10-414/714 – Deep Learning Systems: Algorithms and Implementation

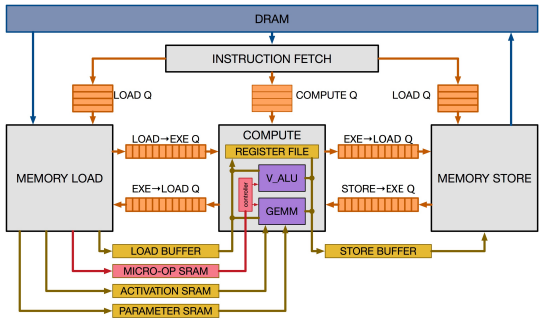
Future Directions

Fall 2021

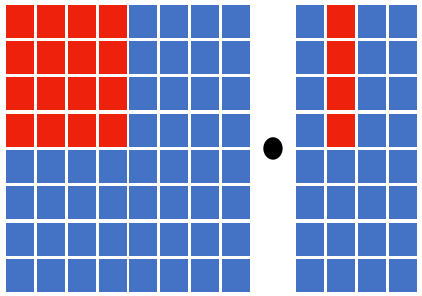
J. Zico Kolter and Tianqi Chen
Carnegie Mellon University

Machine learning compiler for specialized hardware

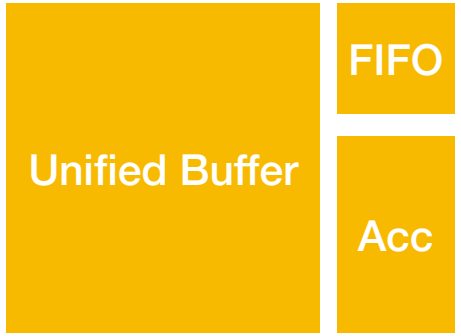
TPUs



Tensor
Compute Primitives



Explicitly Managed
Memory Subsystem



Specialized Hardware

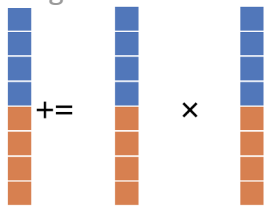
Generic FMA

```
# semantics
C[0] += A[0] * B[0]

# implementation
llvm.fmuladd.f32
```

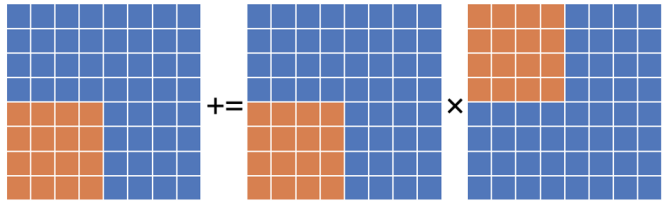
Vector FMA

```
# semantics
for i in range(4):
  C[i] += A[i] * B[i]
# implementation
llvm.fmuladd.v4f32
# diagram
```



Nvidia Tensor Core and NPUs

```
# semantics
for y, x, k in grid(16, 16, 16):
  C[y, x] += A[y, k] * B[k, x]
# implementation
nvvm.wmma.m16n16k16.mma.row.row.f32.f32
# diagram
```



Scalar unit

SIMD,
vector units

Specialized
tensor instructions

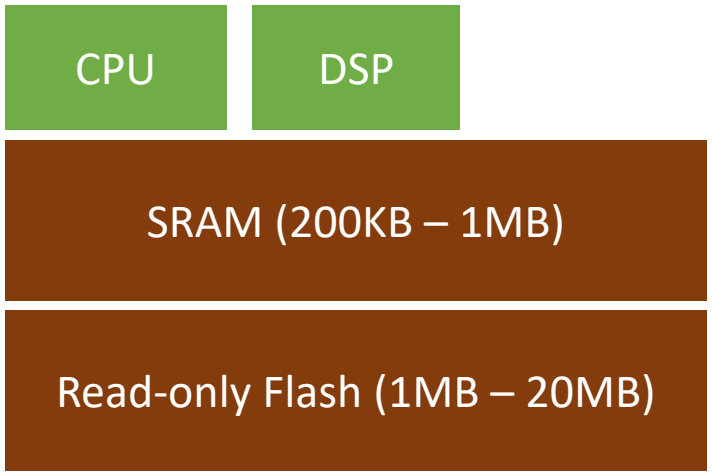
Heterogenous hardware backends

Specialized hardware comes with their own limitations.

We need to start to think about use a mix of them on a single SoC(e.g. Apple M2), or over networks.

Tiny Machine Learning

Tiny Machine Learning



A Typical Tiny Device

- Extremely limited memory resources
- Limited instruction set support(e.g. no floating point units)

Specialized data types

We mostly only looked at float32(and a bit of int32) in our lectures

To get maximum benefits from the hardware acceleration, researchers are looking into more specialized data types

- int8, int4, int1(binary neural network)
- float16, bfloat16 (customized floating points)

Automatic parallelization

We learnt about data parallel training and model parallel training

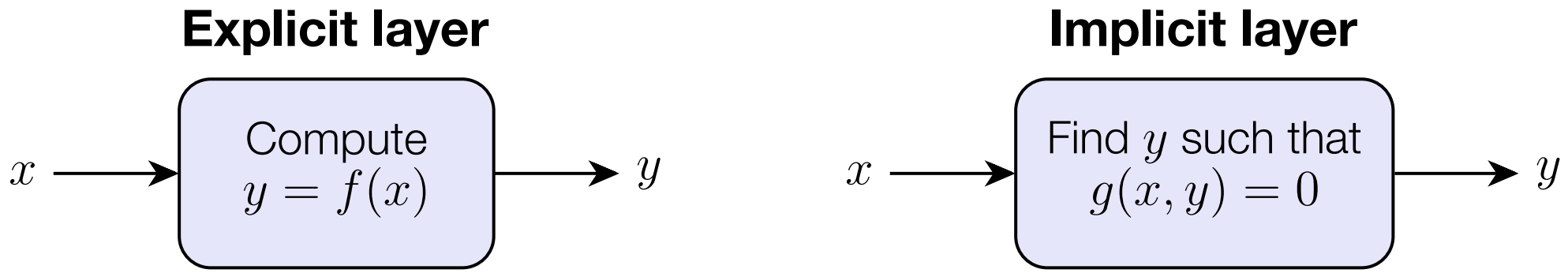
There is an active thread of research on generalized parallelization to scale up big model training in an effective way.

Implicit Layers

Traditional layers in deep networks, like the ones you have implemented, perform some kind of *explicit* computation mapping inputs to outputs

Instead, can define layers *implicity,* as the *solution* to some nonlinear algebraic or differential equation

Can backprop analytically through these layers using implicit differentiation

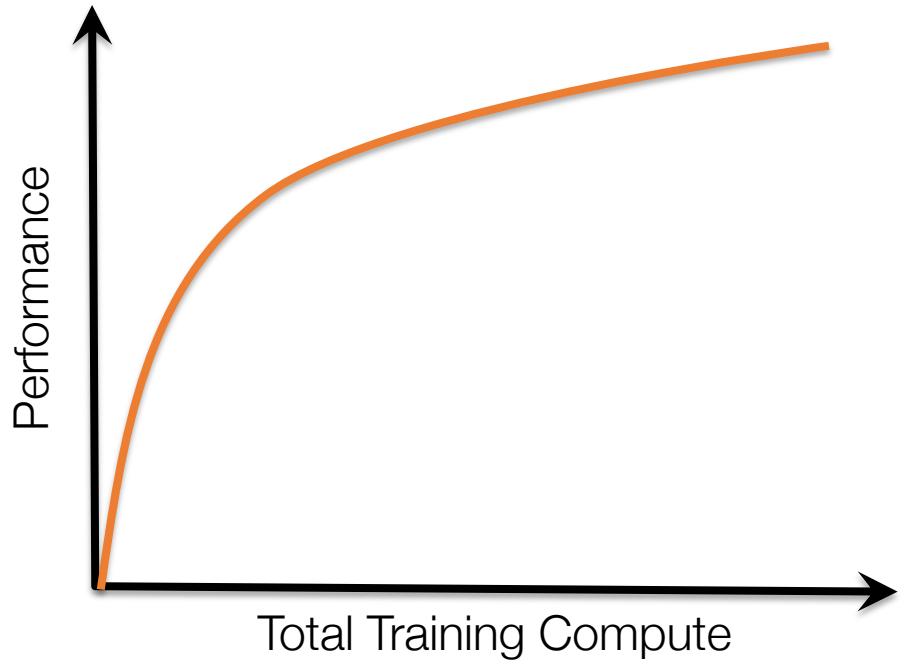


The learning/compute Pareto frontier

How much computation is “necessary” for learning?

There is *some* curve that indications how many operations (FLOPs, but also e.g. data acceses, etc) are needed, in training, to achieve a certain level of performance

But we have very little idea about what this actually



Q&A

We’ve spent a lot of time in this class discussing how hardware and libraries have enabled the advancement of deep learning. On the flip side to what degree do you think these things could have limited the options being considered for deep learning architectures? Do you think there are many ideas that could have been very effective but just didn’t have efficient support/implementations available?

Q&A

I’m interested in the robustness and debuggability of the ML frameworks. While working on assignments, we notice that different implementations of the ML frameworks may return different results. How can we make the ML framework more robust? The non-determinism of the ML framework makes it very hard to debug. How do the existing ML frameworks help developers to identify bugs in their code? Is there a way to improve the debuggability of ML frameworks?

Q&A

I was wondering how you would view the development of various ML chips / accelerators?

Traditionally, we had CPU and GPU dominating the field and Nvidia keeps making larger and faster GPUs that pushed the DL field. Then, FPGA was used to perform inference tasks. Next, Google released TPU and many companies started to develop their own chips. I can easily name quite a few chips from different startups such as Habana, Cerebras, SambaNova, Tenstorrent, Graphcore, Groq, etc. Also, phone chips (Apple) seem to be able to do inference task as well.

Do you think some of those chips may threaten the strength position of Nvidia's GPU? ….

Q&A

(Open questions to class)