Tensorizing Deep Learning

Hrriday Viraj Ruparel

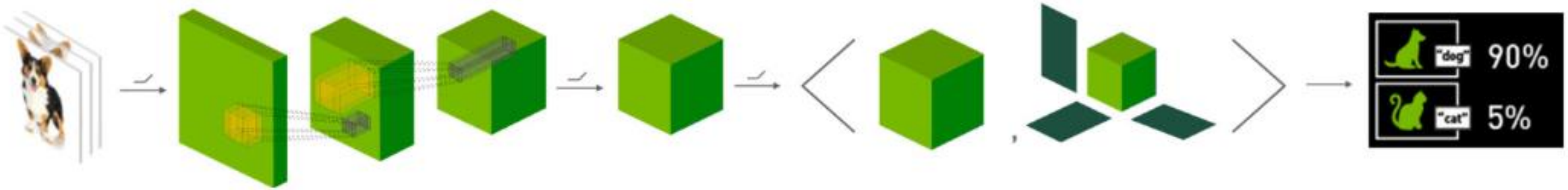
CS 618 – Theoretical Foundations of Machine Learning

IIT Gandhinagar

Date: 20th Nov 2024

Problem Statement

- Want to analyze how Tensors Operations and Tensor Decompositions contribute to the field of Deep Learning.
- Will be looking at: **Tensor-based compression of deep neural networks**
- Based on applications, will also introduce some common techniques of Tensor Decomposition

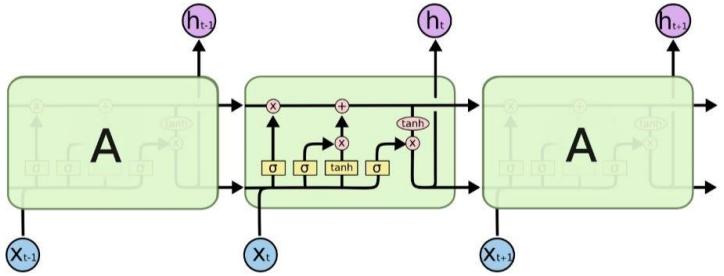


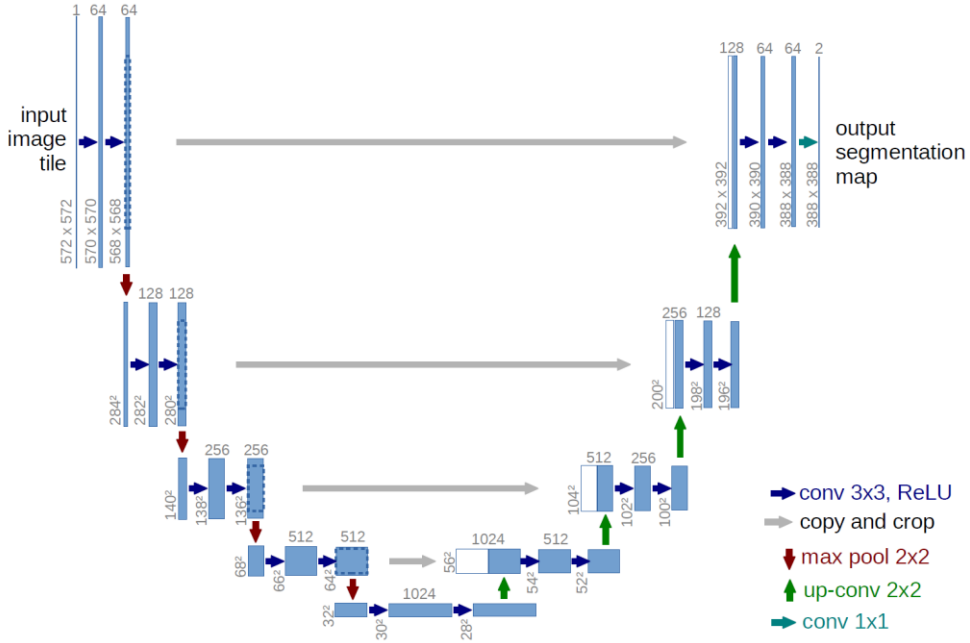
[Sun-Hao-Li; Img Src: NVIDIA Research: Tensors Are the Future of Deep Learning]

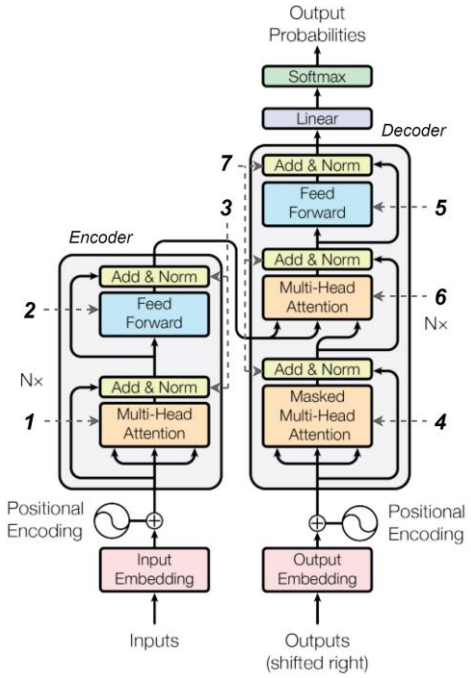
Compressing Deep Neural Networks

Compressing Deep Neural Networks

- **Motivation**: Deep Neural Nets like CNNs are go-to architectures for image-specific tasks. BUT:
  - They are expensive to train (both in time and memory)
  - They are heavy models with millions of parameters
  - They cannot be deployed on mobile devices
- Hence, compressing CNNs is the *need of the hour!*
- Not only an issue with CNNs, but with other architectures as well. Ex: RNNs, LSTMs, GNNs, Transformers, etc.



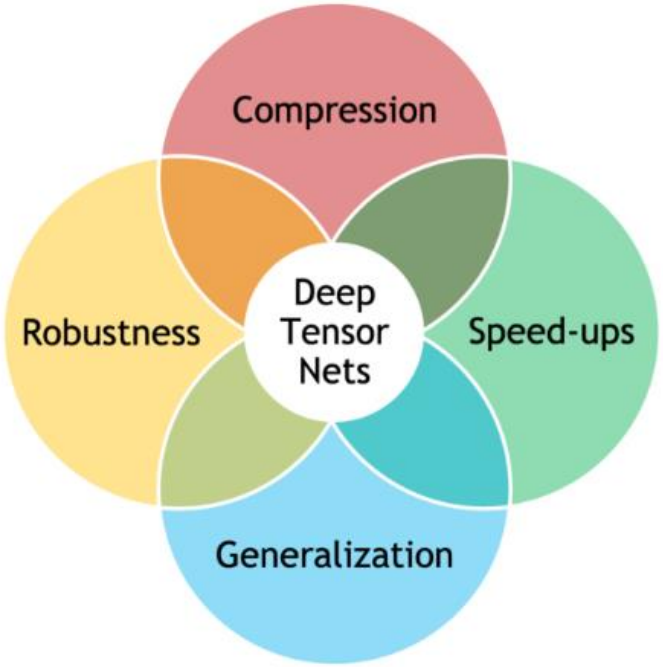




[Sun-Hao-Li]

Compressing Deep Neural Networks

- **Solution**: **Tensor-based Compression of Deep Neural Networks**:
  - Better performance and generalization
  - Improved robustness, from implicit (low-rank structure) or explicit (tensor dropout) regularization
  - Parsimonious models, with a large reduction in the number of parameters
  - Computational speed-ups by operating directly and efficiently on factorized tensors



[NVIDIA Research: Tensors Are the Future of Deep Learning]

Compressing Large Weight Matrices

Compressing Large Weight Matrices

- Input matrix **W** of size $I \times J$, Output tensor $\mathcal{W}$ called **TT-Matrix** $I = I_1 \times I_2 \times \cdots \times I_N$ and $J = J_1 \times J_2 \times \cdots \times J_N$.
  - **W** is reshaped to a higher-order tensor of size $I_1 \times I_2 \times \cdots \times I_N \times J_1 \times J_2 \times \cdots \times J_N$.
  - Permute the dimensions: $I_1 J_1 \times I_2 J_2 \times \cdots \times I_N J_N$
  - Apply **Tensor Train Decomposition** to store in TT-Format.
- Similarly, one would transform a vector to a **TT-vector** (for biases and input vectors)
- Proposed **TT-layer**: Analogue of Linear layer but operates on TT-matrices and TT-vectors.
- Provided equations for forward pass and backpropagation in terms of tensors.
- Even more compression achieved with additional **Weight Quantization** step

Fully-connected layers apply a linear transformation to an N-dimensional input vector $\boldsymbol{x}$:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{4}$$

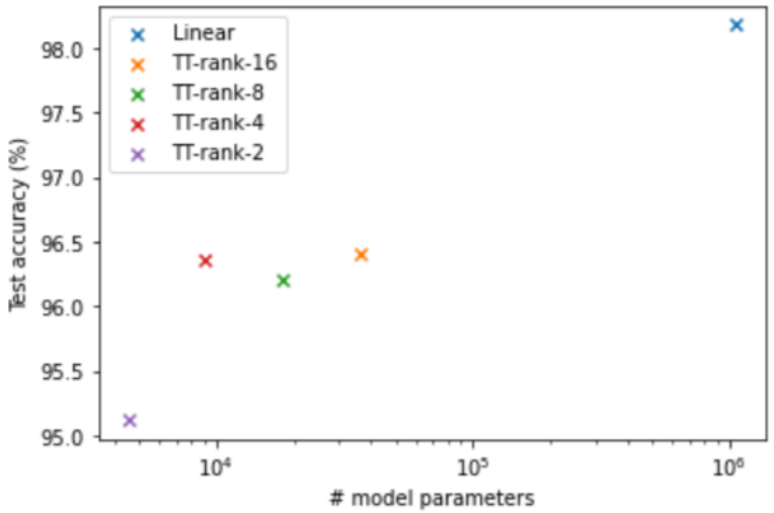
where the *weight matrix* $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ and the *bias vector* $\boldsymbol{b} \in \mathbb{R}^M$ define the transformation.

A *TT-layer* consists in storing the weights $\boldsymbol{W}$ of the fully-connected layer in the TT-format, allowing to use hundreds of thousands (or even millions) of hidden units while having moderate number of parameters. To control the number of parameters one can vary the number of hidden units as well as the TT-ranks of the weight matrix.

A TT-layer transforms a d-dimensional tensor $\boldsymbol{\mathcal{X}}$ (formed from the corresponding vector $\boldsymbol{x}$) to the d-dimensional tensor $\boldsymbol{\mathcal{Y}}$ (which correspond to the output vector $\boldsymbol{y}$). We assume that the weight matrix $\boldsymbol{W}$ is represented in the TT-format with the cores $\boldsymbol{G}_k[i_k, j_k]$. The linear transformation (4) of a fully-connected layer can be expressed in the tensor form:

$$\mathcal{Y}(i_1, \ldots, i_d) = \sum_{j_1, \ldots, j_d} \boldsymbol{G}_1[i_1, j_1] \ldots \boldsymbol{G}_d[i_d, j_d] \, \mathcal{X}(j_1, \ldots, j_d) + \mathcal{B}(i_1, \ldots, i_d). \tag{5}$$

Direct application of the TT-matrix-by-vector operation for the Eq. (5) yields the computational complexity of the forward pass $O(dr^2 m \max\{m, n\}^d) = O(dr^2 m \max\{M, N\})$.



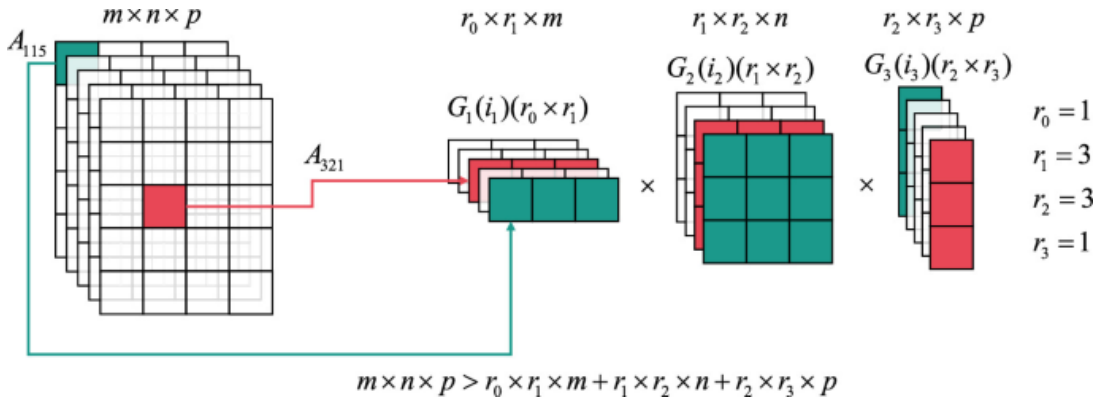
[Panagakis-Koissafi-et al.; Novikov-Podoprikhin-et al.; tt-compression.ipynb]

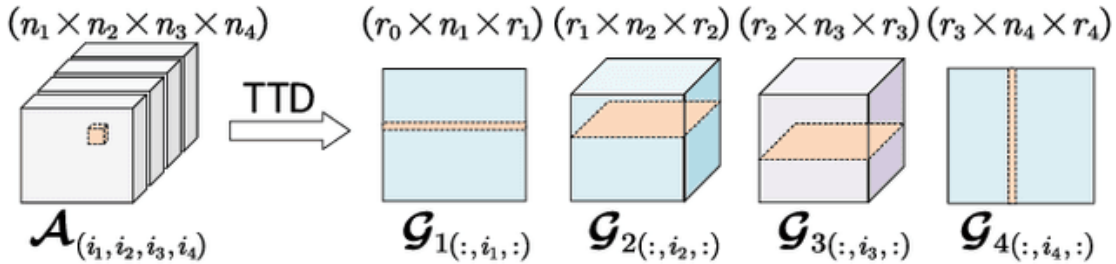
Tensor-Train Decomposition (TT)

- **Representation**: Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ be an order-d tensor. Then, exact TT-decomposition yields the following representation of elements of the tensor:

$$\mathcal{X}(i_1, i_2, \ldots, i_d) = \mathcal{G}_1(i_1) \times \mathcal{G}_2(i_2) \times \cdots \times \mathcal{G}_d(i_d)$$

where $\mathcal{G}_k \in \mathbb{R}^{r_{k-1} \times r_k \times n_k}, \mathcal{G}_k(i_k)$ is a matrix of shape $r_{k-1} \times r_k$ with the constraint that $r_0 = r_d = 1$.





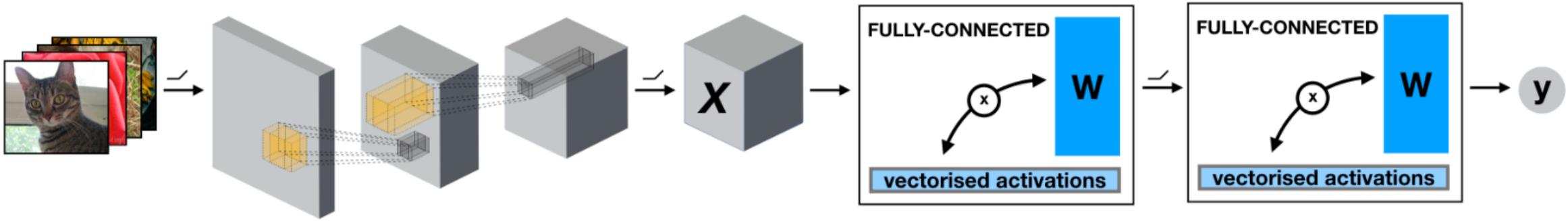
- The ranks $r_k,\ k \in \{0, 1, 2, \ldots, d\}$ are called TT-ranks or compression ranks.

[Oseledets]

Compressing CNNs

Compressing CNNs

- Tensorization can be applied at multiple locations in CNNs:
  - Tensorizing Convolution Operation in Convolutional Layers
  - Tensorizing Contraction Layers (layers that transforms activation maps of convolutional layers to inputs of linear layers)
  - Tensorizing Fully-Connected Linear Layers (Compressing Large Weight Matrices)
  - Tensorizing Regression Layer (at the output)



[Panagakis-Koissafi-et al.]

Tensorizing Convolution Operation

- 1×1 Convolution = Tensor Contractions
  Kernel: $\mathcal{W} \in \mathbb{R}^{T \times C \times 1 \times 1}$, Activation Map: $\mathcal{X} \in \mathbb{R}^{C \times H \times W}$ → Contraction along channel dimension
  Output: $\mathcal{Y} \in \mathbb{R}^{T \times H \times W}$
- **Kruskal Convolution**:
  - Uses CP Decomposition to approximate **learnable separable filter bank**
  - Employs series of separable convolutions
  - Either compute CP Decomposition of pre-trained filter banks or enforce architecture to learn filter banks in CP Decomposed format

$$\mathcal{F}_{t,y,x} = \underbrace{\sum_{r=1}^{R} \mathbf{U}_{t,r}^{(T)} \underbrace{\left[\sum_{i=1}^{W} \mathbf{U}_{i,r}^{(W)} \underbrace{\left(\sum_{j=1}^{H} \mathbf{U}_{j,r}^{(H)} \underbrace{\left[\sum_{k=1}^{C} \mathbf{U}_{k,r}^{(C)} \mathcal{X}(k, j+y, i+x) \right]}_{1\times1 \text{ conv}} \right)}_{\text{depthwise conv}} \right]}_{\text{depthwise conv}}}_{1\times1 \text{ convolution}} \tag{19}$$

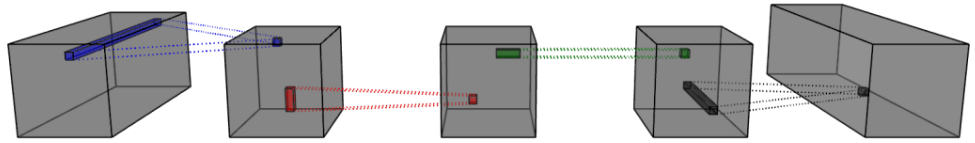


Fig. 13: **Illustration of a 2D Kruskal convolution**, as expressed Eq. (19), with matching colours. First, a 1×1 convolution reduces the number of input channels to the rank (blue). Two depthwise convolutions are then applied on the height and width of the activation tensor (red and green). Finally, a second 1×1 convolution restores the number of channels from the rank of the CP decomposition to the desired number of output channels (black).

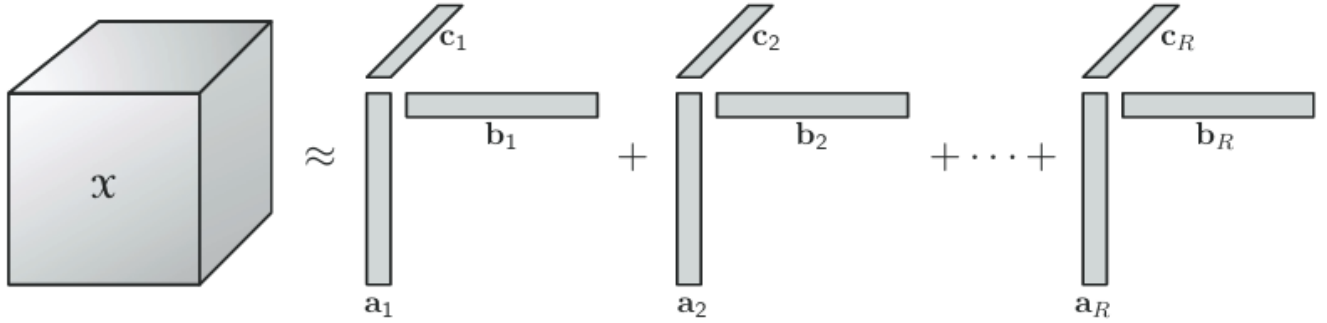
[Lebedev-Ganin-et al.; Panagakis-Koissafi-et al.]

CANDECOMP/PARAFAC (CP)

- **CANonical DECOMPosition/PARAllel FACtors (CP) Decomposition**: The CP decomposition factorizes a tensor into a sum of component rank-one tensors. For example, given a third-order tensor $X \in \mathbb{R}^{I \times J \times K}$, we wish to write it as:

$$X \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r$$

where R is a positive integer and $a_r \in \mathbb{R}^I, b_r \in \mathbb{R}^J$, and $c_r \in \mathbb{R}^K$ for $r \in [R]$, $\circ$ denotes vector outer product.



[Hitchcock; Carroll-Chang; Harshman; Kolda-Bader]

Tensorizing Convolution Operation

- **Tucker Convolution**:
  - Uses Tucker Decomposition to approximate **learnable separable filter bank**
  - Employs series of convolutions
  - Filters don't need to be separable now in the spatial dimensions

$$\mathcal{F}_{t,y,x} = \underbrace{\sum_{r_1=1}^{R_1} \mathbf{U}^{(T)}_{t,r_1} \underbrace{\left[\sum_{j=1}^{H} \sum_{i=1}^{W} \sum_{r_2=1}^{R_2} \mathcal{H}_{r_1,r_2,j,i} \underbrace{\left[\sum_{k=1}^{C} \mathbf{U}^{(C)}_{k,r_2} \mathcal{X}(k, j+y, i+x) \right]}_{1\times 1 \text{ conv}} \right]}_{H\times W \text{ conv}}}_{1\times 1 \text{ conv}} . \tag{20}$$

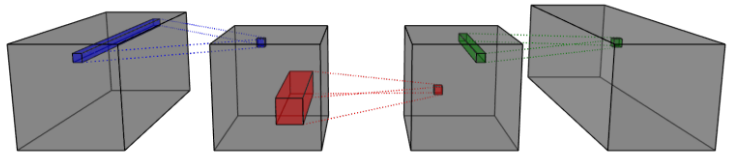


Fig. 14: **Illustration of a Tucker convolution** expressed as a series of small, efficient convolutions, as illustrated in Eq. (20). Note that this is the approach taken by ResNet for the Bottleneck blocks. After the full kernel has been decomposed, the factors along the input and output channels are used to parametrize 1×1 convolutions, applied respectively first (blue) and last (green). The remaining two factors are absorbed into the core and used to parametrize a regular 2D convolution (red).

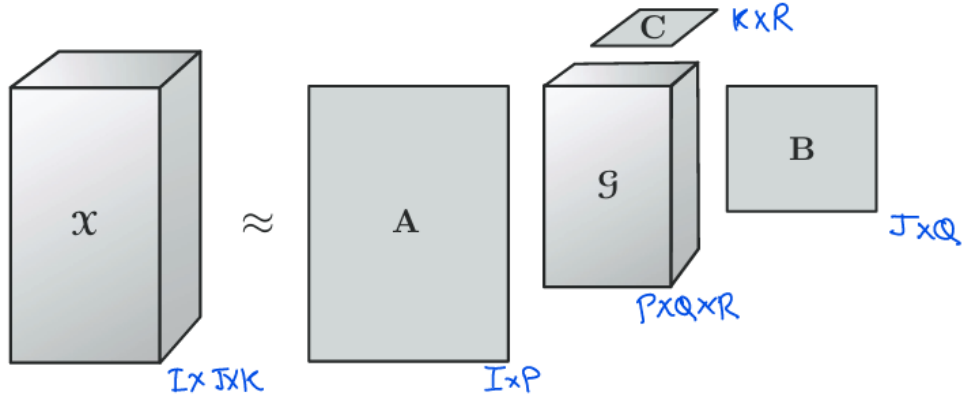
[Kim-Park-et al.; Panagakis-Koissafi-et al.]

Tucker Decomposition

- **Tucker Decomposition**: Factorizes an N-way tensor into a compressed N-way tensor called *core tensor* and N matrices, thus, it is also called higher-order PCA or higher-order SVD. For example, given a third-order tensor $X \in \mathbb{R}^{I \times J \times K}$, Tucker Decomposition gives:

$$X \approx \mathcal{G} \times_1 A \times_2 B \times_3 C \equiv \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \cdot a_p \circ b_q \circ c_r \equiv [[\mathcal{G}, A, B, C]]$$

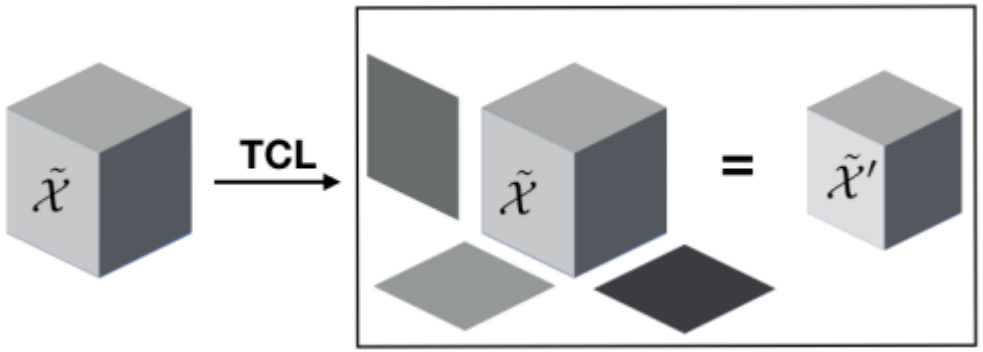
where $A \in \mathbb{R}^{I \times P}, B \in \mathbb{R}^{J \times Q}$, and $C \in \mathbb{R}^{K \times R}$ are *factor matrices* (usually column-wise orthogonal; can be thought of as principal components along each mode) and $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$ is the *core tensor*, $a_p \in \mathbb{R}^I, b_q \in \mathbb{R}^J, c_r \in \mathbb{R}^K$.



[Tucker; Kolda-Bader]

Tensorizing Contraction Layers

- Now, output activation maps of Tensorized Convolutional Layers will be **4-D tensors**
- Naively flattening activation maps to pass on vectorized inputs to Fully-Connected Linear Layers destroys the multilinear structure
- Hence, tensorize **Contraction Layers** (layers that transforms activation maps of convolutional layers to inputs of linear layers)
- Proposition of a new layer called **Tensor Contraction Layer (TCL):**
  - Perform **Tucker Decomposition** of the input tensorized activation maps
  - Pass on the **Tucker Core** to the FC Connected Layers (Remember: Compressing Large Weight Matrices!)
  - Enforce Tucker Decomposition and learn Tucker Core and Factor Matrices while training



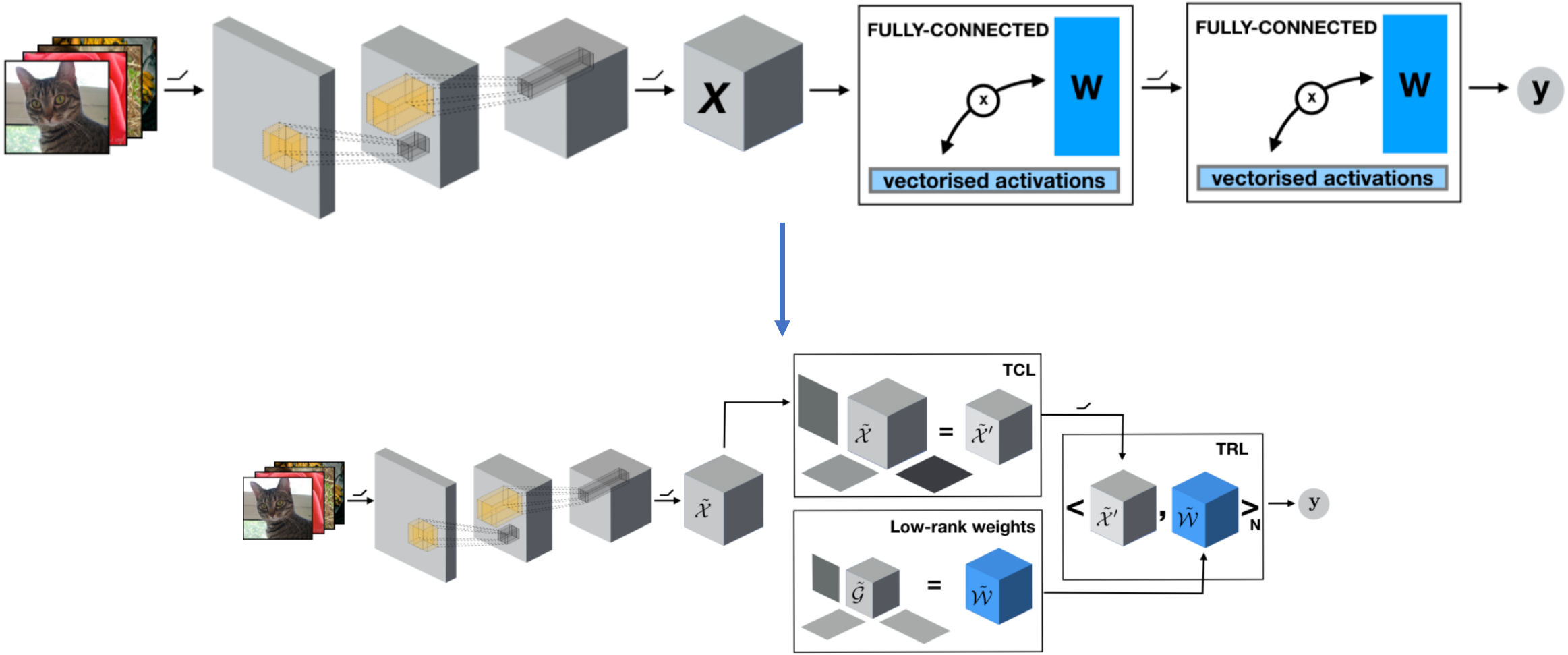
[Kossaifi-Khanna-et al.; Panagakis-Koissafi-et al.]

Tensorizing Regression Layer

- Let's focus on the Output Layer
- **Regression Problem**: The Output Layer receives activation tensors from Tensorized FC Layers and has to output 1 real value.
- Flattening is not an option! (We ❤️ tensors!)
- Introduced **Tensor Regression Layer (TRL):**
  - Tensorized Input: $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$
  - Tensorized Weight Matrix: $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$
  - $y = < \mathcal{X}, \mathcal{W} > + b, \quad y, b \in \mathbb{R}$ [Tensor Inner Product; literally sum of elementwise multiplications]
  - $\mathcal{X}$ and $\mathcal{W}$ can be further admitted in Tucker, CP or TT format

[Kossaifi-Lipton-et al.; Panagakis-Koissafi-et al.; tensor_regression_layer.ipynb]

Compressing CNNs



[Panagakis-Koissafi-et al.]

Compressing RNNs, GRUs, LSTMs

Compressing RNNs, GRUs and LSTMs

- **Remember**: Compressing Large Weight Matrices!
- Weights of RNNs, GRUs, LSTMs compressed using the same straight-forward methodology with additional Weight Sharing property natural to sequential models.
- Use **Tensor Train Decomposition**
- Even better approach:
  - Concatenate all weight matrices corresponding to every "cell" and "gate" and then compress to TT-format
  - This leads to even more compression as compared to compressing individual weight matrices
  - Observed **implicit regularization** because of TT-LSTM: **Less overfitting**, minimizing the issue of **vanishing or exploding gradients**
  - TT-LSTM also **robust to changes in learning rate** unlike Vanilla LSTM

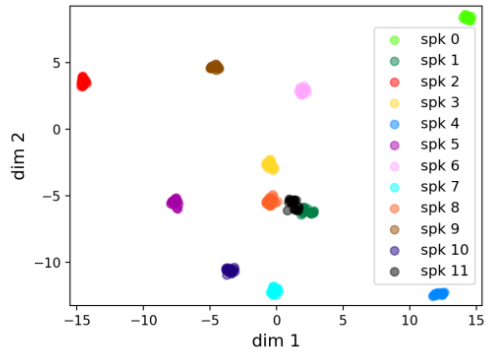


Figure 2: Low dimensional UMAP visualization of embeddings from the TT-LSTM. Each datapoint corresponds to the 256-dimensional embedding of an utterance, where colors reflect the identity of different speakers. A clear clustering pattern is seen amongst the utterances from each speaker.

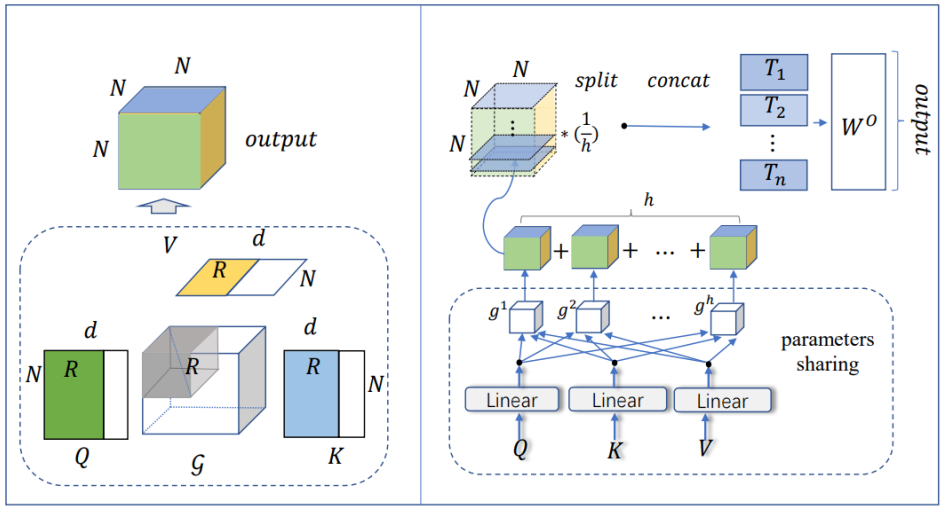
Model	Cores	r	#Params	Compr.	Acc. (%)
LSTM	–	–	266,762	–	89.77
TT-LSTM	2	2	3,434	78	87.98
		4	5,834	46	89.49
		6	8,234	32	89.22
	3	2	1,842	145	85.36
		4	3,354	80	87.18
		6	5,570	48	89.30
GRU	–	–	201,482	–	91.49
TT-GRU	2	2	3,674	55	87.94
		4	5,802	35	89.29
		6	7,930	25	90.26
	3	2	2,282	88	87.62
		4	3,722	54	88.90
		6	5,866	34	89.80

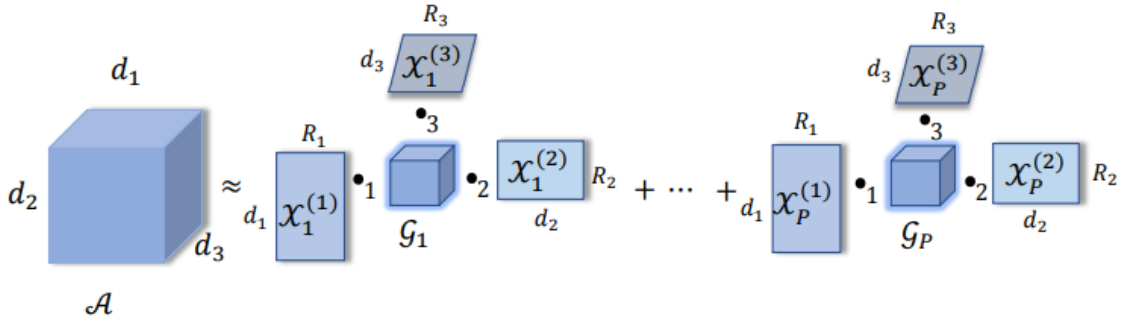
[Onu-Miller-et al.; Novikov-Podoprikhin-et al.]

Compressing Transformers

Compressing Transformers

- Compression applied at multiple sections of the transformer:
  - Tensorizing Embedding Layers
  - Tensorizing Attention Heads
- **Tensorizing Embedding Layers**:
  - Store Embedding Matrix in TT-format (**Remember**: Compressing Large Weight Matrices!)
  - Apart from compression, authors realized and proved that TT-embeddings are **more expressive** than vanilla matrix-based embeddings.
  - Also provided **TT-embedding tensor initialization** method and enforced TT-embedding structure in the model (just compressing pre-trained embedding matrix into TT-format does not exhibit better performance)
- **Tensorizing Attention Heads**:
  - Proposed **Multi-Linear Attention Module** as a replacement of Self Attention
  - Performed Block Term Decomposition to obtain Attention Maps





[Hrinchuk-Khrulkov-et al.; Ma-Zhang-et al.; Rabanser-Shchur-Günnemann; Kolda-Bader; Oseledets]

Thank You