CS230

Predicting Latency of Neural Network Inference

Daniel M. Mendoza
Department of Electrical Engineering
Stanford University
dmendo@stanford.edu

Sijin Wang
Department of Civil Engineering
Stanford University
neversjw@stanford.edu

1 Introduction and Motivation

Many deep learning applications are latency critical, in which the inference latency must be within the bounds specified by a service level objective. Thus, the inference application developer must design or select neural networks (NNs) that can satisfy the service level objective. However, the application developer often does not have the ability to know inference latency of the application until after the NN is deployed to the inference serving system. Inference latency depends upon a multitude of factors including the hardware platform, software environment, and network architecture. In this project, we propose a technique to predict the inference latency of a NN. We implement a prediction algorithm that utilizes graph embedding of the NN architecture to predict the inference latency of the NN on CPUs and GPUs.

Further, with the development of accelerators for machine learning inference comes the challenge to develop networks that fully utilize the capabilities of the accelerators. Thus, inference serving systems often co-locate machine learning applications to increase utilization of the machines. However, co-locating applications often leads to latency-degradation due to intermittent resource contention and interference which may lead to service level objective violations. Thus we must anticipate if a co-location configuration can satisfy the service level objectives by predicting the latency overhead. We implemented an approach to predict the latency degradation due to co-locating NNs on CPUs and GPUs using graph embedding of the inference models.

2 Related work and background

[1] studied latency prediction of NNs on TPUs using graph embedding. However, we differentiate our study by investigating techniques for predicting latency on CPUs and GPUs. Application latency prediction and latency degradation prediction is a well-studied subject in projects such as [2,3,4]. However, these latency prediction techniques are for general workloads, and in this project we specifically target NN inference.

3 Dataset and Features

In this section we detail our datasets for inference latency prediction and co-location overhead prediction. We also describe the graph feature representation we use to train the latency predictor and overhead predictor. We implemented data collections mechanism to profile an inference model in which we obtain its graph feature representation and inference latency or co-locating latency degradation.

3.1 Graph Features

To train a predictor we must obtain features that are related to inference latency. We use the graph representation of each NN architecture as features. The recorded graph features of each example are the attribute list and adjacency matrix of each NN at the Keras operation level. For instance, a standard NN with 3 layers has 3 nodes in its corresponding feature representation. One node corresponds to the input layer (InputLayer in Keras) and the subsequent nodes represent the sequentially fully connected layers (Dense in Keras). The adjacency matrix expresses how the nodes are connected and the attribute list contains the operation type (e.g. Dense or Conv2D) and number of weights for the operation. The operations types are represented by a one hot encoding.

3.2 Randomly Generated Networks for Inference Latency

In this section we describe the dataset used for our experiments in predicting inference latency. We randomly generated 320,000 standard NNs with depth from 3 to 10 and hidden units per layer sampled in base 2 scale from 0 to 14 (min=1,max=16384). We also randomly generated 100,000 plain CNNs with depth from 3 to 10 and channel size in base 2 scale from 0 to 10. The total dataset size is 420,000. When profiling the latency of each network, we take the ground truth label to be the median latency out of 20 runs. We profile the inference models both on a 16 core CPU and a V100 GPU.

3.3 Keras Pre-trained Models for Co-locating Latency

In this section we elaborate on the dataset used for predicing co-location latency overhead. We collect a dataset for pre-trained models provided in Keras. There are a total of 14 different pre-trained models in Keras. These pre-trained models are deployed in real inference applications, and we intend to use them to evaluate our inference prediction method on practical data. To obtain the graph features of each pre-trained model, we implemented a software mechanism which processes a Keras model and returns the graphical feature representation of the model. When profiling the latency of each network, we take the ground truth label to be the median latency out of 20 runs. We have collected co-located latency for pairs and trios of co-located models on a 8 core CPU, 16 core CPU, 32 core CPU, T4 GPU, V100 GPU, P4 GPU, and P100 GPU. The total dataset size for co-locating latency is 16,856.

4 Approach

In this section, we first detail our approach to predicting inference latency. We then explain our strategy to predict latency degradation when co-locating NNs.

4.1 Predicting Inference Latency

We train a predictor using the attribute node list and adjacency matrix of the network at the operator level as input and the individual inference latency as the label for each example. Figure 1 illustrates the prediction pipeline for predicting inference latency on a NN architecture. We trained and evaluated the prediction models using a train-dev-test split. The prediction model's input is the flattened attribute list and adjacency matrix. Our baseline predictor is the linear regression model. We also implemented deep neural networks (DNNs) that input the graphical features of inference models to predict inference latency. To find a DNN architecture that performs well on latency prediction, We searched the hyper-parameter space of number of hidden layers from 1 to 20, and chose the DNN that experiences the lowest mean absolute error (MAE) in seconds on the dev set. The number of hidden units in this search was fixed to 100.

4.2 Predicting Latency Degradation Due to Co-location

We train a predictor using the attribute node lists and adjacency matrices of the co-located networks as input and the latency degradation as the label for each example. Figure 2 shows the prediction pipeline for predicting latency overhead from co-locating multiple NNs. We trained and evaluated the prediction models using a train-dev-test split. Our baseline predictor is the linear regression model. We also implemented a DNN which we show performs better than the baseline model. The

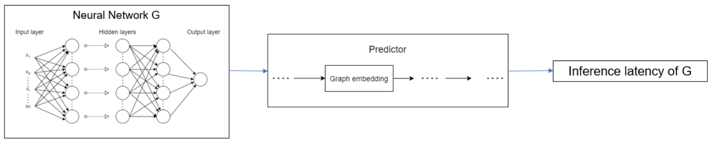


Figure 1: Inference latency prediction on a NN architecture

hyper-parameter search mechanism is the same as the strategy used in the inference latency prediction task in which we vary the number of hidden layers in the DNN and choose the model that achieves the lowest MAE on the dev set.

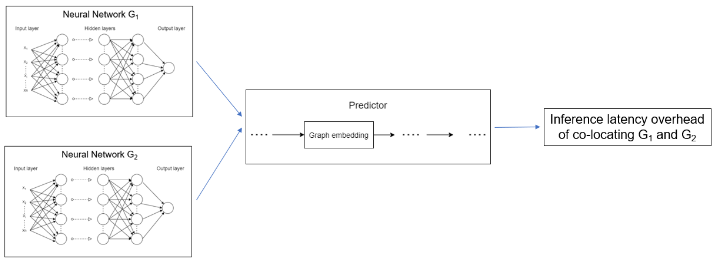


Figure 2: Inference latency degradation prediction on multiple neural network architectures.

5 Evaluation and Analysis

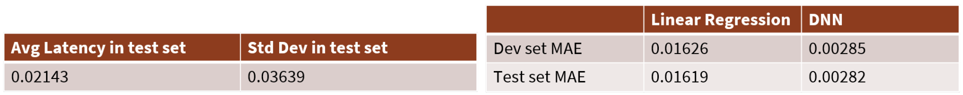
In this section, we show experimental results of our approach to predict inference latency and latency degradation due to collocation.

5.1 Experimental Results on Predicting Inference Latency

We divide this section in separate parts by NN architecture type (standard or convolutional neural network) to evaluate performance prediction on different machines and different types of NN architectures.

5.1.1 Predicting Standard Neural Network Inference Latency

In Figure 3, we show results on predicting inference latency on randomly generated standard NNs on a 16 core CPU. We split the dataset of 320,000 unique standard NNs into a training-dev-test split of 80-10-10. We show the mean and standard deviation of the true latency of the examples in the test set to show the significance of the prediction accuracy. As stated previously, our baseline is the linear regression model, which achieves a mean absolute error (MAE) of 0.01619 seconds on the test set. We trained a NN which achieves an MAE of 0.00282 seconds on the test set. The DNN has 13 hidden layers each with 100 hidden units. The mean latency in the test set is 0.02132 seconds while the DNN achieves an MAE of 0.00282 seconds and performs much better than the baseline linear regression model. This shows an instance in which the inference latency of standard NNs can be accurately predicted on a 16 core CPU using graphical features of the standard NN architecture.



Avg Latency in test set	Std Dev in test set
0.02143	0.03639

	Linear Regression	DNN
Dev set MAE	0.01626	0.00285
Test set MAE	0.01619	0.00282

Figure 3: Results are reported in units of seconds. Illustrates results for predicting inference latency of standard NNs running on a 16 core CPU.

In Figure 4, we show results on predicting inference latency of standard NNs using graph features on a V100 GPU. We split the dataset of 320,000 unique standard NNs into a training-dev-test split of 80-10-10. We trained the baseline linear regression model and a DNN model of 13 hidden layers each with 100 hidden units. The DNN performs better than the baseline with an MAE of 0.00011 seconds on the test set. However, the performance increase is not huge as the linear regression model achieves an MAE of 0.00016 seconds. This may be due to the low variance of latency in the dataset as the test set exemplifies a low standard deviation of inference latency of 0.0002 seconds. This motivates future work to produce a dataset that exemplifies more diversity of inference latency on the V100 GPU.

Avg Latency in test set	Std Dev in test set
0.00177	0.00020

	Linear Regression	DNN
Dev set MAE	0.00016	0.00011
Test set MAE	0.00016	0.00011

Figure 4: Results are reported in units of seconds. Illustrates results for predicting inference latency on standard NNs running on a V100 GPU.

5.1.2 Predicting Convolutional Neural Network Inference Latency

In Figure 5, we show results on predicting inference latency on randomly generated convolutional neural networks (CNNs) on a 16 core CPU. We split the dataset of 100,000 unique CNNs into a training-dev-test split of 70-15-15. The baseline linear regression model achieves an MAE of 0.02225 seconds on the test set. We trained a NN with 19 hidden layers each with 100 hidden units which achieves an MAE of 0.01801 seconds on the test set. In comparison to the mean latency in the test set is 0.39026 seconds, the DNN seems to predict inference latency well and performs better than the linear regression model. This illustrates that the inference latency of CNNs on 16 core CPU can be accurately predicted using graphical features of the CNN architecture.

Avg Latency in test set	Std Dev in test set
0.39026	0.10186

	Linear Regression	DNN
Dev set MAE	0.02210	0.01803
Test set MAE	0.02225	0.01801

Figure 5: Results are reported in units of seconds. Illustrates results for predicting inference latency on CNNs running on a 16 core CPU.

In Figure 6, we show results on predicting inference latency of CNNs using graph features on a V100 GPU. We split the dataset of 100,000 unique CNNs into a train-dev-test split of 70-15-15. The DNN has 5 hidden layers each with 100 hidden units. The DNN performs well as it achieves an MAE of 0.00041 seconds while the mean latency in the test set is 0.01711 seconds. Further, the DNN performs better than the linear regression model which achieves an MAE of 0.00059 seconds. The better performance from the DNN is not so significant. This may be due to the low variance of latency in the dataset as the test set exemplifies a comparably low standard deviation of inference latency of 0.00351 seconds. This motivates future work to produce a dataset of CNNs that exemplifies more diversity of inference latency on the V100 GPU.

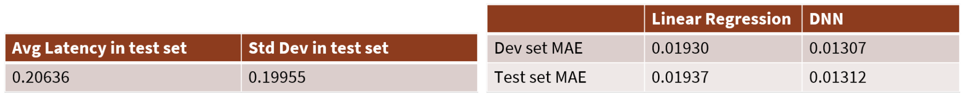
Avg Latency in test set	Std Dev in test set
0.01711	0.00351

	Linear Regression	DNN
Dev set MAE	0.00059	0.00041
Test set MAE	0.00059	0.00041

Figure 6: Results are reported in units of seconds. Illustrates results for predicting inference latency of CNNs on a V100 GPU

5.1.3 Predicting NNs and CNNs Inference Latency

In Figure 7, we show results on predicting inference latency of both standard NNs and CNNs on a 16 core CPU. We combine the datasets of standard NNs and CNNs described in previous sections to train a unified predictor to predict a diverse set of NN architectures. We split the dataset of 420,000 unique NN architectures into a training-dev-test split of 85-7.5-7.5. We balance the dev and test set such that there are an equal amount of standard NNs and CNNs in each set. The linear regression model achieves an MAE of 0.01937 seconds while the DNN achieves an MAE of 0.01312 seconds. The DNN has 13 hidden layers each with 100 hidden units. In comparison to the average latency of 0.20636 seconds in the test set, we can observe that the DNN predictor is accurately predicting the inference latency of each network. This shows that graph features provide useful features for predicting on a diverse set of NN architectures on a 16 core CPU.

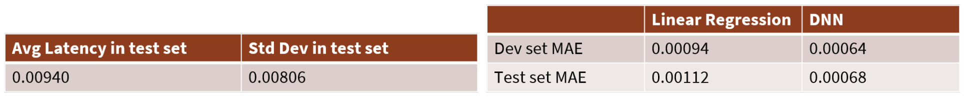


Avg Latency in test set	Std Dev in test set
0.20636	0.19955

	Linear Regression	DNN
Dev set MAE	0.01930	0.01307
Test set MAE	0.01937	0.01312

Figure 7: Results are reported in units of seconds. Illustrates results for predicting inference latency on both standard NNs and CNNs on a 16 core CPU.

In Figure 8, we show results on predicting inference latency of both standard NNs and CNNs on an V100 GPU using graph features. The dataset of 420,000 unique architectures is split into a training-dev-test split of 85-7.5-7.5. The DNN has 19 hidden layers each with 100 hidden units. Observe that the MAE of the DNN is much lower than the average latency of the test set, and thus the model shows accurate predictions on a diverse set of NN architectures on a V100 GPU. Further, the DNN outperforms the baseline linear regression model.



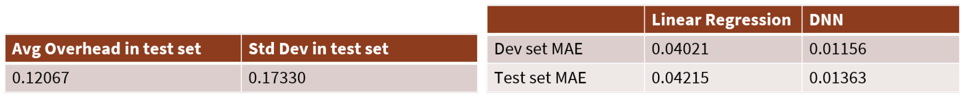
Avg Latency in test set	Std Dev in test set
0.00940	0.00806

	Linear Regression	DNN
Dev set MAE	0.00094	0.00064
Test set MAE	0.00112	0.00068

Figure 8: Results are reported in units of seconds. Illustrates results for predicting inference latency on both standard NNs and CNNs on a V100 GPU.

5.2 Experimental Results on Predicting Latency Degradation Due to Collocation

In Figure 9, we show baseline results on predicting collocating latency degradation on a 8 core CPU, 16 core CPU, 32 core CPU, V100 GPU, P100 GPU, T4 GPU, and P4 GPU. As stated previously, we use the pre-trained Keras models as training/evaluation data. We randomly split the dataset into a train-dev-test by a 70-15-15 split. We use separately trained prediction models for each machine type (7 different machine types), and in Figure 9 we show the cumulative MAE for predicting latency degradation across all machine types. Each DNN model has 10 hidden layers where the hidden layers each have 100 units. We also show the average latency degradation in the dev set for reference. Notice that the DNN performs significantly better than the linear regression model and that the DNN model MAE is much lower than the average latency degradation (overhead) in the test set. This shows that graph features can be used for accurately predicting co-locating latency overhead on a diverse set of machine types and practical NN architectures.



Avg Overhead in test set	Std Dev in test set
0.12067	0.17330

	Linear Regression	DNN
Dev set MAE	0.04021	0.01156
Test set MAE	0.04215	0.01363

Figure 9: Results are reported in units of seconds. Shows cumulative results for predicting co-locating latency degradation of pre-trained Keras models on an 8 core CPU, 16 core CPU, 32 core CPU, V100 GPU, P4 GPU, T4 GPU, and P100 GPU.

6 Conclusion

In conclusion, we showed experimental results for using graph features of NN architectures to predict inference latency and co-locating latency degradation. Overall, we have shown that graph features are useful for these predictions tasks on a diverse set of NN architectures and machine types and can provide accurate results.

7 Contributions

Daniel Mendoza wrote the data collection mechanisms, code for the baseline prediction models, and wrote the introduction, related work, approach, and part of the evaluation section of this report. Sijin Wang implemented the neural network architecture search for the DNN prediction models, extracted prediction results, created the figures for this report, and wrote part of the evaluation section of this report.

References

[1] S. J. Kaufman, Phitchaya Mangpo Phothilimthana, and Mike Burrows *Learned TPU Cost Model for XLA Tensor Programs* in 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada

[2] T. Patel and D. Tiwari, *CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers* in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), San Diego, CA, USA

[3] X. Xu, N. Zhang, M. Cui, J. He, and R. Surana, *Characterization and Prediction of Performance Interference on Mediated Passthrough GPUs for Interference-aware Scheduler* in HotCloud19

[4] C. Delimitrou, and C. Kozyrakis, *Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters* in Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2013)