MODERN NEURAL NETWORKS AND THEIR COMPUTATIONAL CHARACTERISTICS

Paulius Micikevičius, NVIDIA

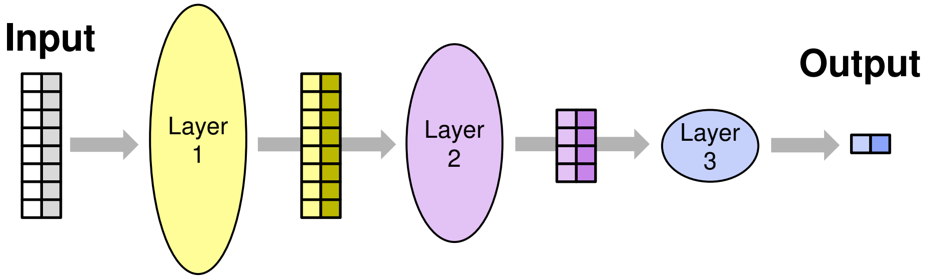
HotChips 2021, Tutorial: ML Performance

Outline

- **Overview and input data categories**
- **Network for various data types**
    - Unstructured data
    - Grid data
    - Sequence data
    - Graph data
- **Summary**

High Level View

- **Neural Networks are composed of "layers"**
- **Data "flows" through the layers**
  - Fwd and bwd for training
  - Fwd for inference
  - Data: vectors/matrices
- **Each layer:**
  - Consumes the inputs
  - Performs some operation (may have it own params/weights)
  - Produces the outputs



Layer Compute Categories

- **Dot-product based (will be referred to as MatMuls)**
  - Output element involves a dot-product
  - Matrix multiplies, also known as: Linear, fully connected, projection, …
  - Convolutions
- **Reductions**
  - Output element involves reducing (accumulating) values over some dimension(s) of a tensor
  - Examples: sum, norm (sum of squares), mean
  - Most normalization (Batch Normalization, Layer Normalization, …) include this primitive
- **Element-wise**
  - Output element depends on a corresponding element in input tensor(s)
  - Non-linearities: ReLU, GeLU, Sigmoid, …
  - Add (add two tensors point-wise), …

Network Input Data Categories

- **Regular structure:**
  - 1D (sequences): text, audio, stock price over time, temperature over time, …
  - 2D: images, …
  - 3D: MRI data, …
- **Irregular/unstructured:**
  - Graphs
  - Point clouds
  - Sets of attributes

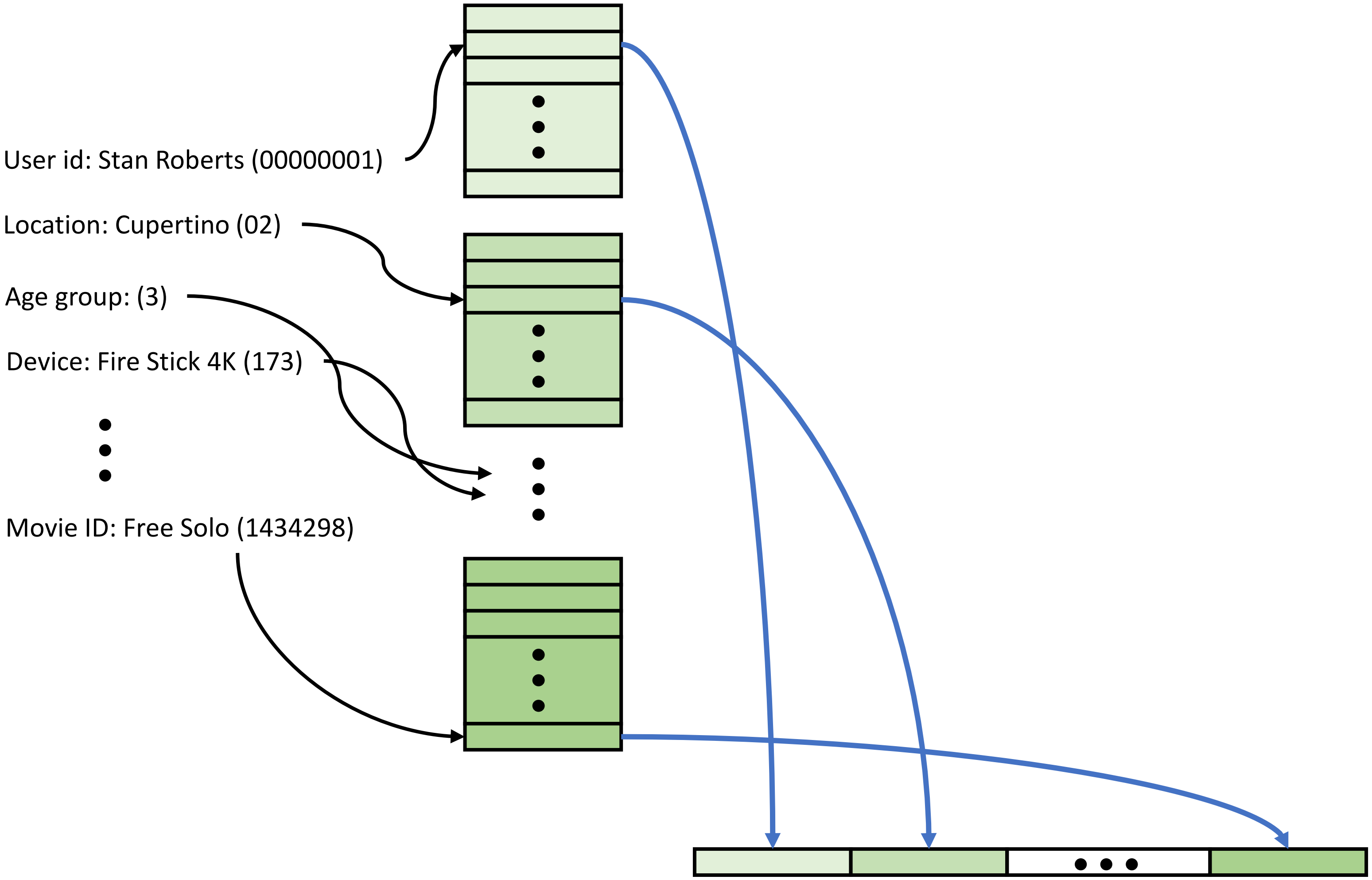
Unstructured Data

Unstructured Data: Set of Attributes

- **Input:**
  - a vector is formed from all the attributes
- **Network:**
  - MLP: a sequence of fully connected layers
  - Example application: recommenders
- **Typical output:**
  - Scalar value: relevance, probability, …
  - Vector: embedding vector, for finding similar items (proximity in vector space)

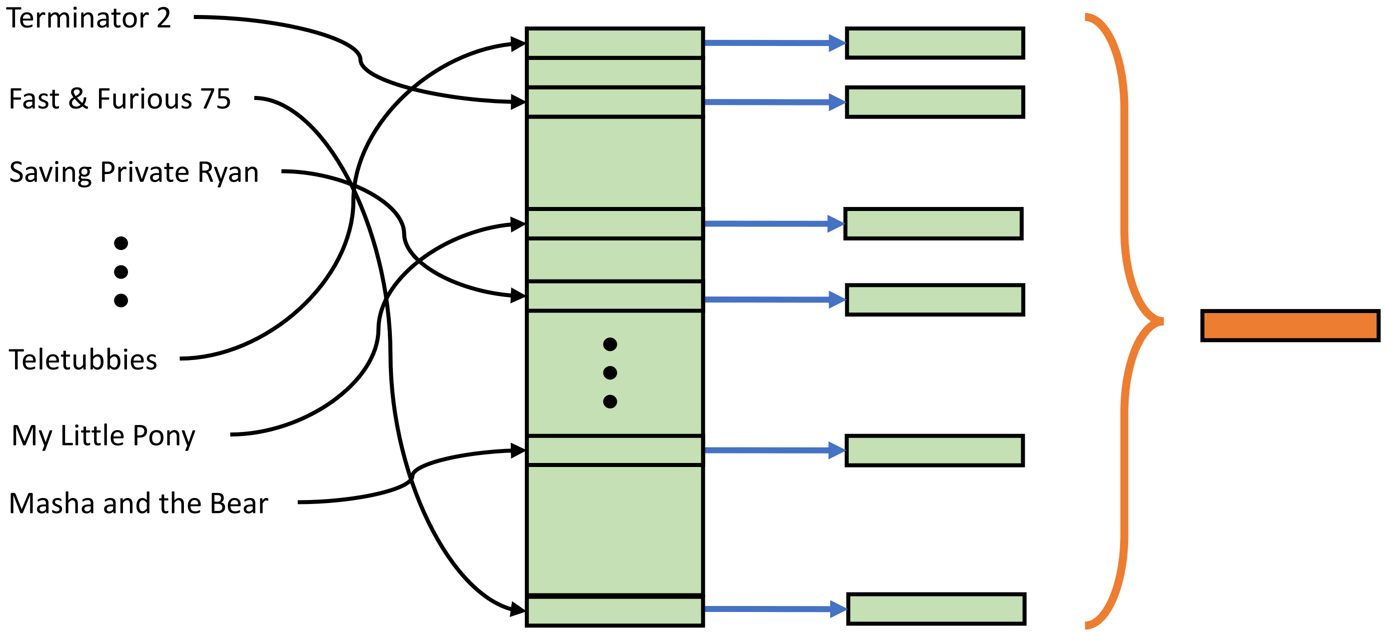
Two Types of Attributes for Input

- **Numerical**
  - May undergo some normalization, become input vector elements
  - Examples: connection bandwidth, weight, price, …
- **Categorical**
  - Examples: user age group, movie ID, …
  - Do not have a numerical meaning -> must be *embedded* to become vectors
  - Embedding: table lookup using category ID
    - Table values are learned
    - Table per attribute (sometimes also known as *feature*)


User id: Stan Roberts (00000001)
Location: Cupertino (02)
Age group: (3)
Device: Fire Stick 4K (173)
Movie ID: Free Solo (1434298)

Multi-Hot Features

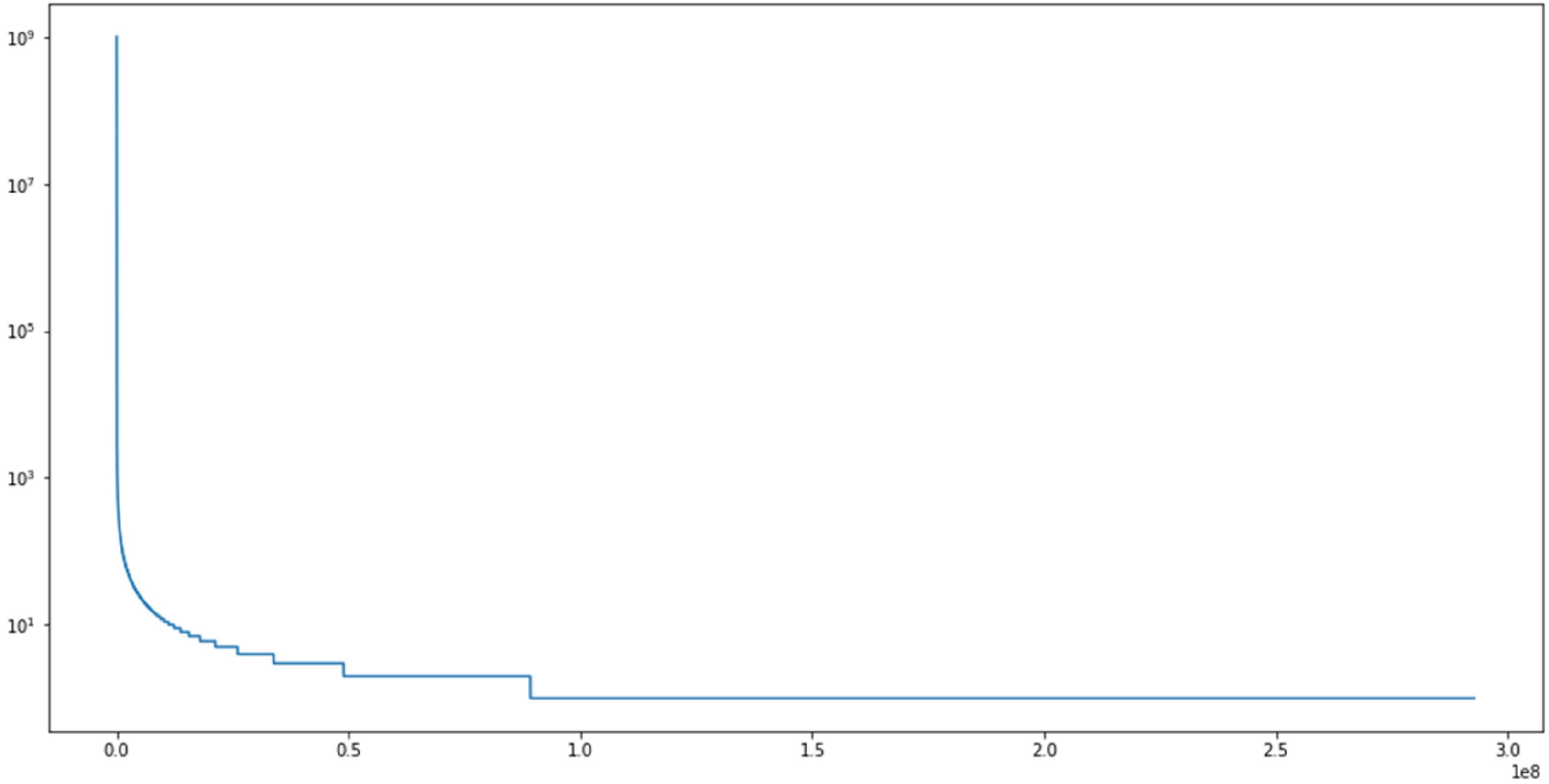
- **Typically a history of *N* items**
- **General operation:**
  - Read *N* vectors from the table
  - Combine element-wise (for example, avg) into a 1 output vector



Recommender Embedding Characteristics

- **Number of reads per write: 1 to 100s**
- **Memory bandwidth limited operations: few math ops per byte accessed**
- **Size varies widely: 100s of MB to 10s of TB**
  - Number of tables (features): 10s to 100s
  - Rows per table: 10s to billions
  - Columns per table (vector dimension): ~10 to 100s
- **Large sizes:**
  - Require high address translation rates
  - Exceed single accelerator memory (10s to ~100 GB), but can be accommodated:
    - Frequency of row accesses tends to follow a power-law like distribution
    - Few frequent items, long tail of very infrequent items
    - Good match for memory hierarchies (that include large host memories, etc.)

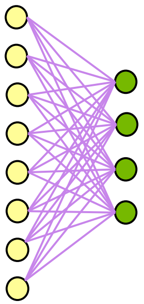
Histogram of Criteo 1TB Feature-19 Categories

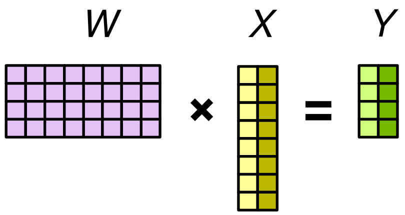


- X-axis: category IDs (~300M)
- Y-axis: number of occurrences in the dataset (note that axis is log10)

Recommender MLP

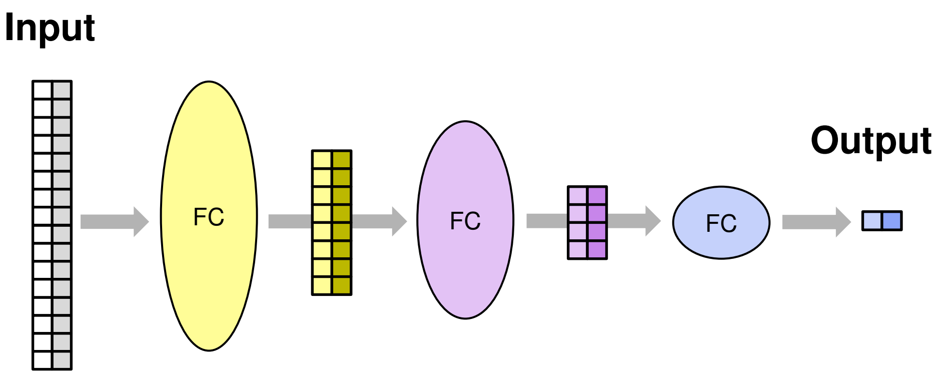
- **Embedding outputs are put through sequences of fully-connected (FC) layers**
  - Also known as Multi Layer Perceptrons (MLP)
- **Each FC layer:**
  - Connects all its input elements to all output elements
    - Fully-connected because there is no known structure or spatial relationship in the data
  - Implemented as a matrix-matrix multiply
    - An KxN matrix of weights projects K-element input vector to N-element output vector
    - Typically operates on a batch of M samples, for training and inference -> [M, K] x [K, N] matmul
- **Each FC layer is typically followed by:**
  - Non-linearity (ReLU, sigmoid, …)
  - Sometimes a normalization (Batch Norm, …)





Recommender MLP

- **Layers often form a "tower": width narrows towards the output**
  - Depths vary in 3-10 range
  - Sample small config: 256-128-64
  - Sample larger config: 2048-1024-512-256



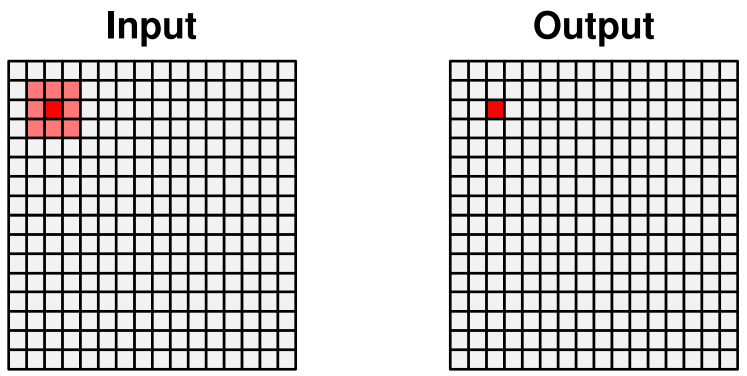
Grid Data

2D Grid Data

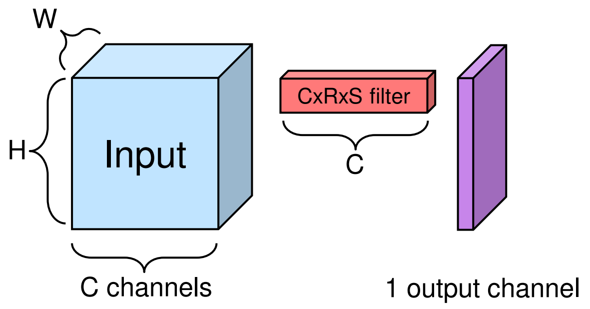
- **Input:**
  - 2D image, can be thought of as:
    - 3D tensor: height x width x channels
    - 2D matrix of vectors: height x width x channels
- **Network:**
  - CNN: sequence of convolutions
  - Transformer (attention-based)
- **Output:**
  - Vector: probabilities for different classes, embedding vector for image search, …
  - Multiple vectors: bounding boxes (2D, 3D) for object detection, …
  - 2D image: segmentation, style transfer, denoising, upscaling, …

Convolution Layer

- **Key observation: there is a spatial relationship among data elements**
- **Convolution takes advantage of this observation:**
  - Each output value is computed from a small (7x7, 5x5, 3x3) neighborhood in the input
    - As opposed to an FC layer, which uses all of the input
  - By stacking multiple convolutions more of input influences output elements
- **Convolution params: K, C, R, S**
  - R, S: height, width of the filter
  - C number of input channels
  - K filters, each produces a single output channel (plane)

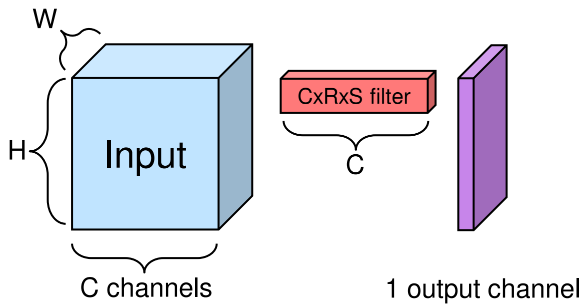


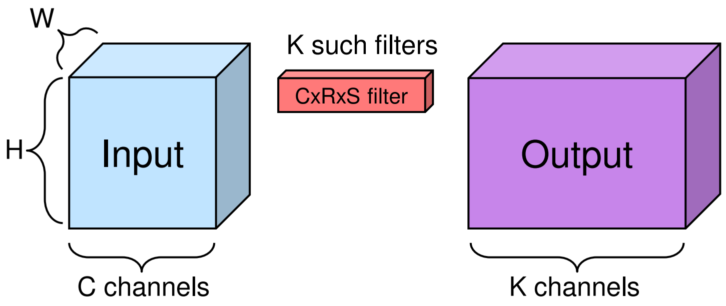
Convolution Layer



- **Input is a 2D grid of vectors**
  - Height x Width x Channels
    - Typical input image has 3 channels (red, green, blue)
    - Channel count typically increases for deeper layers
- **Often batch size is greater than 1**
  - Input becomes a 4D tensor: N, H, W, C

Convolution Layer

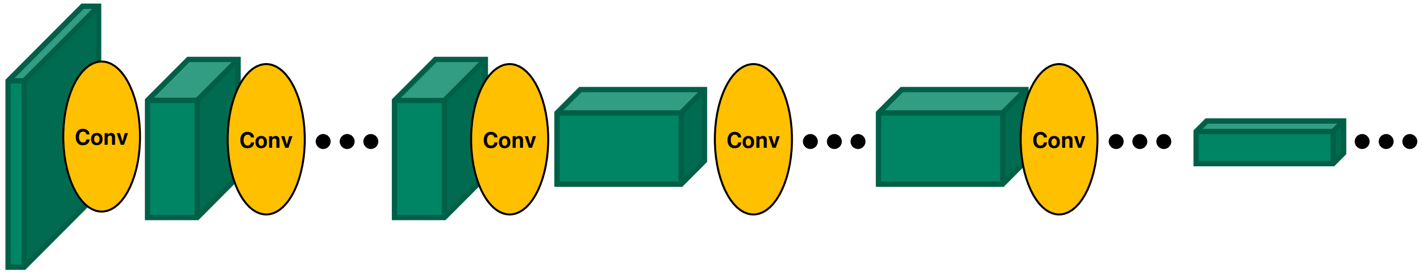




- **Input is a 2D grid of vectors**
  - Height x Width x Channels
    - Typical input image has 3 channels (red, green, blue)
    - Channel count typically increases for deeper layers
- **Often batch size is greater than 1**
  - Input becomes a 4D tensor: N, H, W, C

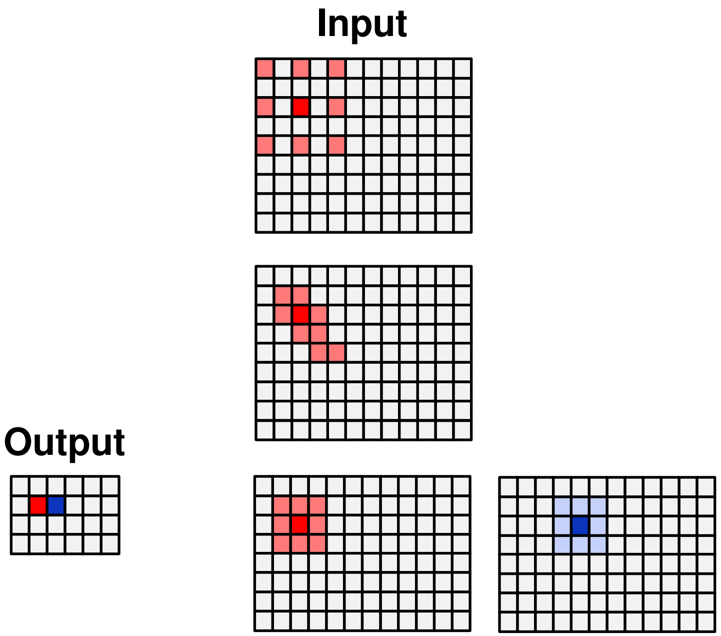
CNNs Typically Consist of Several Stages

- **Stage: sequence of repeated blocks**
  - Each stage typically decreases spatial resolution, increases channel count
- **Block:**
  - Sequence of convolutions (and other layers) with the same WxH
  - Often ~3 convolutions, each followed by a normalization (BN, LN, …) and non-linearity (ReLU, …)



Convolution: Some Spatial Variants

- **Diluted convolution**
  - Filter “taps” are spaced out D elements apart
  - Increases the receptive field at constant flops
- **Deformable convolution**
  - Learns through training how to space out filter “taps”
- **Strided convolution:**
  - Filters are applied at stride U
  - Reduces output resolution by a factor of U
- **Deconvolution:**
  - Inverse of strided conv – increases resolution

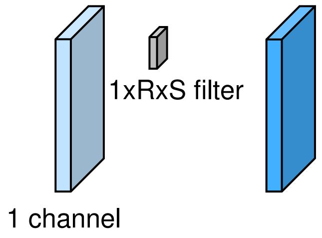


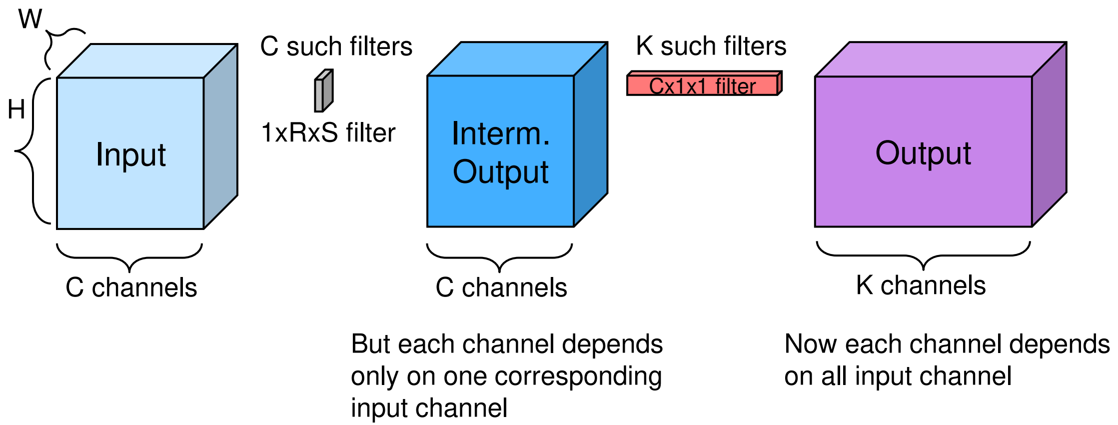
Convolution: Some Channel Variants

- **“Traditional” convolution**
  - Parameters: KCRS
  - Multiply adds: KCRSHW
- **Depth-separable convolution**
  - Parameters: CRS + KC
  - Multiply adds: CRSHW + KCHW
- **Grouped convolution, G groups:**
  - Parameters: KCRS/G
  - Multiply adds: KCRSHW/G

Depthwise Separable Convolution

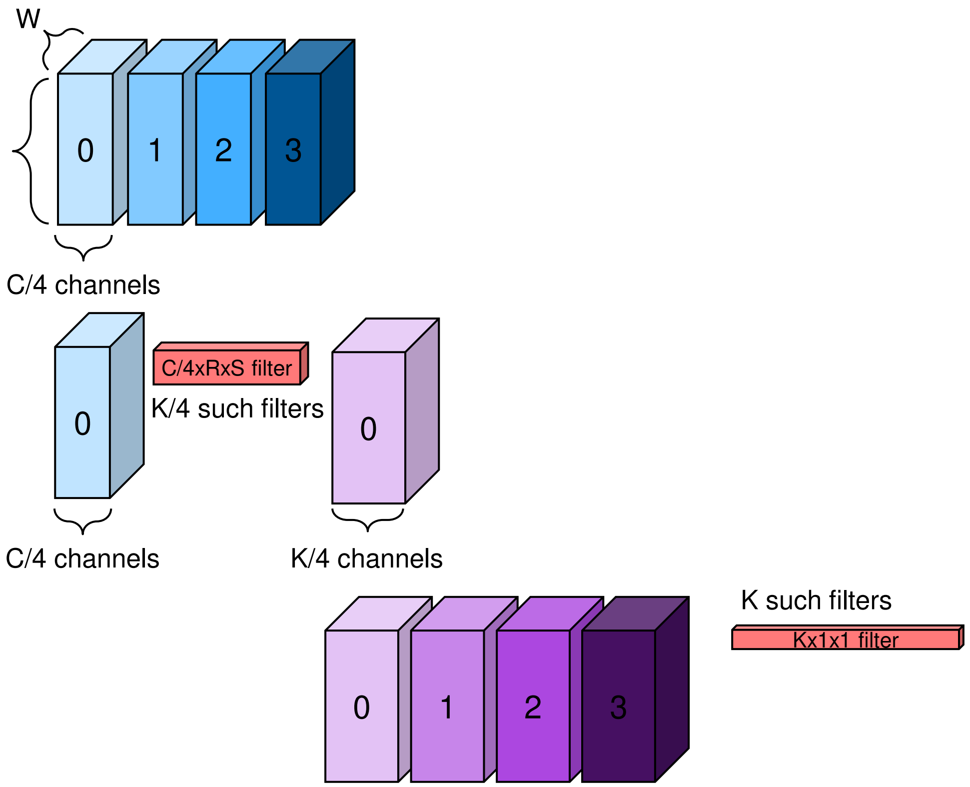
- **Introduced in MobilNets**
- **Sequence of 2 convolutions:**
  - RxS, one per input channel
  - Cx1x1, to “mix” channel data
- **Parameter count: CRS + CK**
- **Multiply adds: CRSHW + CKHW**





Grouped Convolution

- **Partition the input channels into G groups**
  - Each one with C/G channels
- **For each of G groups:**
  - Apply its K/G filters, each is (C/G)RS
- **Concatenate the G outputs**
- **Parameter count: KCRS/G**
- **Multiply adds: KCRSHW/G**
- **Usually followed by a 1x1 convolution to "mix" the channels**
  - Note that depth-wise separable convolution is a special case: G = C



CNN Characteristics

- **Layer types:**
  - Matmuls: various convolutions
  - Reductions: normalizations, pooling, softmax
  - Point-wise: non-linearities, adds (for skip connections)
- **Parameter sizes:**
  - 10s to 100s of MB
  - 10s to 100s of layers
  - Channels: 100s to 1,000s
- **Input dimensions:**
  - 1,000s to millions of pixels
- **Arithmetic intensity varies**

Note on 3D Grid Data

- **Inputs: 3D “images”**
- **Same ideas as for 2D data:**
  - Convolutions, etc.
  - Dimensionality is increased by 1: “2d convolutions” -> “3d convolutions”

Sequence Data

Sequence Data

- **Input:**
  - Sequence of “tokens” (vectors)
- **Network:**
  - Recurrent nets (LSTM, GRU, …)
  - Transformers (attention based)
- **Output:**
  - Sequence of vectors (language transloation, …)
  - Vector of probabilities (language models, …)

Input Data

- **Sequence of tokens:**
  - Words or word pieces
  - Do not have inherent numerical meaning -> must be embedded
- **Language embeddings:**
  - 1-hot look up tables (1 read per write)
  - 1 table (2 for translation), compared to recommender 10s-100s
  - Up to ~50,000 rows, compared to recommender up to billions
  - Embedding vector size: 100s to 1,000s of elements (compared to recommender 10s-100s)

- **Note on vision transformers:**
  - No embedding table
  - Input sequence items are vectors extracted from one of intermediate layers of a CNN

Simplified View of (Dot) Attention

- **For each of N tokens, each represented by a vector:**
  - Compute *key*, *query*, *value* vectors: each is a matrix-vector multiply
  - Compute dot-products of the *query* vector with all N items' *key* vectors: matrix-vector multiply
  - Compute "relevance" of each of the N items: softmax over the vector from above
    - After softmax the N "relevance" scores sum to 1
  - Compute a new vector: sum all N items' vectors, weighted by their relevance: matrix-vector

Simplified View of (Dot) Attention

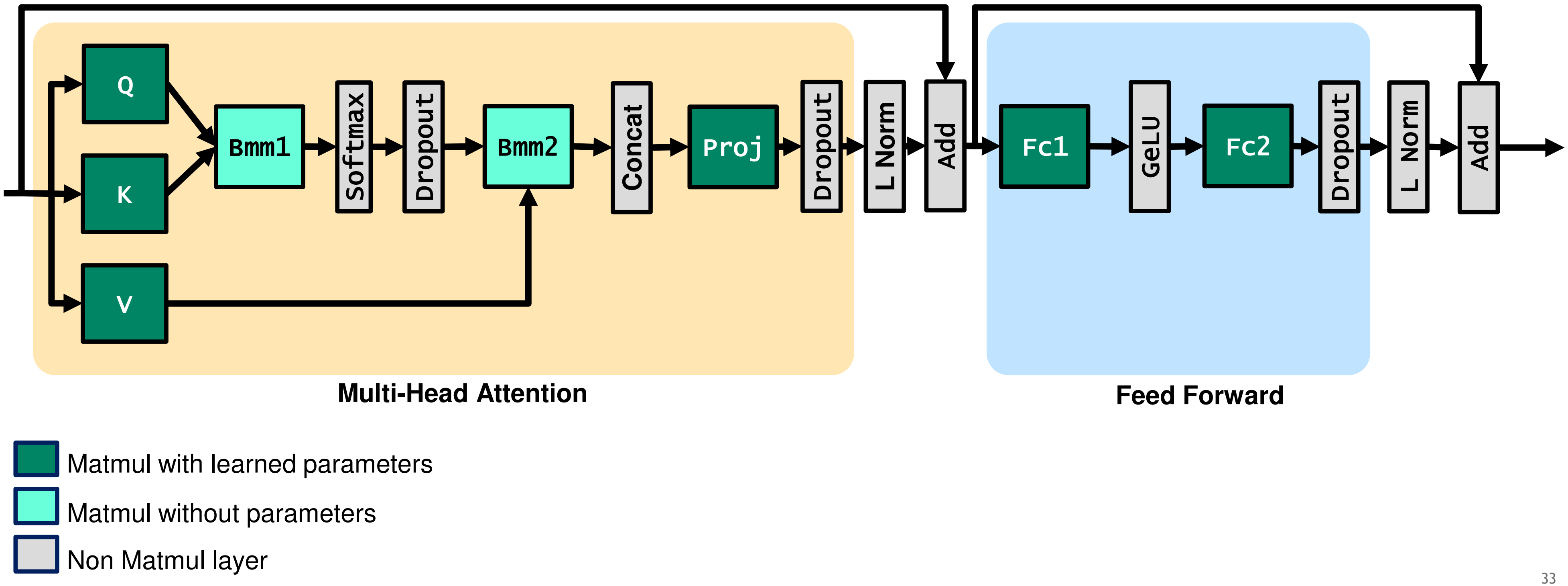
- **For each of N items, each represented by a vector:**
  - Compute *key*, *query*, *value* vectors: each is a matrix-vector multiply
  - Compute dot-products of the *query* vector with all N items' *key* vectors: matrix-vector multiply
  - Compute "relevance" of each of the N items: softmax over the vector from above
    - After softmax the N "relevance" scores sum to 1
  - Compute a new vector: sum all N items' vectors, weighted by their relevance: matrix-vector
- **Combining work for all N items in a sequence for efficiency:**
  - Matrix-matrix multiply: *query*, *key*, *value* projection
  - Matrix-matrix multiply: dot-products of N *query* vectors with N *key* vectors
  - Batched softmax: compute N relevance vectors
  - Matrix-matrix multiply: N weighted sums, each of N vectors

Simplified View of (Dot) Attention

- **For each of N items, each represented by a vector:**
  - Compute *key*, *query*, *value* vectors: each is a matrix-vector multiply
  - Compute dot-products of the *query* vector with all N items' *key* vectors: matrix-vector multiply
  - Compute "relevance" of each of the N items: softmax over the vector from above
    - After softmax the N "relevance" scores sum to 1
  - Compute a new vector: sum all N items' vectors, weighted by their relevance: matrix-vector
- **Combining work for all N items in a sequence for efficiency:**
  - Matrix-matrix multiply: *query*, *key*, *value* projection
  - Matrix-matrix multiply: dot-products of N *query* vectors with N *key* vectors
  - Batched softmax: compute N relevance vectors
  - Matrix-matrix multiply: N weighted sums, each of N vectors
- **Batch of more than 1 sequence further increases matrix sizes or batch for BMMs**

Transformer Building Block

- Input/output for a block: batch_size x seq_len x hidden_size

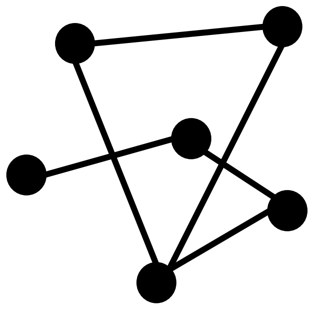


Characteristics

- **Layer types:**
  - Matmuls: matrix-multiplies, batched matrix multiplies
  - Reductions: normalizations (LN, …), softmax
  - Point-wise: adds (skip connections), nonlinearities (GeLU, ReLU, …)
  - Embeddings: 1-hot, relatively small (30K – 50K entries)
- **Parameter sizes:**
  - 10s to ~100 Transformer-like blocks
  - Hidden size (vector size per item): 100s to ~20,000 elements
  - Model sizes: 100s MB to ~1 TB
- **Input Dimensions:**
  - Sequence lengths: 100s to 1,000s of items
  - Batch sizes: 1000s of sequences (training on many accelerators)
- **High arithmetic intensity:**
  - Matrix dimensions for parameter matmuls: 100s to 1,000s
  - Dimensions for BMMs: 100s

Graph Data

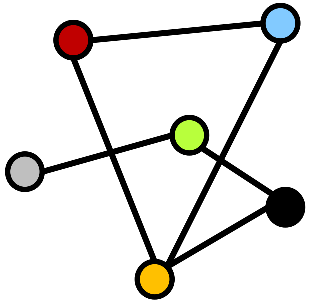
- **Graph examples:**
  - Social graphs:
    - Users are nodes
    - Edges indicates two users are connected
  - Conference attendance
    - Nodes are people and conferences
    - Edge indicates attendance at a conference

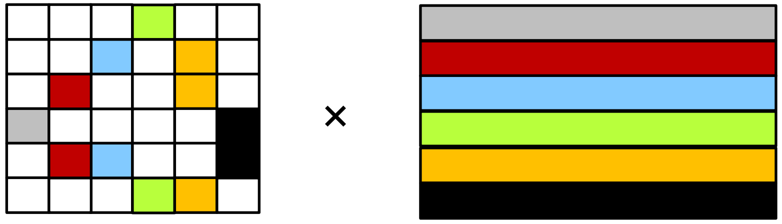


- **Graph tasks:**
  - Edge prediction
  - Node embedding (for searches)
    - compute N-element vectors for nodes, so that “similar” nodes are close in N-dimensional space
  - Node classification (fraud detection, etc.)
  - …

Graph Convolutional Neutworks

- **Mostly layers we have seen before:**
  - Matrix multiplies, non-linearities, adds, normalizations
- **A new one: sparse-dense matrix multiply (aggregating neighbor node vectors)**
  - Dense matrix: vectors for all the nodes in a subgraph
  - Sparse matrix: adjacency matrix for the subgraph
  - Multiplication yields a sum of neighbors' vectors





Characteristics

- **Layer types:**
  - Matmul: matrix multiplies, sparse matrix multiplies
  - Reduction: vector norms
  - Point-wise: add, concat, non-linearities
- **Online data (subgraph) preparation**
  - Sampling the full graph (too large to process all nodes/edges at once)
    - Random walks, sparse adjacency matrix preparations, etc.
  - Time taken can be comparable to neural network fwd/bwd pass
  - Often distributed over multiple compute nodes

Summary

Summary

- **Neural network types usually depend on the input data category**
  - Unstructured data: MLPs
  - Regular grid data: CNNs, CNN+Transformers
  - Sequence data: Transformers (attention nets), RNNs
  - Irregular/graph data: GNNs
- **Operation types:**
  - Matmuls: matrix multiplies (aka FC, Linear layers), BMMs, convolutions, sparse-matrix multiplies
  - Reductions: softmax, normalizations, norms, …
  - Point-wise: non-linearities, adds, concats, …
  - Look up tables: for categorical data
    - Large to very large tables for recommenders, small to medium tables for language networks
  - Skip connections are popular in CNNs and Transformers
  - New operation variants keep getting invented
- **Model size and arithmetic intensity varies**