January 28, 2019

Design of neural networks based on cost estimation

Yair Movshovitz-Attias

Andrew Poon

Ariel Gordon

Elad Edwin Tzvi Eban

Design of neural networks based on cost estimation

ABSTRACT

Certain automatic designs of neural networks not only minimize prediction error but also shrink or prune the network to reduce inference latency. Targeting inference latency directly is difficult; hence, FLOP-count is often used as a proxy for inference latency. However, FLOP-count is only loosely correlated with inference latency.

This disclosure describes techniques for direct computation or measurement of targeted costs such as inference latency, energy consumption, throughput, model size, etc. By integrating such targeted costs into design procedures, high performance neural networks of low inference latency, model size, and energy consumption can be obtained. The techniques find application in domains where fast, low-powered neural networks are advantageous e.g., image classification, language translation, optical character recognition, self-driving cars, interactive augmented/virtual reality, etc.

KEYWORDS

- neural network
- inference latency
- machine learning
- deep learning
- model size
- model pruning
- FLOP cost
- cost modeling
- energy consumption

BACKGROUND

Certain automatic designs of neural networks not only minimize prediction error but also shrink or prune the network to reduce inference latency [1]. Targeting inference latency directly is difficult; hence, FLOP-count is often used as a proxy for inference latency. However, FLOP-count is only loosely correlated with inference latency.

DESCRIPTION

This disclosure describes techniques for direct computation or measurement of targeted costs such as inference latency, energy consumption, throughput, model size, etc. By integrating such targeted costs into automatic design procedures, high performance neural networks of low inference latency, model size, and energy consumption can be obtained. Per the techniques, a targeted cost can be modeled in one of two ways, e.g., analytical and empirical.

Analytical modeling of costs

The analytical model computes cost using a set of closed-form functions associated with operations, e.g., multiplication, addition, convolution etc., in a computational graph that represents a neural network. For example, an analytical model for inference latency can comprise a bottleneck cost between compute-throughput and memory access.

Specifically, let a device have a maximal compute throughput of C floating-point operation per microsecond, and a maximal memory bandwidth of M MB/μsec. Each operation incurs a compute cost c FLOPs and a memory usage m MB. The inference-time cost of an operation is given by $\max(c/C, m/M)$.

Generally, an analytical model of cost is a function that takes into account device-specific characteristics D, operation-specific properties O, and potentially, also some contextual information about the operation: cost = cost(D, O, context).

Example: Analytical computation of the convolution operation

The FLOP-cost for a convolution operation is given by

$$\text{FLOP-cost} = 2NHWRSCK,$$

where:

N is the batch size;

H is the output height;

W is the output width,;

R is the kernel height;

S is the kernel width;

C is the number of input channels; and

K is the number of output channels.

The factor 2 reflects the fact that each element requires two FLOPs: a multiplication and an addition, referred to as a MAC (multiply-accumulate) operation. Compute latency for the operation is derived by dividing FLOP-cost by the peak FLOPs per second throughput of the hardware platform.

The memory cost for a convolutional operation is given by

$$\text{memory-cost} = NHWC + NHWK + RSCK,$$

which is the sum of the memory size for the input tensor and output tensor for the convolution. The memory latency is derived by dividing the memory payload by the peak memory bandwidth (GB/s) of the hardware platform.

Once the compute latency and memory latency are calculated, the net latency of the operation is:

$$\text{latency-of-operation} = \max(\text{compute latency}, \text{memory latency})$$

This assumes that compute and memory access happen in parallel on the device such that the latencies overlap. The total latency cost of the graph is the sum of the costs of individual operations.

The analytical model is a fast and simple approach that leverages accurate knowledge of the inner workings of a device when known. The analytical model generally makes certain assumptions about hardware behavior. The difference between the assumptions and actual hardware behavior determines the accuracy and benefit of this approach.

Empirical modeling of costs

In instances where hardware specifications are not exposed, or where a good analytical model for the cost is not immediately available, empirical modeling can be used to model costs.

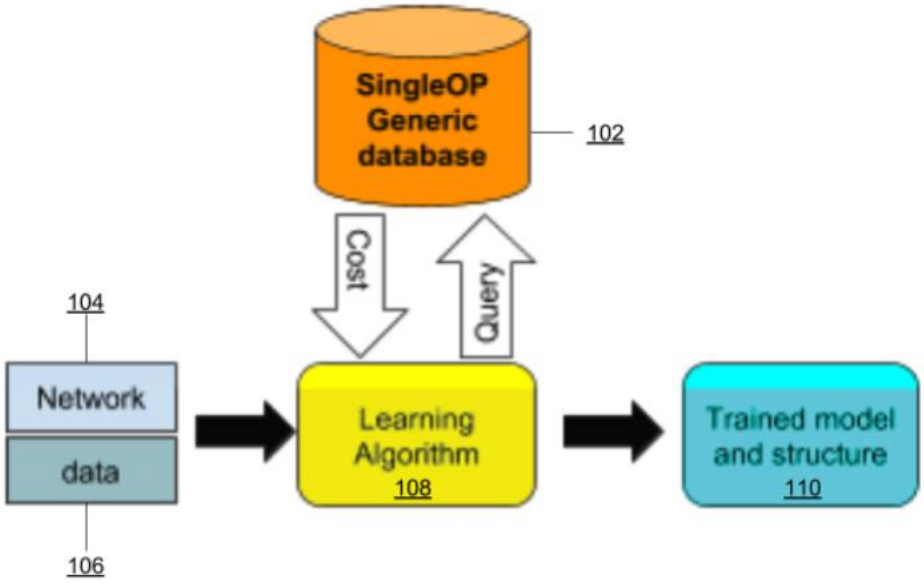

Fig. 1: Empirical modeling of costs

In empirical modeling, illustrated in Fig. 1, a database of single operations (102) is constructed that includes performance data for various configurations of operations on various

platforms, e.g., GPUs, machine-learning processors, etc. The database is populated by actual latency measurements on physical hardware.

A neural network (104) for which costs are to be measured generates data (106) that is fed into a learning algorithm (108). The learning algorithm fetches performance data from the database and trains on it to generate a trained model and structure (110). Data in the database is cached before training. The database is refreshed at regular intervals, e.g., overnight. In this form of empirical modeling, data points within the database are for preset configurations of operations, which may not match configurations that are actually present in the network.

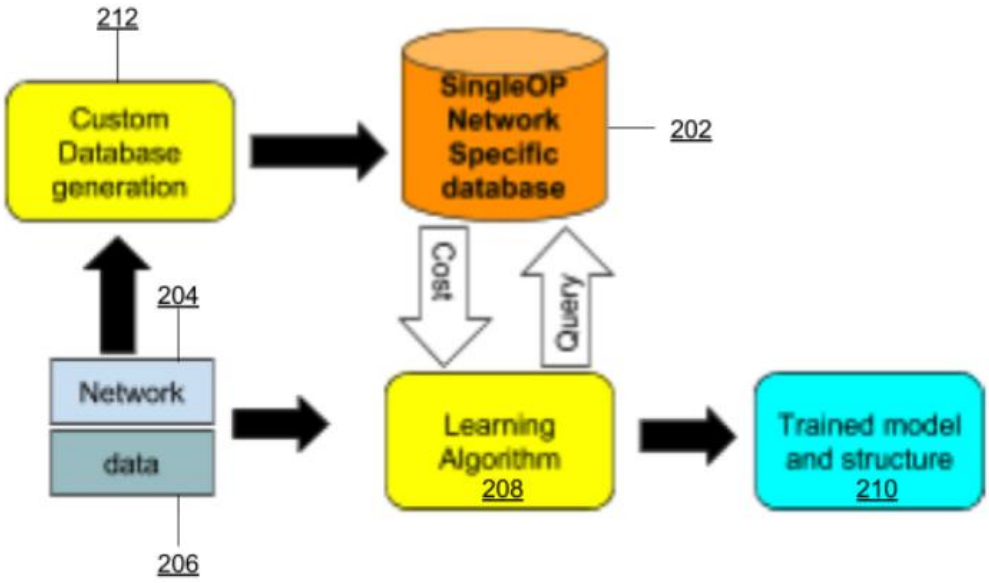

Fig. 2: Empirical modeling of costs

Fig. 2 illustrates another technique for empirical modeling of costs, per techniques of this disclosure. A database of performance data of single operations (202) is populated by actual latency measurements on physical hardware. A neural network (204) for which costs are to be measured generates data (206) that is fed into a learning algorithm (208). The learning algorithm fetches performance data from the database and trains on it to generate a trained model and

structure (210). For the neural network 204, a custom database (212) is generated that includes data pertinent to the architecture and list of configurations of the neural network. Such data is collected on demand, e.g., by a script, and is done once per seed network.

During training, the system refers to the cached data and interpolates costs as needed. The caching mechanism is modular such that swapping the data (e.g., GPU vs. ML processor) is seamless. Data is read from file or cached in memory if the table is small enough. Data points can be spaced either uniformly (e.g., channel step size of 8) or proportionally (e.g., channel step size 10% of original). The choice of data point spacing depends on the cost of data collection, and is a trade-off between the number of configurations, the number of machines available, and developer time. Denser spacing can reduce interpolation errors.

During training, latency cost for operations is calculated from the available data points. If the exact configuration is not available in the table, the latency cost can be interpolated in a one of the following ways.

1. **Interpolate between nearest configurations:** Find the latency cost for configurations (C_lower, K_lower), (C_upper, K_lower), and (C_lower, K_upper), where C is the number of input channels, and K is the number of output channels. Using the plane defined by these 3 data points, interpolate the latency cost for (C, K).
2. **Discrete cost levels:** Due to physical hardware constraints, latency cost may increase in discrete jumps, e.g., when tensors fill another memory chunk. If this effect is strong, it may be more accurate to collect data points at specific intervals, e.g., multiples of 8, and interpolate latency cost as being equal to the next highest interval, e.g., going one over the chunk size bumps up to the next chunk.

Analytical cost modeling is fast, simple, and provides insight into performance bottlenecks. Empirical cost modeling requires no a priori knowledge of the hardware that is used to implement the neural network, and makes no assumptions on hardware behavior. Therefore, it is relatively more accurate, but more complex to perform. Both modeling techniques can handle any hardware that is used to implement the neural network, e.g., GPU, ML processor, CPU, or other accelerator, regardless of manufacturer. Both can target any specific cost of interest, e.g., latency, throughput, power consumption, model size, etc.

Once the cost is computed (using analytical modeling) or measured (using empirical modeling), it is made part of an objective function that the neural network optimizes. For example, the regularizer uses the cost modeling techniques described herein as a black box that provides costs during neural network training.

Making costs part of the objective function causes the neural network to select cost-effective operations to compute an output, e.g., by setting weights across certain pathways to zero. This results in a neural network that not only optimizes prediction error but is also cost-effective. For example, after training, the neural network provides optimal performance subject to a maximum latency, model size, power consumption, throughput, network complexity, or other targeted cost. The techniques find application in domains where fast, low-powered neural networks are advantageous e.g., image classification, language translation, optical character recognition, self-driving cars, interactive augmented/virtual reality, etc.

CONCLUSION

This disclosure describes techniques for direct computation or measurement of targeted costs such as inference latency, energy consumption, throughput, model size, etc. By integrating such targeted costs into design procedures, high performance neural networks of low inference

latency, model size, and energy consumption can be obtained. The techniques find application in domains where fast, low-powered neural networks are advantageous e.g., image classification, language translation, optical character recognition, self-driving cars, interactive augmented/virtual reality, etc.

REFERENCES

[1] Gordon, Ariel, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. "MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks." The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2018, arXiv preprint arXiv:1711.06798v3 (2017).