Linear Algebra

Arthur J. Redfern
arthur.redfern@utdallas.edu

Outline

- Motivation
- Fundamentals
  - Vector spaces
  - Matrix operations
  - Matrix transforms
- Layers built from matrix transforms
  - NN
  - CNN
  - RNN
  - Attention
  - Self attention
  - Average pooling
- References

Disclaimer

- This set of slides is more accurately titled “A brief refresher of a subset of linear algebra for people already somewhat familiar with the topic followed by it’s specific application to xNN related items needed by the rest of the course”

- However, that’s not very catchy so we’ll just stick with “Linear algebra”

- In all seriousness, recognize that linear algebra is a very broad and deep topic that has and will continue to occupy many lifetimes of work; if interested in learning more, please consult the references to open a window into a much larger world

Motivation

Pre Processing

- Pre processing methods simplify feature extraction and prediction

- Understanding linear transformations is a key to understanding many popular pre processing methods

- Example pre processing methods
  - Discrete Fourier transform
  - Principal component analysis
  - Singular value decomposition

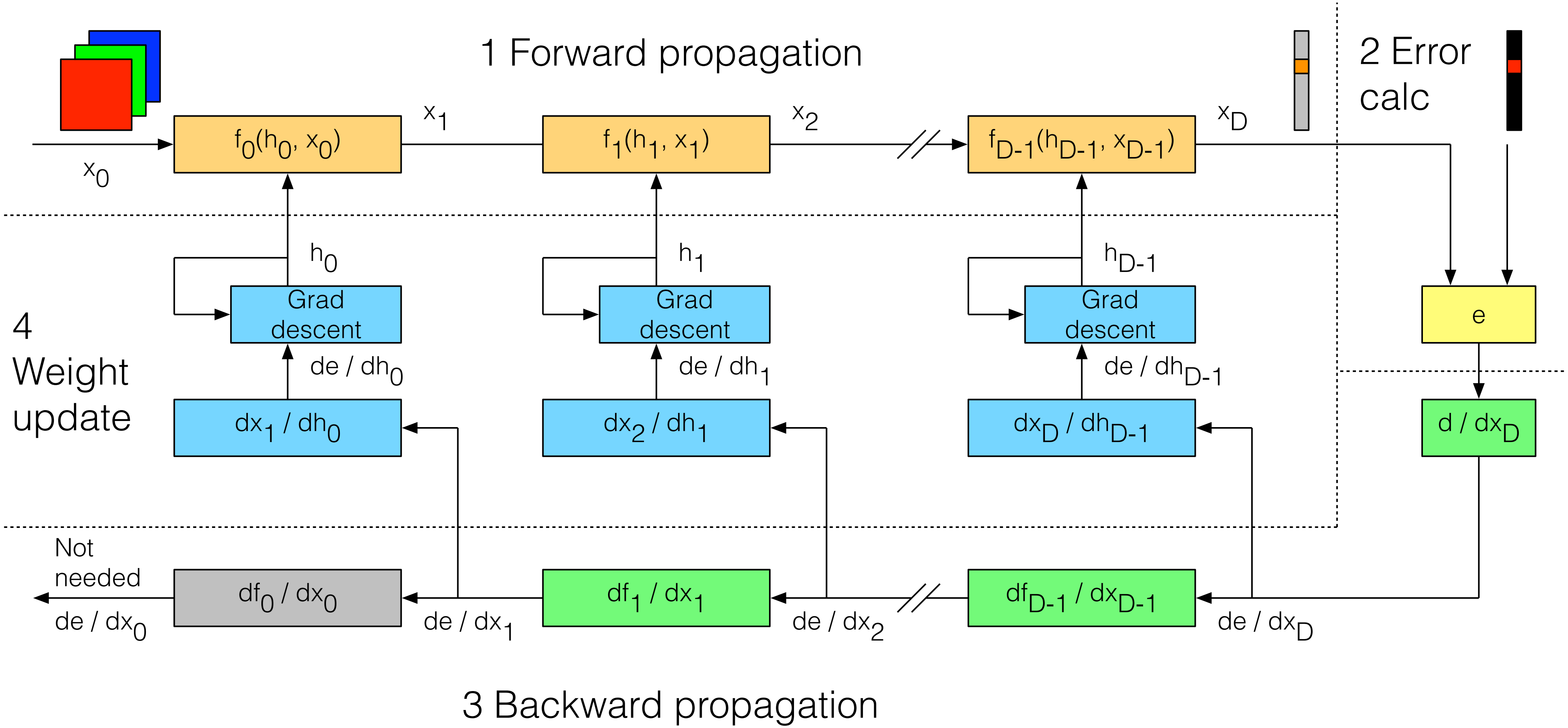
Feature Extraction And Prediction

- xNNs are compositions of nonlinear functions (layers)

- But linear functions with trainable parameters are a component of key layer types that control the network mapping from data space to feature space to information space

- Examples layers that include linear functions
  - Dense layers with single and multiple inputs
  - CNN style 2D convolution layers
  - RNN layers
  - Attention based layers
  - Average pooling layers

Big Picture Reminder



Fundamentals – Vector Spaces

Notation

- Scalars are not bold
    - a
- Vectors are bold lower case
    - **a**
- Matrices and tensors are bold upper case
    - **A**
- Indices start at 0 and go from 0, ..., size – 1
    - 0, ..., N – 1

Set

- A collection of distinct objects

Field

- A set with well defined addition and multiplication operations
  - Associativity: $a + (b + c) = (a + b) + c$ and $a\,(b\,c) = (a\,b)\,c$
  - Commutativity: $a + b = b + a$ and $a\,b = b\,a$
  - Additive identity: $a + 0 = a$
  - Additive inverse: $a + (-a) = 0$
  - Multiplicative identity: $1\,a = a$
  - Multiplicative inverse: $a\,a^{-1} = 1$
  - Distributivity: $a\,(b + c) = (a\,b) + (a\,c)$

- Elements of fields are generally referred to as scalars

- Examples: R (real scalars), C (complex scalars)

Vector

- K tuple of scalars
- Always a column
- Denoted by the field raised to the size
    - F^K
- Examples: R^K and C^K

Matrix

- M x K tuple of scalars

- Collection of K vectors of size M x 1 arranged in columns
  - Leads to column space and right null space
    - What can matrix vector multiplication reach and what can it not
  - Visualize using inner product of matrix vector multiplication

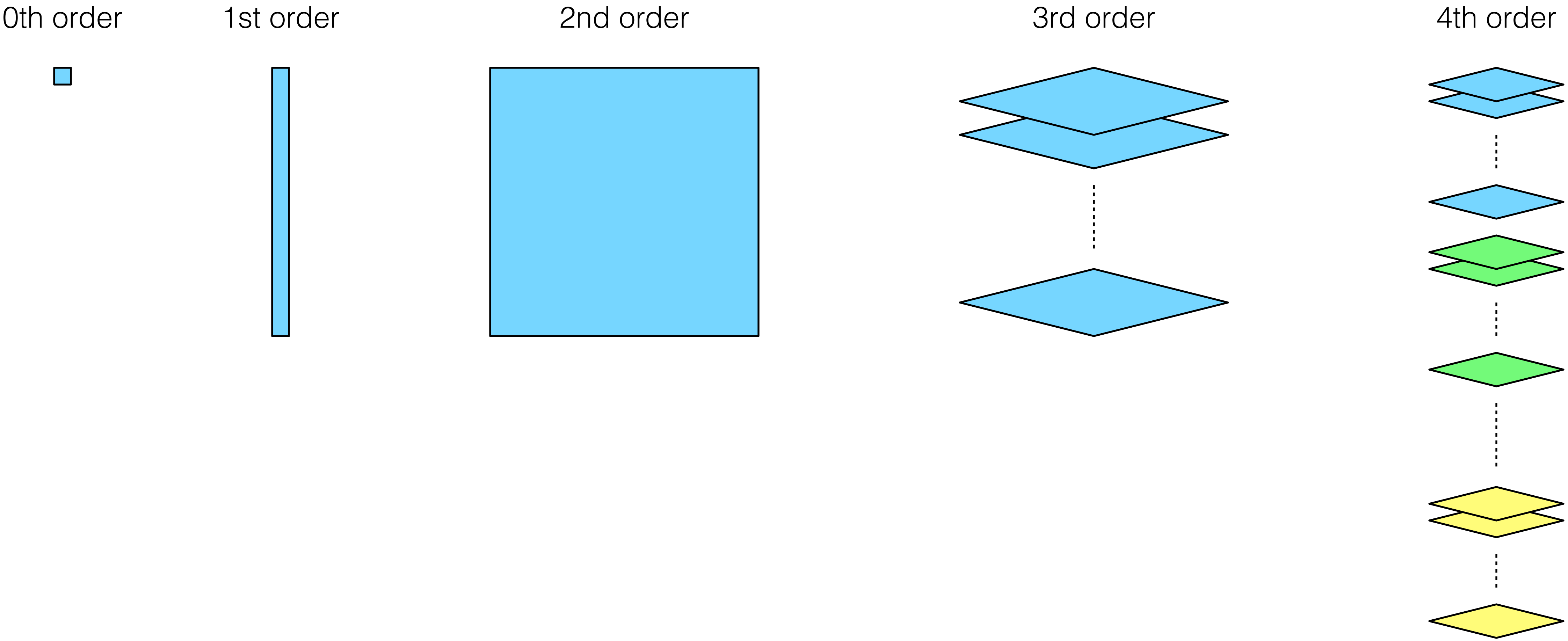
- Collection of M vectors of size K x 1 transposed and arranged as rows
  - Leads to row space and left null space
    - What can vector matrix multiplication reach and what can it not
  - Visualize using outer product of vector matrix multiplication

Tensor

- K_0 x ... x K_{D-1} array of scalars

- Ordering
  - Last dimension is contiguous in memory
  - Working from right to left goes from closest to farthest spacing in memory
  - Feature maps: batch x channel x row x column (sometimes referred to as NCHW ordering)
  - Filter coefficients: output channel x input channel x row x col

Examples Of Tensors



0th order

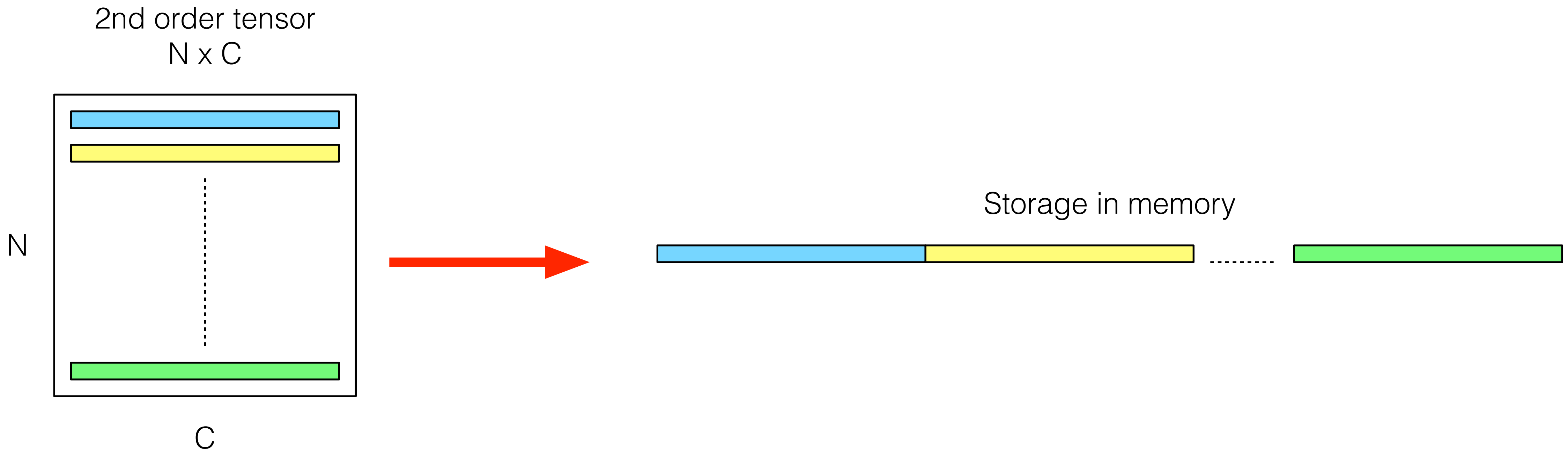
1st order

2nd order

3rd order

4th order

The Layout Of Tensors In Memory



Vector Space

- Set of vectors and linear combinations of those vectors

- Satisfy
  - Associativity: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$
  - Commutativity: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
  - Additive identity: $\mathbf{x} + \mathbf{0} = \mathbf{x}$
  - Additive inverse: $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$
  - Multiplicative compatibility: $a\ (b\ \mathbf{x}) = b\ (a\ \mathbf{x})$
  - Multiplicative identity: $1\ \mathbf{x} = \mathbf{x}$
  - Distributivity: $(a + b)(\mathbf{x} + \mathbf{y}) = a\ \mathbf{x} + a\ \mathbf{y} + b\ \mathbf{x} + b\ \mathbf{y}$

- Examples: R^K, C^K, $R^{K\_0 \text{ x } \ldots \text{ x } K\_D-1}$

Normed Vector Space

- A vector space with a notion of distance

- A norm maps an element of the vector space to a scalar

- Satisfies
  - Non negativity: $||\mathbf{x}|| \geq 0$ and $||\mathbf{x}|| = 0$ iff $\mathbf{x} = \mathbf{0}$
  - Absolute scalability: $||a\,\mathbf{x}|| = |a|\ ||\mathbf{x}||$
  - Triangle inequality: $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$

- Example: l_p norm (common p = 1, 2 and ∞)
  - $||\mathbf{x}||_p = (\Sigma_n(|x(n)|^p))^{1/p}$, $p \geq 1$

Inner Product Space

- A vector space with a notion of distance and angle

- An inner product maps 2 elements of a vector space to a scalar

- Satisfies
  - Positive definiteness: $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ and $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ iff $\mathbf{x} = 0$
  - Conjugate symmetry: $\langle \mathbf{x}, \mathbf{y} \rangle = \text{conj}(\langle \mathbf{y}, \mathbf{x} \rangle)$
  - Linearity: $\langle a\, \mathbf{x}, \mathbf{y} \rangle = a \langle \mathbf{x}, \mathbf{y} \rangle$ and $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$

- Inner products induce norms on a vector space
  - But not all norms have associated inner products (e.g., l_∞)

- Example: dot product
  - $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \bullet \mathbf{y} = \mathbf{x}^H \mathbf{y} = \Sigma_n (\text{conj}(x(n))\, y(n)) = ||\mathbf{x}||_2\, ||\mathbf{y}||_2 \cos(\theta)$

Fundamentals – Matrix Operations

Transpose

- Definition
  - The transpose of matrix $\mathbf{A}$ with entries A(m, k) is matrix $\mathbf{A}^T$ with entries conj(A(k, m))
  - Also referred to as the Hermitian adjoint

- Properties
  - $\mathbf{C}^T = (\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$
  - $\mathbf{C}^T = (\mathbf{A}\ \mathbf{B})^T = \mathbf{B}^T\ \mathbf{A}^T$

Matrix Vector Multiplication

- Definition
  - $\mathbf{c} = [\mathbf{a}_0^T; \ldots; \mathbf{a}_{M-1}^T]\ \mathbf{b} = \mathbf{A}^T\ \mathbf{b}$ where $c(m) = \Sigma_k\ a(m, k)\ b(k)$
  - $$\begin{bmatrix} c(0) \\ \vdots \\ c(M-1) \end{bmatrix} = \begin{bmatrix} a(0,0) & \cdots & a(0,K-1) \\ \vdots & & \vdots \\ a(M-1,0) & \cdots & a(M-1,K-1) \end{bmatrix} \begin{bmatrix} b(0) \\ \vdots \\ b(K-1) \end{bmatrix}$$
- Comments
  - M (output dimension), K (input dimension) is setting up for BLAS notation
  - Inner product of matrix row and vector input to produce each output
  - We’ll use notation similar to this in xNNs when we want to emphasize that the input data (**b**) is in a col and each row of $\mathbf{A}^T$ is used to extract a corresponding output feature or class prediction

Arithmetic Intensity

- Arithmetic intensity
  - Compute = MK (MACs = multiply accumulates)
  - Data movement = K + MK + M (elements)
  - Ratio = (MK)/(K + MK + M) (consider M and K large)
    $\approx$ 1 (memory wall)

- Implementation preview
  - If you want to make matrix vector multiplication run fast, you need to build a fast memory subsystem
  - Typically not an efficient thing to do from an operation per power perspective

Vector Matrix Multiplication

- Definition
  - $\mathbf{c}^T = \mathbf{a}^T [\mathbf{b}_0, \ldots, \mathbf{b}_{N-1}] = \mathbf{a}^T \mathbf{B}$ where $c(n) = \Sigma_k\, a(k)\, b(k, n)$
  - $$[c(0) \quad \cdots \quad c(N-1)] = [a(0) \quad \cdots \quad a(K-1)] \begin{bmatrix} b(0,0) & \cdots & b(0,N-1) \\ \vdots & & \vdots \\ b(K-1,0) & \cdots & b(K-1,N-1) \end{bmatrix}$$

- Comments
  - N (output dimension), K (input dimension) is setting up for BLAS notation
  - Inner product of row vector input and matrix col to produce each output
  - We'll use notation similar to this in xNNs when we want to emphasize that the input data ($\mathbf{a}^T$) is in a row and each col of **B** will be used to extract a corresponding output feature or class prediction

Arithmetic Intensity

- Arithmetic intensity

Compute	= NK	(MACs = multiply accumulates)
Data movement	= K + NK + N	(elements)
Ratio	= (NK)/(K + NK + N)	(consider N and K large)
	$\approx$ 1	(memory wall)

- Implementation preview
  - If you want to make matrix vector multiplication run fast, you need to build a fast memory subsystem
  - Typically not an efficient thing to do from an operation per power perspective

Matrix Matrix Multiplication

- Definition
  - **C** = [$\mathbf{a}_0^T$; …; $\mathbf{a}_{M-1}^T$] [$\mathbf{b}_0$, …, $\mathbf{b}_{N-1}$] = $\mathbf{A}^T$ **B** where c(m, n) = Σ_k a(m, k) b(k, n)
  - $$\begin{bmatrix} c(0,0) & \cdots & c(0,N-1) \\ \vdots & & \vdots \\ c(M-1,0) & \cdots & c(M-1,N-1) \end{bmatrix} = \begin{bmatrix} a(0,0) & \cdots & a(0,K-1) \\ \vdots & & \vdots \\ a(M-1,0) & \cdots & a(M-1,K-1) \end{bmatrix} \begin{bmatrix} b(0,0) & \cdots & b(0,N-1) \\ \vdots & & \vdots \\ b(K-1,0) & \cdots & b(K-1,N-1) \end{bmatrix}$$
- Comments
  - M (output dimension), K (input dimension), N (number of inputs and outputs) is setting up for BLAS notation
  - Can view as matrix vector multiplication applied to multiple inputs stacked next to each other (in the N dimension) with matrix vector multiplication as a special case with N = 1
  - Can view as vector matrix multiplication applied to multiple inputs stacked on top of each other (in the M dimension) with vector matrix multiplication as a special case with M = 1
  - A discussion of different computational options for matrix matrix multiplication (inner product based, outer product based, block based, Strassen style) will be deferred to the implementation section

Arithmetic Intensity

- Arithmetic intensity
  - Compute = MNK (MACs)
  - Data movement = KN + MK + MN (elements)
  - Ratio = (MNK)/(KN + MK + MN) (cube in num, squares in den)
    = $N^3/(3*N^2)$ (special case M = N = K)
    = N/3 (ratio maxed with sq matrix)

- Implementation preview
  - If you want to make matrix matrix mult run fast, if it’s possible choose a large matrix size such that you get multiple ops per element of data moved

- Why are bubbles spherical? Min surface area per volume enclosed
  - Think of surface area as data movement and volume as MACs

Hadamard or Schur Product

- Definition
  - $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$ where $C(m, k) = A(m, k)\ B(m, k)$

- Comments
  - Can be thought of as a point wise or element wise product
  - Used in many FFT algorithms for twiddle factor multiplication
  - Used to combine a gate ([0, 1] limited vector) with an input or output

Fundamentals – Matrix Transforms

Linear

- Important to understand matrices in the context of linear maps (transforms)
  - Every linear map can be represented as a matrix
  - Every matrix represents a linear map

- T: $V_1 \rightarrow V_2$ is a linear map between vector spaces V_1 and V_2
  - Let **x** and **y** be vectors in V_1 and a be a scalar
  - Then T satisfies the following properties
    - Additivity: $T(\mathbf{x} + \mathbf{y}) = T(\mathbf{x}) + T(\mathbf{y})$
    - Homogeneity: $T(a\,\mathbf{x}) = a\,T(\mathbf{x})$

Compositions

- Multiple linear transformations can be composed into a single linear transformation
  - $\mathbf{y} = \mathbf{A}_{D-1}^{T} \ldots \mathbf{A}_{1}^{T} \mathbf{A}_{0}^{T} \mathbf{x}$
    $= \mathbf{A}^{T} \mathbf{x}$, where $\mathbf{A}^{T} = \mathbf{A}_{D-1}^{T} \ldots \mathbf{A}_{1}^{T} \mathbf{A}_{0}^{T}$
- Comments
  - A reason why nonlinearities are included in xNNs
  - Otherwise there would be no depth

Affine

- An affine transformation is a linear transformation + an offset or bias
  - $y = A^T x + b$
  - $y^T = x^T A + b^T$

- Can be implemented as a linear transformation augmented with a nonzero constant input
  - $y = A^T x + b = [A^T\ b]\ [x; 1] = A_{aug}{}^T x_{aug}$
  - $y^T = x^T A + b^T = [x^T\ 1]\ [A; b^T] = x_{aug}{}^T A_{aug}$
  - Note the input dimension has increased by 1

- Many xNN layers take the form of an affine transformation followed by a nonlinearity
  - The bias term is helpful in the constructive universal function approximator proof

Layers Built From Matrix Transforms

Purpose

Different linear transformations mix inputs together in different ways to create each output; these differences are important with respect to a network accomplishing a goal

- You design a network to accomplish a goal
  - Don't ever lose sight of this
  - Network design is not arbitrary
  - So it always makes sense to stop and think how the operations you're including help you accomplish the goal you're trying to achieve
- Ask yourself
  - Why do xNNs include these layers?
  - How do these layers map from data to features to predictions?
  - How are the input features combined to generate output features?
- The purpose of the next few slides is to introduce layers which include linear transformations and help build intuition on how they map from data to features to classes

What Maps To What?

- A brief preview of where we’re going (it’s easy to get lost in the middle of things, so come back to this as necessary)

- Densely connected layers
  - Map 1 input vector to 1 output vector
  - In the general case all input features are mixed together to produce each output feature

- RNN layers
  - Map 1 input vector and 1 previous output vector to 1 output vector
  - In the general case all input features are mixed together and all previous output features are mixed together, then the 2 are combined to produce each output feature

What Maps To What?

- CNN style 2D convolution
  - Map N_i input feature maps to N_o output feature maps
  - In the general case a local window of all input features are mixed together to produce each output feature

- Self attention layers
  - Map M inputs vectors to M output vectors
  - In the general case mixing occurs first across all M input vectors then across all features within each vector to produce M output feature vectors

Layers – NN

Layer Definition

- Densely connected or fully connected layer
  - $\mathbf{y}^T = f(\mathbf{x}^T \mathbf{H} + \mathbf{v}^T)$
  - $\mathbf{x}^T \mathbf{H}$ is multiplication of a 1 x K input vector $\mathbf{x}^T$ with a K x N weight matrix $\mathbf{H}$
  - $\mathbf{v}^T$ is a 1 x N bias vector
  - f is a (sub) differential nonlinearity applied point wise (to each element individually)
    - ReLU: 0 out the negative values and pass the positive unchanged
    - Sigmoid: monotonic nonlinear map to (0, 1)
    - Tanh: monotonic nonlinear map to (-1, 1)
    - ... many other options possible
  - $\mathbf{y}^T$ is a 1 x N output vector
- A traditional neural network is composed of multiple densely connected layers
  - Transform from data to weak features
  - Transform from weak features to strong features
  - Transform from strong features to classes

Intuition

- Some intuition of feature extraction and prediction for a densely connected layer
  - Sometimes linear classification is viewed as template matching where each row is a different template and the predicted class is the maximum output
- The inner product depends on magnitude and angle
  - Inner product definition reminder: $\mathbf{a}^T \mathbf{b} = \langle \mathbf{a}, \mathbf{b} \rangle = ||\mathbf{a}||_2 \, ||\mathbf{b}||_2 \cos(\theta)$
- Each linearly extracted output feature is the inner product of input $\mathbf{x}^T$ and a col of $\mathbf{H}$
  - $z(n) = \mathbf{x}^T \, \mathbf{H}(:, n)$
  - $z(n)$ is the extracted feature or prediction
  - $\mathbf{x}^T$ is the input
  - $\mathbf{H}(:, n)$ is the feature extractor or predictor

Intuition

- How strong or important is a feature extractor? $||\mathrm{H}(:, n)||_2$
  - Note that the input mag contributes the same to each extracted feature $||\mathbf{x}||_2$
  - So here input magnitude only matters relative to bias
  - But input magnitude will also matter for network structures with branches that come together
  - Input magnitude will also matter when the same feature extractor is applied to different inputs

- How aligned is the feature extractor with the input? θ
  - In same direction: positive feature
  - Orthogonal: 0 feature
  - In opposite direction: negative feature

Intuition

- Intuition of bias
  - Affine transformation
  - Allows the dividing line to shift
  - Implementation as a rank 1 outer product
  - Will use bias in a constructive variant of the universal approximation proof
- Intuition of ReLU
  - Removes negatively aligned features or predictions
  - Allows depth
  - Subsequent layers combine positively aligned extracted features
- Intuition of other nonlinearities
  - Also allow depth
  - Sigmoid acts as a gate
  - Tanh acts as a positive / negative map

Intuition

- Intuition of the size of K (input length) and N (output length)
  - Small K to large N
    - Different combinations of a small number of features to predict a large number of classes
  - Large K to small N
    - 1 feature or a combination of features to predict a small number of classes is now possible
  - Example: ImageNet classification and final fully connected layer size
  - Example: the game of 20 questions

Use In Classification

- In general, for the final classification layer, it’s better to have K > N
    - Classification goal is to create matrix and bias that takes K features and makes the correct 1 of the N elements at the output much larger (closer to + ∞) than all the others
    - To get a feel for this consider 2 extreme cases
        - 2 features K linearly combined + bias then ReLU to predict 200 classes N
        - 200 features K linearly combined + bias then ReLU to predict 2 classes N
    - Create example features, matrices and biases for both cases
    - Look at sensitivity of the prediction to errors in the features for each
        - Can relate to matrix condition number
        - Show the condition number of K > N is always less than that of K < N
        - Now vary K and N and show diminishing returns after some point too
        - Generalize to considerations in a hierarchical head design

Batch Processing

- If a densely connected layer is applied to a batch of M inputs
  - $\mathbf{Y}^T = f(\mathbf{X}^T \mathbf{H} + \mathbf{1}\,\mathbf{v}^T)$
  - $\mathbf{X}^T \mathbf{H}$ is multiplication of a M x K input matrix $\mathbf{X}^T$ composed of M input vectors of size 1 x K with a K x N weight matrix $\mathbf{H}$
  - $\mathbf{1}$ is a M x 1 vector of 1s and the same 1 x N bias vector $\mathbf{v}^T$ is added to each matrix multiplication output as the product $\mathbf{1}\,\mathbf{v}^T$ is the same $\mathbf{v}^T$ per row
  - f is a (sub) differential nonlinearity applied point wise (to each element individually)
  - $\mathbf{Y}^T$ is a M x N output matrix composed of M output vectors of size 1 x N

- As M becomes larger densely connected layers go from being vector matrix multiplication to matrix matrix multiplication
  - Important from an implementation perspective

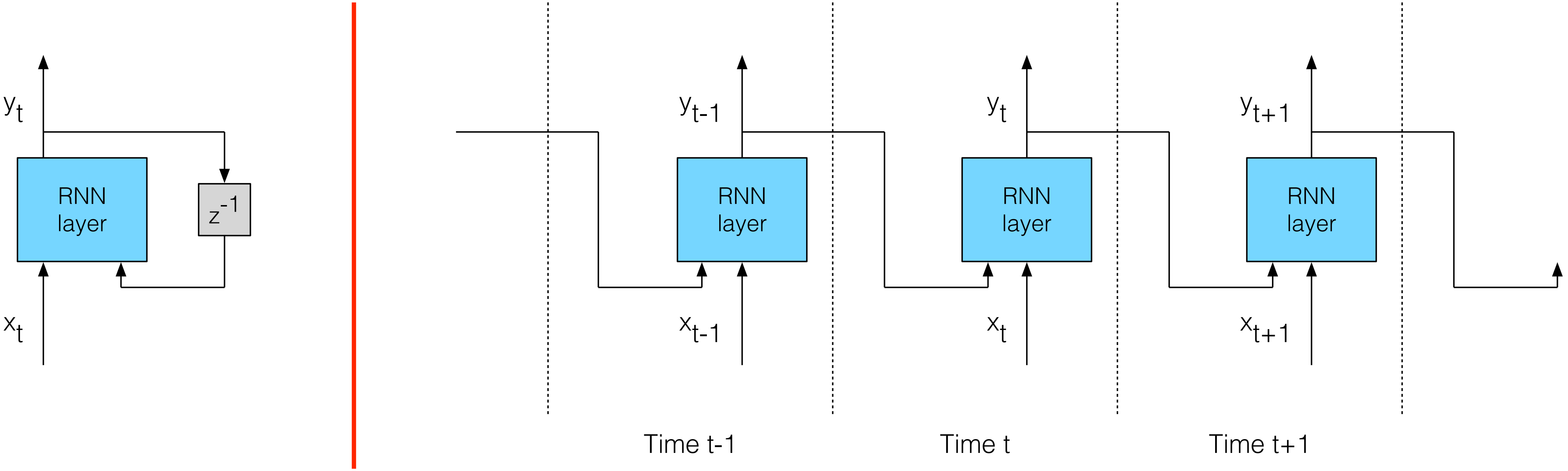
Layers – RNN

Layer Definition

- RNN layer
  - $\mathbf{y}_t^T = f(\mathbf{x}_t^T \mathbf{H} + \mathbf{y}_{t-1}^T \mathbf{G} + \mathbf{v}^T)$
  - t is the current time step, t – 1 is the previous time step
  - $\mathbf{x}_t^T \mathbf{H}$ is the multiplication of the 1 x K input vector $\mathbf{x}_t^T$ with the K x N matrix $\mathbf{H}$
  - $\mathbf{y}_{t-1}^T \mathbf{G}$ is the multiplication of the 1 x N previous output vector $\mathbf{y}_{t-1}^T$ with the N x N matrix $\mathbf{G}$
  - $\mathbf{v}^T$ is the 1 x N bias vector
  - f is the pointwise nonlinearity as in the case of a densely connected layer
  - $\mathbf{y}_t^T$ is the 1 x N current output vector
- A traditional RNN is composed of multiple RNN layers
  - Transform from data to weak features to strong features
  - All sorts of structural configurations: stacked, bi directional, pyramidal
  - All sorts of variants to improve memory: GRU, LSTM
  - These will be discussed in later lectures in the context of language and speech

Unrolling In Time

A RNN layer illustrated with feedback (left) and unrolled in time (right)



Intuition

- RNN intuition
  - Current output features are dependent on features extracted from the input and features extracted from the previous output
  - So it’s like a densely connected layer with the addition of a term that depends on the previous output
  - This allows for information to flow sequentially

- Inputs X can be batched similarly to that of dense layers
  - But previous outputs y must be used sequentially in the computation

Layers – CNN

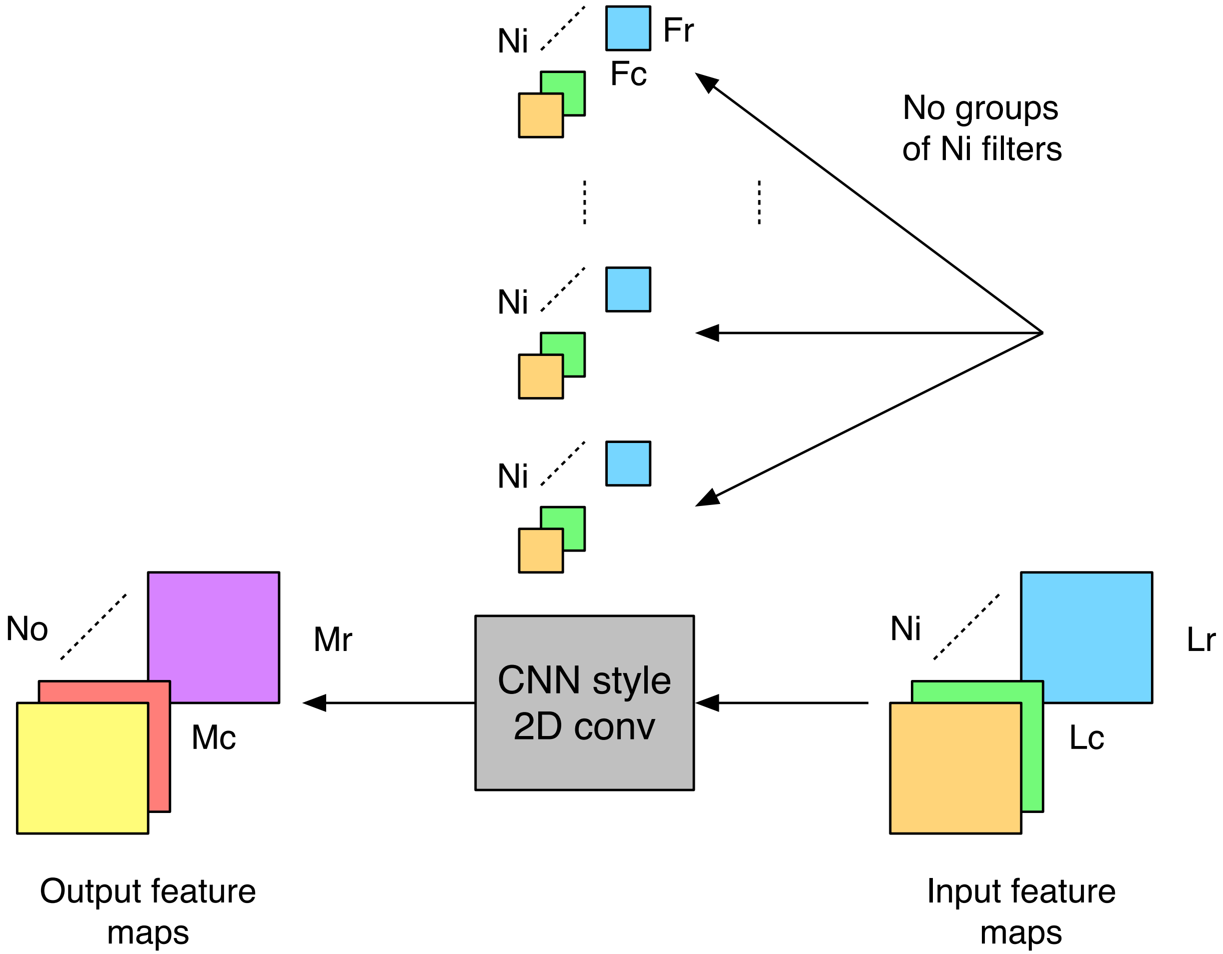
Layer Definition

- CNN style 2D convolution layer
  - $\mathbf{Y}^{3D} = f(\mathbf{H}^{4D} \circledast \mathbf{X}^{3D} + \mathbf{V}^{3D})$
  - $\mathbf{H}^{4D} \circledast \mathbf{X}^{3D}$ is CNN style 2D convolution of a N_o x N_i x F_r x F_c tensor $\mathbf{H}^{4D}$ with a N_i x L_r x L_c input vec $\mathbf{X}^{3D}$
  - $\mathbf{V}^{3D}$ is a N_o x M_r x M_c bias tensor that is a constant for each output feature map
  - f is a pointwise nonlinearity as in the case of a densely connected layer with ReLU being common
  - $\mathbf{Y}^{3D}$ is a N_o x M_r x M_c output tensor
  - Batching can be thought of as adding a loop that applies this layer to multiple input / output pairs resulting in the addition of a dimension to X and Y after concatenation
    - $\mathbf{X}^{4D}$ is a B x N_o x M_r x M_c output tensor
    - $\mathbf{Y}^{4D}$ is a B x N_o x M_r x M_c output tensor

- A traditional CNN is composed of CNN style 2D convolution layers (in addition to other layer types and branching structures)
  - Transform from data to weak features
  - Transform from weak features to strong features

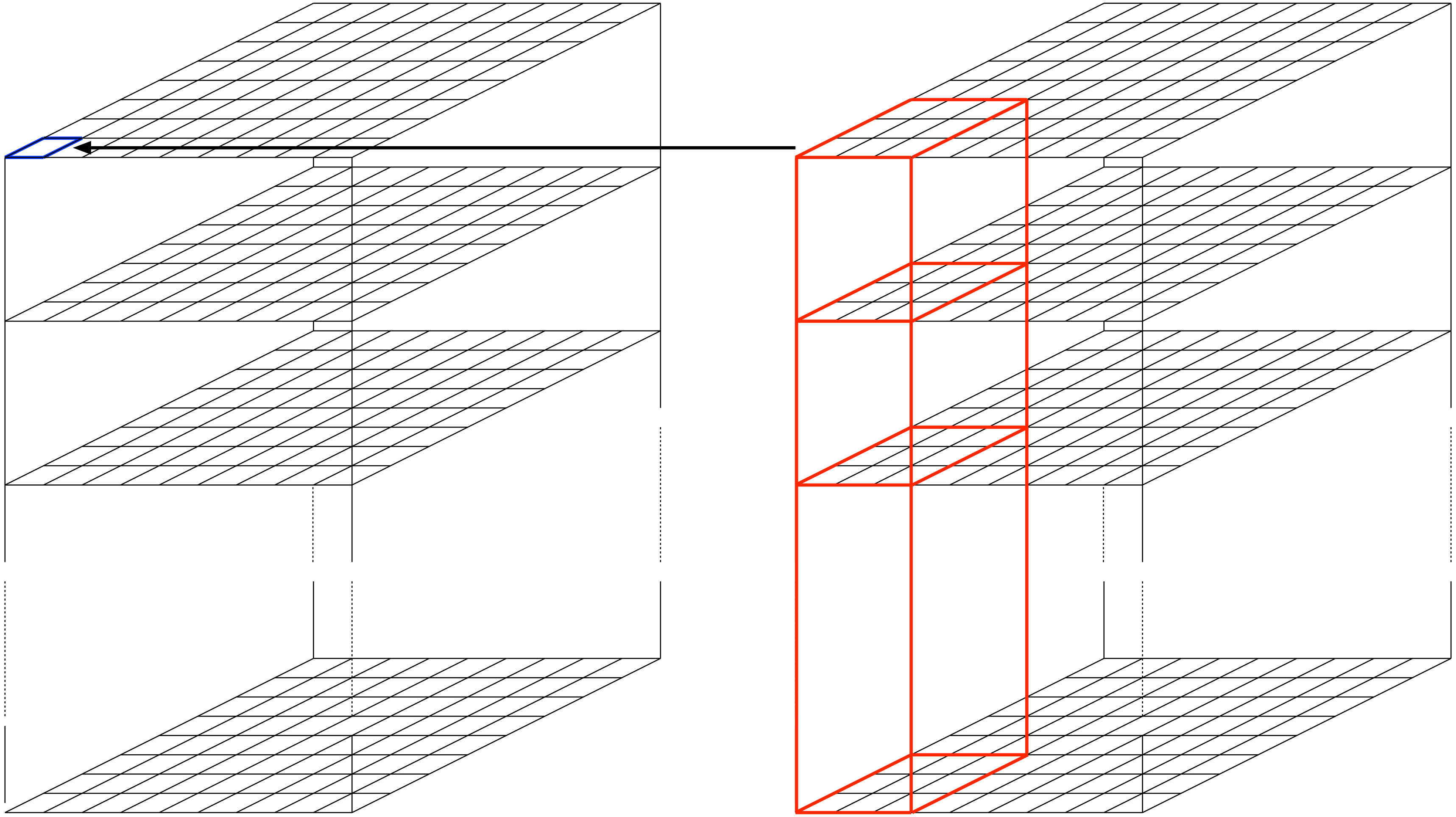
CNN Style 2D Convolution

- Input feature maps
  - 3D tensor
  - N_i inputs x L_r rows x L_c cols
- Filter coefficients
  - 4D tensor
  - N_o outputs x N_i inputs x F_r rows x F_c cols
- Output feature maps
  - 3D tensor
  - N_o outputs x M_r rows x M_c cols



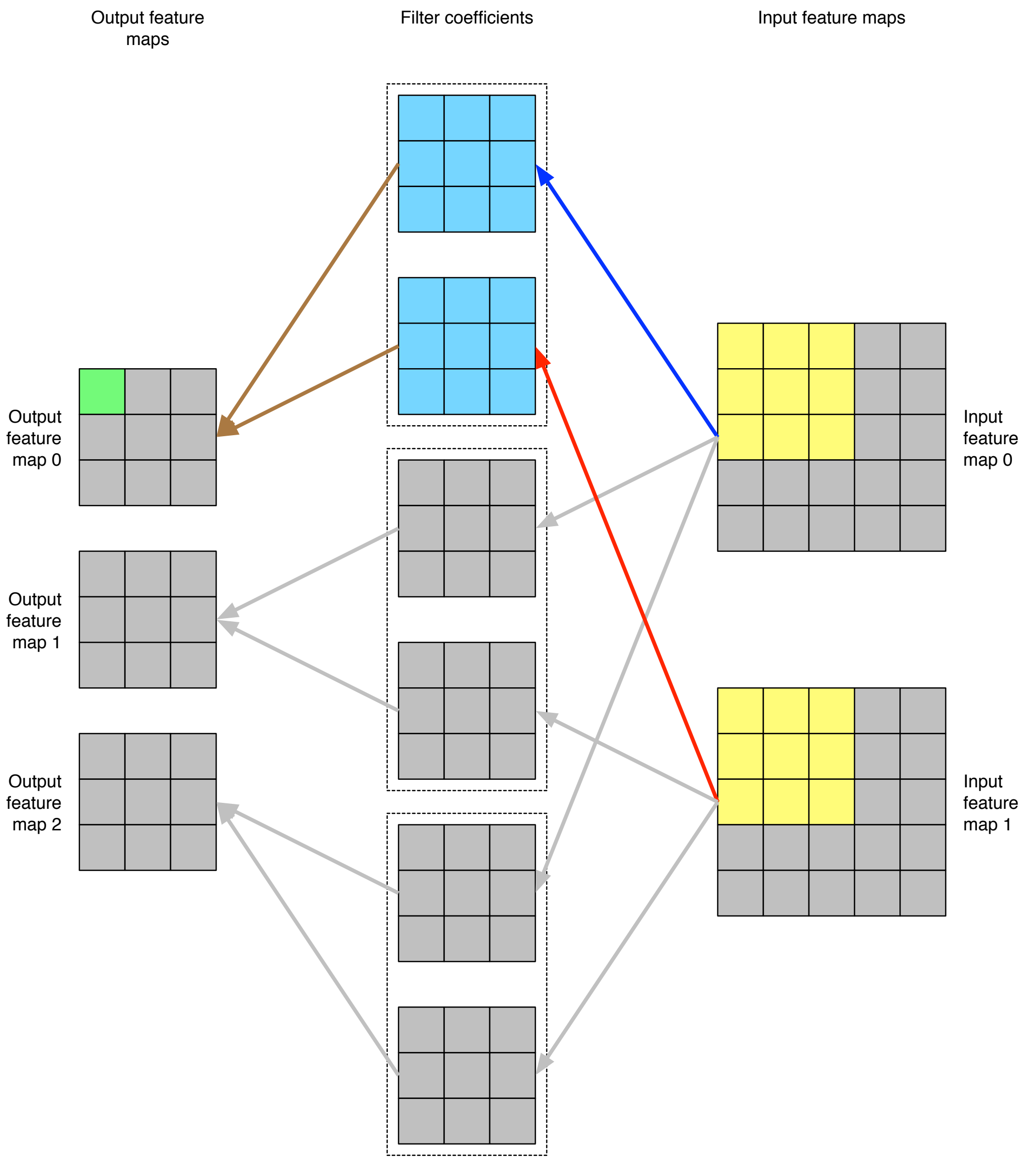
<Filter, Input> Inner Product View

An illustration of the input features used by CNN style 2D convolution with 3x3 filters to produce each output feature



Example

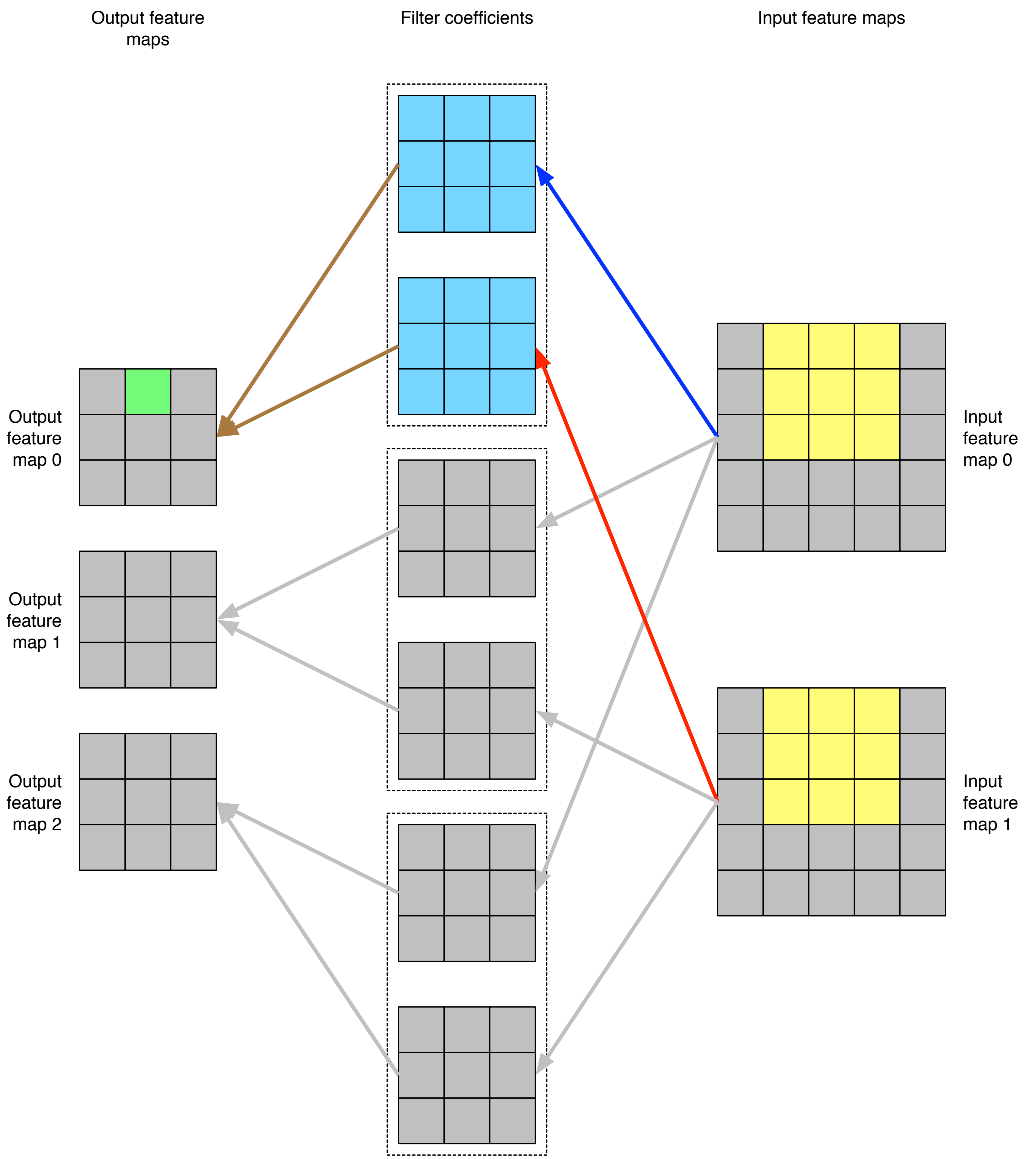
- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

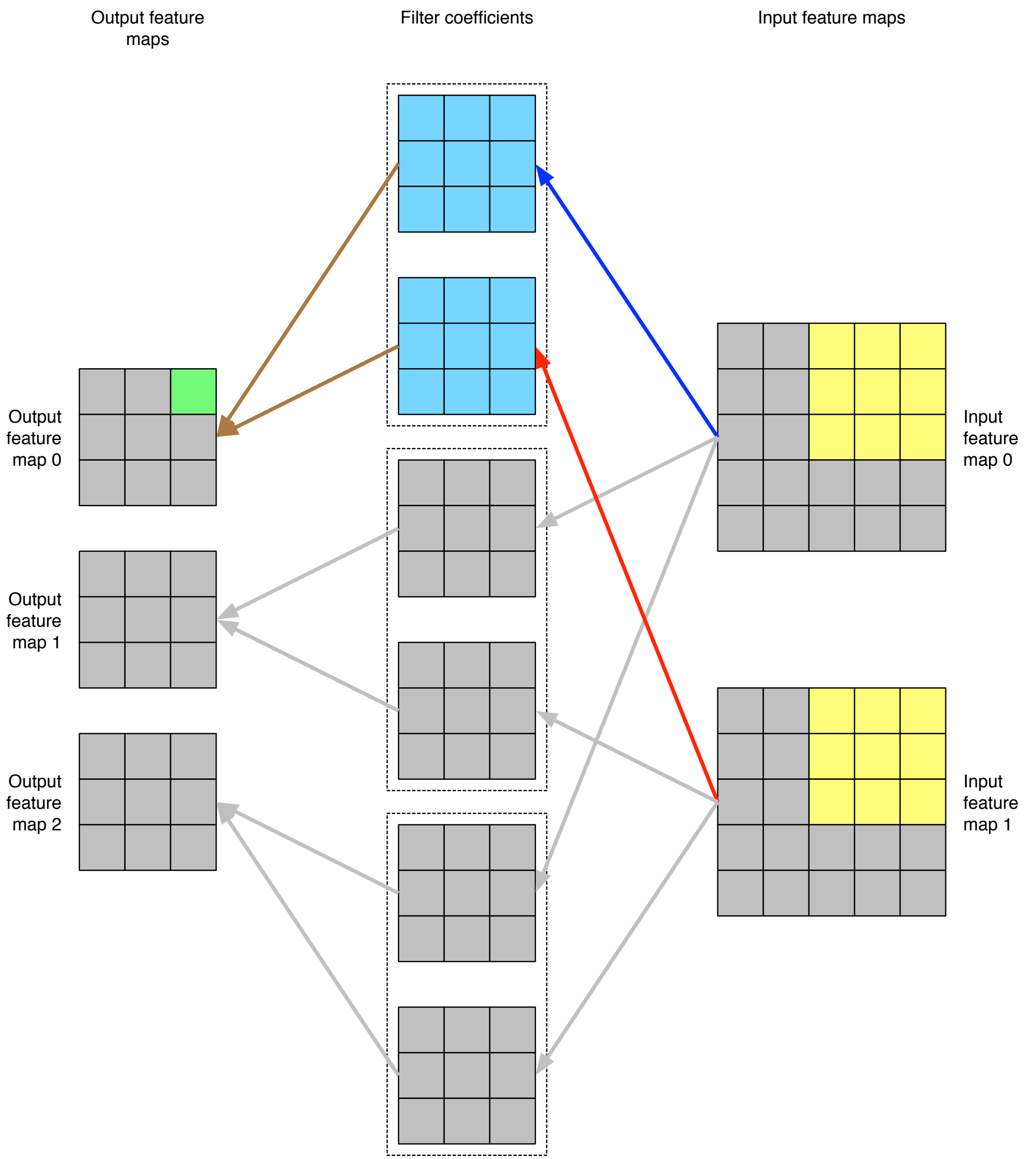
- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



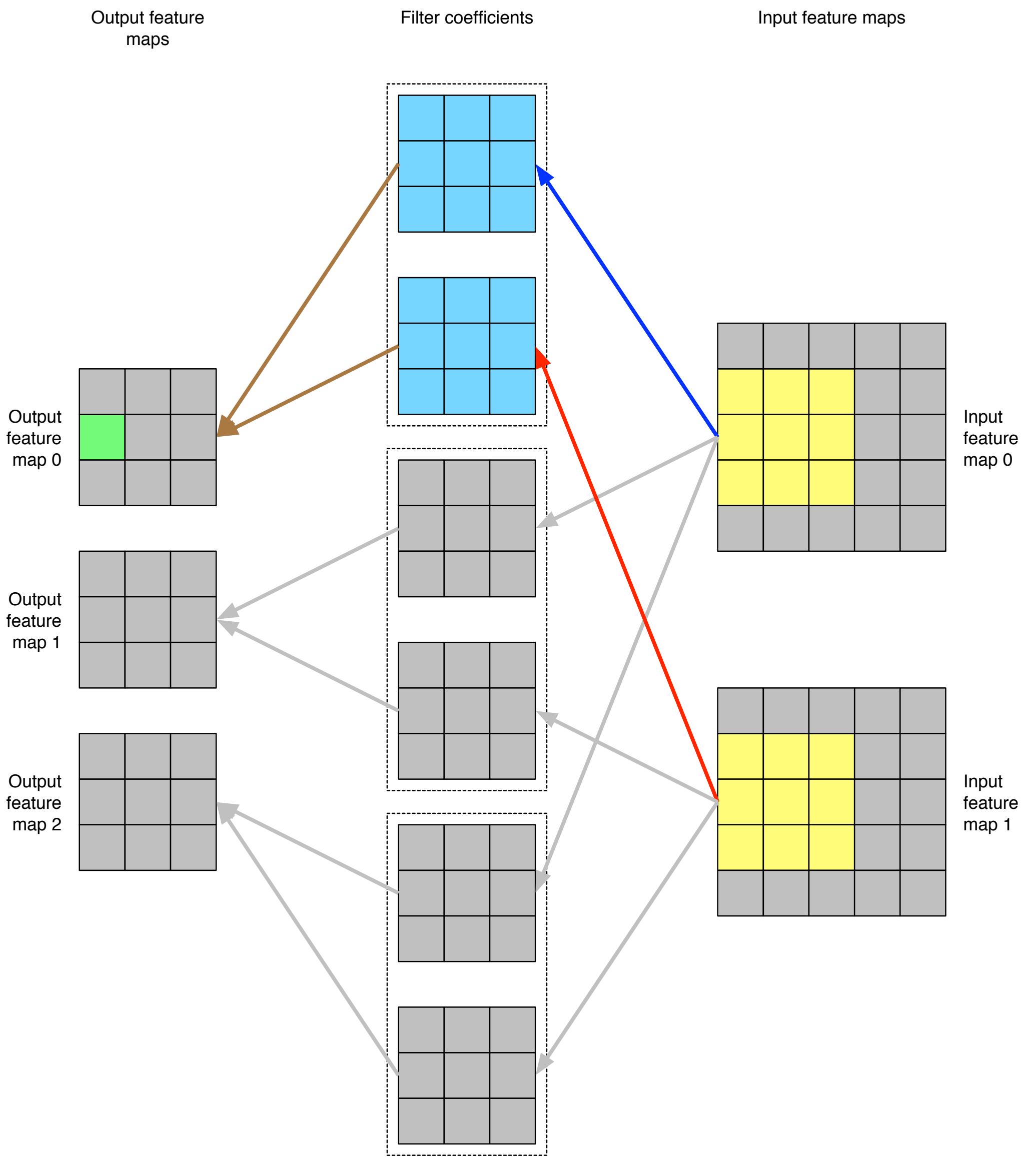
Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



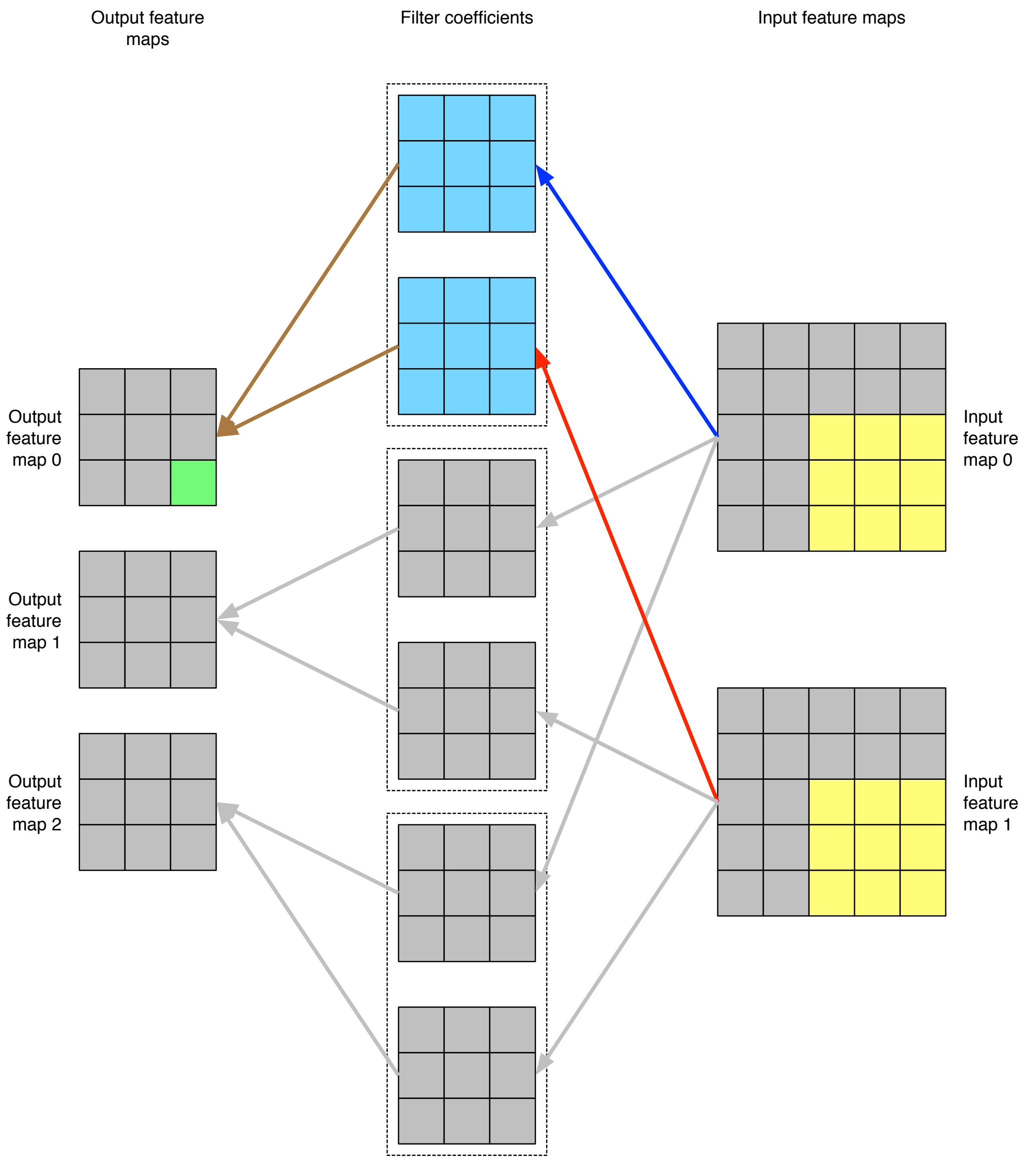
Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output

...

Example

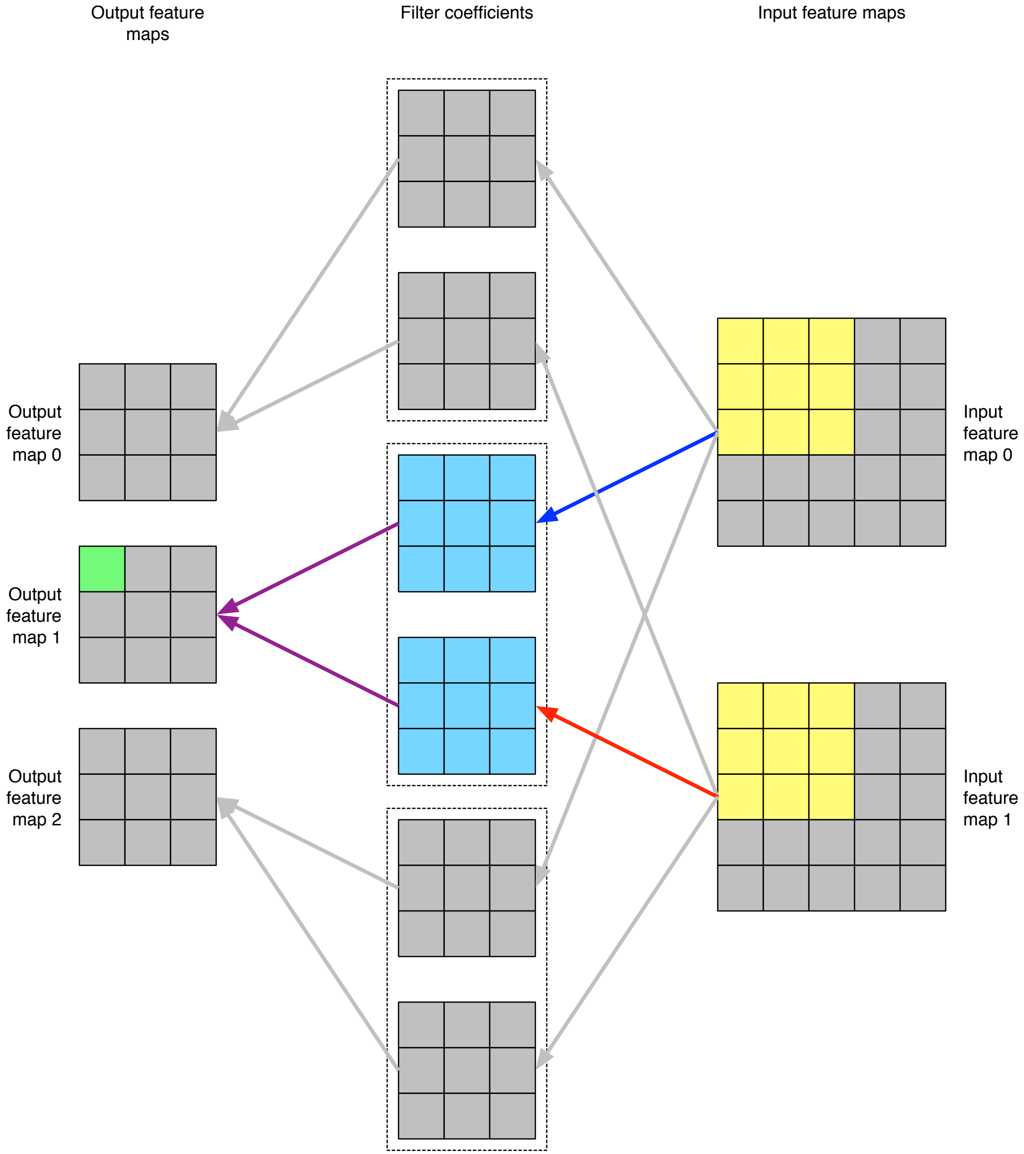
- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

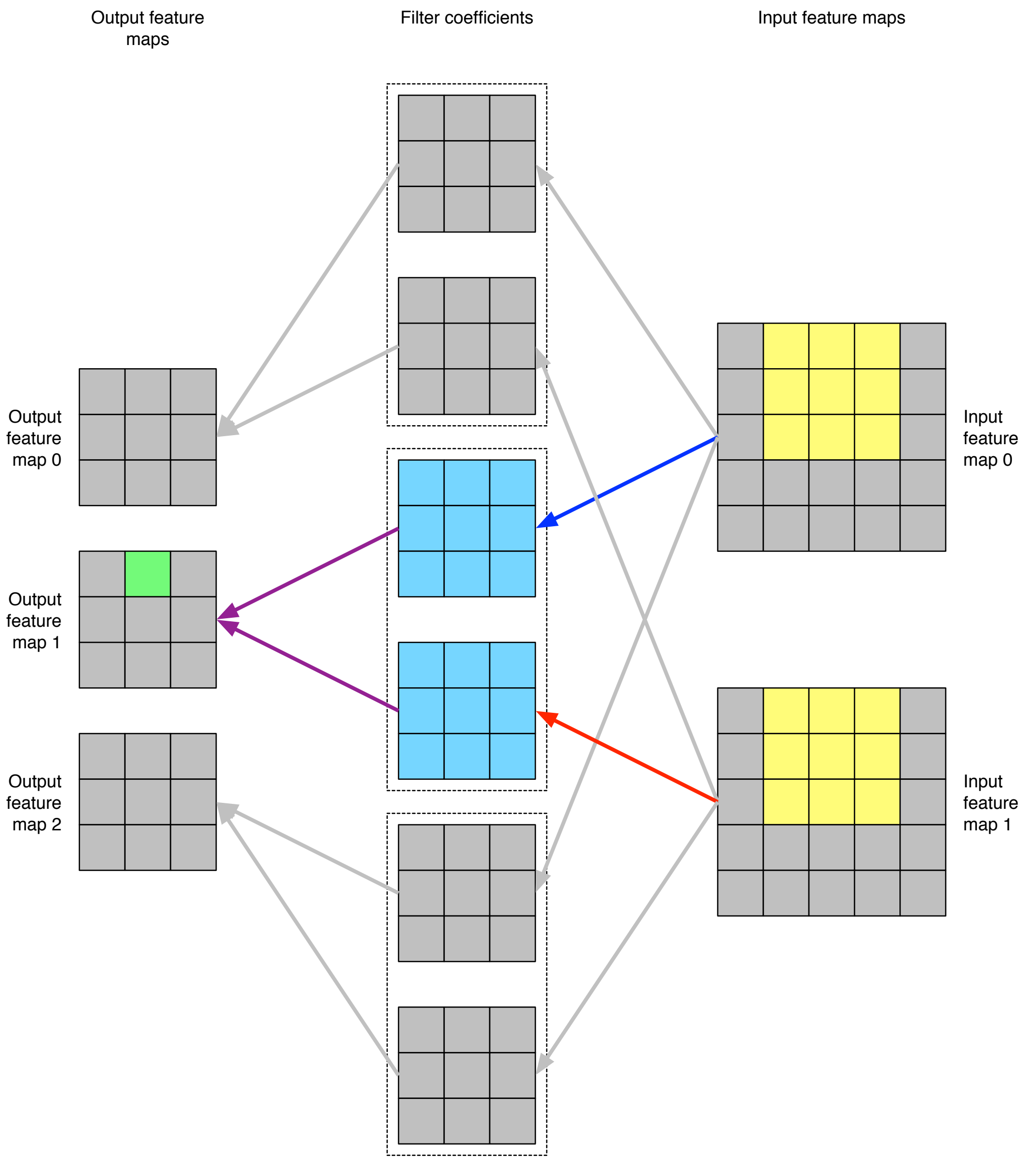
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

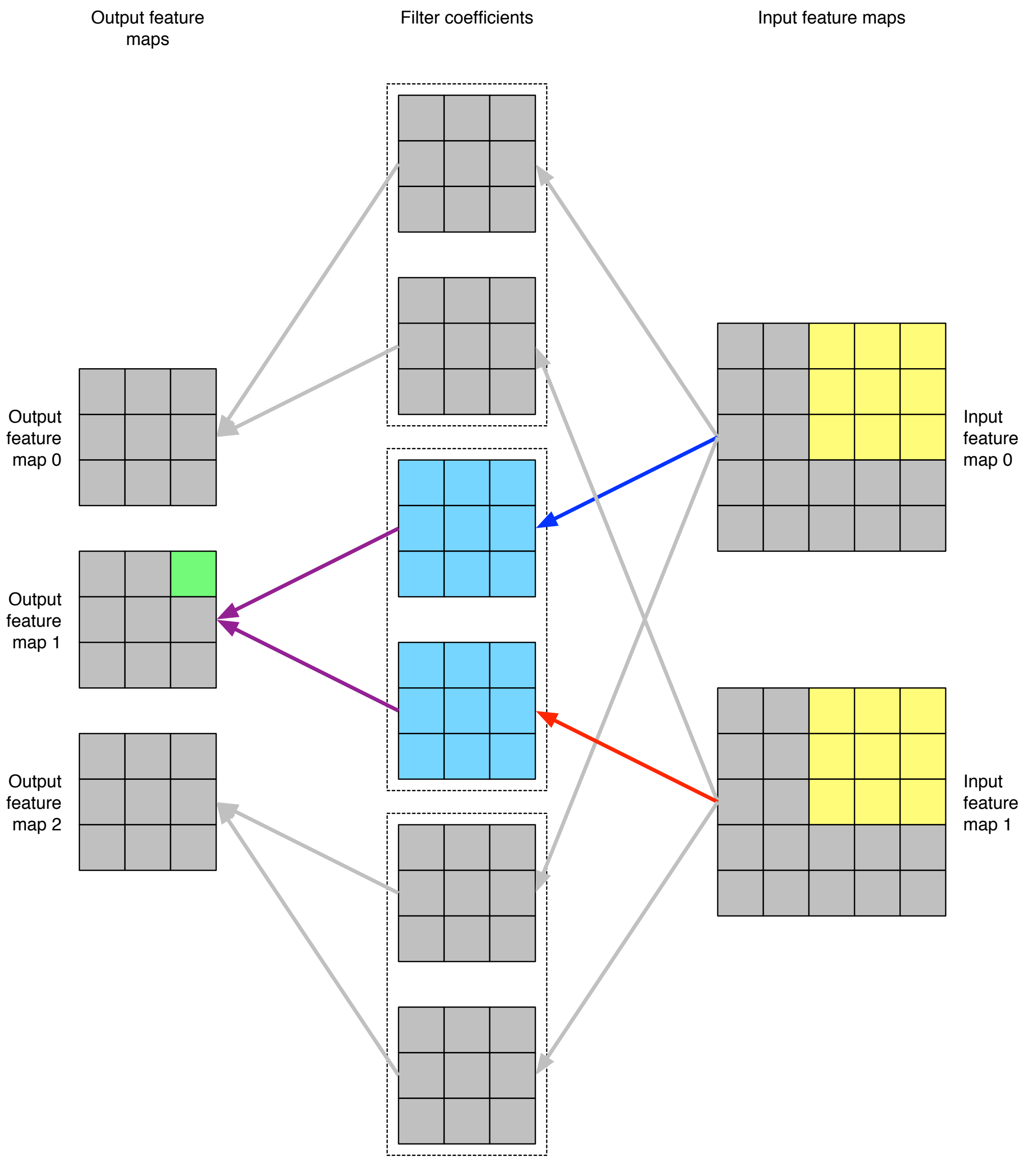
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

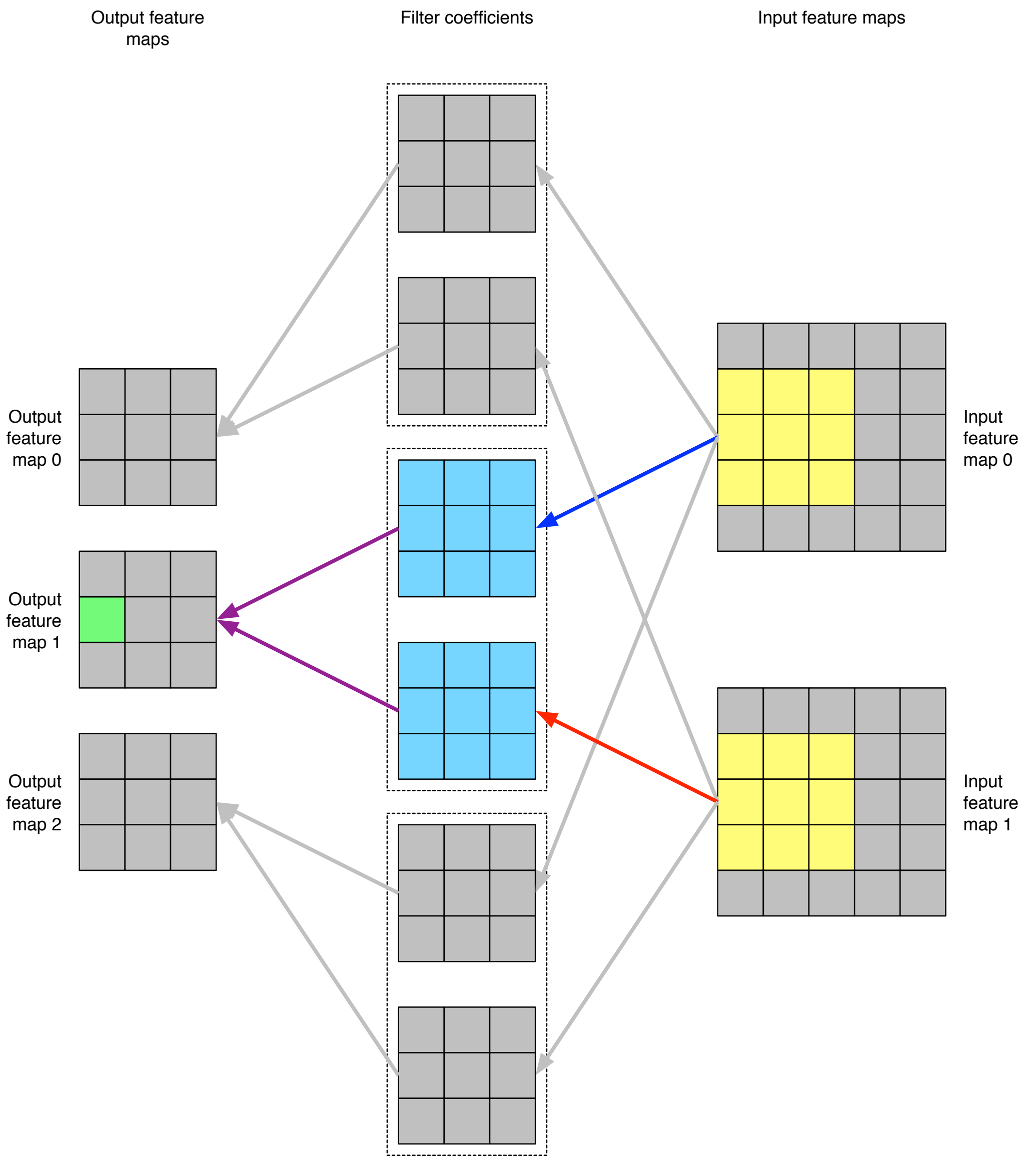
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

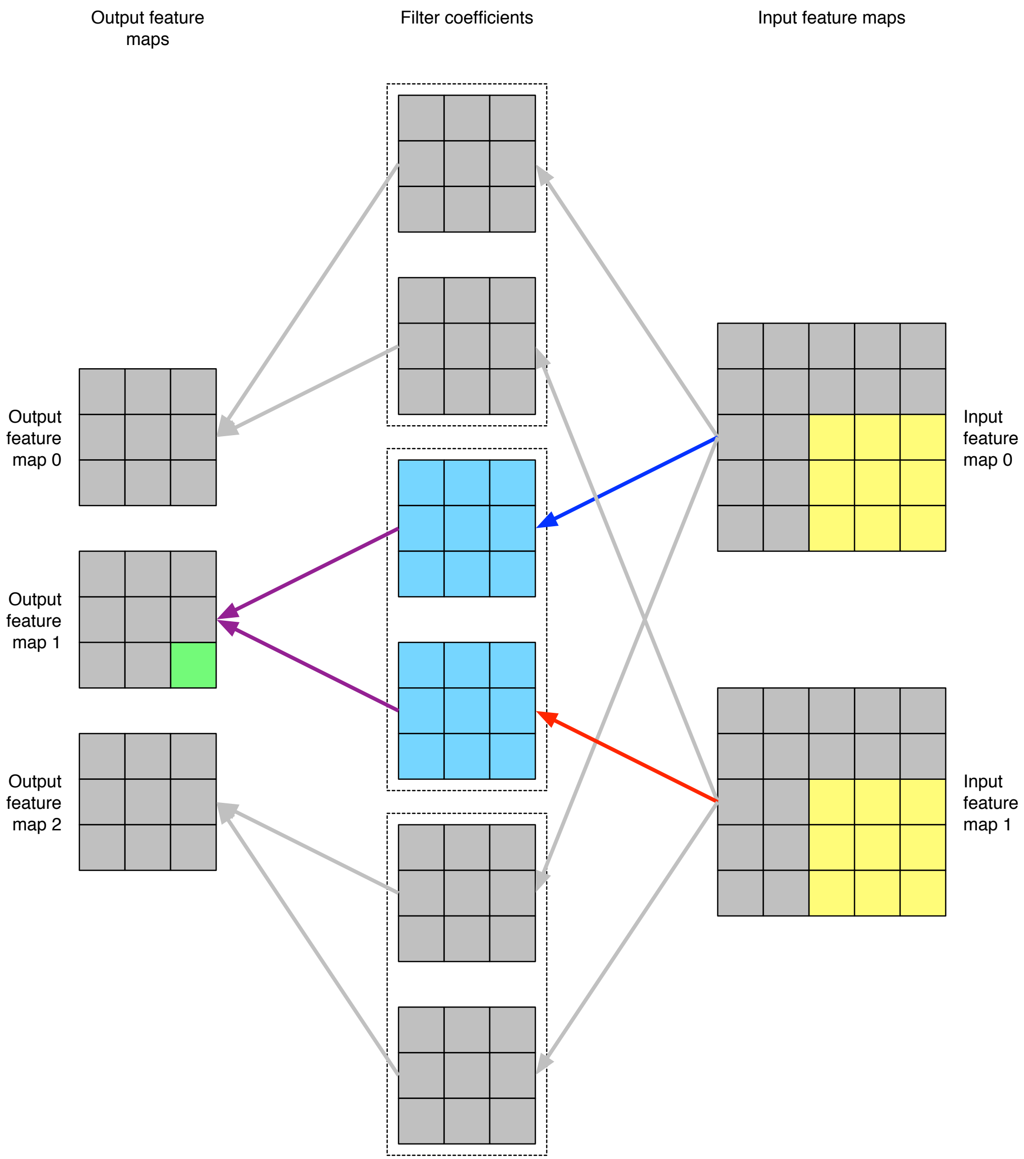
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output

...

Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

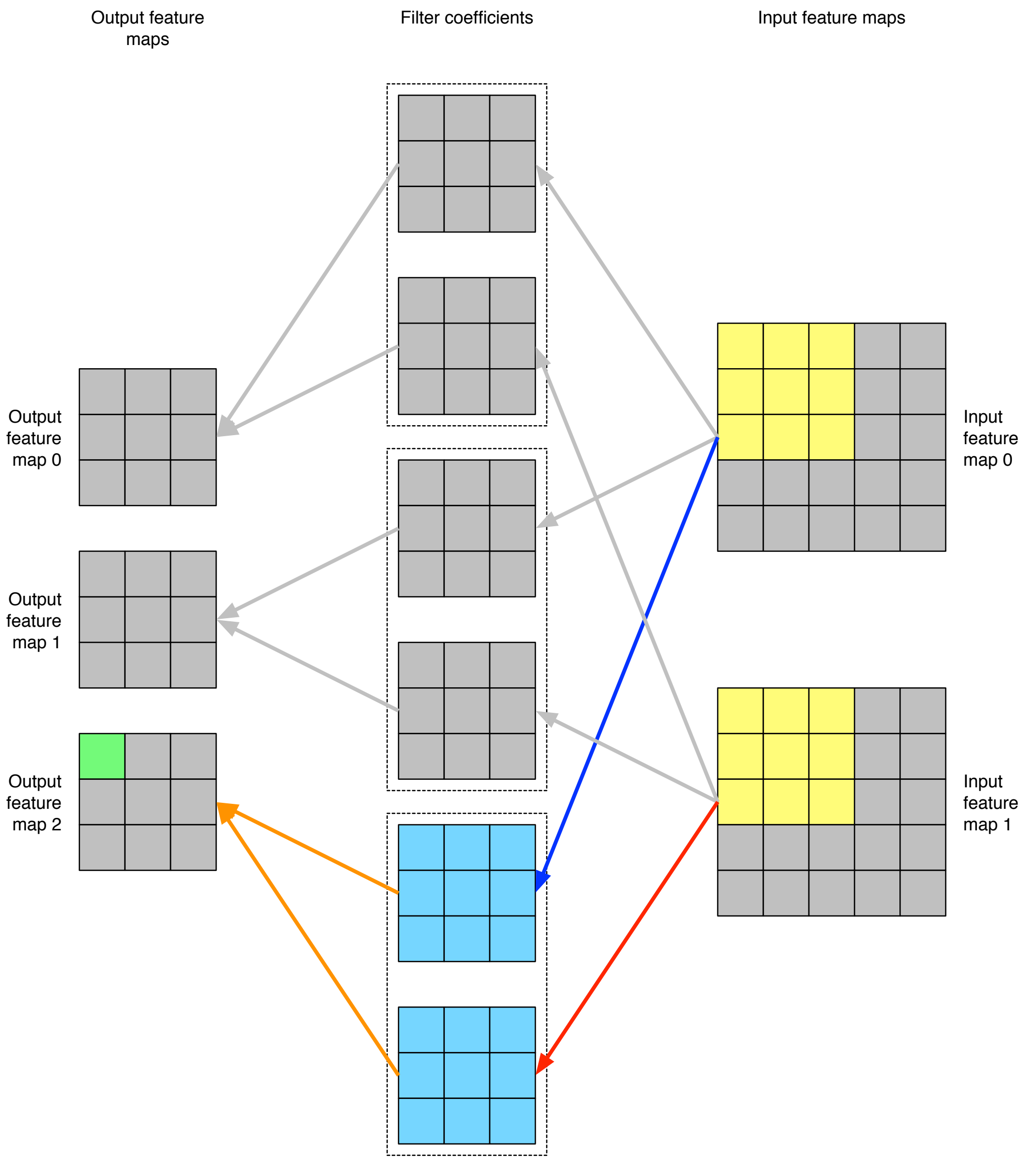
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

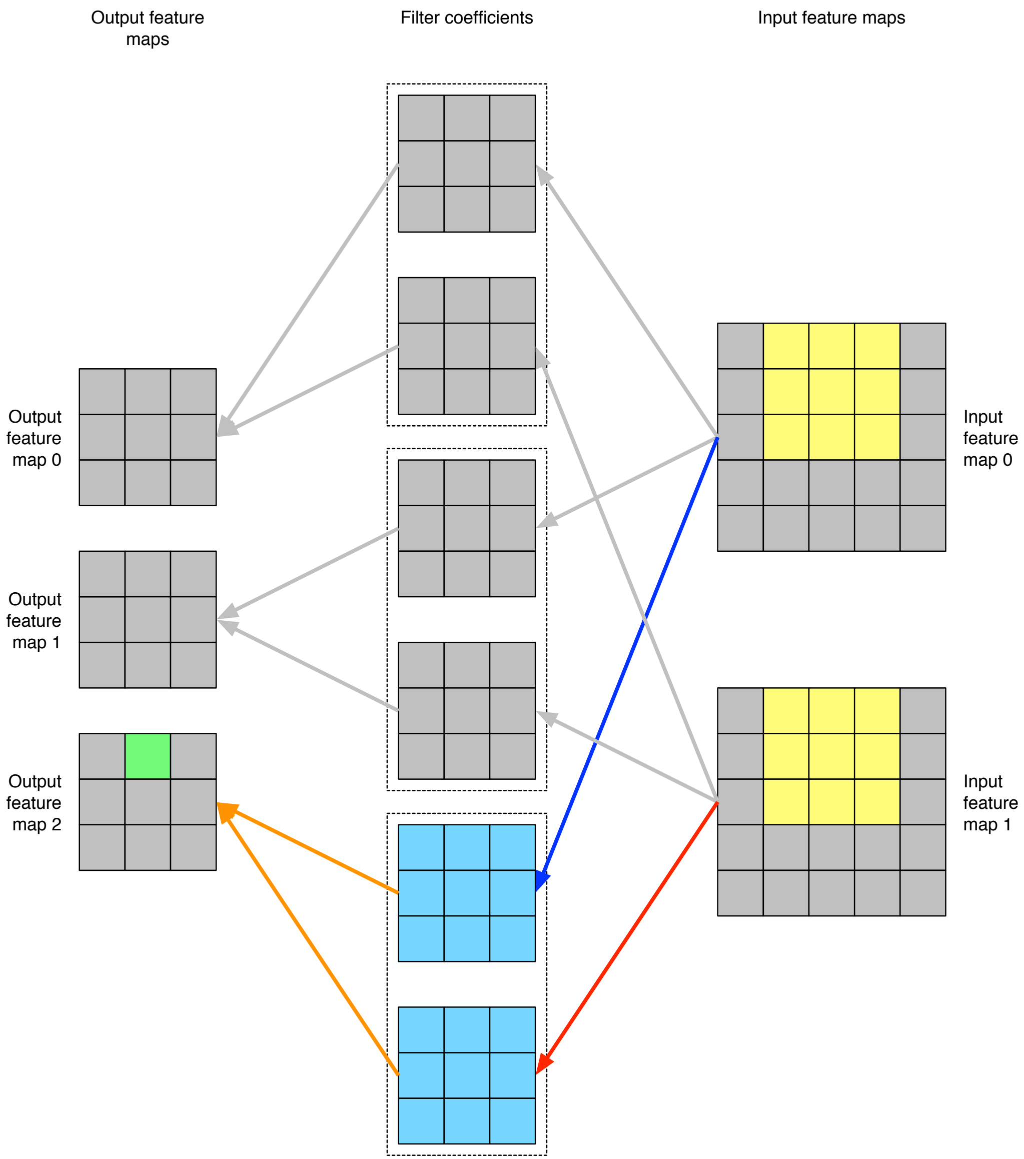
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

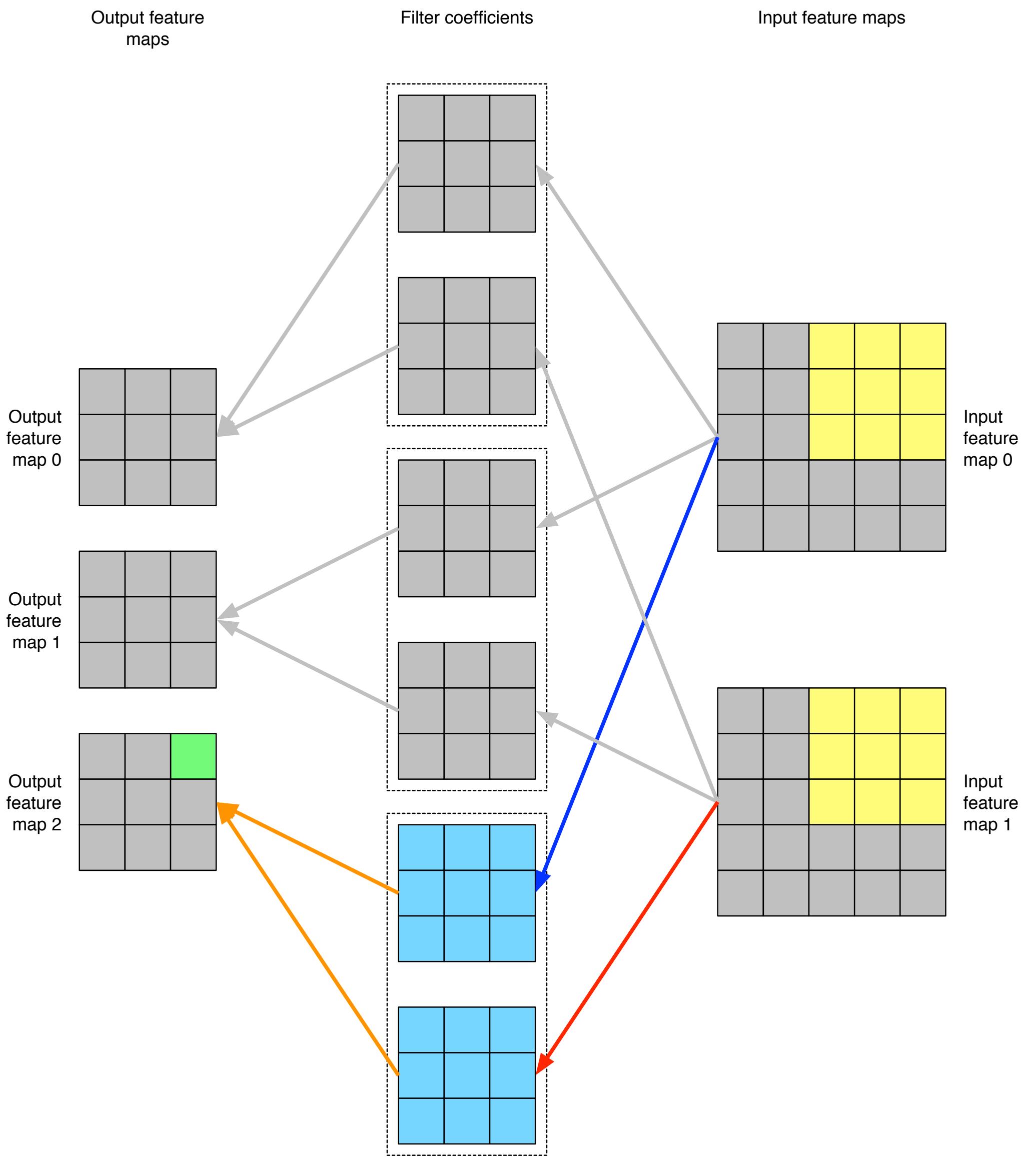
- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

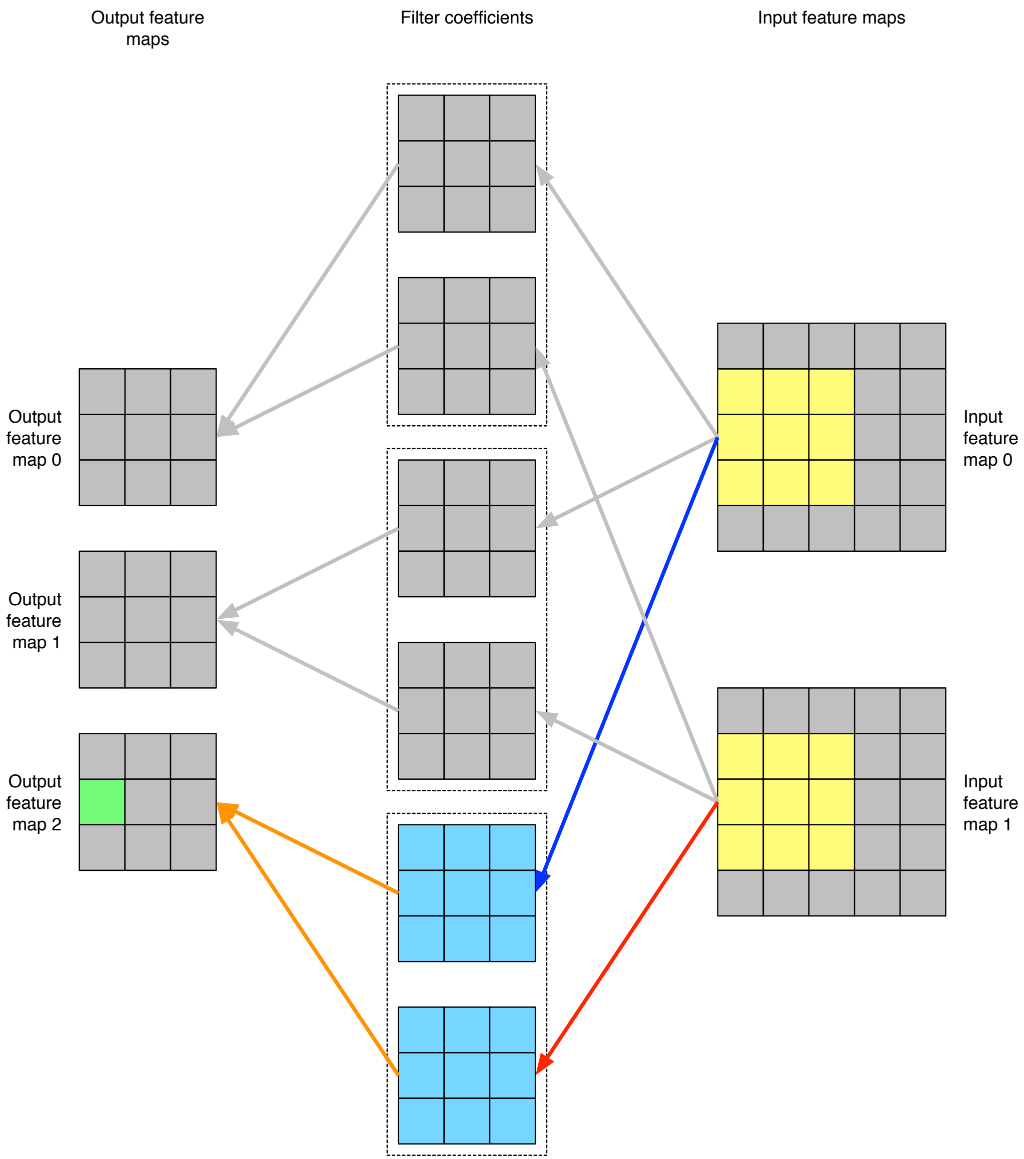
- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



Example

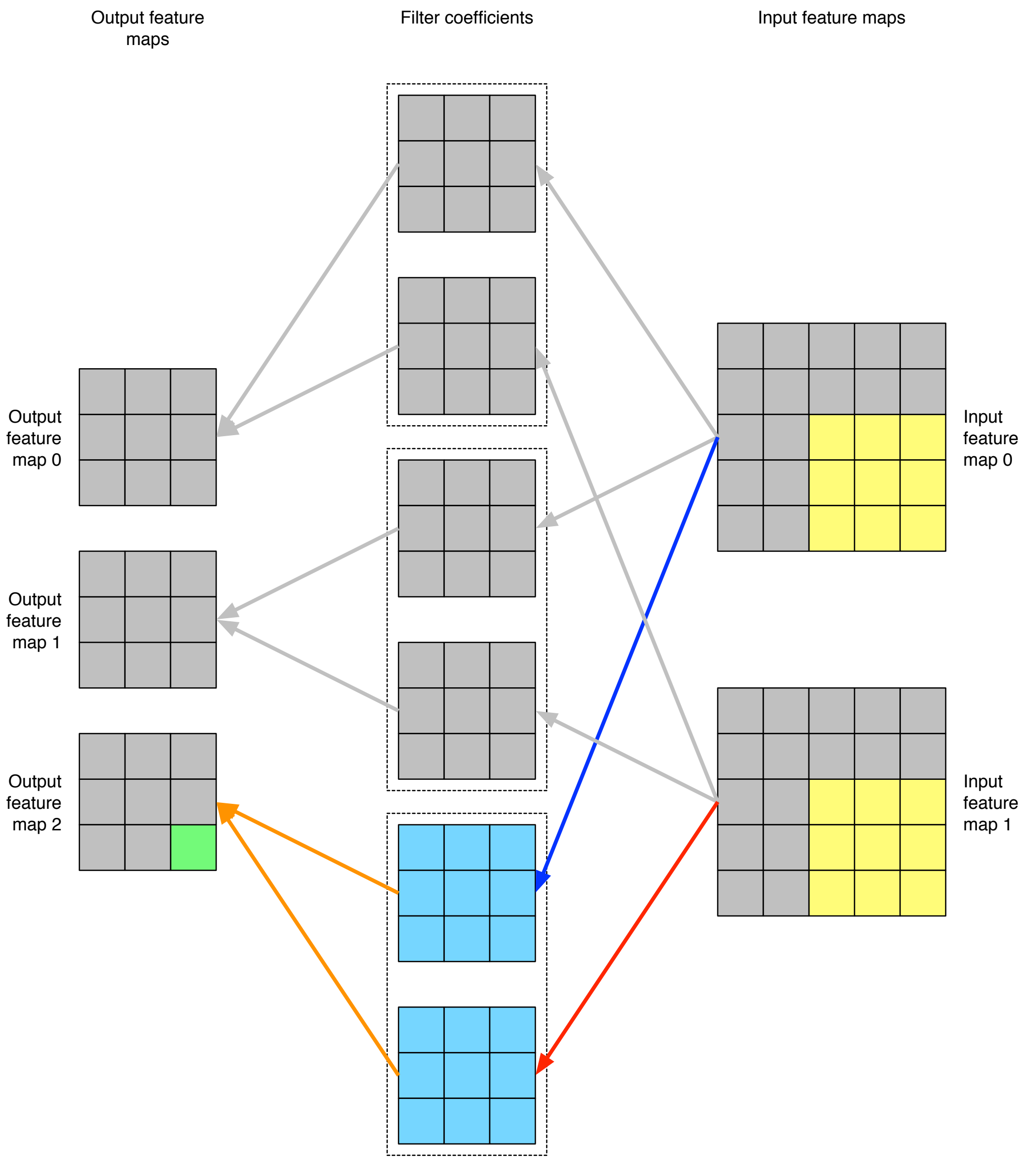
- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps

- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output

…

Example

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



6 Loops View

- Note that 2D correlation is typically used instead of 2D convolution
  - Equivalent with a flip of the filter and indexing change
  - But will still refer to it at CNN style 2D convolution and not CNN style 2D correlation

- Mathematically it's 6 loops (listed from common outer to inner)

Output feature map channel	$n_o = 0, \ldots, N_o - 1$
Output feature map row	$m_r = 0, \ldots, L_r - F_r = M_r - 1$
Output feature map col	$m_c = 0, \ldots, L_c - F_c = M_c - 1$
Input feature map channel	$n_i = 0, \ldots, N_i - 1$
Filter row	$f_r = 0, \ldots, F_r - 1$
Filter col	$f_c = 0, \ldots, F_c - 1$

- For each output n_o, m_r and m_c
  - $y(n_o, m_r, m_c) = \Sigma_{ni}\ \Sigma_{fr}\ \Sigma_{fc}\ h(n_o, n_i, f_r, f_c)\ x(n_i, m_r + f_r, m_c + f_c)$

6 Loops View

- For each output n_o, m_r and m_c
  - $y(n_o, m_r, m_c) = \Sigma_{ni}\ \Sigma_{fr}\ \Sigma_{fc}\ h(n_o, n_i, f_r, f_c)\ x(n_i, m_r + f_r, m_c + f_c)$

- Can be viewed as an inner product (by expanding the summations)
  - Of a vector formed from N_i F_r F_c filter coefficients
  - With a vector formed from F_r F_c elements of each N_i input feature maps
  - To produce a single output at the corresponding row col of an output feature map

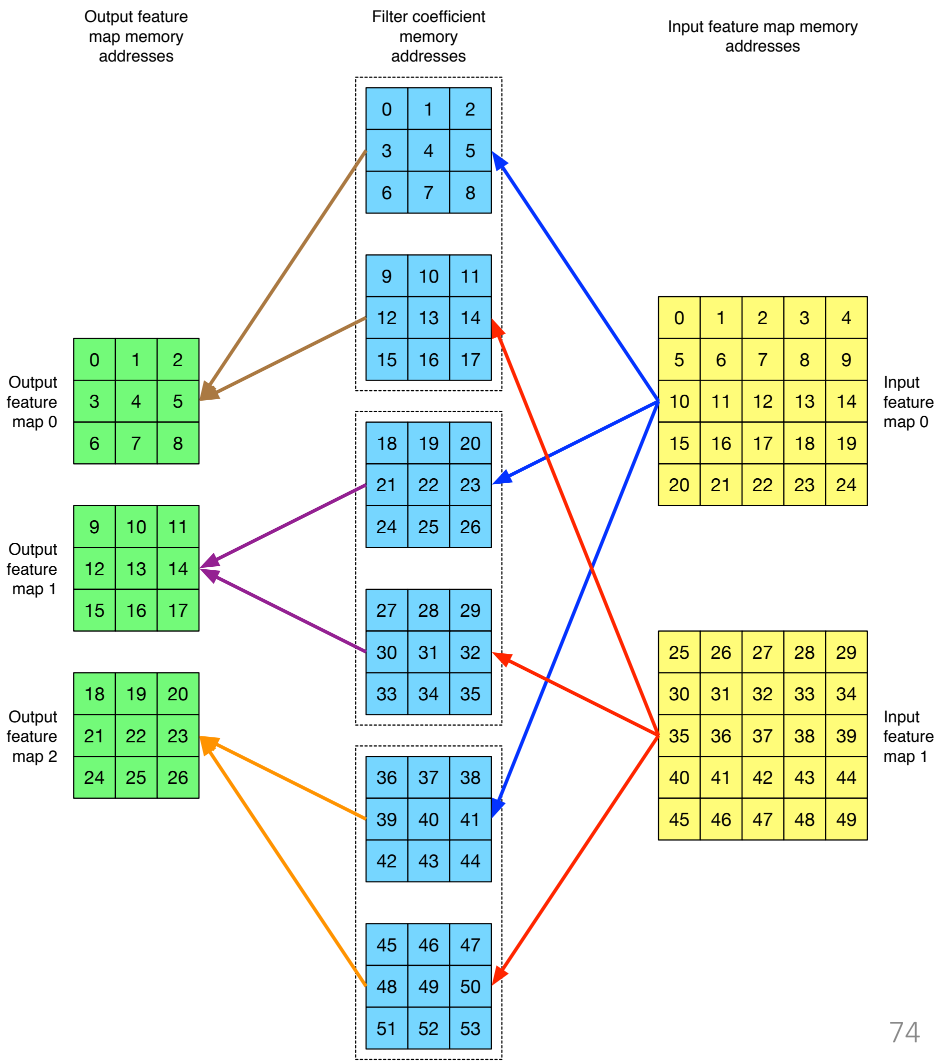
- Repeated
  - For all row col values of the output feature map using the same filter coefficients
  - For all output feature map channels using different filter coefficients for each output feature map channel

Lowering To Matrix Matrix Multiplication

- High performance implementations of CNN style 2D convolution do not explicitly use 6 loops (but compute the same thing)
- The key realization is that CNN style 2D convolution can be written as (lowered to) matrix matrix multiplication: $Y^{2D} = (H^{2D})^T X^{2D}$
  - $(H^{2D})^T$ = reshape 4D filter coefficient tensor to 2D matrix
    - Trivial, nothing actually needs to be reshaped in practice
  - X^{2D} = form 3D input feature map tensor into 2D Toeplitz style filtering matrix
    - This is the key (challenge for efficient implementations)
    - Will generate blocks of this on the fly as each input element is repeated ~ F_r F_c times
  - Y^{2D} = compute 2D matrix of output feature maps
    - Matrix matrix multiplication is efficient in hardware
    - Trivial to reshape to 3D output feature map tensor, nothing actually needs to be done in practice

Lowering To Matrix Matrix Multiplication

- An example showing CNN style 2D convolution is matrix matrix multiplication using 2 x 5 x 5 input feature maps, 3 x 2 x 3 x 3 filters and 3 x 3 x 3 output feature maps
- This figure illustrates memory addresses (specifically offsets to the initial pointer for each array)
- The next page shows where the memory addresses go in matrix matrix multiplication



Lowering To Matrix Matrix Multiplication

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26

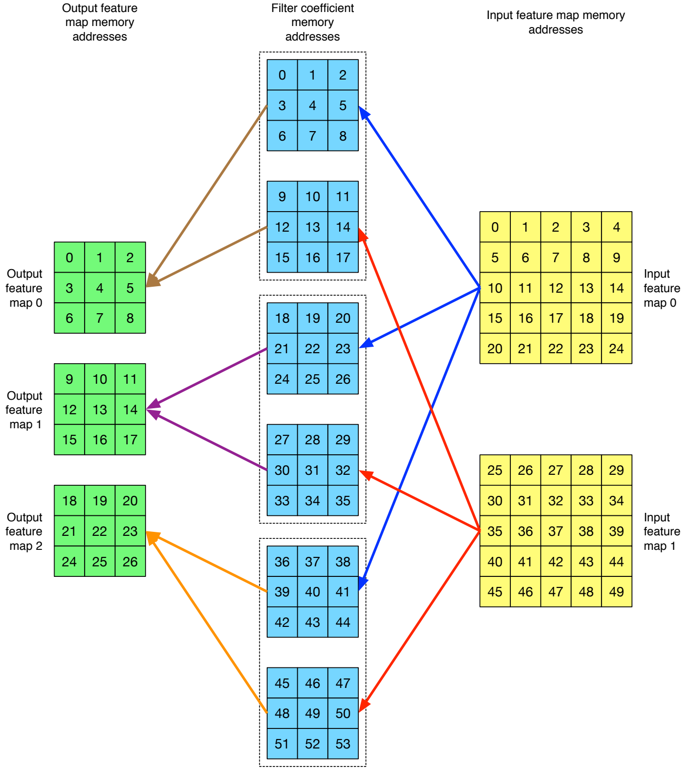
Output feature map memory addresses
(note vectorization)

=

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53

Filter coefficient memory addresses
(note vectorization)

- Main figure is matrix form
- Small figure is convolution form from previous page for reference



Input feature map memory addresses (note Toeplitz filtering matrix structure)

0	1	2	5	6	7	10	11	12
1	2	3	6	7	8	11	12	13
2	3	4	7	8	9	12	13	14
5	6	7	10	11	12	15	16	17
6	7	8	11	12	13	16	17	18
7	8	9	12	13	14	17	18	19
10	11	12	15	16	17	20	21	22
11	12	13	16	17	18	21	22	23
12	13	14	17	18	19	22	23	24
25	26	27	30	31	32	35	36	37
26	27	28	31	32	33	36	37	38
27	28	29	32	33	34	37	38	39
30	31	32	35	36	37	40	41	42
31	32	33	36	37	38	41	42	43
32	33	34	37	38	39	42	43	44
35	36	37	40	41	42	45	46	47
36	37	38	41	42	43	46	47	48
37	38	39	42	43	44	47	48	49

Limiting Cases

- Limiting cases illustrated via depth wise separable convolution that splits CNN style 2D convolution into 2 layers
  - Traditional 2D convolution followed by CNN style 2D convolution with 1 x 1 filters
  - Each is less general vs the original mixing, but back to back them give 1 extra level of depth

- Traditional 2D convolution to mix across space ($N_i = N_o = 1$)
  - Can also get small values of N_i and N_o via grouping
  - Equivalent to vector matrix multiplication
  - Note that K dimension reduces from ($N_i F_r F_c$) to ($F_r F_c$)

- CNN style 2D convolution with 1 x 1 filters to mix across channel
  - Equivalent to standard matrix matrix multiplication
  - Note that K dimension reduces from ($N_i F_r F_c$) to N_i

Traditional 2D Convolution

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26

Output feature map memory addresses
(note vectorization)

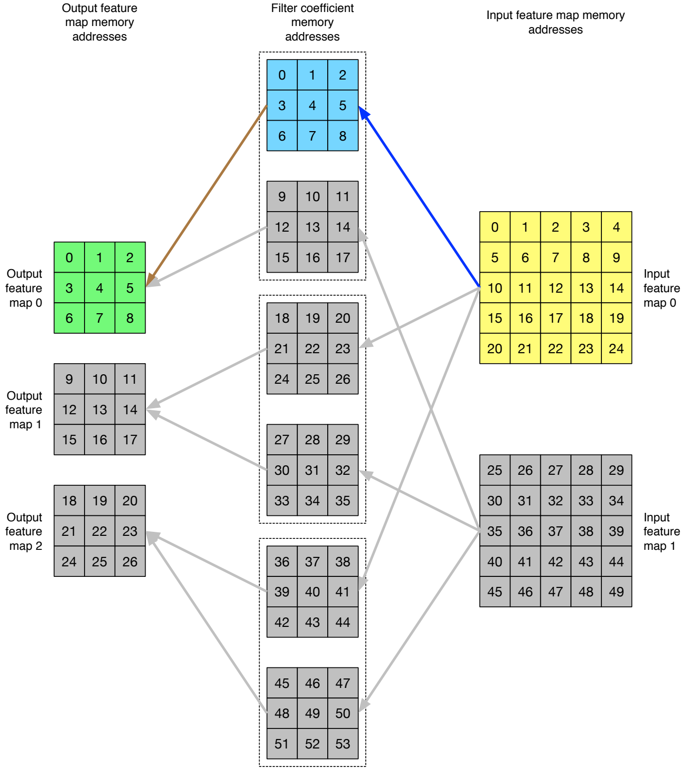
=

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53

Filter coefficient memory addresses
(note vectorization)

- Traditional convolution

- Equivalent to vector matrix multiplication

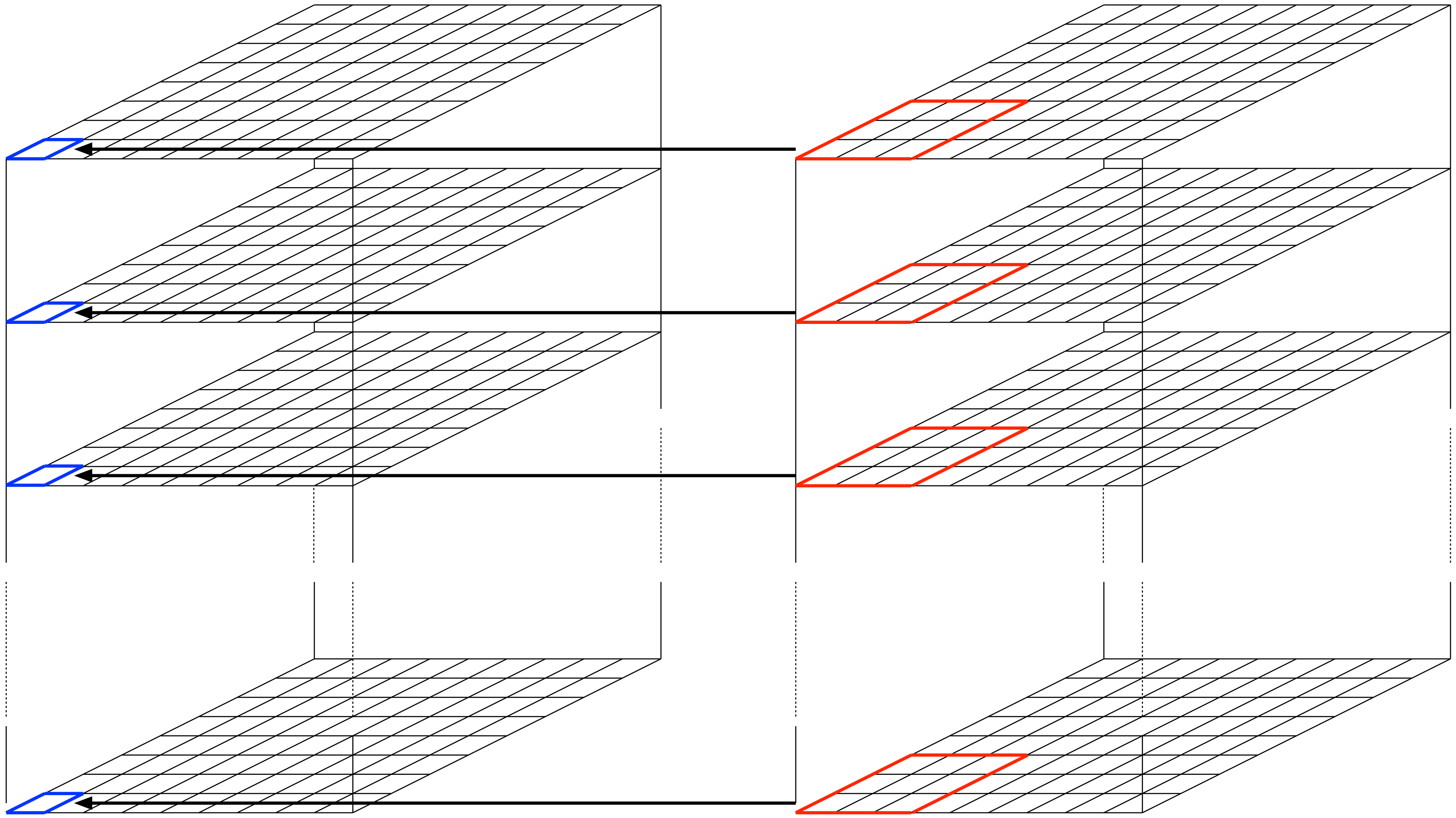


Input feature map memory addresses (note Toeplitz filtering matrix structure)

0	1	2	5	6	7	10	11	12
1	2	3	6	7	8	11	12	13
2	3	4	7	8	9	12	13	14
5	6	7	10	11	12	15	16	17
6	7	8	11	12	13	16	17	18
7	8	9	12	13	14	17	18	19
10	11	12	15	16	17	20	21	22
11	12	13	16	17	18	21	22	23
12	13	14	17	18	19	22	23	24
25	26	27	30	31	32	35	36	37
26	27	28	31	32	33	36	37	38
27	28	29	32	33	34	37	38	39
30	31	32	35	36	37	40	41	42
31	32	33	36	37	38	41	42	43
32	33	34	37	38	39	42	43	44
35	36	37	40	41	42	45	46	47
36	37	38	41	42	43	46	47	48
37	38	39	42	43	44	47	48	49

Traditional 2D Convolution

An illustration of the input features used by traditional 2D convolution with 3x3 filters (equivalent to fully grouped CNN style 2D convolution with 3x3 filters) to produce each output feature



CNN Style 2D Convolution With 1x1 Filters

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26

Output feature map memory addresses
(note vectorization)

=

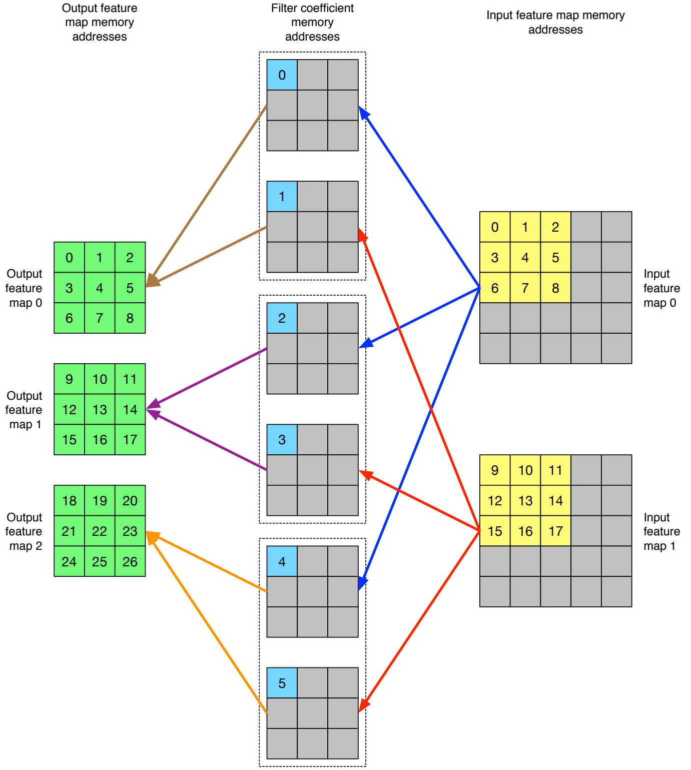
0									1								
2									3								
4									5								

Filter coefficient memory addresses
(note vectorization)

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17

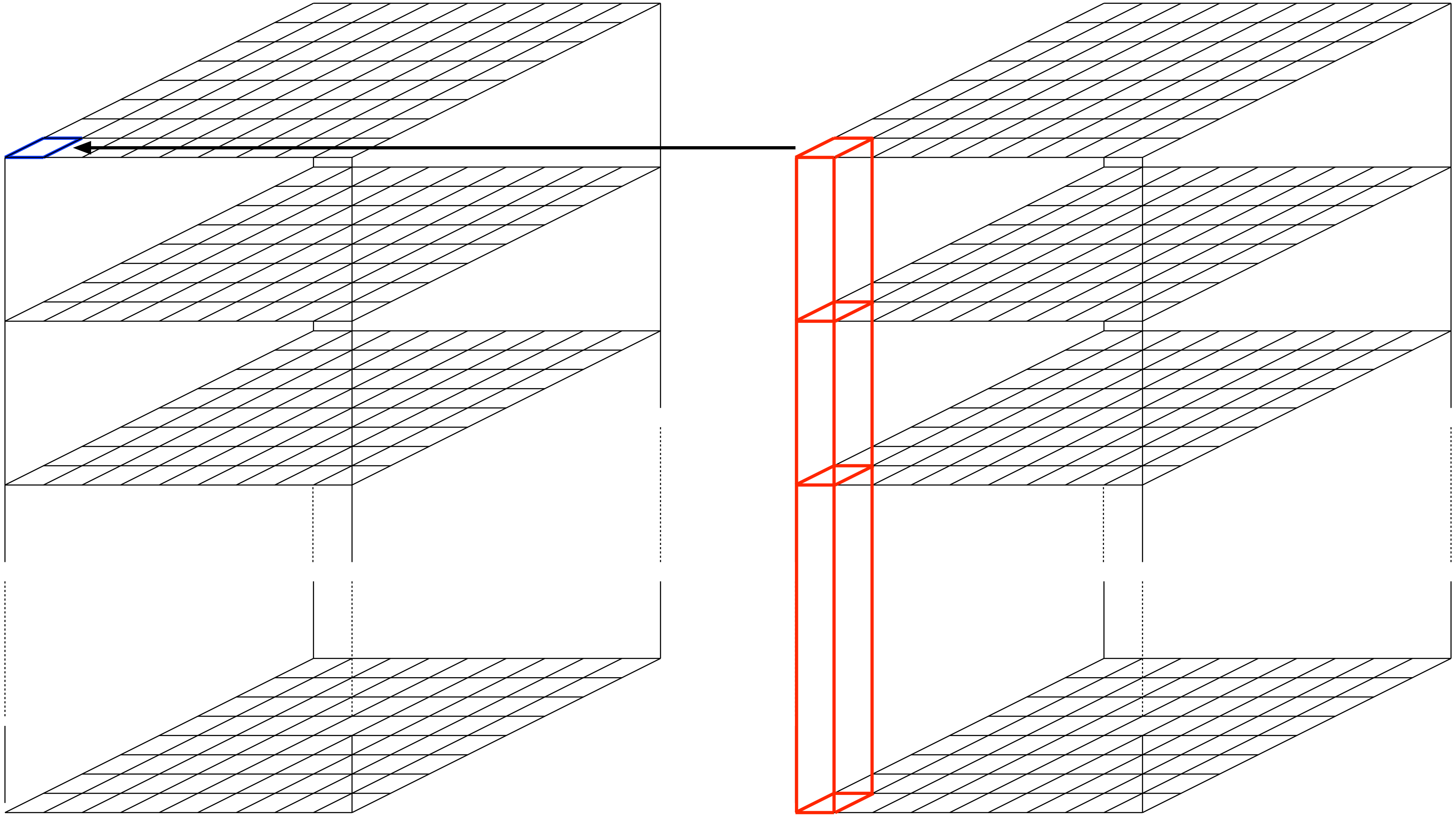
Input feature map memory addresses (note Toeplitz filtering matrix structure)

- CNN style 2D convolution with 1x1 filters
- Equivalent to pure matrix matrix multiplication



CNN Style 2D Convolution With 1x1 Filters

An illustration of the input features used by CNN style 2D convolution with 1x1 filters to produce each output feature



Arithmetic Intensity

- Memory
  - Formulas
    - Input feature maps: $N_i L_r L_c$
    - Output feature maps: $N_o M_r M_c$
    - Filter coefficients: $N_i N_o F_r F_c$
  - Early in the network feature map memory tends to dominate
  - Deeper in the network filter coefficient memory tends to dominate

- Compute
  - Formula for MACs
    - $(N_o) (M_r M_c) (N_i F_r F_c) = (N_o M_r M_c) (N_i F_r F_c)$
      = (number of outputs) (number of input MACs per output)

Arithmetic Intensity

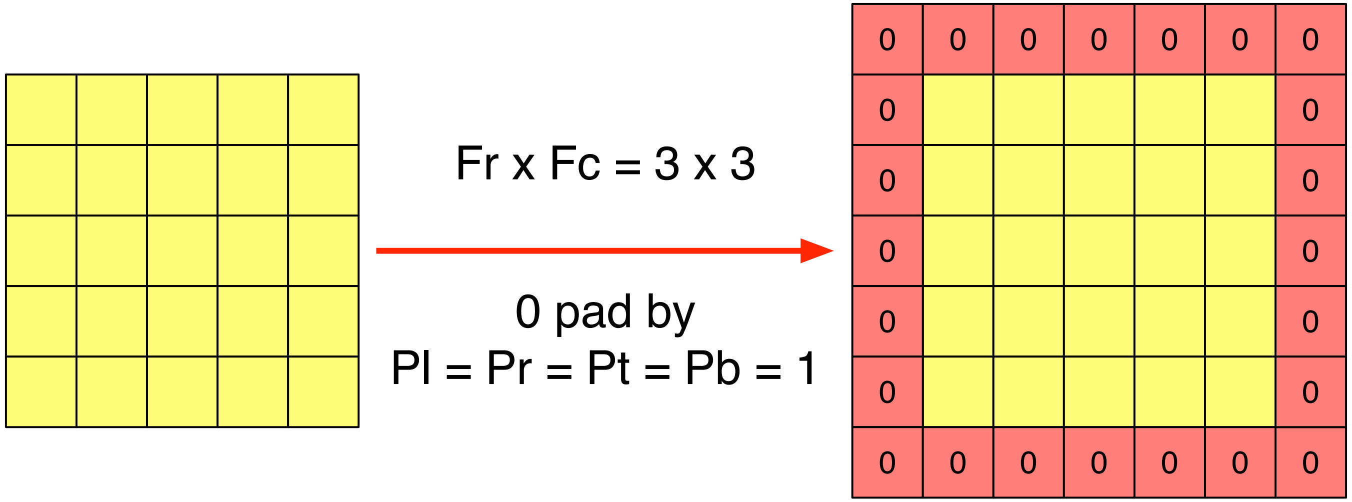
- Arithmetic intensity
  - Compute = $N_i N_o F_r F_c M_r M_c$ (MACs)
  - Data movement = $N_i L_r L_c + N_o M_r M_c + N_i N_o F_r F_c$ (elements)
  - Ratio = compute / data movement

Alternate View

- Alternate view of CNN style 2D convolution
  - Add together F_r * F_c CNN style 2D convolutions with 1x1 filter size
    - Reminder: CNN style convolution with 1x1 filters is pure matrix matrix multiplication
    - Input size is reduced to M_r x M_c for each with an appropriate shift / offset of the original input
  - Tradeoffs
    - Advantage of simpler input feature map matrix structure and associated data movement logic
    - Drawback of additional input feature map memory movement
  - Side benefit: useful for understanding back propagation through a CNN style 2D convolution layer

0 Padding

- Variant: input feature map 0 padding
  - P_l left, P_r right, P_t top, P_b bottom
  - Typically $P_l + P_r = F_c - 1$ and $P_t + P_b = F_r - 1$
  - Used for same size input / output feature maps
  - Implementation key is efficient 0 insert

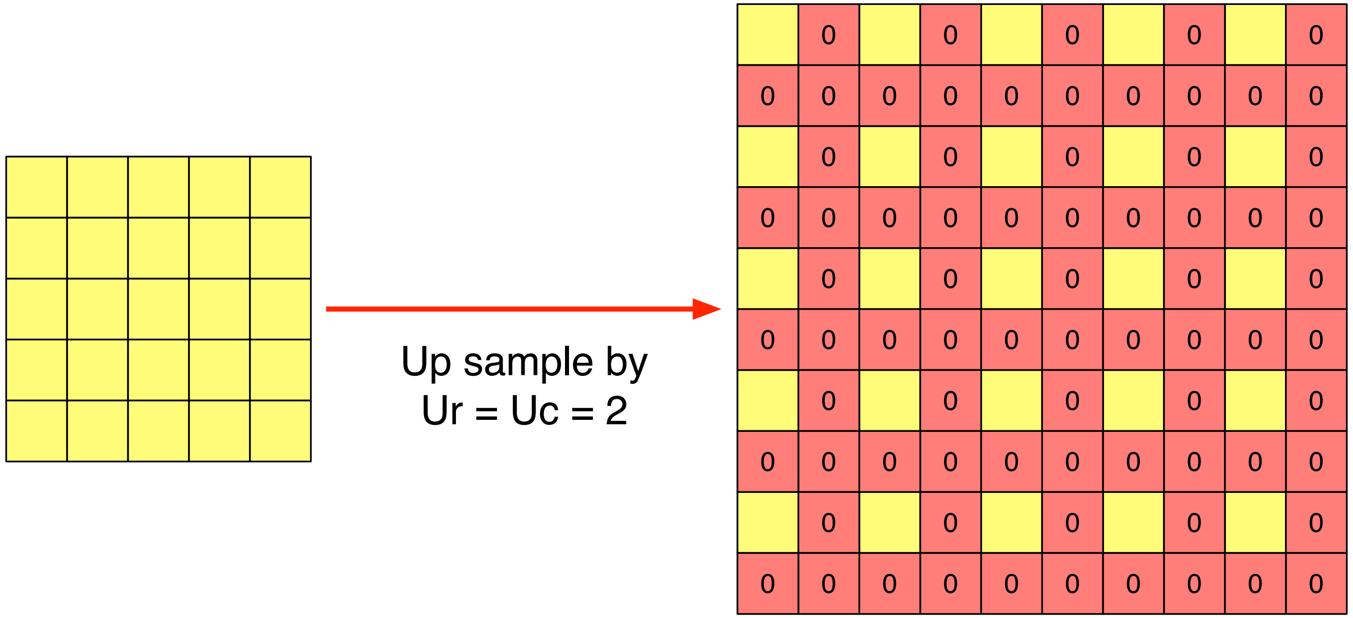


Fr x Fc = 3 x 3

0 pad by
Pl = Pr = Pt = Pb = 1

Up Sampling Input Feature Maps

- Variants: input feature map up sampling
  - U_r rows, U_c cols
  - Frequently called transposed convolution, fractionally strided convolution or deconvolution
  - Used in decoder style head designs
  - Implementation key is input memory reuse
  - Alternatives are bilinear and nearest neighbor interpolation

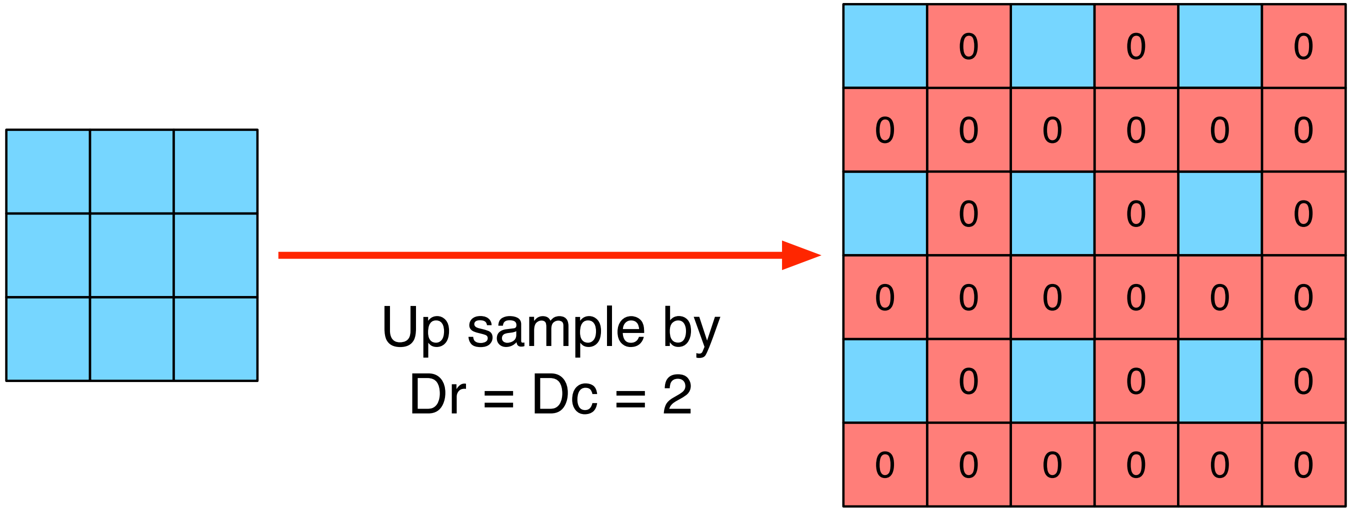


Up sample by
Ur = Uc = 2

	0		0		0		0		0
0	0	0	0	0	0	0	0	0	0
	0		0		0		0		0
0	0	0	0	0	0	0	0	0	0
	0		0		0		0		0
0	0	0	0	0	0	0	0	0	0
	0		0		0		0		0
0	0	0	0	0	0	0	0	0	0
	0		0		0		0		0
0	0	0	0	0	0	0	0	0	0

Up Sampling Filters

- Variants: filter coefficient up sampling
  - D_r rows, D_c cols
  - Typically called dilated or Atrous convolution
  - Used to maintain spatial resolution with large receptive field
  - Implementation key is input feature map filtering matrix row removal

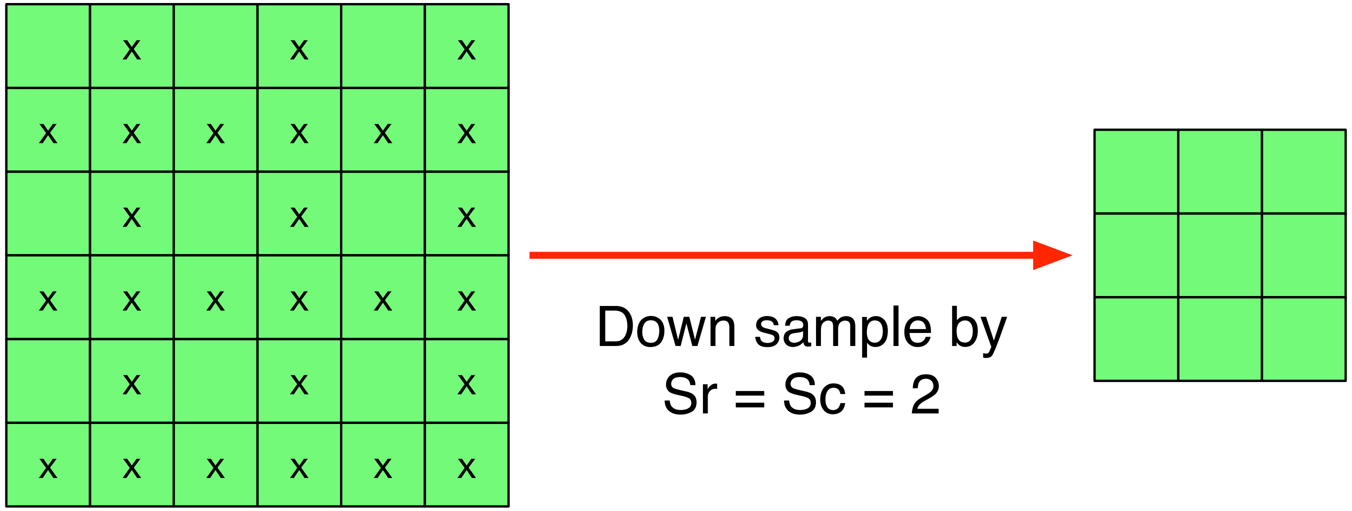


Up sample by Dr = Dc = 2

	0		0		0
0	0	0	0	0	0
	0		0		0
0	0	0	0	0	0
	0		0		0
0	0	0	0	0	0

Down Sampling Output Feature Maps

- Variants: output feature map down sampling
  - S_r rows, S_c cols
  - Typically called strided convolution
  - Used to reduce spatial resolution
  - Implementation key is input feature map filtering matrix column removal
  - Alternative is pooling



	x		x		x
x	x	x	x	x	x
	x		x		x
x	x	x	x	x	x
	x		x		x
x	x	x	x	x	x

Down sample by Sr = Sc = 2

Intuition

- Intuition of feature extraction
    - CNN style 2D convolution is a linear transformation
    - Output feature maps matrix = filter coefficient matrix * input feature maps filtering matrix
    - Matrix vector multiplication as used in a fully connected layer of a neural network had the intuition of matching features to inputs over channel
    - For CNN style 2D convolution it has the intuition of matching features to inputs over channel and space
        - How far can it see in space? For 1 layer? For repeated layers?
        - How many features does it work over?

Intuition

- Intuition of bias
  - Add a constant to all elements in an output feature map
  - Can be a different constant for each output feature map
  - Affine transformation
  - Allows the dividing line to shift
  - Implementation using a rank 1 outer product

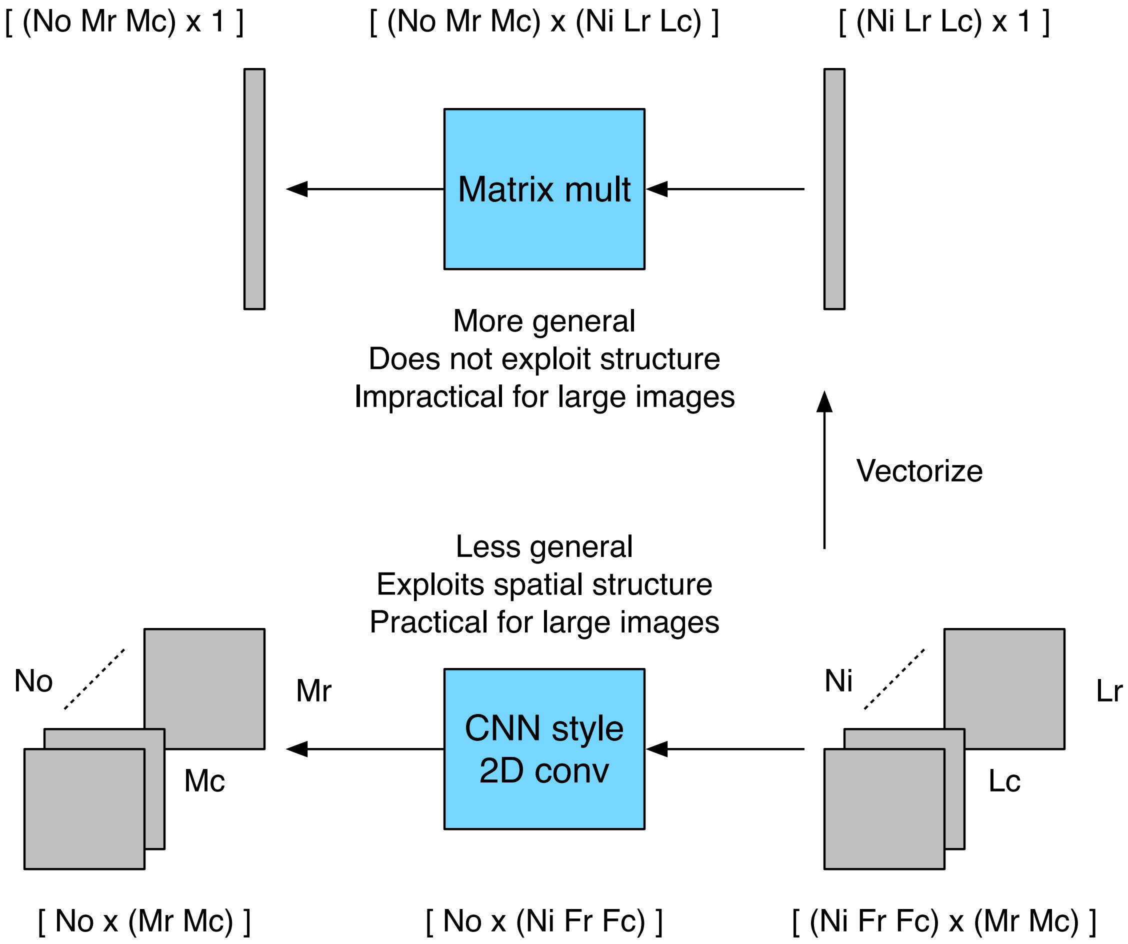
- Intuition of ReLU
  - Removes negatively aligned features or predictions
  - Allows depth
  - Subsequent layers combine positively aligned extracted features

Why Use A CNN Layer Vs a NN Layer?

- Why use CNN style 2D convolution instead of vectorizing the input?

- Consider applying a standard neural network to an image
  - Dimensions / memory / arithmetic intensity make it unreasonable to apply normal neural network linear layer to large images

- Use CNN style 2D convolution layer instead
  - It's a less general transformations but if the input / problem has translational invariance then perhaps the loss of generality is ok
  - Very high (but not unreasonable) memory and compute for modern hardware



Layers – Self Attention

Layer Definition

- Multi headed self attention layer
    - Intra feature vector mixing via value $\mathbf{V}_i$ and output $\mathbf{O}_i$ matrices
    - Inter feature vector mixing via the input dependent softmax() term which can be though of as the self attention matrix $\mathbf{A}_i^T$
    - Note that trained parameter matrices query $\mathbf{Q}_i$, key $\mathbf{K}_i$, value $\mathbf{V}_i$, output $\mathbf{O}_i$ and vector bias $\mathbf{b}_i$ are independent of the sequence length N
    - $Z^T = \text{SelfAttention} (X^T)$

applied per row (N x dm)(dm x dk)(dk x dm)(dm x N) (N x dm)(dm x dv)(dv x dm) (N x 1)(1 x dm)

$$= \sum_{i=0}^{\text{heads-1}} \text{softmax} (X^T \mathbf{Q}_i \mathbf{K}_i^T X / \text{scale}) X^T \mathbf{V}_i \mathbf{O}_i^T + 1\,\mathbf{b}$$

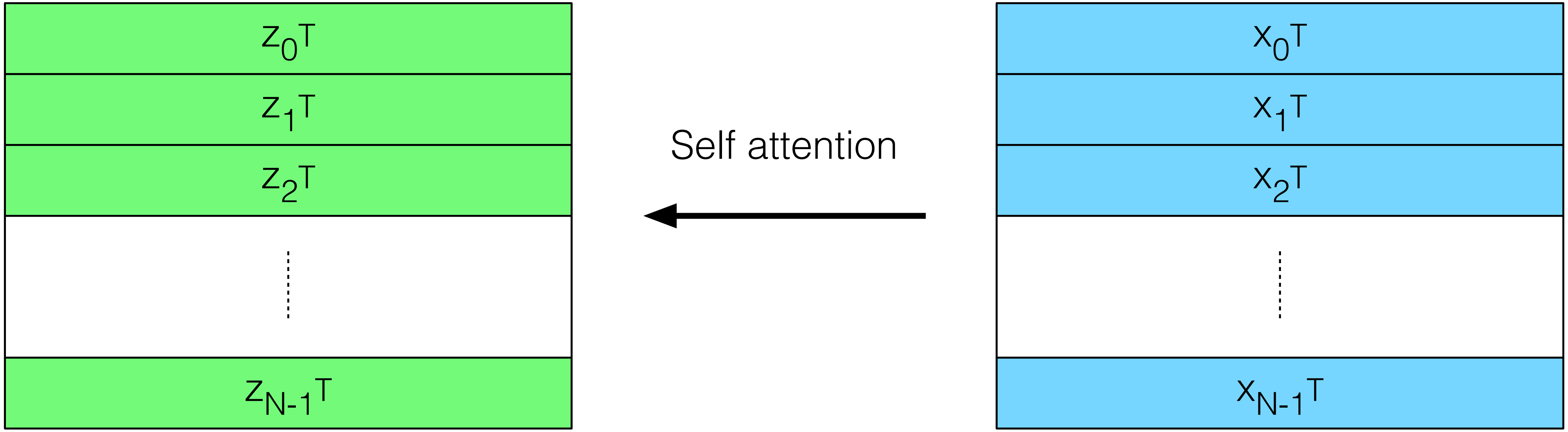
$$= \sum_{i=0}^{\text{heads-1}} \mathbf{A}_i^T X^T \mathbf{V}_i \mathbf{O}_i^T + 1\,\mathbf{b}$$

For now think of softmax() as a function that maps input vectors to a probability mass function (a vector whose elements are non negative and sum to 1); when applied to a matrix the function $\text{softmax}_{\text{row}}()$ is applied separately per row

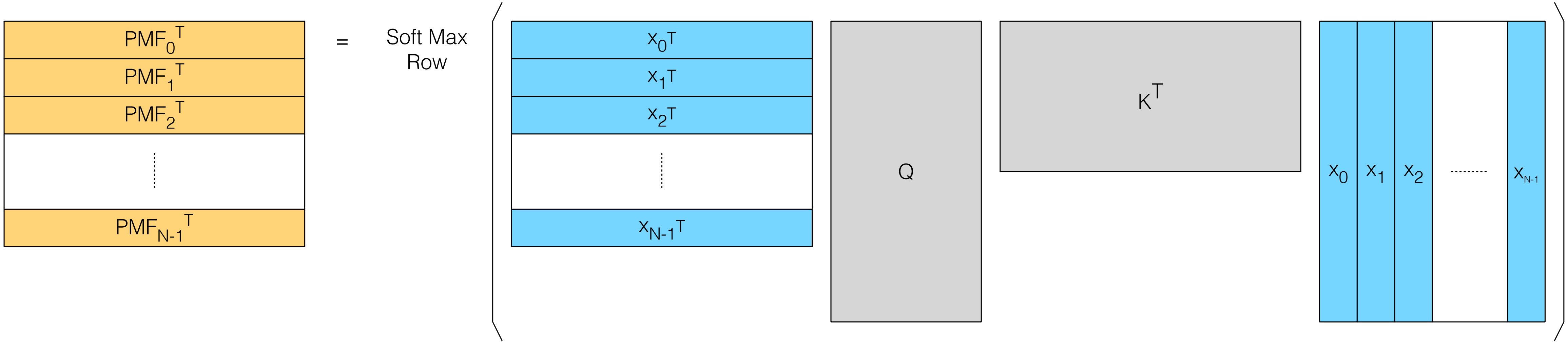
Computation notes

- The complexity is quadratic in the sequence length (N x N matrix multiplies a N x d_m matrix for a total of $N^2 d_m$ MACs)
- Modified variants have complexity that's linear in the sequene length (good) but typically at the expense of something like sequence length dependent params (bad)
- If d_k isn't < 1/2 d_m then it seems appropriate to replace $Q_i K_i^T$ with a single d_m x d_m weight matrix; choosing d_k << 1/2 d_m implies creating a low dimension projection to determine the attention distribution
- The same comments apply to d_v

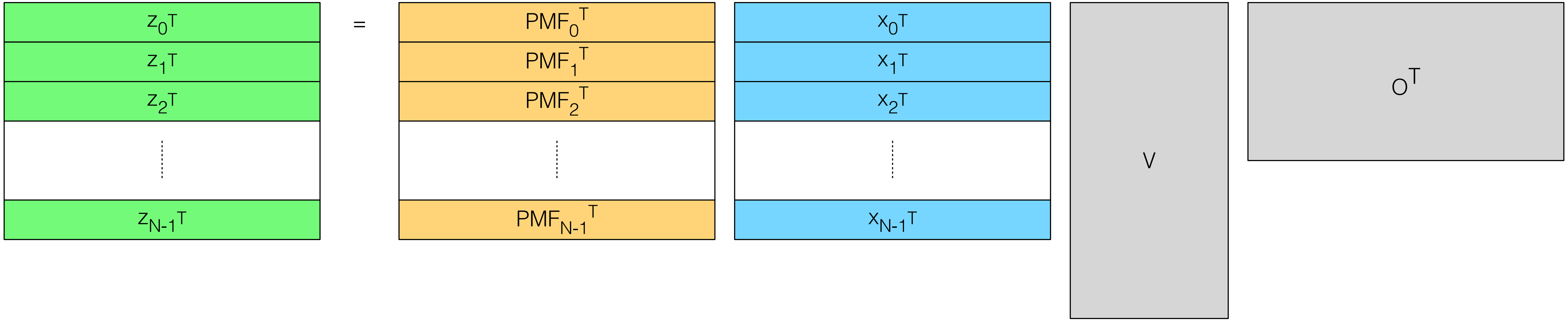
Map Input Vectors To Output Vectors



1: Create A PMF For Mixing Across Vectors



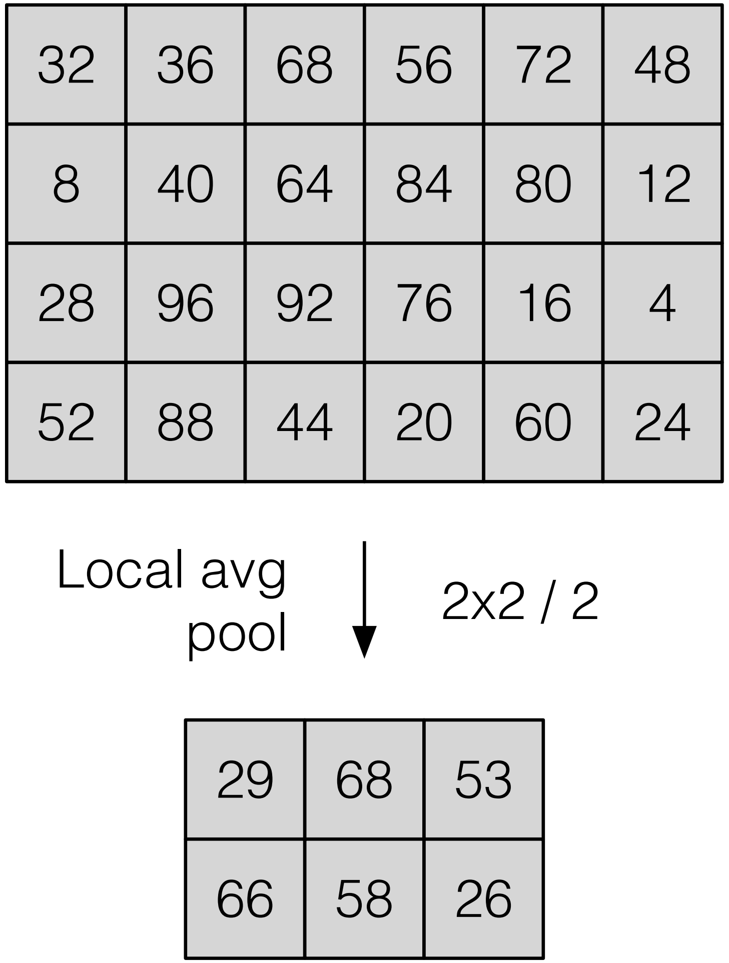
2: Mix Across And Within Feature Vectors



Layers – Average Pooling

Layer Definition

- Average pooling maps a region of an input feature map to a single value that's the average of the elements in that region
  - Local average pooling repeats this process across the whole input feature map in a periodic pattern
    - Implicitly a down sampling mechanism
    - Occasionally used between convolutional building blocks (though max pooling is more common)
  - Global average pooling averages a whole input feature map to a single value
    - Frequently used between convolutional layers and a final densely connected layer in a classification network
    - Aggregates all spatial information, loses all spatial resolution



32	36	68	56	72	48
8	40	64	84	80	12
28	96	92	76	16	4
52	88	44	20	60	24

Local avg pool

2x2 / 2

29	68	53
66	58	26

Layers – Recap

Think About What Mixes Together

- To return to the “Ask yourself” questions in the 1st slide of this section: think how the layers we’ve seen map from data to weak features to strong features to classes in an xNN

- Dense layer
    - Maps an input feature vector to an output feature vector
    - Combines information across the whole input
    - Universal approximator when stacked (will prove in the Analysis slides)
    - Practically limited in the size of the input that they can be applied to because the weight memory scales proportional to N_i*N_o

- RNN layer
    - Maps an input feature vector and an input state vector to an output feature vector and an output state vector
        - The output and state can be the same or different
    - Combines information sequentially
    - Computationally universal as a consequence of including a memory state (will mention but not prove in Algorithms slides)
    - Practically limited in the size of the input that they can be applied to because the weight memory scales proportional to N_i*N_o

Think About What Mixes Together

- CNN layer
  - Maps input feature maps (3D tensors) to output feature maps (3D tensors)
  - Combines information locally spatially across rows and columns and depth wise across channels
  - Practical to use with features stored in 3D tensors as the weight memory is independent of the tensor rows and cols
    - Assuming that the product of N_i and N_o is reasonable (which it typically is)

- Self attention layer
  - Maps an input feature matrix to an output feature matrix
  - Combines information across the full input feature matrix in an input dependent manner

Backup

Function

- Mapping f: X → Y from domain to co domain
  - Injective: one to one; each y produced by at most one x
  - Surjective: onto; each y produced by at least one x
  - Bijective: one to one and onto (invertible)

- An infinite set is
  - Countably infinite if there's a bijection between the natural numbers and elements of the set
  - Un countably infinite if there's not

Vector Space

- Span
  - The span of a set of vectors $\{\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}\}$ is the set of all finite linear combinations of the vectors
  - Vectors $\mathbf{x}$ in the span can be written as $\mathbf{x} = a_0\, \mathbf{x}_0 + \ldots + a_{N-1}\, \mathbf{x}_{N-1}$
  - The span of a set of vectors is a vector space

- Linear independence
  - A set of vectors is linearly dependent if at least 1 vector in the set is a linear combination of the others
  - A set of vectors is linearly independent if no vector in the set can be written as a linear combination of the others

Vector Space

- Basis
  - A basis for a vector space V is any linearly independent set of vectors that span V

- Dimension
  - The dimension of a vector space V is the number of vectors required to form a basis of V
  - Only finite dimensional vector spaces are considered here (with apologies to Prof Hilbert)

- Rank
  - The rank of a matrix X is the dimension of the vector space generated by the span of the column vectors forming the matrix
  - It is the same as the dimension of the space spanned by the rows of X

Normed Vector Space

- The matrix norm induced by the l_p vector norm for a M x K matrix **H** is
  - $||\mathbf{H}||_p = \sup_{\mathbf{x} \neq \mathbf{0}} ||\mathbf{H}\,\mathbf{x}||_p \,/\, ||\mathbf{x}||_p$
  - The l_1 induced matrix norm is the maximum absolute column sum of **H**
  - The l_2 induced matrix norm is the largest singular value of **H**
  - The l_∞ induced matrix norm is the maximum absolute row sum of **H**

- The matrix norm expressed as a vector norm applied first across columns then to the resulting vector is
  - $||\mathbf{H}||_{p,q} = (\Sigma_k (\Sigma_m |H(m, k)|^{p})^{q/p})^{1/q}$, $1 \leq p, q \leq \infty$
  - If p = q = 1 then the matrix norm is the absolute value of all matrix entries
  - If p = q = 2 then the matrix norm is the square root of the sum of the squares of all matrix entries and referred to as the Frobenius norm
  - If p = q = ∞ then the matrix norm is the maximum of the absolute value of all matrix entries

Inner Product Space

- Matrix inner product is the Frobenius inner product
  - $\langle H, G\rangle_F = \Sigma_m \Sigma_k \operatorname{conj}(H(m, k))\, G(m, k)$
  - If the matrices were flattened by stacking the rows or columns end to end then the Frobenius inner product would be equivalent to the vector dot product

Addition

- Definition
  - **C** = **A** + **B** where C(m, k) = A(m, k) + B(m, k)

Multiplication – Matrix Scalar

- Definition
  - **C** = a **B** where C(m, k) = a B(m, k)

Inversion

- Square
  - A K x K square matrix **A** has an inverse matrix **B** = $\mathbf{A}^{-1}$ when the column vectors comprising **A** are linearly independent
  - $\mathbf{A}\ \mathbf{B} = \mathbf{B}\ \mathbf{A} = \mathbf{I}_K$
  - Properties
    - $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$
    - $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
    - $(\mathbf{A}\ \mathbf{B})^{-1} = \mathbf{B}^{-1}\ \mathbf{A}^{-1}$
  - Diagonal
    - Invertible if diagonal entries are non zero
    - B(k, k) = 1/A(k, k)

Inversion

- Non square
  - A M x K matrix $\mathbf{A}$
  - When the rank of $\mathbf{A}$ is M then $\mathbf{A}$ has a right inverse $\mathbf{B}$ such that $\mathbf{A}\ \mathbf{B} = \mathbf{I}_M$
  - When the rank of $\mathbf{A}$ is K then $\mathbf{A}$ has a left inverse $\mathbf{B}$ such that $\mathbf{B}\ \mathbf{A} = \mathbf{I}_K$

- Unitary
  - A K x K unitary matrix $\mathbf{U}$ has orthogonal unit norm columns
  - $\mathbf{U}^H\ \mathbf{U} = \mathbf{U}\ \mathbf{U}^H = \mathbf{I}_K$
  - Unitary matrices preserve inner products $\langle \mathbf{U}\ \mathbf{x}, \mathbf{U}\ \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle$

- Orthogonal
  - A K x K orthogonal matrix $\mathbf{Q}$ has orthogonal unit norm columns with only real valued elements
  - $\mathbf{Q}^T\ \mathbf{Q} = \mathbf{Q}\ \mathbf{Q}^T = \mathbf{I}_K$
  - Orthogonal matrices preserve inner products $\langle \mathbf{Q}\ \mathbf{x}, \mathbf{Q}\ \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle$

Kronecker Product

- Definition
  - $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ where $\mathbf{C}(m, k) = A(m, k)\,\mathbf{B}$
- Comments
  - Generalizes vector outer products to matrix outer products
  - Not commutative in general

Vectorization

- Definition
  - $\mathbf{y} = \mathrm{vec}(\mathbf{A})$ where $\mathbf{y}$ is formed from stacking columns of $\mathbf{A}$

- Identities
  - $\mathrm{vec}(\mathbf{A}\ \mathbf{B}\ \mathbf{C}) = (\mathbf{C}^T \otimes \mathbf{A})\ \mathrm{vec}(\mathbf{B})$

    $= (\mathbf{I}_N \otimes \mathbf{A}\ \mathbf{B})\ \mathrm{vec}(\mathbf{C})$

    $= (\mathbf{B}^T\ \mathbf{C}^T \otimes \mathbf{I}_K)\ \mathrm{vec}(\mathbf{A})$

Trace

- Definition
    - The trace of a K x K matrix **A** is the sum of the elements on the principal diagonal
- Comments
    - The trace is also equal to the sum of the eigenvalues of **A**

Determinant

- Definition
  - The determinant of a K x K matrix **A** is the product of the matrix eigenvalues

- Comments
  - Can be thought of as the volume of a polytope defined by the column vectors of **A**

Eigen Decomposition

- An eigenvector **v** of a K x K matrix **A** is a nonzero vector that satisfies **A v** = λ **v**
  - λ is a scalar referred to as the associated eigenvalue
  - Matrix **A** scales the eigenvector **v** but does not change its direction

- If **A** has K linearly independent eigenvectors then **A** can be factored as $\mathbf{A} = \mathbf{Q}\,\mathbf{D}\,\mathbf{Q}^{-1}$
  - **Q** is a matrix with eigenvectors as columns (**Q** is an orthogonal matrix if **A** is real symmetric)
  - **D** is a diagonal matrix with associated eigenvalues as diagonal elements
  - The eigen decomposition is frequently calculated via a power method and deflation

- Given an eigen decomposition of **A** it's straightforward to find $\mathbf{A}^{-1}$ and $\mathbf{A}^{n}$
  - $\mathbf{A}^{-1} = (\mathbf{Q}\,\mathbf{D}\,\mathbf{Q}^{-1})^{-1}$, $= \mathbf{Q}\,\mathbf{D}^{-1}\,\mathbf{Q}^{-1}$, this exploits the inversion formula for diagonal matrices and products of matrices
  - $\mathbf{A}^{n} = (\mathbf{Q}\,\mathbf{D}\,\mathbf{Q}^{-1})^{n} = \mathbf{Q}\,\mathbf{D}^{n}\,\mathbf{Q}^{-1}$

Singular Value Decomposition

- The SVD of a M x K matrix **A** is the weighted outer product **A** = **U S V**^{H}
  - **U** is a M x M unitary matrix (orthogonal if **A** is real)
  - **S** is a M x K diagonal matrix is singular values
  - **V**^{H} is a K x K unitary matrix (orthogonal if **A** is real)

- The columns of **U** are the eigenvectors of **A A**^{H} = **U S V**^{H} **V S**^{H} **U**^{H} = **U S S**^{H} **U**^{H}
  - Let **Q** = **U**, **D** = **S S**^{H} and **Q**$^{-1}$ = **U**^{H} in the eigen decomposition
  - Initial columns corresponding to nonzero singular values span the column space of **A**
  - Last columns corresponding to zero singular values span the left null space of **A**

- The number of nonzero singular values is the rank of **A** and the ratio of the largest to smallest singular value is the condition number of **A**

- The columns of **V**^{H} are the eigenvectors of **A**^{H} **A** = **V S**^{H} **U**^{H} **U S V**^{H} = **V S**^{H} **S V**^{H}
  - Let **Q** = **V**, **D** = **S**^{H} **S** and **Q**$^{-1}$ = **V**^{H} in the eigen decomposition
  - Initial columns corresponding to nonzero singular values span the row space of **A**
  - Last columns corresponding to zero singular values span the null space of **A**

4 Fundamental Subspaces

- 4 fundamental subspaces
  - The column space, image or range of T is the vector subspace of V_2 comprising all vectors T can produce and is denoted by range(T) = $\{T(\mathbf{x}) \in V_2 : \mathbf{x} \in V_1\}$
  - The null space or right null space of T is the vector subspace of V_1 comprising all vectors T maps to $\mathbf{0}$ and is denoted by null(T) = $\{\mathbf{x} \in V_1 : T(\mathbf{x}) = \mathbf{0}\}$
  - The row space or co image of T is the vector subspace of V_1 comprising all vectors T^T can produce and is denoted by range(T^T) = $\{T^T(\mathbf{y}) \in V_1 : \mathbf{y} \in V_2\}$
  - The left null space or co kernel of T is the vector subspace of V_2 comprising all vectors T^T maps to $\mathbf{0}$ and is denoted by null(T^T) = $\{\mathbf{y} \in V_2 : T^T(\mathbf{y}) = \mathbf{0}\}$

- Consider the linear transformation between finite dimensional vector spaces $\mathbf{y} = \mathbf{A}\,\mathbf{x}$
  - **A** is a M x K matrix representing linear map T, **x** is a length K input and **y** is a length M output
  - The range of **A** is the vector space formed by the span of the column vectors of **A**
  - The number of linearly independent columns of **A** is the rank of **A** and satisfies rank(**A**) ≤ min(M, K)

Principal Component Analysis

- Note
  - Some of this is dependent on probability for parts of the understanding

- Setup
  - M x K data matrix **X**
  - Each row is a different trial (ex: point in time)
  - Each column is a different measurement from that trial (ex: different stock)
  - Columns are normalized to 0 mean
  - Columns are potentially linearly correlated

- Goal
  - Linearly transform to a new M x K matrix **Y** via a K x K matrix **Q** by **Y** = **X Q**
  - **Q** is chosen such that columns of **Y** are orthogonal and ordered from largest to smallest variance
  - For dimensionality reduction keep first L < K columns

Principal Component Analysis

- Mechanics for finding **Q**
  - Decompose **X** via the SVD as **X** = **U S V**^{T}
  - Select **Q** = **V** such that **Y** = **X Q** = **U S V**^{T} **V** = **U S**

- Example
  - Statistical arbitrage (e.g., SPY, MDY and IJR)
  - Stock 0 time series in col 0, stock 1 time series in col 1, …, stock K – 1 time series in col K – 1
  - 0 th principal component for trend trading (you would keep this for feature extract)
  - K – 1 th principal component for stat arb (throw away for feature extract)

Discrete Fourier Transform

- The DFT is a linear transformation from domain to 1/domain via a projection onto a complex exponential basis: $y(k) = (1/\mathrm{sqrt}(K))\ \Sigma_n\ x(n)\ e^{-i(2\pi/K)nk}$
  - k = 0, ..., K – 1 and n = 0, ..., K – 1
  - Example domains: time to 1/time = frequency

- Equivalent to a K x K DFT matrix $\mathbf{F}_K$ that transforms input vectors **x** to output vectors **y**
  - $\mathbf{y} = \mathbf{F}_K\ \mathbf{x}$ where $F_K(a, b) = (1/\mathrm{sqrt}(K))\ e^{-i(2\pi/K)ab}$
  - $\mathbf{F}_K$ is a unitary matrix so it's invertible (conj transpose = inverse, called the IDFT)
  - Output is typically circular complex Gaussian (will discuss implications later)
  - Efficient implementations are possible
    - O(K log K) for fast Fourier transform (FFT)
    - Vs $O(K^2)$ for DFT

Discrete Fourier Transform

- Use: data transformation
  - Sometimes it’s easier to do feature extraction in the frequency domain vs time domain
    - Common example of this is speech to text
    - DFTs are used for creating MFCCs
  - Unitary so invertible (no information lost (until you read the next sub bullet))
    - Effectively lets the network decide what data to keep and what data to throw away

- Use: dimensionality reduction
  - The DFT frequently concentrates the majority of information in naturally occurring signals to $L < K$ basis components
  - A common dimensionality reduction strategy is to keep the L main components and get rid of the rest

CNN Style 2D Convolution

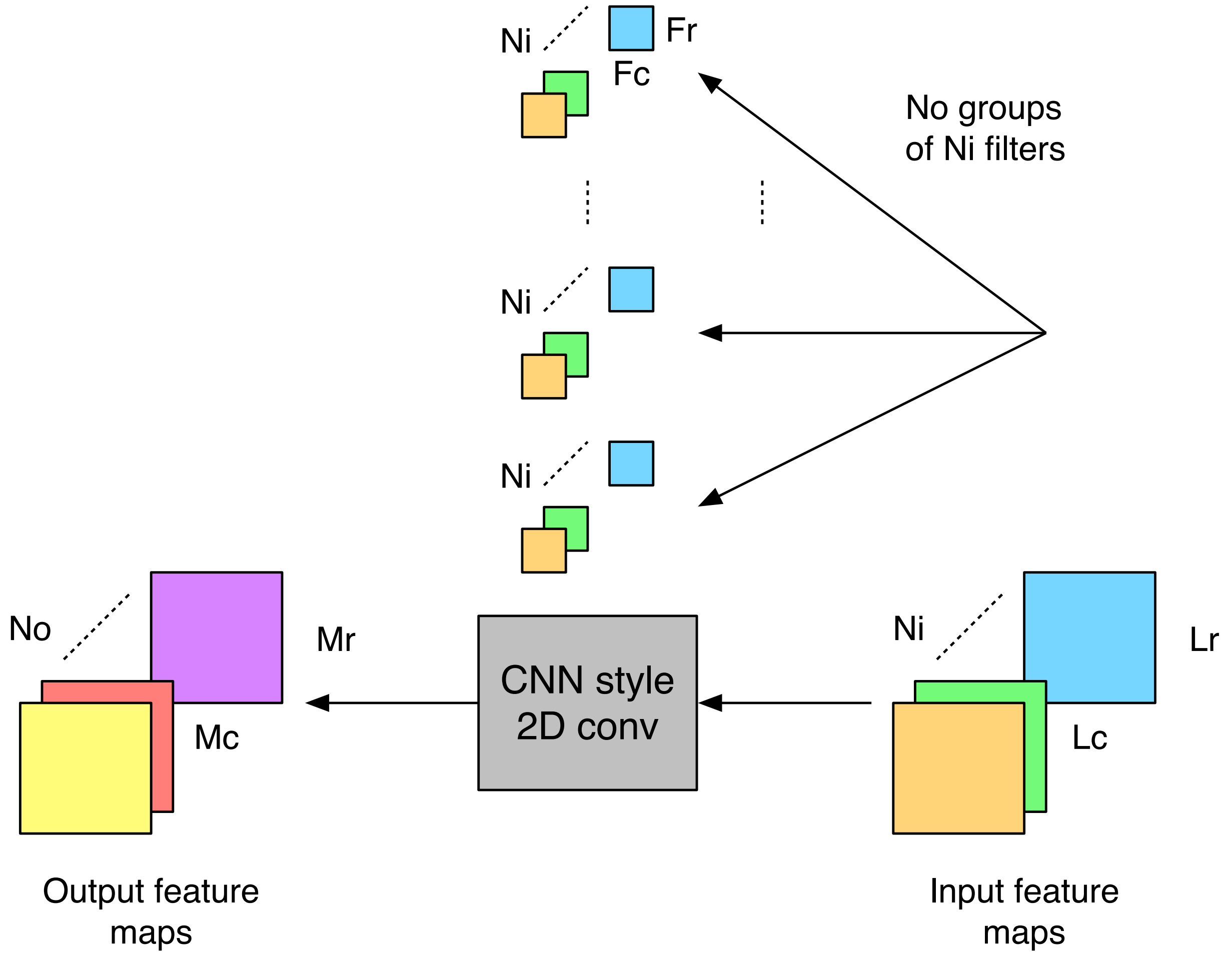
- Common types of filtering / convolution
  - 1D
  - 2D
  - CNN style 2D

- Common methods for speeding up filtering / convolution for various cases
  - Frequency domain
  - Winograd

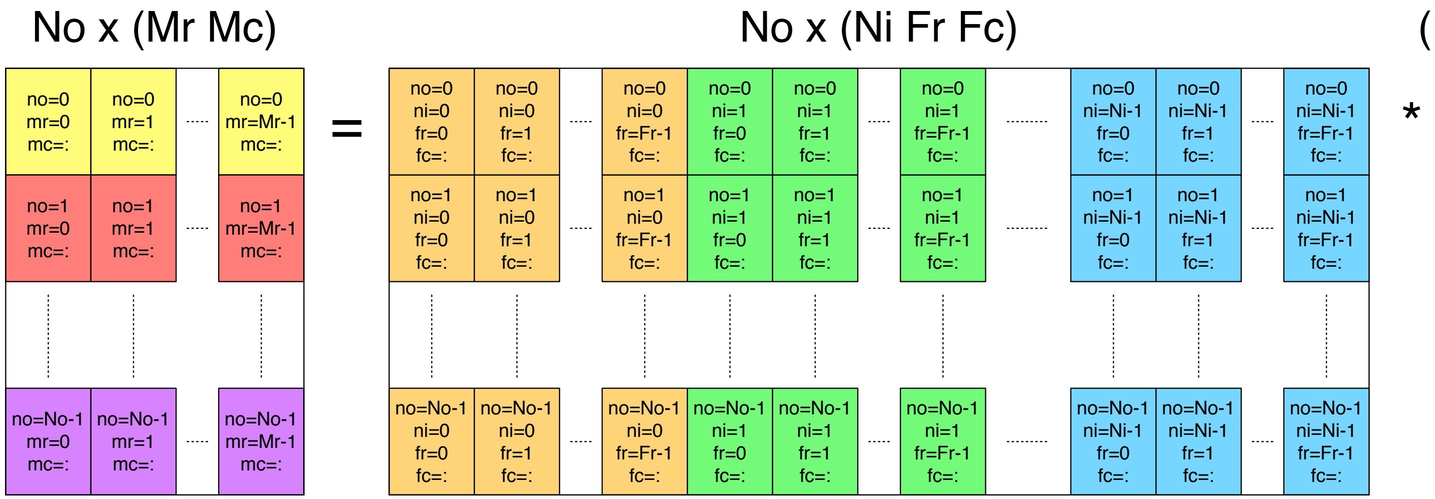
- This set of slides will only consider CNN style 2D convolution in the time domain
  - 1D and 2D convolution can be viewed as special cases
  - Tensor to matrix lowering for computation is also included

Lowering To Matrix Matrix Multiplication

- Starting point / reminder
- Input feature maps
  - 3D tensor
  - N_i inputs x L_r rows x L_c cols
- Filter coefficients
  - 4D tensor
  - N_o outputs x N_i inputs x F_r rows x F_c cols
- Output feature maps
  - 3D tensor
  - N_o outputs x M_r rows x M_c cols



Lowering To Matrix Matrix Multiplication



No x (Mr Mc)

no=0
mr=0
mc=:
	no=0
	mr=1
	mc=:
		......	no=0
			mr=Mr-1
			mc=:

no=1
mr=0
mc=:
	no=1
	mr=1
	mc=:
		......	no=1
			mr=Mr-1
			mc=:

⋮	⋮		⋮

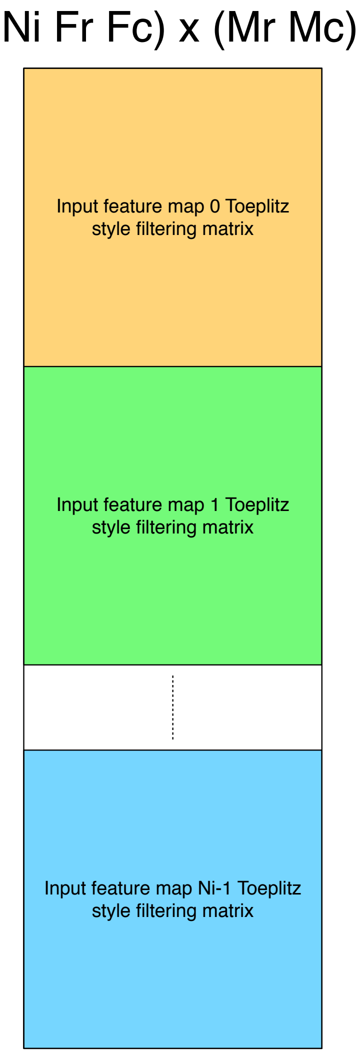
no=No-1
mr=0
mc=:
	no=No-1
	mr=1
	mc=:
		......	no=No-1
			mr=Mr-1
			mc=:

=

No x (Ni Fr Fc)

no=0 ni=0 fr=0 fc=:	no=0 ni=0 fr=1 fc=:	......	no=0 ni=0 fr=Fr-1 fc=:	no=0 ni=1 fr=0 fc=:	no=0 ni=1 fr=1 fc=:	......	no=0 ni=1 fr=Fr-1 fc=:	....	no=0 ni=Ni-1 fr=0 fc=:	no=0 ni=Ni-1 fr=1 fc=:	......	no=0 ni=Ni-1 fr=Fr-1 fc=:
no=1 ni=0 fr=0 fc=:	no=1 ni=0 fr=1 fc=:	......	no=1 ni=0 fr=Fr-1 fc=:	no=1 ni=1 fr=0 fc=:	no=1 ni=1 fr=1 fc=:	......	no=1 ni=1 fr=Fr-1 fc=:	....	no=1 ni=Ni-1 fr=0 fc=:	no=1 ni=Ni-1 fr=1 fc=:	......	no=1 ni=Ni-1 fr=Fr-1 fc=:
⋮	⋮		⋮	⋮	⋮		⋮		⋮	⋮		⋮
no=No-1 ni=0 fr=0 fc=:	no=No-1 ni=0 fr=1 fc=:	......	no=No-1 ni=0 fr=Fr-1 fc=:	no=No-1 ni=1 fr=0 fc=:	no=No-1 ni=1 fr=1 fc=:	......	no=No-1 ni=1 fr=Fr-1 fc=:	....	no=No-1 ni=Ni-1 fr=0 fc=:	no=No-1 ni=Ni-1 fr=1 fc=:	......	no=No-1 ni=Ni-1 fr=Fr-1 fc=:

*



(Ni Fr Fc) x (Mr Mc)

Input feature map 0 Toeplitz style filtering matrix
Input feature map 1 Toeplitz style filtering matrix
⋮
Input feature map Ni-1 Toeplitz style filtering matrix

- CNN style 2D convolution written as matrix matrix multiplication
  - Output feature maps (each box is 1 x M_c elements)
  - Filter coefficients (each box is 1 x F_c elements)
  - Input feature maps (ordering not shown)

Layer Definition

- Input X^T is a M x K matrix composed of M input vectors with K features per vector
- Compute query, key and value matrices (H's are weights)
  - Query (Q): $X_{q,i}^T = X^T H_{q,i}$, (M x P) = (M x K) (K x P)
  - Key (K): $X_{k,i}^T = X^T H_{k,i}$, (M x P) = (M x K) (K x P)
  - Value (V): $X_{v,i}^T = X^T H_{v,i}$, (M x L) = (M x K) (K x L)
- Single headed output Y_i^T is a M x L matrix composed of M output vectors with L features per vector
  - Y_i^T $= \text{softmax}_{\text{row}}(X_{q,i}^T X_{k,i} / P^{1/2}) X_{v,i}^T$
    $= \text{softmax}_{\text{row}}(X^T H_{q,i} H_{k,i}^T X / P^{1/2}) X^T H_{v,i}$
    $= A_i^T X^T H_{v,i}$

For now think of softmax() as a function that maps input vectors to a probability mass function (a vector whose elements are non negative and sum to 1); when applied to a matrix the function $\text{softmax}_{\text{row}}()$ is applied separately per row

Single headed self attention layer

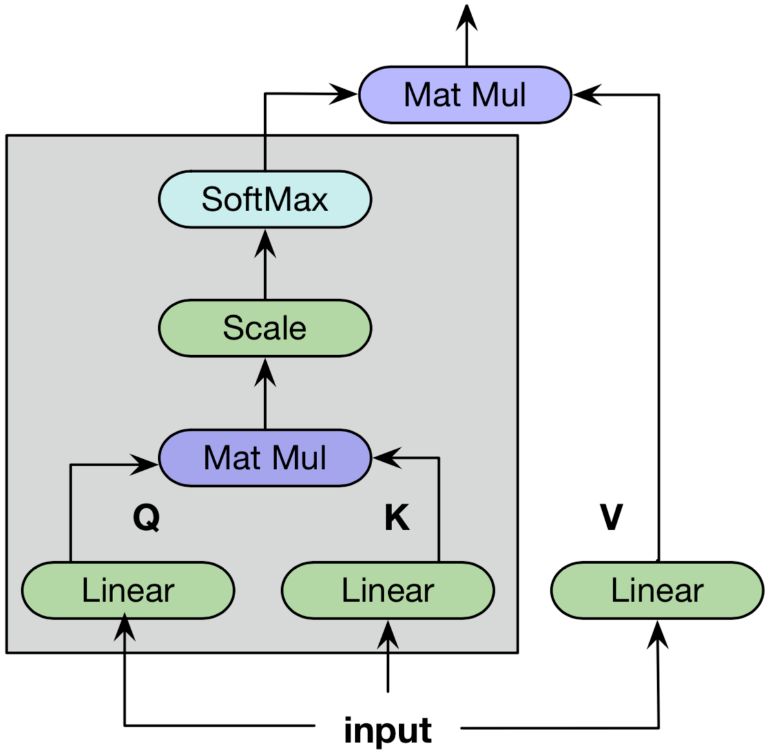


Figure from https://arxiv.org/abs/1901.10430

Intuition

- Single headed self attention notes
  - Think of $\mathbf{A}_i^T$ as 1 pmf per row
  - $\mathbf{A}_i^T$ $\mathbf{X}^T$ will effectively be M different pmf weighted mixings of the input vectors
  - If P isn't < 1/2 K then it seems appropriate to replace $\mathbf{H}_{q,i}$ $\mathbf{H}_{k,i}^T$ with a K x K weight matrix $\mathbf{H}_{qk,i}$; choosing P << 1/2 K implies creating a low dimension projection to determine the attention distribution
  - The $\mathbf{H}_{v,i}$ weight matrix allows the mixing of features within a row of $\mathbf{X}^T$ to create new features and / or change the number of features in that row
  - The result is a per head mapping from M input vectors with K features per input vector to M output vectors with L features per output vector

Single headed self attention layer

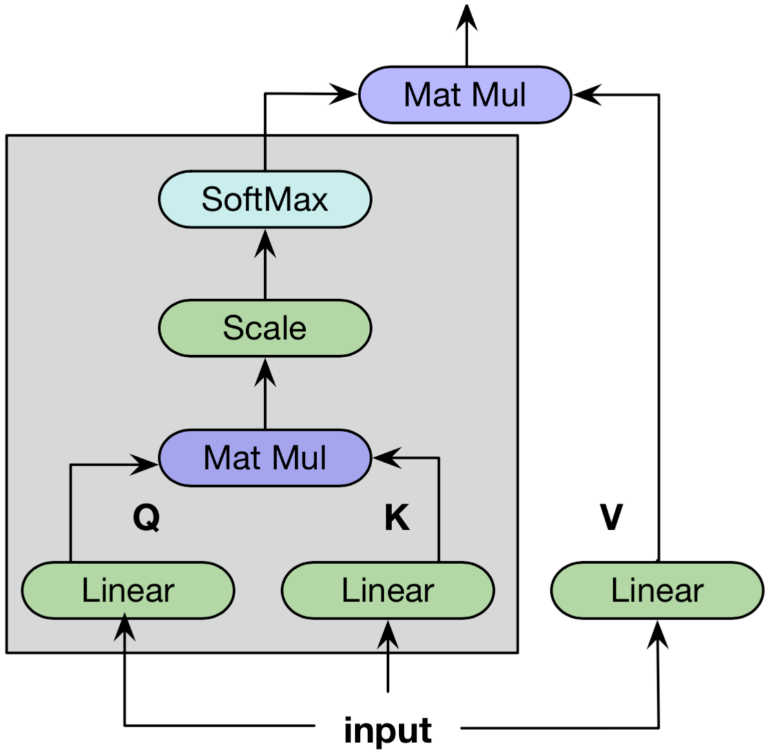


Figure from https://arxiv.org/abs/1901.10430

Multi Headed

- In multi headed self attention, the outputs of multiple single headed self attention transforms fed by the same input are concatenated together and transformed to create the M x N output $\mathbf{Y}^T$ composed of M output vectors with N feature per vector
  - $\mathbf{H}_{o,i}$ is a L x N weight matrix
  - $\mathbf{Y}^T = \Sigma_i \mathbf{Y}_i^T \mathbf{H}_{o,i}$, i = 0, ..., num heads – 1
    $= \Sigma_i \mathbf{A}_i^T \mathbf{X}^T \mathbf{H}_{v,i} \mathbf{H}_{o,i}$, i = 0, ..., num heads – 1

- Multi headed self attention notes
  - If KL + LN < KN then it seems appropriate to combine $\mathbf{H}_{v,i}$ $\mathbf{H}_{o,i}$ into a weight matrix $\mathbf{H}_{vo,i}$ of size K x N
  - The result is an overall mapping from M input vectors with K features per input vector to M output vectors with N features per output vector

Multi headed self attention layer concatenates outputs of single headed self attention and multiplies by an additional weight matrix

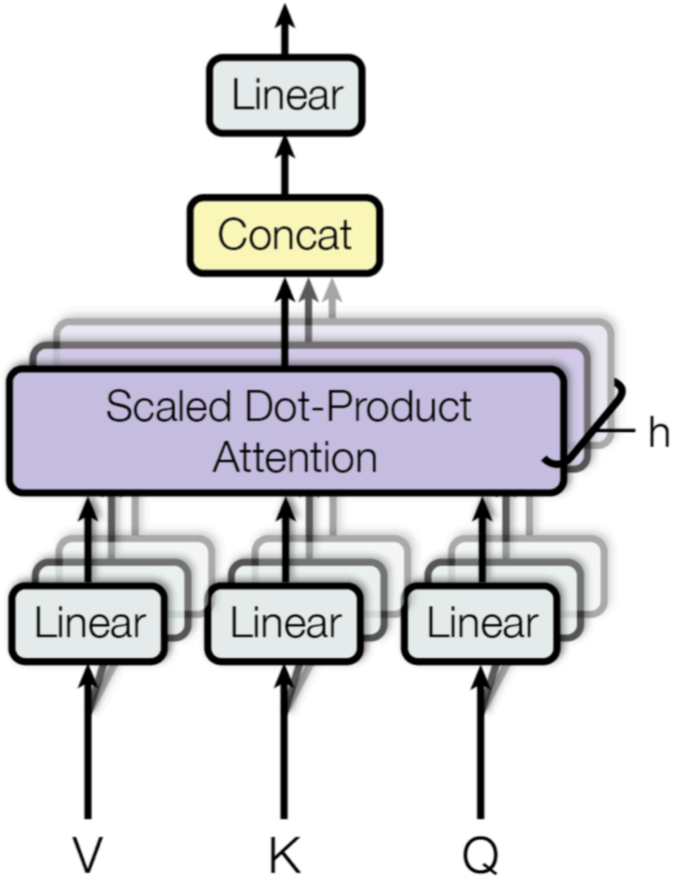


Figure from https://arxiv.org/abs/1706.03762

Layer Definition

- Encoder – attention – decoder layer
  - X^T is a M x K input, $\mathbf{z}_d^T$ is a 1 x P query at position d and $\mathbf{H}_{k,0}^T$ is a P x K trainable weight matrix
  - $\mathbf{a}_{0,d}^T = \text{softmax}(\mathbf{z}_d^T \mathbf{H}_{k,0}^T \mathbf{X} / P^{1/2})$ is a 1 x M attention probability mass function for position d (this is the generalized dot product variant, other possibilities exist)
  - $\mathbf{y}_{0,d}^T = \mathbf{a}_{0,d}^T \mathbf{X}^T$ is a 1 x K attention output; the output row vector is a pmf weighted sum of the input row vectors

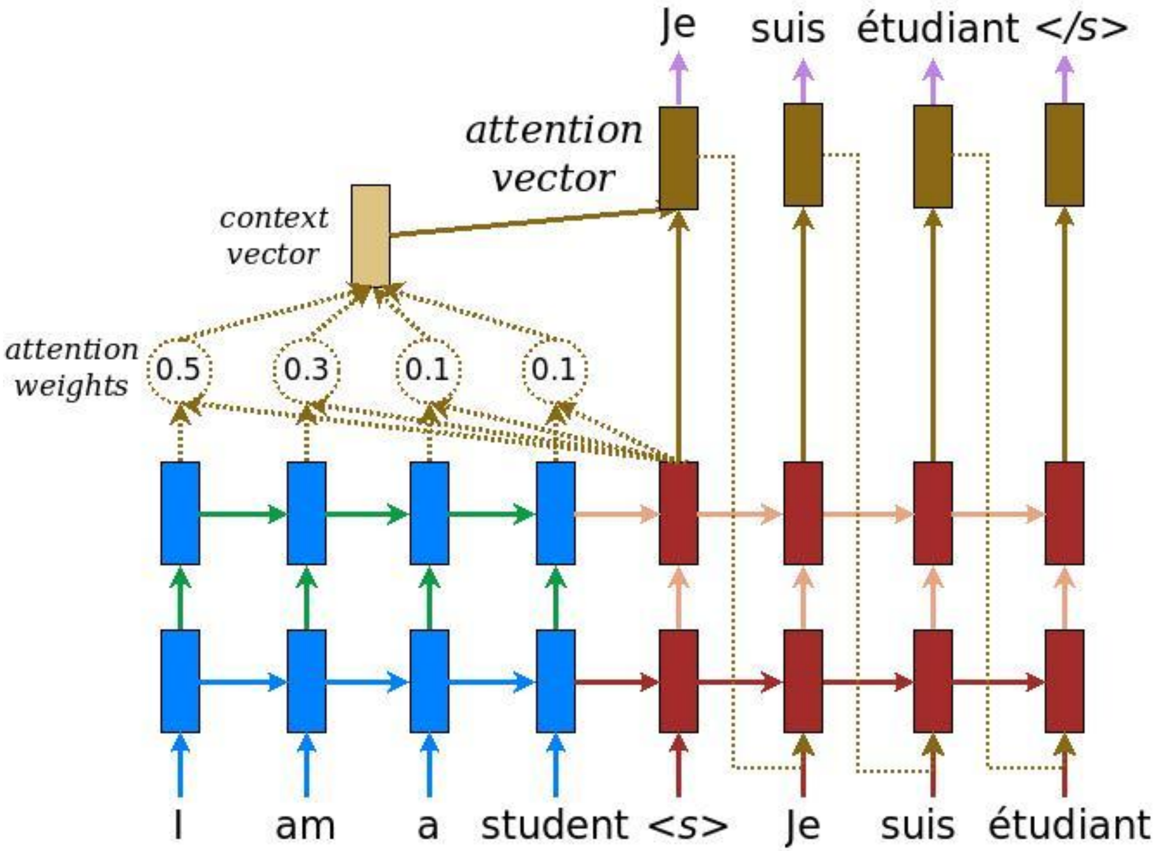


Figure from https://github.com/tensorflow/nmt

Variants

- This is just scratching the surface of attention and there are many possibilities
  - Different scoring functions used to generate attention distributions
  - Different input windows (global / soft, local / hard fixed and adaptive)
  - Different choices of query and binding location for the output

- Aside: RPN and RoI align layers can be thought of as attention mechanisms in object detection and object segmentation networks

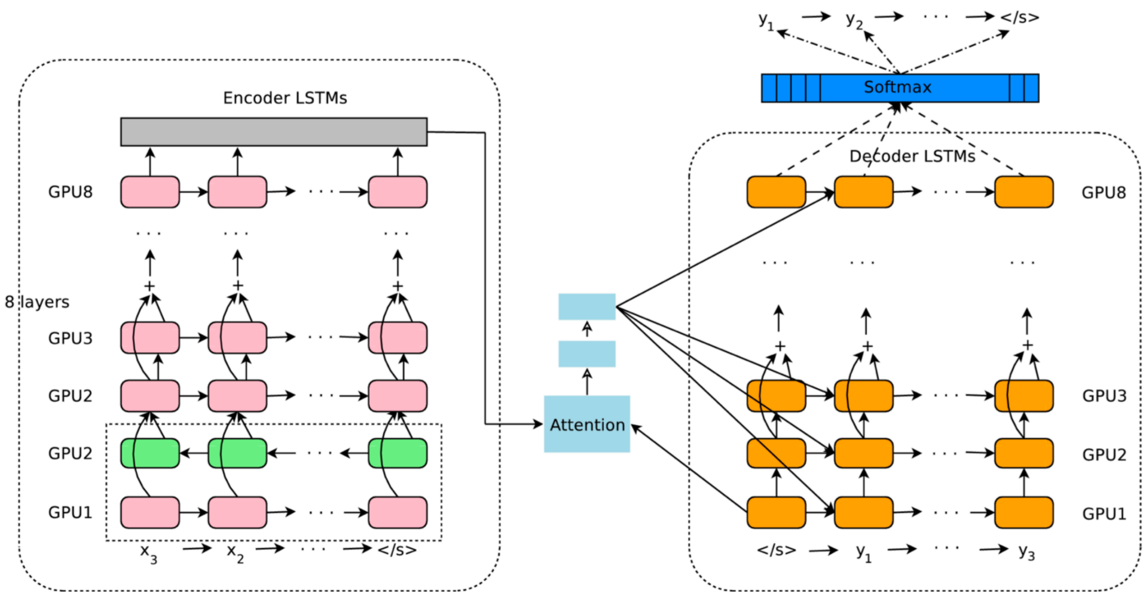


Figure from https://arxiv.org/abs/1609.08144

Intuition

- Attention intuition
  - Depending on what you're interested in you focus on different things
    - Different parts of an image for captioning
    - Different parts of a sound waveform for speech to text transduction
    - Different parts of a sentence for language to language translation
  - Attention provides a way of doing this
  - The output is a weighted combination of inputs based on a query (what you're interested in)

- A note on computation
  - Transposing matrices is bad, so instead compute $\mathbf{a}_{0,d}^T = (\text{softmax}(\mathbf{X}^T \mathbf{H}_{k,0} \mathbf{z}_d / P^{1/2}))^T$
  - $\mathbf{X}^T$ is the format that the input data is already in
  - Directly store and update $\mathbf{H}_{k,0}$ so no transpose is needed there
  - Transposing the result is transposing a vector so that is just bookkeeping

Relation To Self Attention

- Note that you can "derive" encoder – attention – decoder from self attention
  - Replace the M x P input queries with a 1 x P query from the decoder: $\mathbf{X}^T \mathbf{H}_{q,0} \rightarrow \mathbf{z}_d^T$
    - This allows the number of attention outputs D (d = 0, ..., D – 1) to be decoupled from the number of input vectors M, a difference from self attention
  - Simplify
    - Let the number of output features N = the number of input features K
    - Let L = N such that there's no reduced rank projection
    - Let $\mathbf{H}_{v,0}$ $\mathbf{H}_{o,0}$ = $\mathbf{I}_K$ as any additional projection can be handled by the decoder
  - Historically, though, self attention came from / was motivated by attention

- Question
  - Does it make sense to consider a multi headed version of attention (similar to the multi headed version of self attention)? Should the multiple heads be concatenated instead of added?

References

Vectors And Matrices

- Personal communication with T. Lahou
- Linear algebra
    - https://www.math.ucdavis.edu/~linear/linear-guest.pdf
- Linear algebra abridged
    - http://linear.axler.net/LinearAbridged.pdf
- Essence of linear algebra
    - https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab
- The matrix cookbook
    - https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

Layers Built From Linear Transforms

- Christopher Olah's blog
  - http://colah.github.io
- A guide to convolution arithmetic for deep learning
  - https://arxiv.org/abs/1603.07285
- A critical review of recurrent neural networks for sequence learning
  - https://arxiv.org/abs/1506.00019
- Effective approaches to attention-based neural machine translation
  - https://arxiv.org/abs/1508.04025