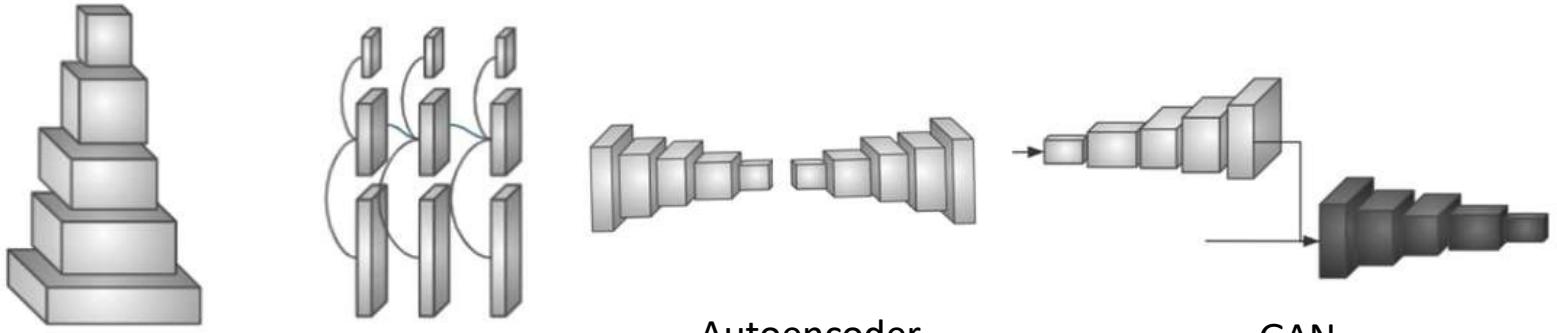
Types of Deep Learning

- Supervised Learning:
  - learning a function that maps an input to an output from datasets consisting of input-output pairs
  - Example: prediction of house price based on location
- Unsupervised Learning:
  - … input data without labeled responses.
  - Example: recommend channels based on history
- Semi-supervised Learning
  - … a small amount of labeled data with a large amount of unlabeled data
  - Example: detection of CAR from photos with a few labels
- Reinforcement Learning
  - learning in an interactive environment using feedback from its own actions and experiences
  - Example: game AI

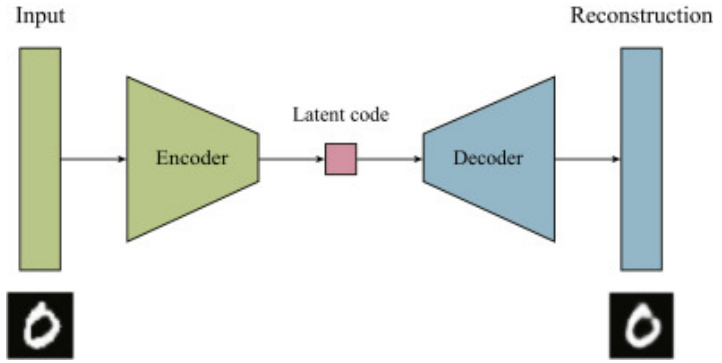
Model architectures

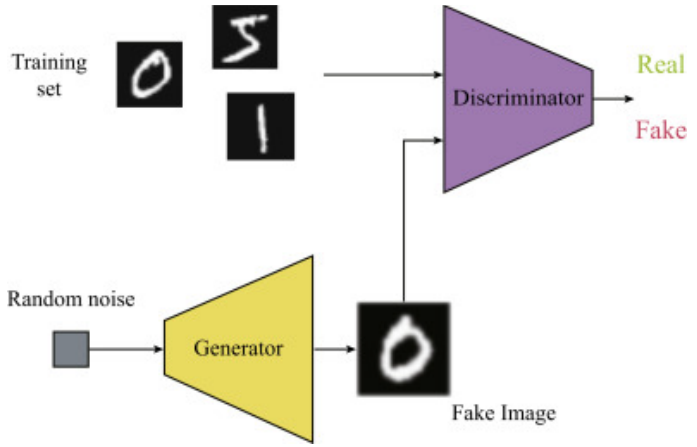
- Convolutional Neural Network (CNN):
  - feed-forward structure consists of a stack of layers
- Recurrent Neural Network (RNN):
  - transform an input sequence into an output sequence
- Transformers:
  - Input to output sequence. Leverage “attention” module
- Autoencoder:
  - learn a low-dimensional representation of a high-dimensional data set
- Generative Adversarial Network (GAN):
  - consist of two (CNN) models competing against each other in a two-player game framework
  - create new fake data by learning through the real data



Example: An Autoencoder

- Two NNs: encoder generating a compressed ”latent code” so that reconstruction loss is minimized
  - The latent code can serve as a reduced dimension representation (similar to PCA)
- Generative Adversarial Networks (GANs) use to the encoder component but with a different loss function
  - To generate “realistic” fake inputs
- Related concept: “embedding layers”: maps input information from a high-dimensional to a lower-dimensional space
  - Encoders can generate an embedding (of an image, text, graph, …)
  - Embedding can be as simple as a one-hot encoding of words in a vocabulary





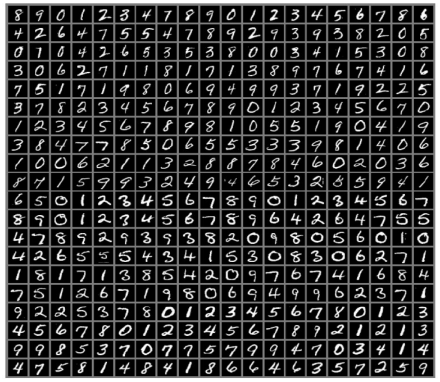
[Autoencoders – ScienceDirect]

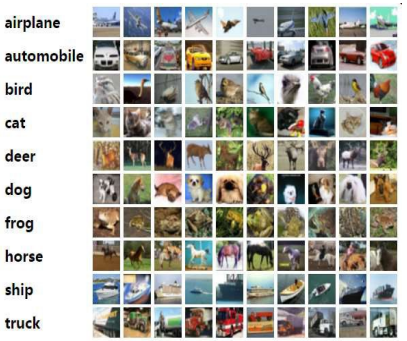
A Full Cycle of DL in Practice

- Training:
  - Data Cleaning
  - Data Input
  - Forward Computation
  - Loss Calculation
  - Backward Computation
  - Parameter Update
  - Converge Check / Stop Criteria
  - Save Model (Parameters)
- Testing/Inference:
  - Data Cleaning
  - Data Input
  - Load Saved Model
  - Forward Computation
  - Results Interpretation

Common Computer Vision Datasets

- MNIST (1998)
  - scans of handwritten digits and associated labels describing which digit 0–9 is contained in each 28x28 image.
  - The training set contains 60000 examples, and the test set 10000 examples.
  - A trivial dataset by modern standards
- CIFAR10 (2009)
  - 60000 32x32 color images in 10 classes, with 6000 images per class.
  - 50000 training images and 10000 test images.
  - Has a 100 class CIFAR100 variant
- IMAGENET (2014)
  - ~14 million labeled images, 20k classes
  - Average image size 469x387 usually downsampled to 224x224
  - Images gathered from Internet
  - Human labels via Amazon MTurk



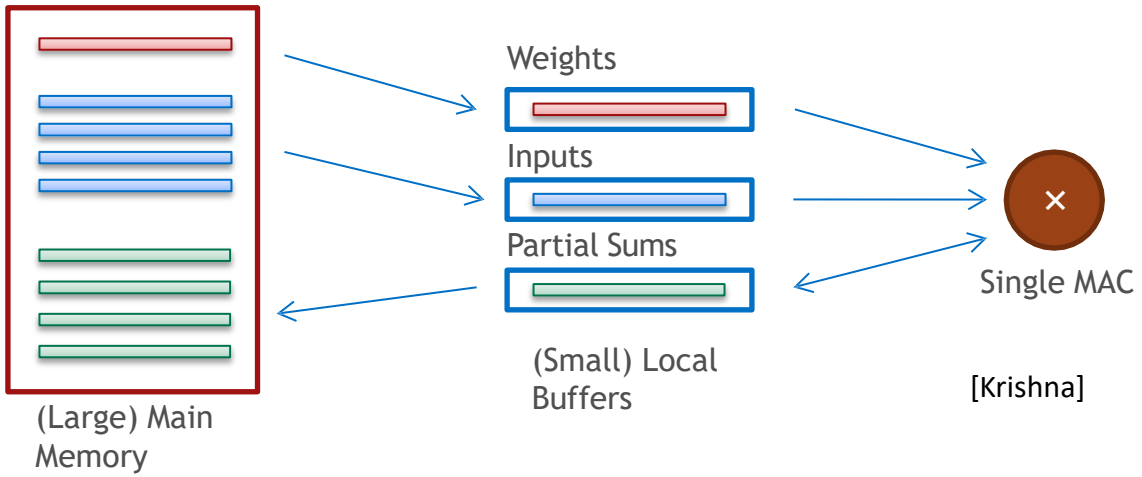


Lecture 2: Dataflows

Puneet Gupta

Some slides from Lei He

A Simple NN Hardware Concept



[Krishna]

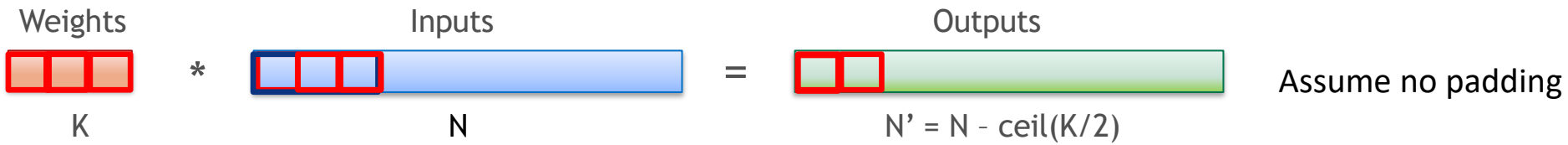
Single MAC → long dot product is computed by reading partial sums and adding a new multiplication result to compute the next partial sum

Lets assume a MAC takes a cycle

- Main memory access is “expensive” compared to local buffer access
  - Large size, further away
  - Similar to a cache argument

  →want to reuse values in local buffers as much as possible before sending them back to main memory (similar to cache hit rate)
    - Local buffers limited in size, so cant keep everything!

Simple 1D CONV Example



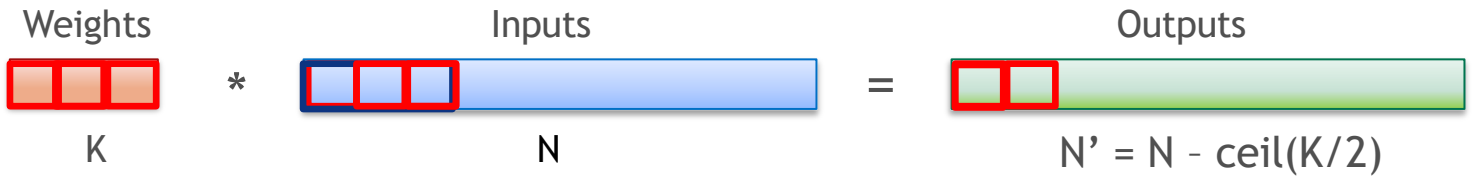
Assume no padding

- How often is a new input read ?
- How often is a new weight read ?
- How often is a new output read ?
- Output changes the least → output "stationary" dataflow
- What is the size of buffer needed so that everything is read from main memory exactly once ?
  - Inputs: K
  - Weights: K
  - Outputs: 1
- How many times is the buffer read?
  - For inputs ?, weights ?, outputs ? N'K

```
int i[N];        # Input activations
int w[K];        # Filter weights
int o[N];        # Output activations

for (x = 0; x < N'; x++) {
        for (y = 0; y < K; y++) {
            o[x] += i[x+y]*w[y];
}
```

Another way to implement 1D CONV



Assume appropriate padding

- In this, output and input need to be reloaded every cycle
- Weights are reloaded every N' cycles

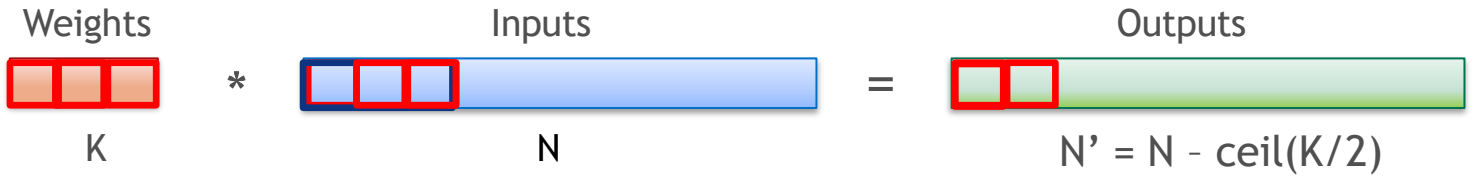
→Weight stationary dataflow

- What is the size of buffer needed so that everything is read from main memory exactly once ?
  - Inputs: N'
  - Weights: 1
  - Outputs: N'
- How many times is the buffer read?
  - For inputs ?, weights ?, outputs ?: N'K

```
int i[N];        # Input activations
int w[K];        # Filter weights
int o[N];        # Output activations

for (y = 0; y < K; y++) {
        for (x = 0; x < N'; x++) {
            o[x] += i[x+y]*w[y];
}
```

Yet Another way to implement 1D CONV



Assume appropriate padding

- In this, output and weight need to be reloaded every cycle
- Inputs are reloaded every N' cycles

→Input stationary dataflow

- What is the size of buffer needed so that everything is read from main memory exactly once ?
  - Inputs: 1
  - Weights: K
  - Outputs: N'
- How many times is the buffer read?
  - For inputs ?, weights ?, outputs ?: N'K

```
int i[N];        # Input activations
int w[K];        # Filter weights
int o[N];        # Output activations

for (x = 0; x < N'; x++) {
        for (y = 0; y < K; y++) {
            o[x-y] += i[x]*w[y];
}
```

What about energy ?

- Total Energy = Main memory access energy + Buffer access energy + MAC energy
- MAC Energy = N'K *E(MAC)
- Main memory energy = E(Mem_read)*(K+N) + E(Mem_write)*N'
- Buffer access energy: Remember access energy of a buffer will depend on its size (for SRAM buffers, roughly linearly increases with size) →EBUF(X) is access energy of a buffer of size X
  - OS: N'K*(EBUF(K) + EBUF(K) + 2*EBUF(1))
  - WS: N'K*(EBUF(1) + EBUF(N) + 2*EBUF(N))
  - IS: N'K*(EBUF(1) + EBUF(K) + 2*EBUF(N'))
  - Recall output partial sums need to be read and written equal number of times to the buffer

What about latency ? OS Example

- Naieve implementation:
  1. load buffers from main memory as needed
  2. load MAC from buffers: TBUF(K)+ TBUF(K) +TBUF(1)
  3. do MAC: TMAC
  4. Write output partial sum to buffer: TBUF(1)
  5. If output fully computed write to main memory: T(MEM_write) + TBUF (1)
  6. Go to step 2 till done (N'K times)
  - Total Step 1 time: (N+K)* (T(Mem_read) + TBUF(K))
  - Total Step 2 time: N'K*(TBUF(K) + TBUF(K) +TBUF(1))
  - Total Step 3 time: N'K* TMAC
  - Total Step 4 time: N'K * TBUF(1)
  - Total Step 5 time: N' * (T(Mem_write) + TBUF(1))
  - Total latency: sum of all steps
- Plenty of ways to speed things up
  - What if we make the buffers "dual ported". I.e., they can be read and written at the same time
    - Step 2 and step 4 can happen simultaneously → Step 4 time reduces to 0
    - Step 1 time may be possible to reduce but not by much since T(Mem_read) >> TMAC + TBUF
  - What if we add a pipeline stage between BUF and MAC → Step 2, 3, 4 can be overlapped → Step (2+3+4) time becomes N'K*max ((TBUF(K) + TBUF(K) +TBUF(1)), TMAC, TBUF(1))

Real-Life Complications

- Bigger compute
  - You may have more than one MAC → multiple filters or multiple inputs or multiple parts of inputs (sliding window/Im2Col) at the same time
  - Size of the MAC may be different
    - E.g., it could do 16-way dot product at a time → less partial sum read/writes
  - Need more buffer bandwidth to support this
- Buffer is not large enough → will have to read write from main memory during CONV
  - This is the common case and the hard problem to optimize as T(MEM) > 10X TBUF > 2X TMAC → once your read something from memory, *reuse* it as much as you can before you kick it out
    - Total parameter + activation storage requirements: 10KB – 10TB
    - On-chip SRAM: 10KB – 10 MB
- MACs and their local buffers (i.e., a processing element or PE) may be "distributed" with a network connecting them → NoC congestion, latency….
- More levels of memory hierarchy
  - Small SRAM + Large SRAM/Flash + DRAM + HDD
- Many more….

Logistics

- Colab + Pytorch short tutorial now by Shurui Li
- Project 1 to be announced end of next week.
- If you request a PTE, clearly mention
  - Your UID
  - Which courses have you taken in computer architecture, ML, VLSI to qualify
  - You know Verilog and Python

5 min break