Università
della
Svizzera
italiana

Facoltà
di
scienze
informatiche

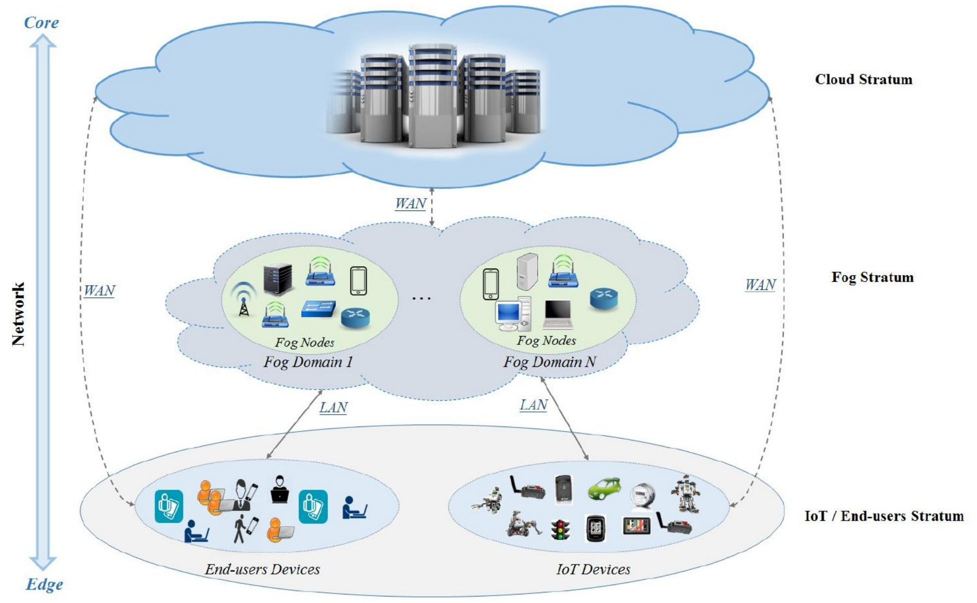
Edge Computing in the IoT

Edge Computing

Alberto Ferrante (alberto.ferrante@usi.ch)

Edge/Fog/Cloud Computing

- Fog computing extends traditional cloud computing to the edge of the network
  - The processing can take place at the edge of the network (**fog nodes**)
  - Other processing can happen in the cloud
- Edge computing: computation is moved into IoT nodes



https://arxiv.org/pdf/1710.11001.pdf

Pro and Cons of Moving the Computation To the Edge

- No real pro and cons list: it all depends on the specific case!
- Things to consider:
  - Latency of computation w.r.t. requirements
    - For cloud computing it includes Network latency
      - For example, sending the data to the cloud and receiving the results of the computation
  - Energy/power of computation vs. energy of networking
  - Possibility of software updates on edge/fog devices vs. cloud
  - Number of nodes and amount of data:
    - Many nodes and/or relevant flow of data imply relevant computational and network resources on the cloud side
  - Security and privacy
    - E.g., cloud computing might be problematic for privacy

Pro and Cons of Moving the Computation To the Edge

- For simple sensing and control logic, edge computing is usually the best option
  - Lower latency, lower energy, low computational resources required
- For real-time applications cloud computing is usually not ideal
  - Networking in general, with the exception of few dedicated protocols, introduce too much uncertainty

Design Space Exploration

- Define the metric (energy for a task, latency, …) + objective (i.e., minimize or maximize the metric) + requirements
  - For example we may consider energy as a metric to minimize, but impose a requirement on max latency
  - Usually, we are more interested in node energy than global energy including cloud
- Define the design space: which are the different choices that we have available?
  - All computation on cloud
  - All computation on edge nodes
  - Pre-processing on edge, then on cloud
  - ...
- Explore the design space and choose the optimal point(s) w.r.t. the metric
  - Implementing and testing in real life all design points is usually impossible →
    - (partial) Simulation
    - Evaluations
    - Preliminary tests
    - Models

Design Space Exploration – Example 1

- IoT nodes collect images with the purpose of counting dogs present in different dog areas; entrance is closed if no. of dogs is higher than a threshold
  - IoT nodes collect 1 image (approx 2MB) every 30s
  - A cloud infrastructure collects data about dog presence (no. of dogs over time) and shows it on a map
- Let’s concentrate on the task of counting dogs
  - Metric: node energy, provided that the computation is completed before the arrival of a new image (i.e., 30s latency)
  - Design space:
    - Sending images to the cloud and running models for counting on the cloud
    - Running models for counting on the nodes and sending only the number of dogs to cloud
    - Pre-processing images (e.g., extract objects that can be dogs) on the node and sending pre-processed images to the cloud for counting

Design Space Exploration – Example

- Suppose that we know the following
  - Networking node-gateway (BLE): 0.001 mWh/KB
  - Running the full model on edge node requires:
    - 35s of computation
    - 0.1 mWh
    - 2 bytes of data to be sent as count
  - Image pre-processing requires:
    - 22s of computation
    - 0.075mWh
    - Average of 1MB of data to be sent to the cloud

- Exploring the design space

1) All on cloud:
   - Energy per image: 2MB*1000*0.001mWh/KB=2mWh
   - Computation time: image sent in less than 2s
2) All on node:
   - Energy per image: 0.100002mWh
     - Network: 2/1000*0.001mWh/KB=0.000002mWh
     - Computation: 0.1mWh
   - Computation time: 35s + less than 0.5s for communication
3) Pre-processing on node:
   - Energy per image: 1.075mWh
     - Network: 1MB*1000*0.001mWh/KB=1mWh
     - Computation: 0.075mWh
   - Computation time: 22s + less than 2s for sending the image

The optimal solution is no. 2, but it does not satisfy the requirement on latency: we opt for solution 3

Intermittent Computations

- Often, IoT devices need to work intermittently:
  - Processing is not continuous
  - Periodic (with low periodicity) event-based computation
- Examples:
  - A sensor node that needs to sense some parameters with low periodicity, process data and send it to the cloud
    - E.g., sense temperature every 10 minutes
  - A device that needs to respond to a specific rare event
    - E.g., A sensor signals that water is over a certain threshold, the device should activate an alarm and send a message by using a 4G network
  - A device that takes a picture every hour, performs some processing (e.g., a machine learning model), and send the results to the cloud

Intermittent Computations

- To save energy, the devices can make use of sleep modes, depending on the type of device/microcontroller
  - Wake-up on events (interrupts)
  - Walke-up periodically (timer-based events)
  - Switch on/off parts of the system depending on the computational needs
    - E.g., in a 2-core setup, one core can be always on, the other can be waken-up only when more computational resources are required
      - Typical case: one core used for sensing/controlling; the other core is used for computations (e.g., image/signal processing or machine learning)
      - One core uses a microphone to detect if there is any sound over a certain volume threshold, if there is any, the second core processes the sound to detect if it corresponds to steps or a broken window

Intermittent Computing

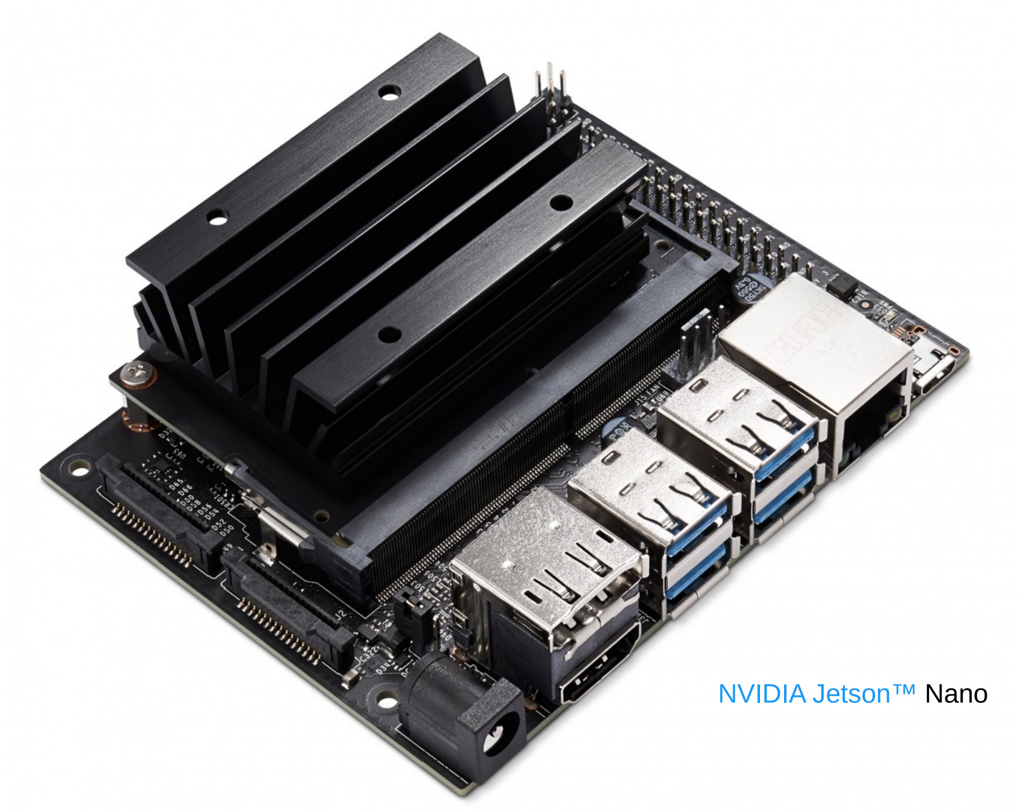
- Ambient energy harvesting enables battery-less embedded sensing
  - E.g., RFID-based devices
- Intermittent computing stem from the erratic energy patterns caused by energy harvesting:
  - Computations unpredictably terminate whenever energy is insufficient
- See for example: F. Bambusi, F. Cerizzi, Y. Lee and L. Mottola, "The Case for Approximate Intermittent Computing," 2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2022, pp. 463-476, doi: 10.1109/IPSN54338.2022.00044.

Accuracy Vs. Performance/Energy

- In some cases accuracy of considered algorithms can to be traded for performance/energy
  - Less complex algorithms
  - Use of fixed point numbers instead of floating point ones
  - Use of simplified models
  - …
- Which is the accuracy loss that my system can tolerate?
- We can use a metric to evaluate the trade-off
  - e.g., we maximize accuracy/energy

Machine Learning and Edge Computing

- Machine learning in general and deep learning in particular has increased its presence in edge devices:
  - Availability of more powerful hardware (e.g., 32 bit microcontrollers equipped with GPUs) at low cost
  - Applications that benefit from local processing
  - Increase in amount of data collected
- There exist devices that were specifically designed for deep learning applications
- Deep learning models can have memory and/or computational requirements that are still incompatible with the resources provided by edge devices
  - What about fog?



NVIDIA Jetson™ Nano

Workflow of Machine Learning

- Define the task
- Develop a model
  - Prepare the data
  - Select a model evaluation protocol and a baseline
  - Train the model
- Deploy the model
  - Deploy the model to your final device/server/…
  - Monitor the model’s performance
  - Collect data for improving the model

Workflow of Machine Learning with Edge Devices

Phase	Task	Cloud	Edge device
Model development	Collect data	X	X
	Prepare the data	X	
	Training	x	
Model deployment	Deploy the model	x	x
	Collect data for improving the model	x	x
	Improve the model	x	

- In general, more computationally intensive operations (e.g., training) are performed on cloud
- Data are collected by using edge devices, similar to the final ones (+ available datasets)
- Deployment may be performed on the cloud or on edge devices (partly or fully)

Machine Learning and Edge Devices – Example

- Let’s consider the system that counts dogs again and let’s suppose that the model is going to run on edge devices
- The ideal model training process is as follows:
  - Edge devices similar to the final ones (or at least with similar cameras) are used to collect images (e.g., by sending them to the cloud, or by saving them on local memories that are later collected)
  - On the cloud
    - Images are preprocessed (e.g., resizing, elimination of duplicates, ...)
    - Once a sufficient number of images has been collected, the training, test, and validation datasets are formed; the model is trained and validated
  - The model is deployed on edge nodes along with required preprocessing modules (e.g., image resizing)
    - Images captured by the camera are fed into the model deployed on edge nodes after suitable preprocessing

Machine Learning and Edge Devices – Model Maintenance

- During the lifetime of the model, it might require to be maintained
  - Model is improved with new training data
    - How to collect new data if processing is performed on edge nodes?
  - Model tuning to different situations/users
- Updates usually happen offline (i.e., not on edge devices), as the initial training
- The updated model is deployed to edge devices (e.g., OTA updates)
  - In most cases, only weights need to be updated

Model Optimizations for Edge Computing

- Models might be too big to fit into edge devices memory and too computational resource hungry for microcontrollers
  - Optimizations are necessary both to reduce memory footprint and computational resource requirements
  - A reduction of model accuracy might be accepted to make it fit into the edge device (e.g., by reducing the number of layers)

Model Optimizations for Edge Computing

- Common methods for reducing memory footprint are
  - Model pruning
  - Quantization of weights
  - Knowledge Distillation
  - Factorization
    - The redundancy present within the convolutional filters is exploited to derive approximations that significantly reduce the required computation [*]

[*] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In 28th Annual Conference on Neural Information Processing Systems 2014, NIPS 2014, pages 1269–1277. Neural information processing systems foundation, 2014.

Model Pruning

- Not all parts of large neural networks are still useful after the training process is over
  - Eliminate these parts without impacting network's performance
    - Done before, during or after training
    - Not all kinds of pruning actually allow for accelerating neural networks
- What to prune?
  - Prune parameters: unstructured pruning
    - Not always useful in accelerating networks, still, it might allow to have more compact models
  - Pruning larger structures, such as whole neurons or filters: structured pruning

https://towardsdatascience.com/neural-network-pruning-101-af816aaea61

Model Pruning

- Pruning criteria
  - E.g., weight magnitude criterion: pruning weights whose absolute value (or “magnitude”) is the smallest
    - Under the constraint of a weight-decay, those which do not contribute significantly to the function are expected to have their magnitude shrink during training
- How to apply pruning: most common method
  - Train
  - Prune
  - Fine-tune
  - We may iterate on the prune and fine-tune steps to prune even more

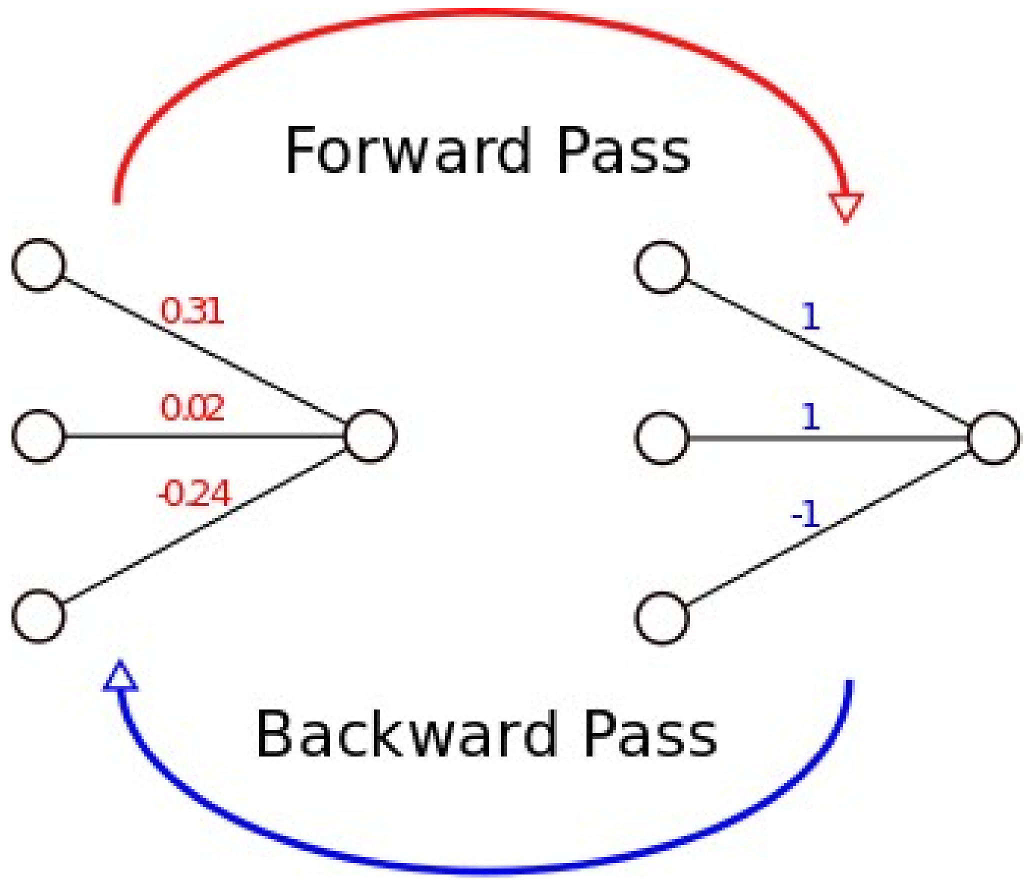
https://towardsdatascience.com/neural-network-pruning-101-af816aaea61

Model Quantization

- The process of approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers
  - Performed on network parameters
  - E.g., 16-bit floats or integers (or fixed point numbers) instead of 32-bit floats
    - Strictly speaking, it is intended as integers
- It reduces the memory footprint of the model, but it might also increase performance, for example by reducing floating point operations
- It can be performed:
  - After training
    - It leads to lower accuracy
  - After training + tuning
    - Partly compensates for the loss of accuracy
  - During training

Model Quantization During Training

- The forward pass of the neural network uses a scheme for rounding float-precision parameters to lower precision
- In the backwards pass, the high-precision parameters are updated using gradients calculated during the forward pass
- Only the reduced-precision network is saved for inference



https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b

Knowledge Distillation

- A procedure for model compression
  - A small (student) model is trained to match a large pre-trained (teacher) model
  - Knowledge is transferred from the teacher model to the student by minimizing a loss function, aimed at matching
    - Softened teacher outputs (before normalization → logits)
    - Ground-truth labels

https://keras.io/examples/vision/knowledge_distillation/

In Short

- Advantages and disadvantages of moving the computation towards edge nodes
- Optimal placement of tasks on different elements of the system (cloud, edge, fog)
- Intermittent computations
- Optimization of algorithms for edge devices
- Machine learning on edge devices