Neurosurgeon

Collaborative Intelligence Between the Cloud and Mobile Edge

by Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars and L. Tang

Stefanos Laskaridis
sl829@cam.ac.uk

R244: Large-Scale Data Processing and Optimisation
Summary

Given the observation that ideal fine-grained DNN partition points depend on the layer compositions of the DNN, and constituent layers can achieve far superior end-to-end latency to cost due to the high data transfer overhead, locally executing on mobile devices is attractive. However, with this approach, large amounts of the computation are sent to the cloud for processing, as shown between the cloud and mobile device.

While data transfer becomes the latency and energy bottleneck, performance and energy efficiency of modern mobile hardware have continued to improve through powerful mobile SoC integration. Although recent research has shown that using distributed computing to partition DNN computation between the mobile and cloud systems is a win-win situation for both the mobile and cloud systems, we investigate how computation can be pushed out of the cloud and onto the mobile devices on the edge to execute computing at the mobile device (Figure 1).

In this section, we provide an overview of Deep Neural Networks (DNN) and describe how computer vision, speech, and natural language processing applications leverage DNNs. (Section 2.1) Furthermore, we find that instead of limiting the computation to be either executed in the cloud or on the mobile devices, we can achieve far superior end-to-end latency by intelligently partitioning DNN computation by 59.5%, and improves datacenter throughput by 1.5x. (Section 2.2)

In Section 3, we study the key contributions to the efficiency of DNNs in the cloud and on the mobile device. (Section 3.1) We provide an in-depth layer-level characterization of DNN layers, we show that partitioning DNN at optimal points to partition computation for reduced latency and mobile energy consumption across a suite of application domains. (Section 3.2) We demonstrate a system to intelligently partition DNN computation across the mobile edge – cloud datacenter – mobile device (Figure 3).

The contributions of this paper are as follows: (1) We provide an in-depth layer-level characterization of DNN compute and data size characteristics study – (2) significant differences in the compute and size characteristics of DNN layers, we show that partitioning DNN at optimal points to partition computation for reduced latency and mobile energy consumption across a suite of application domains. (3) We show that using distributed computing to partition DNN computation between the mobile and cloud systems is a win-win situation for both the mobile and cloud systems. (4) We demonstrate a system to intelligently partition DNN computation across the mobile edge – cloud datacenter – mobile device.

Figure 1: Status quo, mobile-only and the mobile-only Approach

Summary

Figure 3: Approach

a. Status quo Approach

b. Mobile-only Approach

c. Neurosurgeon Approach

Image taken from [1]
Status Quo
Status Quo

• Deep Neural Networks in “intelligent” applications
  • Apple Siri, Google Now, Microsoft Cortana

• Deep Neural Network applications are mostly offloaded to powerful private or public clouds for computation
  • Computer Vision
  • Natural Language Processing
  • Speech Recognition

• Large volume of data transfers cause latency and energy consumption.

• However, SoC advancements urged authors to revisit the problem.
The Mobile edge
Experiment Setup

Power Consumption
Watts Up? meter

Software
• Deep Learning: Caffe
• mCPU: OpenBLAS
• GPU: cuDNN

Server Platform
• 4U Intel Dual CPU Chassis, 8xPCIe 3.0 x 16 slots
• 2x Intel Xeon E5-2620, 2.1 GHz
• 1TB HDD
• 16x16GB DDR3 1866MHz ECC
• NVIDIA Tesla K40 M-class 12GB PCIe

Mobile Platform
• Tegra K1 SoC
• 4+1 quad core ARM Cortex A15 CPU
• 2GB DDR3L 933MHz
• NVIDIA Kepler with 192 CUDA cores
Testing the Mobile Edge

- Experiment running an Image of 152KB image through AlexNet [3]

- Measuring:
  - **Communication Latency**: 3G, LTE, WiFi
  - **Computation Latency**: mCPU, mGPU, cloud GPU
  - **End-to-end Latency**
  - **Energy Consumption**
Testing the Mobile Edge

Transmission has the dominating cost.

More power but shorter bursts.

Images taken from [1]
Neurosurgeon:
Partitioning between Cloud and Mobile
cNN

Convolution

Pooling

Images taken from [2]
DNN Layer types

- **Fully Connected Layer** (*fc*)
  All neurons are connected with all the neurons of the previous layer. Depth is the number of filters. Stride is how much we slide the filter each time. [2]

- **Convolutional & Local Layer** (*conv, local*)
  Convolves an image with one or more filters to produce a set of maps.

- **Pooling Layer** (*pool*)
  Downsamples an image to simplify representation. Can be average, max, or L2. [2]

- **Activation Layer** (*sig, relu, htanh*)
  Applies non-linear function to its input (sigmoid, Rectified Linear Unit, Tanh)

- **Normalisation layer** (*norm*)
  Normalises features across feature map.

- **Softmax Layer** (*softmax*)
  Probability distribution over possible classes.

- **Argmax Layer** (*argmax*)
  Chooses class with higher probability.

- **Dropout Layer** (*dropout*)
  Randomly ignores neurons as regularisation to prevent overfitting.
Images taken from [1] and [3]

Partition points (after each layer)
AlexNet

- **Convolutional layers** produce a lot of data.
- **Pooling layers** reduce a lot of data.
- **Fully connected layers** operate on few data but have high latency.

Images taken from [3]
Partitioning

- First layers have most of the data (convolutions and pooling)
- Later layers have most of the latency (fully connected layers)
- **Key idea**: Compute locally until the point it make sense and then offload to cloud.
## More Applications

<table>
<thead>
<tr>
<th>Category</th>
<th>Abbreviation</th>
<th>Network</th>
<th>Input</th>
<th>Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Classification</td>
<td>IMC</td>
<td>AlexNet</td>
<td>Image</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>VGG</td>
<td>VGG</td>
<td>Image</td>
<td>46</td>
</tr>
<tr>
<td>Facial Recognition</td>
<td>FACE</td>
<td>DeepFace</td>
<td>Image</td>
<td>10</td>
</tr>
<tr>
<td>Digit Recognition</td>
<td>DIG</td>
<td>MNIST</td>
<td>Image</td>
<td>9</td>
</tr>
<tr>
<td>Speech Recognition</td>
<td>ASR</td>
<td>Kaldi</td>
<td>Speech</td>
<td>13</td>
</tr>
<tr>
<td>Part-of-speech Tagging</td>
<td>POS</td>
<td>SENNA</td>
<td>Word vectors</td>
<td>3</td>
</tr>
<tr>
<td>Named Entity Recognition</td>
<td>NER</td>
<td>SENNA</td>
<td>Word vectors</td>
<td>3</td>
</tr>
<tr>
<td>Word Chunking</td>
<td>CHK</td>
<td>SENNA</td>
<td>Word vectors</td>
<td>3</td>
</tr>
</tbody>
</table>
5. Neurosurgeon

sion DNNs sometimes have better partition points in the each marked by

Figure 8: End-to-end latency when choosing different partition points. Each bar represents the end-to-end latency if the DNN is partitioned after each

25
30
70

18% more energy efficient than the cloud-only approach. As marked in Figure

energy cost of wireless data transfer, transferring the input

and the server to generate performance prediction models

two stages: deployment and runtime.

Partition points

Latency (s)

Data communication energy

Mobile processing energy

Energy (J)

Partition points

Layer latency

Size of output data

Latency (ms)

Data size (MB)

Images taken from [1]
The best way to partition a DNN depends on the characteristics of the DNN and the environment. For example, mobile devices' wireless communication latency can affect the data transfer latency. Datacenters typically experience diurnal load patterns.

The layer breakdown of VGG, FACE, and DIG reveals similar characteristics among them. Similar to latency, due to the high complexity of the back-end layers, reducing data size can provide significant latency and energy efficiency improvements. The data size in the back-end layers is often much lower than that in the front-end layers.

To choose the best partition point for a DNN architecture, we need to consider both end-to-end latency and mobile device energy consumption. The best point to partition the DNN to optimize for either end-to-end latency or mobile device energy consumption is not always the same. For example, cloud-only processing may not always be the most energy-efficient choice.

At deployment, we need to address this need and select the best point to partition the DNN in its DNN query service time. Due to these dynamic factors such as state of the wireless network and data size in the back-end layers, an intelligent DNN partitioning engine is required to generate performance prediction models for choosing an optimal partitioning point.

Images taken from [1]
The best partition point for a DNN architecture depends on strong variations in the best partition point suggest there is a need for an automatic system to intelligently select the best point to partition the DNN to optimize for end-to-end latency or mobile device energy consumption. To characterize the trade-off space of these performance metrics, the partition points are between the intermediate layers of the DNN. Note that partitioning at the layer granularity can affect the data transfer latency. Datacenters typically experience high variances in network inter-layer communication, while the per-layer computation generally decreases through the execution.  

1) In DNNs with convolution and pooling layers, there is a need for an automatic system to intelligently select the best point to partition the DNN to optimize for end-to-end latency or mobile device energy consumption. To characterize the trade-off space of these performance metrics, the partition points are between the intermediate layers of the DNN. Note that partitioning at the layer granularity can affect the data transfer latency. Datacenters typically experience high variances in network inter-layer communication, while the per-layer computation generally decreases through the execution.  

2) DNNs with only fully-connected layers, there is a need for an automatic system to intelligently select the best point to partition the DNN to optimize for end-to-end latency or mobile device energy consumption. To characterize the trade-off space of these performance metrics, the partition points are between the intermediate layers of the DNN. Note that partitioning at the layer granularity can affect the data transfer latency. Datacenters typically experience high variances in network inter-layer communication, while the per-layer computation generally decreases through the execution.  

3) The best way to partition a DNN depends on the application. For computer vision applications, there is a benefit to partition near the input layer. For speech applications, there is a benefit to partition near the end. For general-purpose DNNs, there is an advantage to partition near the middle of the DNN, while it is more beneficial to partition near the back-end layers for speech applications.  

Similar to latency, due to the high downs are shown in Figures 8 and 9. The front-end layers (e.g., conv1, conv2) of each DNN starting from the first non-input layer to conv5.3, the last of which is the fully-connected layer. In addition, the data size of each layer is shown in Figures 8 and 9. The best partition point for each DNN architecture depends on the neural network architecture. As marked in Figure 8, the partition points for best energy are each marked by a red star. The wireless network configuration is LTE. The partition points for best latency are each marked by a red square. The partition points for best energy are each marked by a red star. The partition points for best latency are each marked by a red square.

Images taken from [1]
Key Observations

1) In DNNs with convolution and pooling layers, the best partition point depends on its topology and constituent layers. Computer vision applications, which consist of a set of connected convolution and pooling layers, show strong variations in the best partition point, suggesting there is a need for a system to partition DNN computation between the cloud and mobile devices.

2) The best way to partition a DNN depends on its topology and constituent layers. For example, the back-end layers of an ASR DNN sometimes have better partition points in the cloud, while the front-end layers of an NLP DNN are partitioned at the mobile device.

3) The best partition point for a DNN architecture depends on its topology and constituent layers. For instance, the last pooling layer in an ASR DNN is sometimes partitioned at the mobile device, while the first fully-connected layer is sometimes partitioned in the cloud.

Figure 8: End-to-end latency when choosing different partition points. Each bar represents the end-to-end latency if the DNN is partitioned after each layer, where the left-most bar represents cloud-only processing (i.e., partitioning at the beginning) while the right-most bar represents mobile-only processing.

Figure 9: Mobile energy consumption when choosing different partition points. Each bar represents the mobile energy consumption if the DNN is partitioned after each layer.
POS

The best partition point for a DNN architecture depends on the suite (ASR, POS, NER and CHK) only consist of fully-convolutional layers and the server to generate performance prediction models. 2) DNNs with only fully-connected layers (e.g., Computer Vision applications), the data size increases through the execution. 3) DNNs with only fully-connected layers, there is a need for an automatic system to intelligently address this need, we present the design of a system to intelligently address this need, we present the design of

- Server processing latency
- Data communication latency
- Mobile processing latency

- Data communication energy
- Mobile processing energy

- Layer latency
- Size of output data

Images taken from [1]
Key Observations

1) In DNNs with convolution and pooling layers, partitioning at a convolution layer achieves the lowest latency, as marked in Figure 8. DNNs with only fully-connected layers experience higher latency after each fully-connected layer.

2) DNNs with only fully-connected layers sometimes have better partition points in the middle of the DNN, while it is more beneficial to partition at the beginning or the end for ASR and NLP DNNs. The wireless network configuration is LTE. The partition points for best energy are similar or smaller than the original input data.

3) The best way to partition a DNN depends on the application and the DNN's topology, which manifests itself in the computation DNNs sometimes have better partition points in the middle of the DNN, while it is more beneficial to partition at the beginning or the end for ASR and NLP DNNs. 3) The best way to partition a DNN depends on the application and the DNN's topology, which manifests itself in the computation DNNs sometimes have better partition points in the middle of the DNN, while it is more beneficial to partition at the beginning or the end for ASR and NLP DNNs. 3) The best way to partition a DNN depends on the application and the DNN's topology, which manifests itself in the computation DNNs sometimes have better partition points in the middle of the DNN, while it is more beneficial to partition at the beginning or the end for ASR and NLP DNNs.

For cloud-only processing is not the most energy-efficiency choice for cloud-only processing is not the most energy-efficiency choice. 4) To partition a DNN at the layer granularity can provide significant latency and energy efficiency improvements. In addition, dynamic data communication latency and data size variations of each layer.

figure 8: End-to-end latency when choosing different partition points. Each bar represents the end-to-end latency if the DNN is partitioned after each layer, where the left-most bar represents cloud-only processing (i.e., partitioning at the beginning) while the right-most bar represents mobile-only execution (i.e., partitioning at the end). The wireless network configuration is LTE. The partition points for best energy are similar or smaller than the original input data.

Images taken from [1]
Figure 8: End-to-end latency when choosing different partition points. Each bar represents the end-to-end latency if the DNN is partitioned after each layer, where the left-most bar represents cloud-only processing (i.e., partitioning at the beginning) while the right-most bar represents mobile-only execution (i.e., partitioning at the end).

The wireless network configuration is LTE. The partition points for best energy are shown in Figures 9 and 10.

Layer latency Size of output data

Images taken from [1]
Neurosurgeon
Neurosurgeon

- Partitions DNN based on:
  - DNN Topology
    - Computation latency
    - Data size output
  - Dynamic factors
    - Wireless network
    - Datacenter workload
Neurosurgeon

• **Profiles** device and cloud server
  
  • To generate prediction models
  
  • One time, in advance
  
  • Results stored in device for decision-making

• Two, distinct goals:
  
  • Latency minimisation
  
  • Energy optimisation
We observe that for each layer type, there is a large latency gap between mobile and cloud. This gap arises primarily due to the difference in hardware resources available on mobile and cloud platforms. The computation requirement of the layer approaches the limit of the available hardware resources. Based on the layer type, we use either a logarithmic or linear approach to predict the latency and power of the layer based on its configuration. We use GFLOPS (Giga Floating Point Operations per Second) as our performance metric.

To support future neural network architectures without additional profiling overhead, we establish a regression model for each layer type and power consumption for each configuration. Using these configurable parameters of the layer and measure the latency, we struct the prediction model for each layer type, we vary the regression model variables later in this section. We describe each layer's regression model variables, including the input feature map dimension, number, and pooling layers, we use the size of the input and output features in the input feature maps, and size and stride of the filters. The regression model for convolution, local and pooling layers, we use the number of input neurons and number of output neurons as the regression model variables.

As previously mentioned, it is a one-time profiling step to support future neural network architectures without additional profiling overhead. This set of prediction models are stored on the mobile and server platforms; per-application profiling is not needed. This allows the approach to support future neural network architectures without additional profiling overhead. This set of prediction models are stored on the mobile and later used to predict the latency and energy consumption for executing each layer on the mobile device and later used to predict the latency and energy consumption of arbitrary neural network architecture. This allows the approach to support future neural network architectures without additional profiling overhead. Each layer's type and configuration (at lines 11 and 12 of Algorithm 5.1) generate a set of prediction models. The models enable dynamic factors.

During the execution of a DNN-based intelligent application on the mobile device, the steps are as follows:

1. **Deployment Phase**
   - Extract layer configurations
   - Predict layer performance
   - Evaluate partition points
   - Run the DNN

2. **Execution**
   - Neurosurgeon
   - Cloud

**Algorithm 5.1**

1. Generate prediction models

**Prediction Models:**

- CONV
- FC
- POOL
- ACT

**Target Application:**

- Prediction Model
- Prediction Model
- Prediction Model

**Runtime Phase:**

4. Partitioned Execution
Regression model per DNN Layer

Linear or logarithmic regression model. GFLOPS for performance.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Regression Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>(filter_size/stride)^2 *</td>
</tr>
<tr>
<td></td>
<td>(# filters)</td>
</tr>
<tr>
<td>Local, Pooling</td>
<td>input, output feature maps</td>
</tr>
<tr>
<td>Fully Connected</td>
<td># Input/Output neurons</td>
</tr>
<tr>
<td>Softmax, Argmax</td>
<td># Input/Output neurons</td>
</tr>
<tr>
<td>Activation, Normalisation</td>
<td># neurons</td>
</tr>
</tbody>
</table>
Partitioning Algorithm

1: **Input:**
2: \( N \): number of layers in the DNN
3: \( \{ L_i | i = 1 \cdots N \} \): layers in the DNN
4: \( \{ D_i | i = 1 \cdots N \} \): data size at each layer
5: \( f, g(L_i) \): regression models predicting the latency and power of executing \( L_i \)
6: \( K \): current datacenter load level
7: \( B \): current wireless network uplink bandwidth
8: \( PU \): wireless network uplink power consumption
9: **procedure** PARTITIONDECISION
10: **for each** \( i \) **in** \( 1 \cdots N \) **do**
11: \( TM_i \leftarrow f_{mobile}(L_i) \) // Latency of mobile execution
12: \( TC_i \leftarrow f_{cloud}(L_i, K) \) // Latency of cloud execution
13: \( PM_i \leftarrow g_{mobile}(L_i) \) // Power of mobile execution
14: \( TU_i \leftarrow D_i/B \) // Transfer latency
15: **if** \( \text{OptTarget} == \text{latency} \) **then**
16: \( \text{return } \arg \min_{j} \left( \sum_{i=1}^{j} TM_i + \sum_{k=j+1}^{N} TC_k + TU_j \right) \)
17: **else if** \( \text{OptTarget} == \text{energy} \) **then**
18: \( \text{return } \arg \min_{j} \left( \sum_{i=1}^{j} TM_i \times PM_i + TU_j \times PU \right) \)

Device calculation
Cloud Calculation
Power consumed for local calculations
Uplink power consumption
Benchmark Results - Latency Optimisation

<table>
<thead>
<tr>
<th>Mobile</th>
<th>Wireless network</th>
<th>IMC</th>
<th>VGG</th>
<th>FACE</th>
<th>DIG</th>
<th>ASR</th>
<th>POS</th>
<th>NER</th>
<th>CHK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Wi-Fi</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LTE</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3G</td>
<td>argmax</td>
<td>input</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>Wi-Fi</td>
<td>pool5</td>
<td>input</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LTE</td>
<td>argmax</td>
<td>argmax</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3G</td>
<td>argmax</td>
<td>argmax</td>
<td>argmax</td>
<td>argmax</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
</tr>
</tbody>
</table>

Mispredicts when performance is close to one another.

Images taken from [1]
Benchmark Results - Power Optimisation

<table>
<thead>
<tr>
<th>Mobile</th>
<th>Wireless network</th>
<th>IMC</th>
<th>VGG</th>
<th>FACE</th>
<th>DIG</th>
<th>ASR</th>
<th>POS</th>
<th>NER</th>
<th>CHK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Wi-Fi</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LTE</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>fc3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3G</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>fc3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>Wi-Fi</td>
<td>input</td>
<td>input</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>fc3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LTE</td>
<td>pool5</td>
<td>input</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>fc3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3G</td>
<td>argmax</td>
<td>argmax</td>
<td>input</td>
<td>argmax</td>
<td>input</td>
<td>fc3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Even in suboptimal cases, 24.2% less energy than status quo
Testing under Network Variation

Bad network to offload computation.

Offloading makes sense now

In real world scenarios, network quality may vary.

Cloud-only will suffer the consequences.

Neurosurgeon dynamically adapts offloading to mitigate the problem.
Testing under Server Load Variation

Offload to server.

Server is too loaded now. Compute locally.

End-to-end latency of AlexNet
Mobile CPU-only, transfers via Wi-Fi

Current server load is determined by pinging.

Avoid latency, by taking server load into consideration.

This strategy is further **dropping server load**, allowing for more user queries to be served.
Results

- **End-to-end latency** improvement:
  - average: 3.1x
  - up to: 40.7x

- **Energy consumption** improvement:
  - average: 59.5%
  - up to: 94.7%

- **Datacenter throughput** improvement:
  - average: 1.5x
  - up to: 6.7x
Relevant work
Popular computation offloading solutions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No need to transfer program state</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Data-centric compute partitioning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Low/no runtime overhead</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Requires no application-specific profiling</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>No programmer annotation needed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Server load sensitive</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Benchmark Result - MAUI

![Graph showing latency speedup for different benchmarks using MAUI and Neurosurgeon](image)

MAUI scheduling for a layer depends on previous invocations.

Images taken from [1] and [4]
Privacy Preserving Shared Models

• Based on the edge computing paradigm.

• Models where a general model is trained in the cloud and online learning is supplementing this model on the device [8].
Review & Critique
Review: The good parts

• The name! :)

• Brand new paper published in ASPLOS ’17 (1 citation from Cambridge [5])

• Rational extension of current model of execution based on SoC developments.

• All around benchmarks, substantial speedups.

• Inclusive of GPU computation and different network setups.
Critique

- DNN specific (in contrast with MAUI)

- Profiling has hardcoded the regression models for each type of layer (difficult to expand, does not learn how to assess)
  - How would an rNN get split with Neurosurgeon?

- Profiler assumes one type of hardware server-side.
  - Different sized containers based on load.
  - Different datacenter forwarding behind load balancer.

- Adoption: NVIDIA Tegra K1 is a high-end SoC
  - Lower-end processors may shift offloading to the cloud.
Critique

- Distinct optimisation for latency, energy efficiency.

- Why not offer a Pareto’s optimality curve and pick point based on user profile?
Critique

Algorithm 1

1: Input:
2: \( N \): number of layers in the DNN
3: \( \{L_i| i = 1 \cdots N\} \): layers in the DNN
4: \( \{D_i| i = 1 \cdots N\} \): data size at each layer
5: \( f, g(L_i) \): regression models predicting the latency and power of executing \( L_i \)
6: \( K \): current datacenter load level
7: \( B \): current wireless network uplink bandwidth
8: \( PU \): wireless network uplink power consumption
9: procedure PARTITIONDECISION
10: for each \( i \) in \( 1 \cdots N \) do
11: \( TM_i \leftarrow f_{mobile}(L_i) \)
12: \( TC_i \leftarrow f_{cloud}(L_i, K) \)
13: \( PM_i \leftarrow g_{mobile}(L_i) \)
14: \( TU_i \leftarrow D_i/B \)
15: if OptTarget == latency then
16: return arg min (\( \sum_{j=1}^{N} TM_i + \sum_{k=j+1}^{N} TC_k + TU_j \))
17: else if OptTarget == energy then
18: return arg min (\( \sum_{j=1}^{N} TM_i \times PM_i + TU_j \times PU \))

Smartphones support multitasking. Why not include \( K_{mobile} \)?
Suggestions

• Work with model decomposition and compression algorithms to push more computation locally (such as DeepX [6])

• Other hardware could be taken into consideration (e.g. DSP) for further efficiency (such as DeepEar [7])

• Could Reinforcement Learning be of any help in learning how to partition instead of static profiler?

• Offloading to devices in local network. (MAUI [4])
Thank you

Q&A
References


2. Stanford CS231n - Andrej Karpathy
   http://cs231n.github.io/convolutional-networks/


