (i.e., all nodes computed) median latencies over 100 runs of the evaluated neural networks. *PyTorch* bars represent the inference latency of out-of-the-box PyTorch. Fig. 3 demonstrates that slo-nns exhibit low overhead even if no computation is dropped out. In this paper, we focus on latency-critical online inference where batch inference is often too slow (e.g., due to queuing delays [35]). Many real-time inference systems implement a batch size of 1 [8, 31, 35] and most are restricted to a small batch size [20, 35]. We plan to investigate batch inference in future work.

5 Preliminary Results

5.1 slo-nns vs. existing dropout frameworks

Fig. 4 showcases the ability of slo-nns to select the *most important* nodes—those that optimize accuracy—to serve inference queries when performing dropout. The x-axes represent the number of nodes computed during an inference query. The y-axes report inference accuracy, averaged across all test set samples. Fig. 4 compares three dropout schemes—slo-nn, Mongoose, and *random*—to the baseline accuracy of the full neural network (where all nodes are computed). Mongoose is the most similar prior work to slo-nns which proposes LSH-based dropout at training [6].

Fig. 4 shows that for each dataset, slo-nns significantly outperform Mongoose and *random* dropout. Given the same number of active nodes, the slo-nn is up to 50% more accurate than Mongoose. We expect this discrepancy is due to the differing LSH training procedures of slo-nns and Mongoose. Specifically, Mongoose never realizes the entire activation of a data input in order to achieve faster forward propagation and gradient update, given their goal of *training*. Mongoose only considers *subsets* of node activations when training its LSH, which leads to imprecise node importance ranks. Training can tolerate and adapt to inaccurate node importance ranks since the inaccuracy emulates random adaptive dropout, whereas inference requires node importance lists to have higher degrees of precision. slo-nns leverage complete node activations during LSH training to establish node importance, which results in better accuracy.

slo-nns quickly reach and sustain full neural network accuracy with as few as 0.01% of the total nodes and as many as 94%. The point at which maximum accuracy is achieved is marked with a yellow dot in each graph. Interestingly,

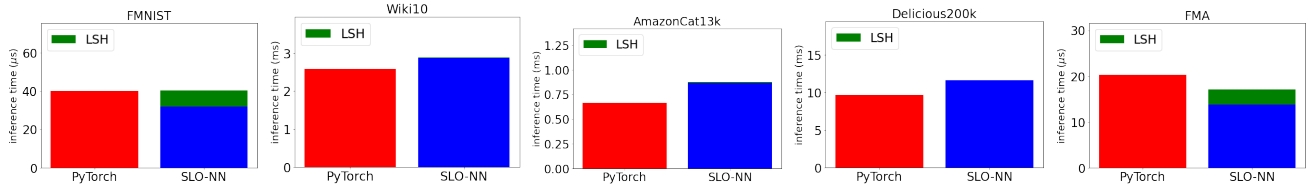

Figure 3. Performance breakdown of full neural network inference latency (all nodes computed) for a baseline *PyTorch* implementation and an SLO-NN. The SLO-NN is using FREEHASH. The time spent in the LSH includes the time spent computing hash functions. Our framework performs similarly to PyTorch, demonstrating the practicality of our SLO-NN approach.

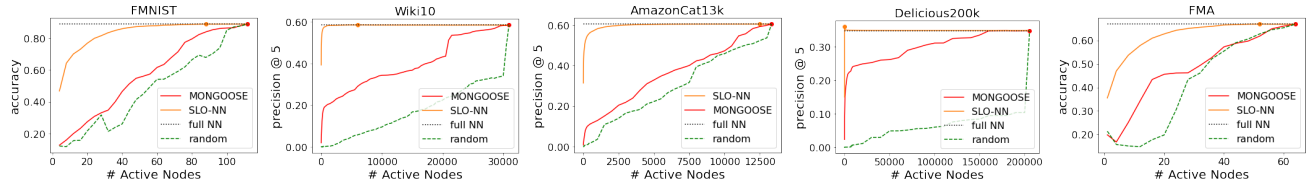

Figure 4. Number of computed nodes per layer vs. accuracy. Yellow dots indicate where SLO-NNs first reach maximum accuracy.

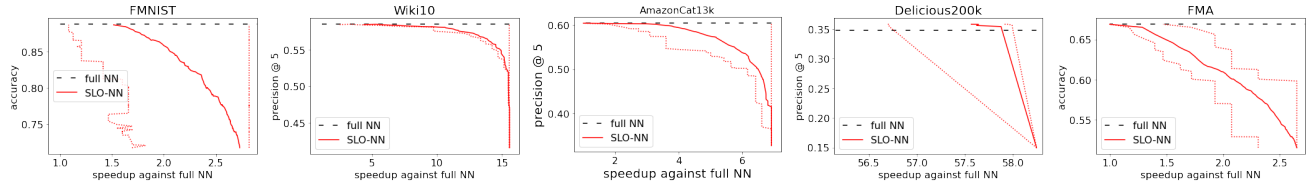

Figure 5. Solid (resp. dotted) lines indicate the average (resp. min/max) speedup against the full neural network. Speedup depends on the number of computed nodes in the neural network for each data input.

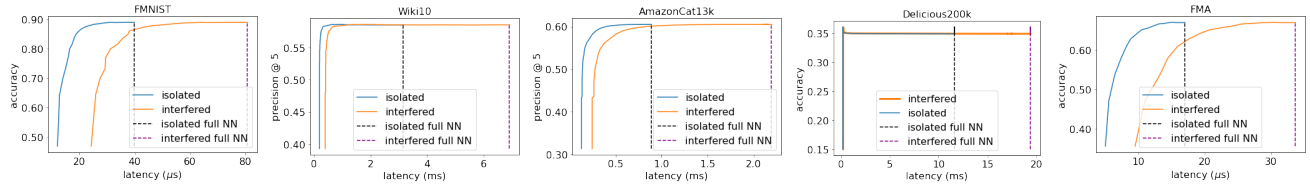

Figure 6. Accuracy-latency tradeoff for LCAO SLO-NNs. The *isolated* curves show the performance of various SLO-NNs when running alone on the worker machine. The *interfered* curves show performance when a two instances of the same SLO-NN are co-located and queried simultaneously.

for Delicious200k, SLO-NNs achieve higher accuracy than the original neural network when computing only 0.01% of the nodes. Its accuracy converges to that of the full neural network with more computed nodes. Overall, SLO-NNs effectively identify and selectively compute the most important nodes in a neural network on a *per-input basis.*

5.2 SLO-NNs with an ACLO Optimization Target

The ACLO optimization target directs an SLO-NN to minimize inference latency given an accuracy SLO target. For a given input example ACLO involves minimizing the number of computed nodes given an accuracy constraint. In theory, "easy" inputs can be computed faster than "hard" ones. Fig. 5 compares inference speedup of SLO-NNs over a full neural network (x-axes) with the achieved accuracy (y-axis). Specifically, these experiments are the result of supplying SLO-NNs with an accuracy target and asking it to minimize inference latency (i.e., minimize the number of computed nodes) for each input example in the test set.

Fig. 5 shows the minimum (left curve), average (middle curve), and maximum (right curve) achieved by SLO-NNs at various accuracy targets. Overall, SLO-NNs exhibit a high range of inference speedup. For example, for a high accuracy target (within 0.3% accuracy of the full neural network), SLO-NNs exhibit speedups of $1.1 - 2.8\times$ for FMNIST, $8.4 - 15.6\times$

for Wiki10, 1.8 − 6.9× for AmazonCat13k, 56.7 − 57.9× for Delicious200k, and 1.2 − 1.7× for FMA.

Overall, SLO-NNs are able to achieve significant latency improvements while retaining accuracy.

5.3 SLO-NNs with an LCAO Optimization Target

The LCAO optimization target directs an SLO-NN to minimize dropout (so as to optimize accuracy) given a latency target. Furthermore, LCAO takes into consideration information about current machine utilization and pre-computed Latency Profiles (Fig. 2) to account for the effects of intermittent co-location interference on latency. Fig. 6 compares inference latency of SLO-NNs (x-axes) with inference accuracy (y-axes) when operating under the LCAO optimization target. The dotted black and purple vertical lines show full model neural network inference latency when inference is run in isolation versus when it experiences co-location interference, respectively. Here, our co-location interference scenario considers a second co-located copy of the same inference model, serving back-to-back inference requests. The blue/orange curves illustrate the accuracy latency tradeoff for SLO-NNs in isolated/interfered execution scenarios.

Notably, Fig. 6 demonstrates that zero latency degradation (with respect to full network latency) can be achieved by SLO-NNs *when interfered*. Wiki10, Delicious200k, FMNIST, AmazonCat13k, and FMA exhibit 0%, 0%, 2.1%, 0.3%, and 4.1% accuracy drop compared to the full neural network, respectively, while retaining the latency of un-interfered inference. Overall, Fig. 6 illustrates that the LCAO SLO-NN is able to simultaneously meet a range of latency SLOs even under intermittent co-location interference while maintaining high accuracy.

6 Related Work

Inference Serving Systems Most modern model serving systems (e.g., Clipper [9], Amazon Sagemaker, Microsoft AzureML, INFaaS [36], Horus [40], Perseus [29]) treat ML inference as a black box. These approaches must train and manage many models to meet diverse SLOs under varying query loads. As observed by [29], model load times are often significantly longer (up to 100× slower) than inference; thus, switching models online is likely to lead to latency SLO violations. SLO-NNs circumvent these issues by managing a single model which can dynamically adapt to a diversity of SLOs, changing query load, and co-location interference.

ML Inference Optimizations Model pruning is a popular technique employed to compress a neural network by permanently removing connections between neurons or the neurons themselves and often incurs accuracy loss [4, 18, 30]. Static model pruning is oblivious to the notion that some input queries are "easy" and thus cannot leverage per query activation sparsity for inference acceleration. SLO-NNs are complementary to static model pruning as the framework can take as input a statically pruned model.

LSH for Neural Networks Reformer [26] propose an LSH-based transformer model where they replace the attention layers of a transformer model with LSH tables to produce a more compressed transformer model. Their technique is only applicable to transformer models, and requires the LSH-based transformer model to be trained with the LSH tables. Mongoose [6] applies an LSH-based dropout-at-training scheme to speed up neural network training. The technique is an extension of prior work which maps adaptive dropout as a maximum inner product search problem [7, 37]. Mongoose only considers partial node activation when training its LSH which leads to inaccurate node importance ranks. SLO-NNs take into account the full node activation at LSH training which, as we demonstrate, leads to significant improved performance.

Dynamic Neural Networks Recent work has proposed dynamic neural network architectures which exhibit conditional computation based on SLO targets [16, 22, 41]. However, these designs restrict either the model architecture or training procedure, and may not achieve state-of-the-art accuracy [16, 22, 41]. In contrast, SLO-NNs make no such restrictions as it provides a general framework to develop facilities for conditional computations *at inference*.

7 Future Work and Conclusions

In this paper, we focus on latency-critical online inference for Multi-Layer Perceptrons networks. In ongoing work, we are investigating the application SLO-NNs to other architectures, such as convolutional neural networks. We also plan to investigate batch inference for SLO-NNs. Many solutions to batch inference with SLO-NNs are possible, such as using LSH to cluster batch inputs into parallel micro-batches or dividing the selected nodes across inputs according to a weighting scheme that accounts for input difficulty. Scheduling with SLO-NN batch inference is difficult as it requires making adaptive batch size decisions under varying co-location interference, queuing delay, and query load.

Our current experiments evaluate SLO-NNs running on CPUs with a maximum of two co-located models. We plan to scale up our evaluation by adding more complex query patterns, co-location configurations, and hardware platforms. Along these lines, we are also interested in understanding how our SLO-NNs can be to designed to accelerate inference under shifting query data distributions by employing lightweight online updates to the Node Activator.

Finally, while the Node Activator in SLO-NNs is based on LSH, and we plan to study other node ranking mechanisms.

In summary, we present SLO-NNs as a type of neural network which can dynamically adapt inference computation according to SLO optimization targets and co-location interference on a *per-query basis*. SLO-NNs place no restrictions on

training, enable a variety of SLOs to be met with just a single model, and exhibit benefits beyond what can be achieved by static model pruning techniques.

References

[1] AWS. 2019. Deliver high performance ML inference with AWS Inferentia. https://d1.awsstatic.com/events/reinvent/2019/REPEAT_1_Deliver_high_performance_ML_inference_with_AWS_Inferentia_CMP324-R1.pdf.

[2] Lei Jimmy Ba and Brendan Frey. 2013. Adaptive Dropout for Training Deep Neural Networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 3084–3092.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems* (1st ed.). O'Reilly Media, Inc.

[4] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the State of Neural Network Pruning? arXiv:2003.03033 [cs.LG]

[5] Guy Blanc and Steffen Rendle. 2018. Adaptive Sampled Softmax with Kernel Based Sampling. arXiv:1712.00527 [cs.LG]

[6] Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Christopher Ré. 2021. MONGOOSE: A Learnable LSH Framework for Efficient Neural Network Training. In *ICLR*.

[7] Beidi Chen, Tharun Medini, and Anshumali Shrivastava. 2019. SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems. *CoRR* abs/1903.03129 (2019). arXiv:1903.03129 http://arxiv.org/abs/1903.03129

[8] Marshall Choy. 2020. Accelerating the Modern Machine Learning Workhorse: Recommendation Inference. https://sambanova.ai/blog/accelerating-the-modern-ml-workhorse-recommendation-inference/

[9] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. arXiv:1612.03079 [cs.DC]

[10] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. 2017. FMA: A Dataset For Music Analysis. arXiv:1612.01840 [cs.SD]

[11] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. *SIGPLAN Not.* 49, 4 (Feb. 2014), 127–144. https://doi.org/10.1145/2644865.2541941

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[14] Matthew F. Dixon, Igor Halperin, and Paul Bilokon. 2020. *Machine Learning in Finance*. Springer International Publishing. https://doi.org/10.1007/978-3-030-41068-1

[15] Facebook Research. 2021. An implementation of a deep learning recommendation model (DLRM). https://github.com/facebookresearch/dlrm.

[16] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. 2019. Dynamic Channel Pruning: Feature Boosting and Suppression. arXiv:1810.05331 [cs.CV]

[17] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture*.

[18] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. arXiv:1506.02626 [cs.NE]

[19] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[20] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[22] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. 2019. Channel Gating Neural Networks. arXiv:1805.12549 [cs.LG]

[23] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 604–613. https://doi.org/10.1145/276698.276876

[24] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. 2017. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology* 2, 4 (2017), 230–243. https://doi.org/10.1136/svn-2017-000101 arXiv:https://svn.bmj.com/content/2/4/230.full.pdf

[25] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*.

[26] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. arXiv:2001.04451 [cs.LG]

[27] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 1–6.

[28] Kevin Lee, Vijay Rao, and William Arnold. 2019. Accelerating Facebook's infrastructure with application-specific hardware. https://engineering.fb.com/2019/03/14/data-center-engineering/accelerating-infrastructure/.

[29] Matthew LeMay, Shijian Li, and Tian Guo. 2020. Perseus: Characterizing Performance and Cost of Multi-Tenant Serving for CNN Models. arXiv:1912.02322 [cs.DC]

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning Filters for Efficient ConvNets. arXiv:1608.08710 [cs.CV]

[31] machynist and kippinitreal. 2020. How We Scaled Bert To Serve 1+ Billion Daily Requests on CPUs. https://blog.roblox.com/2020/05/scaled-bert-serve-1-billion-daily-requests-cpus/

[32] Julian McAuley and Jure Leskovec. 2013. Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text. In *Proceedings of the 7th ACM Conference on Recommender Systems* (Hong Kong, China) *(RecSys '13)*. Association for Computing Machinery, New York, NY, USA, 165–172. https://doi.org/10.1145/2507157.2507163

[33] Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Interference-Aware Scheduling for Inference Serving. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (Online, United Kingdom) *(EuroMLSys '21)*. Association for Computing Machinery, New York, NY, USA, 80–88. https://doi.org/10.1145/3437984.3458837

[34] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. 2020. Shredder: Learning Noise Distributions to Protect Inference Privacy. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[35] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, Juan Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. arXiv:1811.09886 [cs.LG]

[36] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*.

[37] Ryan Spring and Anshumali Shrivastava. 2016. Scalable and Sustainable Deep Learning via Randomized Hashing. arXiv:1602.08194 [stat.ML]

[38] Robert Wetzker, Carsten Zimmermann, and Christian Bauckhage. 2008. Analyzing Social Bookmarking Systems: A del.icio.us Cookbook. In *Mining Social Data (MSoDa) Workshop Proceedings*. ECAI 2008, 26–30. http://robertwetzker.com/wp-content/uploads/2008/06/wetzker_delicious_ecai2008_final.pdf

[39] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747 [cs.LG]

[40] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. 2020. Horus: An Interference-Aware Resource Manager for Deep Learning Systems. In *Algorithms and Architectures for Parallel Processing*, Meikang Qiu (Ed.). Springer International Publishing, Cham, 492–508.

[41] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2018. Slimmable Neural Networks. arXiv:1812.08928 [cs.CV]

[42] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[43] Arkaitz Zubiaga. 2012. Enhancing Navigation on Wikipedia with Social Tags. arXiv:1202.5469 [cs.IR]