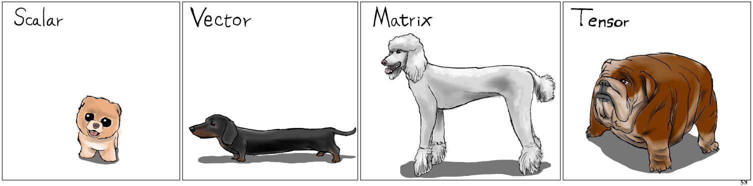
Role of Tensors in Deep Learning

Anima Anandkumar

Principal Scientist

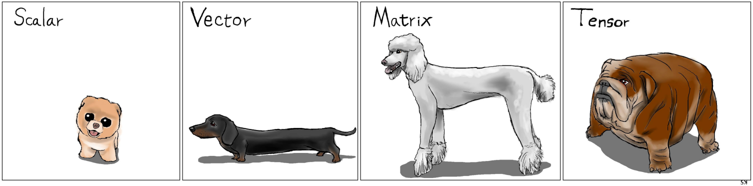
Amazon AI

Tensors: Beyond 2D world

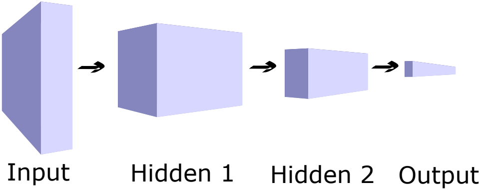


Modern data is inherently multi-dimensional

Tensors: Beyond 2D world



Modern data is inherently multi-dimensional

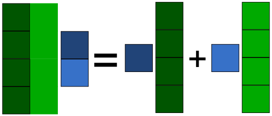


Tensor Contraction

Extends the notion of matrix product

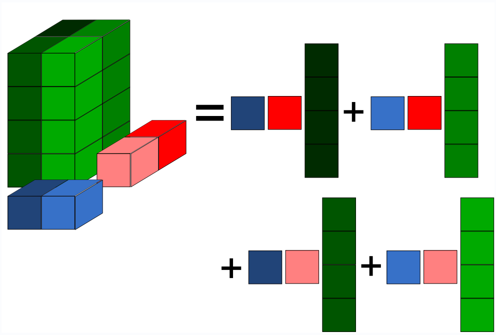
Matrix product

$$Mv = \sum_j v_j M_j$$

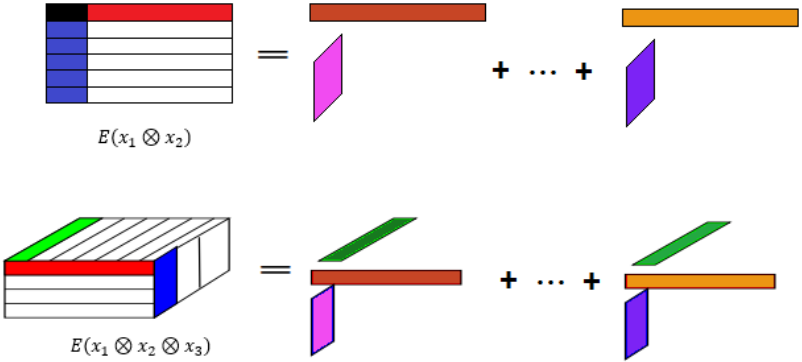


Tensor Contraction

$$T(u, v, \cdot) = \sum_{i,j} u_i v_j T_{i,j,:}$$

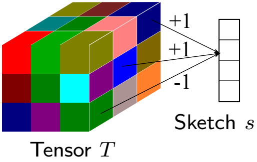


Tensor Decompositions



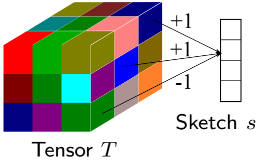
Tensor Sketches

- Randomized dimensionality reduction through sketching.
  - Complexity independent of tensor order: exponential gain!



Tensor Sketches

- Randomized dimensionality reduction through sketching.
  - Complexity independent of tensor order: exponential gain!

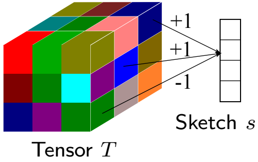


Applications

- Tensor Decomposition via Sketching by Wang, Tung, Smola, A. NIPS'15.
- Compact Tensor Pooling for Visual Question and Answering by Shi, Anubhai, Furlanello, A, CVPR 2017 VQA workshop. **Poster, this afternoon 2:30-4**.

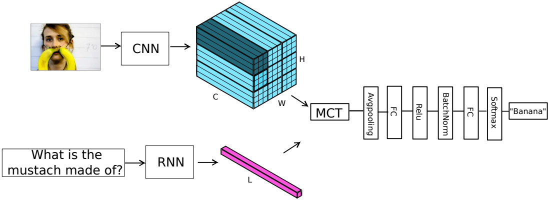
Tensor Sketches

- Randomized dimensionality reduction through sketching.
  - Complexity independent of tensor order: exponential gain!



Applications

- Tensor Decomposition via Sketching by Wang, Tung, Smola, A. NIPS‘15.
- Compact Tensor Pooling for Visual Question and Answering by Shi, Anubhai, Furlanello, A, CVPR 2017 VQA workshop. **Poster, this afternoon 2:30-4**.



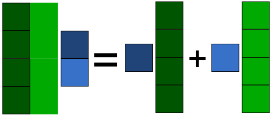
Outline

Tensor Contraction

Extends the notion of matrix product

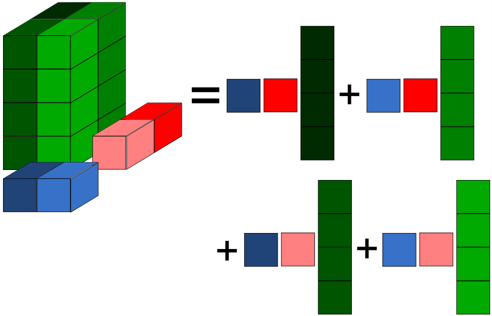
Matrix product

$$Mv = \sum_j v_j M_j$$

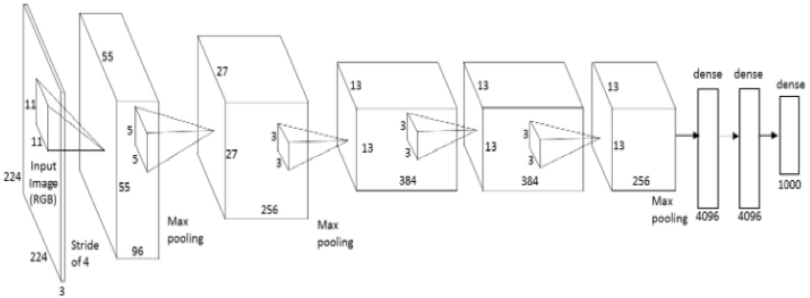


Tensor Contraction

$$T(u, v, \cdot) = \sum_{i,j} u_i v_j T_{i,j,:}$$

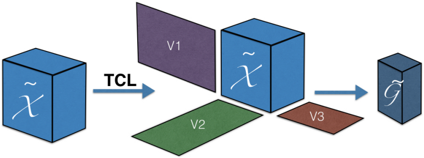


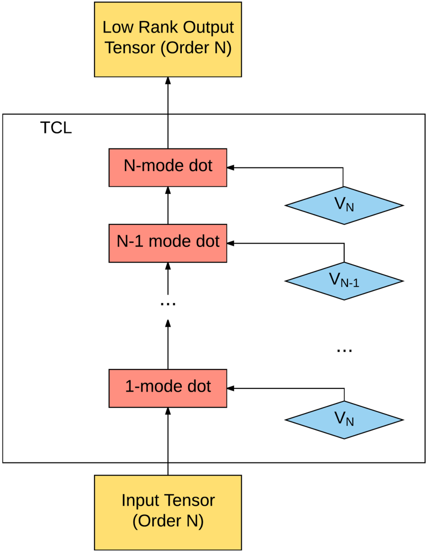
Employing Tensor Contractions in Alexnet



Replace fully connected layer with tensor contraction layer

Enabling Tensor Contraction Layer in Mxnet





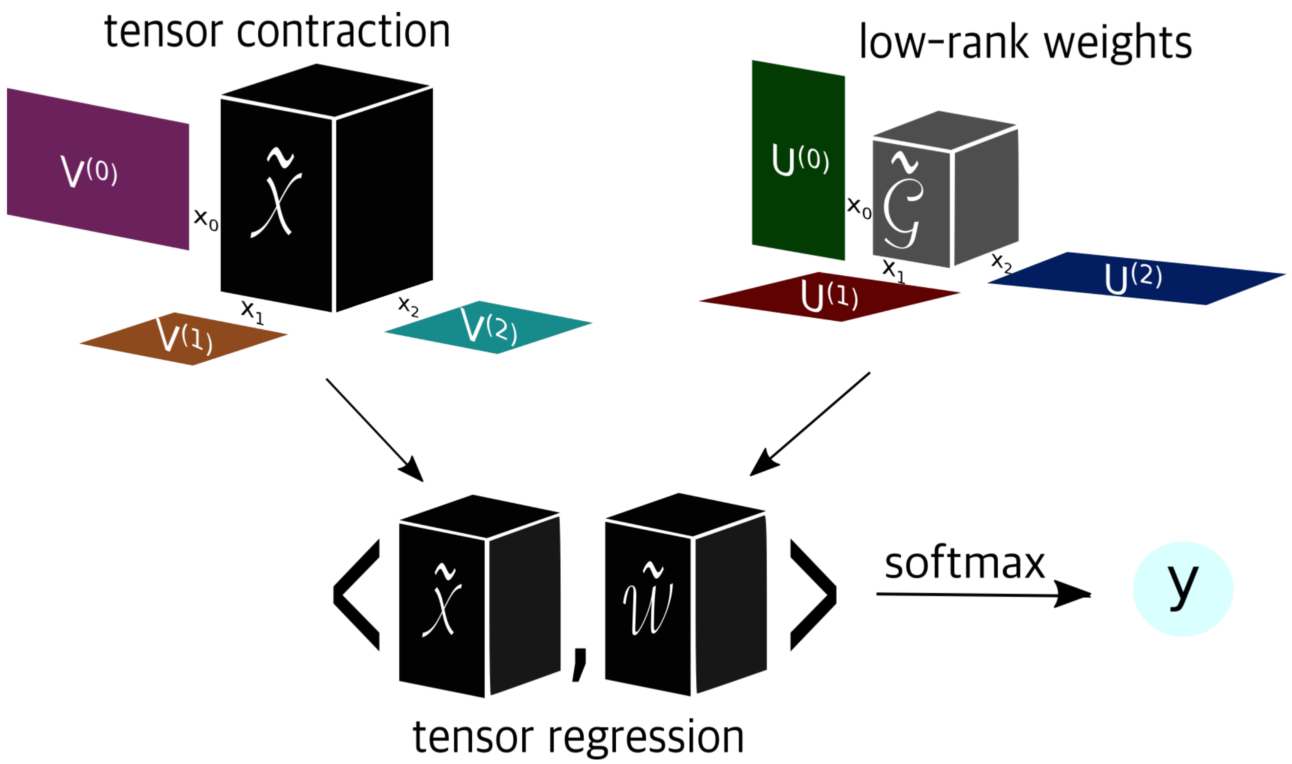
Performance of the TCL

- Trained end-to-end

- On ImageNet with VGG:
  - 65.9% space savings
  - performance drop of 0.6% only

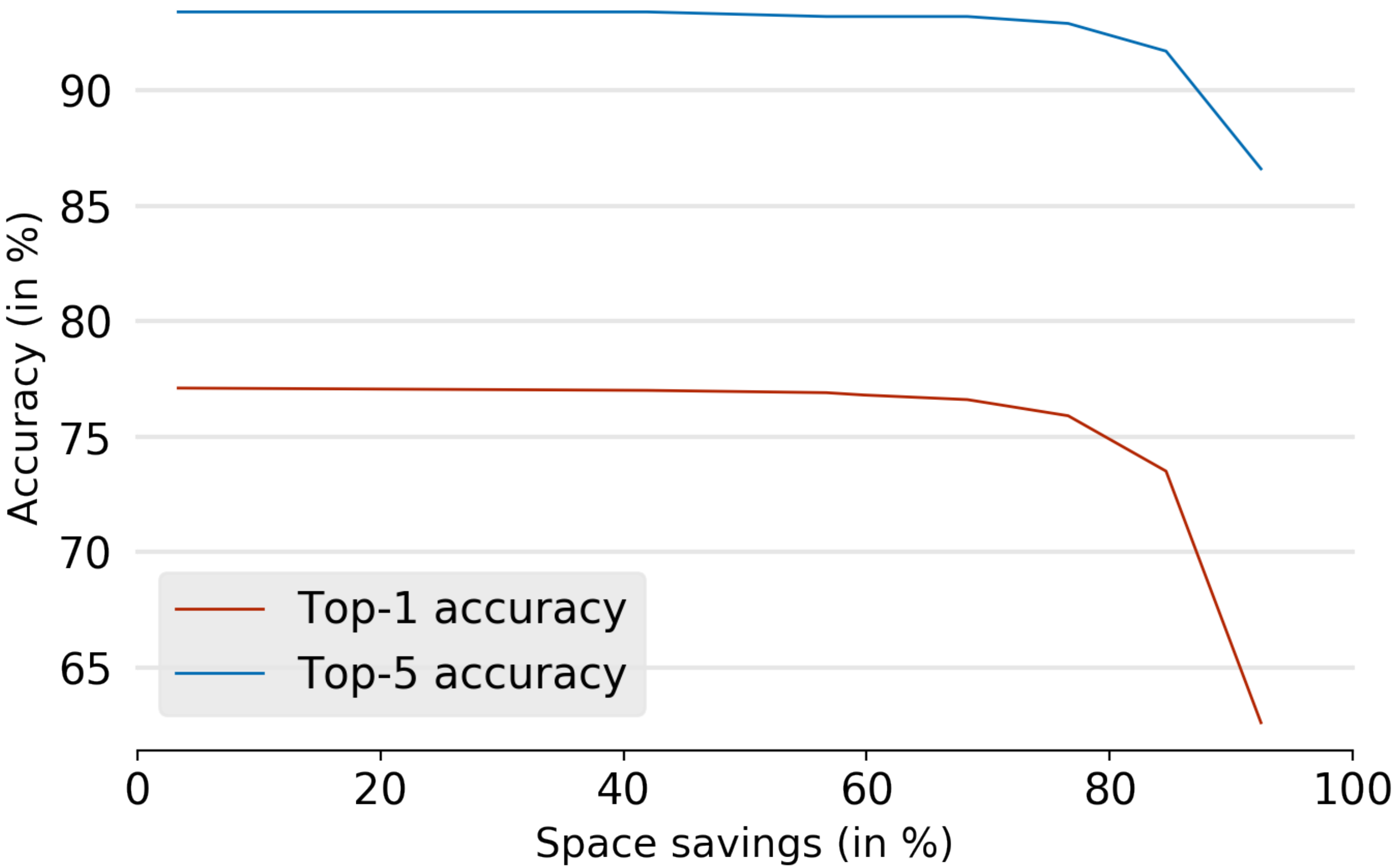
- On ImageNet with AlexNet:
  - 56.6% space savings
  - Performance improvement of 0.5%

Low-rank tensor regression



Tensor Regression Networks, *J. Kossaifi, Z.C.Lipton, A.Khanna, T.Furlanello and A.Anandkumar*, ArXiv pre-publication

Performance and rank



Outline

Speeding up Tensor Contractions

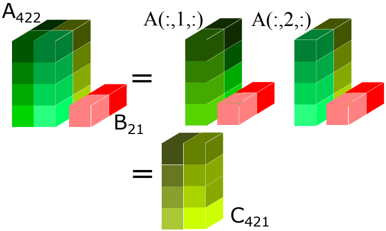
1. Tensor contractions are a core primitive of multilinear algebra.
2. BLAS 3: Unbounded compute intensity (no. of ops per I/O)

Consider single-index contractions: $C_{\mathcal{C}} = A_{\mathcal{A}} \, B_{\mathcal{B}}$

Speeding up Tensor Contractions

1. Tensor contractions are a core primitive of multilinear algebra.
2. BLAS 3: Unbounded compute intensity (no. of ops per I/O)

Consider single-index contractions: $C_{\mathcal{C}} = A_{\mathcal{A}} \, B_{\mathcal{B}}$



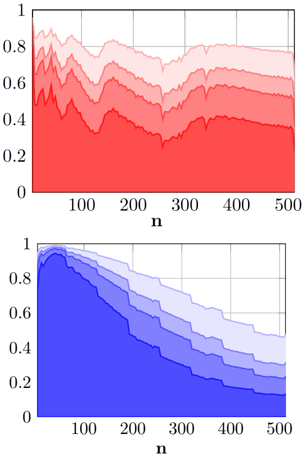
e.g. $C_{mnp} = A_{mnk} \, B_{kp}$

Speeding up Tensor Contraction

Explicit permutation dominates, especially for small tensors.

Consider $C_{mnp} = A_{km}\, B_{pkn}$.

1. $A_{km} \rightarrow A_{mk}$
2. $B_{pkn} \rightarrow B_{kpn}$
3. $C_{mnp} \rightarrow C_{mpn}$
4. $\boxed{C_{m(pn)} = A_{mk}\, B_{k(pn)}}$
5. $C_{mpn} \rightarrow C_{mnp}$



(Top) CPU. (Bottom) GPU. The fraction of time spent in copies/transpositions. Lines are shown with 1, 2, 3, and 6 transpositions.

Existing Primitives

GEMM

- Suboptimal for many small matrices.

Pointer-to-Pointer BatchedGEMM

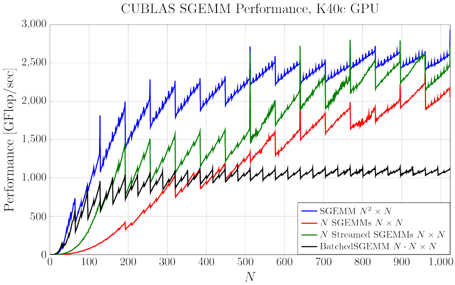
- Available in MKL 11.3β and cuBLAS 4.1

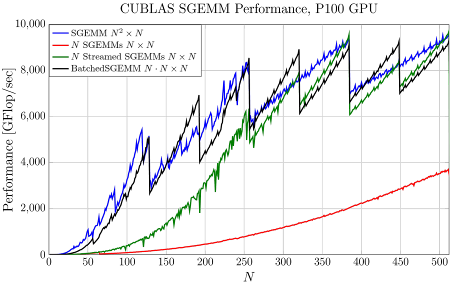
$$C[p] = \alpha \, \mathsf{op}(A[p]) \, \mathsf{op}(B[p]) + \beta \, C[p]$$

```
cublas<T>gemmBatched(cublasHandle_t handle,
                     cublasOperation_t transA, cublasOperation_t transB,
                     int M, int N, int K,
                     const T* alpha,
                     const T** A, int ldA,
                     const T** B, int ldB,
                     const T* beta,
                     T** C, int ldC,
                     int batchCount)
```

Existing Primitives

Pointer-to-Pointer BatchedGEMM





A new primitive: StridedBatchedGEMM

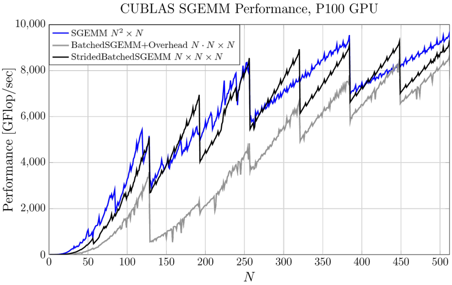
$$C[p] = \alpha \, \mathsf{op}(A[p]) \, \mathsf{op}(B[p]) + \beta \, C[p]$$

- Pointer-to-Pointer BatchedGEMM requires memory allocation and pre-computation.
- Solution: StridedBatchedGEMM with fixed strides.
  - Special case of Pointer-to-pointer BatchedGEMM.
  - No Pointer-to-pointer data structure or overhead.

```
cublas<T>gemmStridedBatched(cublasHandle_t handle,
                            cublasOperation_t transA, cublasOperation_t transB,
                            int M, int N, int K,
                            const T* alpha,
                            const T* A, int ldA1, int strideA,
                            const T* B, int ldB1, int strideB,
                            const T* beta,
                            T* C, int ldC1, int strideC,
                            int batchCount)
```

A new primitive: StridedBatchedGEMM

- Performance on par with pure GEMM (P100 and beyond).



StridedBatchedGEMM

Documentation in cuBLAS 8.0:

```
$$ grep StridedBatched -A 17 /usr/local/cuda/include/cublas_api.h
2320:CUBLASAPI cublasStatus_t cublasSgemmStridedBatched (cublasHandle_t handle,
2321-                                                      cublasOperation_t transa,
2322-                                                      cublasOperation_t transb,
2323-                                                      int m,
2324-                                                      int n,
2325-                                                      int k,
2326-                                                      const float *alpha,  // host or device pointer
2327-                                                      const float *A,
2328-                                                      int lda,
2329-                                                      long long int strideA,   // purposely signed
2330-                                                      const float *B,
2331-                                                      int ldb,
2332-                                                      long long int strideB,
2333-                                                      const float *beta,   // host or device pointer
2334-                                                      float *C,
2335-                                                      int ldc,
2336-                                                      long long int strideC,
2337-                                                      int batchCount);
...
```

Tensor Contraction with Extended BLAS Primitives

$$C_{mnp} = A_{**} \times B_{***}$$

$$C_{mnp} \equiv C[m + n \cdot \mathsf{ldC1} + p \cdot \mathsf{ldC2}]$$

Case	Contraction	Kernel1	Kernel2	Case	Contraction	Kernel1	Kernel2
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	4.1	$A_{kn}B_{kmp}$	$C_{mn[p]} = B_{km[p]}^{\top}A_{kn}$	
1.2	$A_{mk}B_{kpn}$	$C_{mn[p]} = A_{mk}B_{k[p]n}$	$C_{m[n]p} = A_{mk}B_{kp[n]}$	4.2	$A_{kn}B_{kpm}$	$C_{mn[p]} = B_{k[p]m}^{\top}A_{kn}$	
1.3	$A_{mk}B_{nkp}$	$C_{mn[p]} = A_{mk}B_{nk[p]}^{\top}$		4.3	$A_{kn}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A_{kn}$	
1.4	$A_{mk}B_{pkn}$	$C_{m[n]p} = A_{mk}B_{pk[n]}^{\top}$		4.4	$A_{kn}B_{pkm}$		
1.5	$A_{mk}B_{npk}$	$C_{m(np)} = A_{mk}B_{(np)k}^{\top}$	$C_{mn[p]} = A_{mk}B_{n[p]k}^{\top}$	4.5	$A_{kn}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A_{kn}$	
1.6	$A_{mk}B_{pnk}$	$C_{m[n]p} = A_{mk}B_{p[n]k}^{\top}$		4.6	$A_{kn}B_{pmk}$		
2.1	$A_{km}B_{knp}$	$C_{m(np)} = A_{km}^{\top}B_{k(np)}$	$C_{mn[p]} = A_{km}^{\top}B_{kn[p]}$	5.1	$A_{pk}B_{kmn}$	$C_{(mn)p} = B_{k(mn)}^{\top}A_{pk}^{\top}$	$C_{m[n]p} = B_{km[n]}^{\top}A_{pk}^{\top}$
2.2	$A_{km}B_{kpn}$	$C_{mn[p]} = A_{km}^{\top}B_{k[p]n}$	$C_{m[n]p} = A_{km}^{\top}B_{kp[n]}$	5.2	$A_{pk}B_{knm}$	$C_{m[n]p} = B_{k[n]m}^{\top}A_{pk}^{\top}$	
2.3	$A_{km}B_{nkp}$	$C_{mn[p]} = A_{km}^{\top}B_{nk[p]}^{\top}$		5.3	$A_{pk}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A_{pk}^{\top}$	
2.4	$A_{km}B_{pkn}$	$C_{m[n]p} = A_{km}^{\top}B_{pk[n]}^{\top}$		5.4	$A_{pk}B_{nkm}$		
2.5	$A_{km}B_{npk}$	$C_{m(np)} = A_{km}^{\top}B_{(np)k}^{\top}$	$C_{mn[p]} = A_{km}^{\top}B_{n[p]k}^{\top}$	5.5	$A_{pk}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A_{pk}^{\top}$	$C_{m[n]p} = B_{m[n]k}A_{pk}^{\top}$
2.6	$A_{km}B_{pnk}$	$C_{m[n]p} = A_{km}^{\top}B_{p[n]k}^{\top}$		5.6	$A_{pk}B_{nmk}$		
3.1	$A_{nk}B_{kmp}$	$C_{mn[p]} = B_{km[p]}^{\top}A_{nk}^{\top}$		6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B_{k(mn)}^{\top}A_{kp}$	$C_{m[n]p} = B_{km[n]}^{\top}A_{kp}$
3.2	$A_{nk}B_{kpm}$	$C_{mn[p]} = B_{k[p]m}^{\top}A_{nk}^{\top}$		6.2	$A_{kp}B_{knm}$	$C_{m[n]p} = B_{k[n]m}^{\top}A_{kp}$	
3.3	$A_{nk}B_{mkp}$	$C_{mn[p]} = B_{mk[p]}A_{nk}^{\top}$		6.3	$A_{kp}B_{mkn}$	$C_{m[n]p} = B_{mk[n]}A_{kp}$	
3.4	$A_{nk}B_{pkm}$			6.4	$A_{kp}B_{nkm}$		
3.5	$A_{nk}B_{mpk}$	$C_{mn[p]} = B_{m[p]k}A_{nk}^{\top}$		6.5	$A_{kp}B_{mnk}$	$C_{(mn)p} = B_{(mn)k}A_{kp}$	$C_{m[n]p} = B_{m[n]k}A_{kp}$
3.6	$A_{nk}B_{pmk}$			6.6	$A_{kp}B_{nmk}$		

Tensor Contraction with Extended BLAS Primitives

Case	Contraction	Kernel1	Kernel2	Kernel3
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	$C_{m[n]p} = A_{mk}B_{k[n]p}$
6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B_{k(mn)}^{\top}A_{kp}$	$C_{m[n]p} = B_{km[n]}^{\top}A_{kp}$	

Example: Mappings to Level 3 BLAS routines

- Case 1.1, Kernel2: $C_{mn[p]} = A_{mk}B_{kn[p]}$

```
cublasDgemmStridedBatched(handle,
                          CUBLAS_OP_N, CUBLAS_OP_N,
                          M, N, K,
                          &alpha,
                          A, ldA1, 0,
                          B, ldB1, ldB2,
                          &beta,
                          C, ldC1, ldC2,
                          P)
```

Tensor Contraction with Extended BLAS Primitives

Case	Contraction	Kernel1	Kernel2	Kernel3
1.1	$A_{mk}B_{knp}$	$C_{m(np)} = A_{mk}B_{k(np)}$	$C_{mn[p]} = A_{mk}B_{kn[p]}$	$C_{m[n]p} = A_{mk}B_{k[n]p}$
6.1	$A_{kp}B_{kmn}$	$C_{(mn)p} = B^{\top}_{k(mn)}A_{kp}$	$C_{m[n]p} = B^{\top}_{km[n]}A_{kp}$	

Example: Mappings to Level 3 BLAS routines

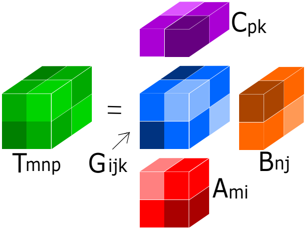
- Case 6.1, Kernel2: $C_{m[n]p} = B^{\top}_{km[n]}A_{kp}$

```
cublasDgemmStridedBatched(handle,
                          CUBLAS_OP_T, CUBLAS_OP_N,
                          M, P, K,
                          &alpha,
                          B, ldB1, ldB2,
                          A, ldA1, 0,
                          &beta,
                          C, ldC2, ldC1,
                          N)
```

Y. Shi, U.N. Niranjan, **A.**, C. Cecka "Tensor Contractions with Extended BLAS Kernels on CPU and GPU" in HiPC2016.

Applications: Tucker Decomposition

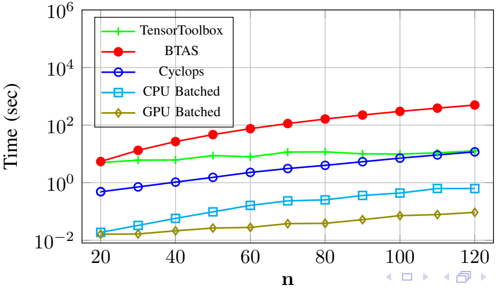
$$T_{mnp} = G_{ijk} A_{mi} B_{nj} C_{pk}$$



Main steps in the algorithm

- $Y_{mjk} = T_{mnp} B^t_{nj} C^t_{pk}$
- $Y_{ink} = T_{mnp} A^{t+1}_{mi} C^t_{pk}$
- $Y_{ijp} = T_{mnp} B^{t+1}_{nj} A^{t+1}_{mi}$

Performance on Tucker decomposition:



Applications: FFT

Low-Communication FFT for multiple GPUs involves tensor contractions.

$$
\begin{aligned}
T_{pib} &= S2T^{(p)}_{ijs}\, S_{pj(b+s)} && \Longrightarrow && T_{pib} = S2T^{(p)}_{i(js)}\, S_{p(js)b} \\
M_{pqb} &= S2M_{qi}\, S_{pib} && \Longrightarrow && M_{pq[b]} = S_{pi[b]}\, S2M^T_{qi} \\
M_{pqb'} &= M2M^-_{qm}\, M_{pmb^-} + M2M^+_{qm}\, M_{pmb^+} && \Longrightarrow && M_{pq[b']} = M_{pM[b]}\, M2M^T_{qM} \\
r_p &= 1_{ib}\, S_{pib} = 1_{qb}\, M_{pqb} && \Longrightarrow && r_p = 1_{(qb)}\, M_{p(qb)} \\
L_{pnb} &= M2L^{(p)}_{nms}\, M_{pm(b+s)} && \Longrightarrow && L_{pnb} = M2L^{(p)}_{n(ms)}\, M_{p(ms)b} \\
L_{pqb^\pm} &= L2L^\pm_{qm}\, L_{pmb'} && \Longrightarrow && L_{pq[b]} = L_{pM[b']}\, M2M_{qM} \\
T_{pib} &= L2T_{iq}\, L_{pqb} && \Longrightarrow && T_{pi[b]} = L_{pq[b]}\, S2M_{qi}
\end{aligned}
$$

- StridedBatchedGEMM composes 75%+ of the runtime.
  - Essential to the performance.
  - Two custom kernels are precisely interleaved GEMMs.

- 2 P100 GPUs: 1.3x over cuFFTXT.
- 8 P100 GPUs: 2.1x over cuFFTXT.

C. Cecka "Low-Communication FFT with Fast Multipole Method" GTC 2017.

Summary of this Section

- Tensor contractions provide significant space savings in deep learning.
- Fast tensor contractions using extended BLAS primitive: StridedBatchedGEMM
- Avoids explicit transpositions or permutations.
- **10x**(GPU) and **2x**(CPU) speedup on small/moderate sized tensors.
- **Available in cuBLAS 8.0**

Summary of this Section

- Tensor contractions provide significant space savings in deep learning.
- Fast tensor contractions using extended BLAS primitive: StridedBatchedGEMM
- Avoids explicit transpositions or permutations.
- **10x**(GPU) and **2x**(CPU) speedup on small/moderate sized tensors.
- **Available in cuBLAS 8.0**
- Future work:
  - Exceptional case kernels/performance/interface??
  - Library Optimizations
    - Matrix stride zero – Persistent Matrix Strided Batched GEMM

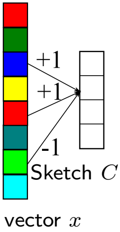
Outline

Tensor Sketching

- Dimensionality reduction through sketching.

Count Sketch for vector x

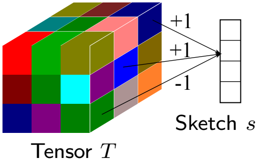
- $C[h[i]] += s[i]x[i]$, for $s[i] \in \{-1, +1\}^n$



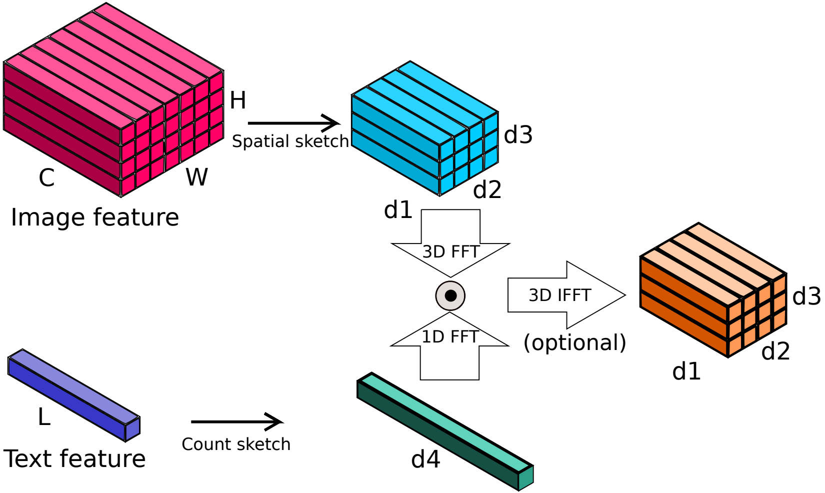
Count Sketch for outer products $x \otimes y$

- Convolution of count sketches

$$C(x \otimes y, h, s) = C(x, h, s) * C(y, h, s)$$
$$= FFT^{-1}(FFT(C(x, h, s))FFT(C(y, h, s)))$$

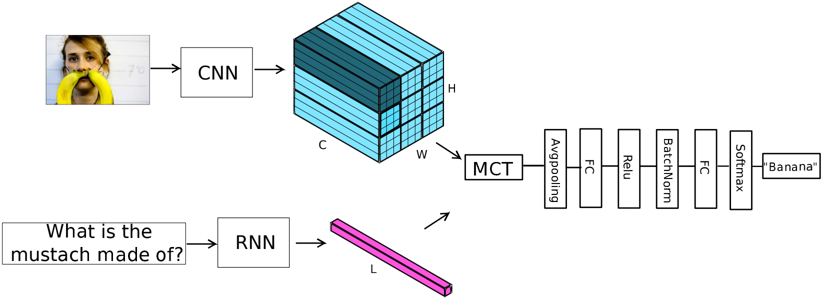


Multimodal Tensor Pooling



MCT in Visual Question & Answering

- More on this at the poster at VQA workshop **2:30-4pm today**



Outline

Conclusion

Tensors are the future of ML

- Tensor contractions: space savings in deep architectures.
  - More on this at spotlight talk by Jean Kossaifi at this workshop **2:10pm today**
- New primitives speed up tensor contractions: extended BLAS
- Tensor sketches compress efficiently while preserving information
- Tensor sketches enable multi-modal tasks such as visual Q&A
  - More on this at the poster at VQA workshop **2:30-4pm today**