Advanced NN Architectures

Convolutional Neural Networks

Learning goals

- Understand the convolution operation and layer
- Understand applicatoin of CNNs in image analysis
- Understand applicatoin of CNNs in NLP

NEURAL NETWORK ARCHITECTURES

- An architecture is an abstract design for a neural network
- Loosely speaking: Which nodes connect to which nodes? Which connections share parameters?
- Examples of architectures:
  - Fully Connected Neural Networks
  - Convolutional Neural Networks (CNNs)
  - Recurrent Neural Networks (RNNs)
    - Subclasses: Vanilla RNNs, LSTMs, GRUs, QRNNs, ...
  - Transformers (self-attention)
  - ...
- The choice of architecture is often informed by assumptions about the data, based on our domain knowledge

ASSUMPTION OF LOCALITY

- Computer vision: Pixels that make up a meaningful objects tend to be located in a coherent (“local”) area:

- Not always true for words in NLP:
  - *sie sollen sich heute bei ihm im büro* ***vorstellen***
  - ***stellen*** *sie sich bitte heute bei ihm im büro* ***vor***

ASSUMPTION OF TRANSLATION INVARIANCE

- The features that make up a meaningful object do not depend on that object's absolute position in the input.
- Computer vision:

- NLP:
  - ***[the yellow house]*** *[rest of sentence]* $\rightarrow$ noun phrase
  - *[rest of sentence]* ***[the yellow house]*** $\rightarrow$ still a noun phrase

ASSUMPTION OF SEQUENTIALITY

- NLP: Sentences should be processed left-to-right or right-to-left. This one is falling out of fashion, since people are replacing recurrent neural networks with self-attention networks.

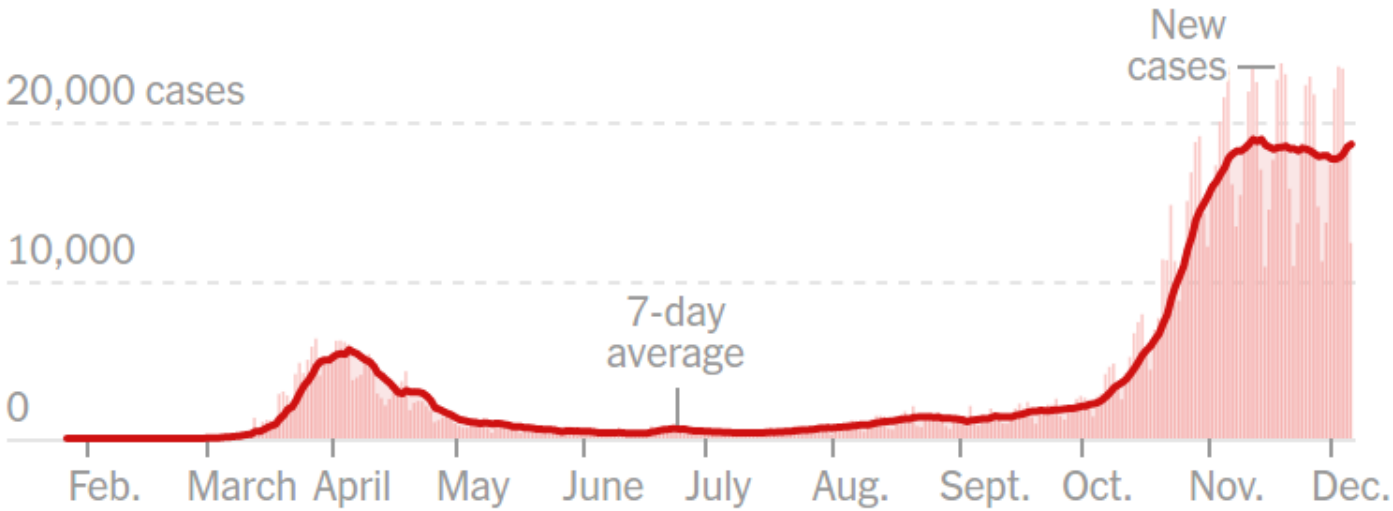
ARE THESE ASSUMPTIONS (ALWAYS) TRUE?

- Of course not!
- But they are a good way of thinking about why certain architectures are popular for certain problems.
- Also, for limiting the search space when deciding on an architecture for a given project.

CONVOLUTIONAL LAYERS

- Technique from Computer Vision (esp. object recognition), by LeCun et al., 1998
- Adapted for NLP (e.g., Kalchbrenner et al., 2014)
- Filter banks with trainable filters
- So what is a filter? What is a filter bank?

EXAMPLE: 7 DAY ROLLING AVERAGE FILTER



`https://www.nytimes.com/`, December 07, 2020

EXAMPLE: 7 DAY ROLLING AVERAGE FILTER

- Let $\mathbf{x} \in \mathbb{R}^J$ be a vector that represents a data series (e.g., number of positive tests per day, over J consecutive days)
- Let $\mathbf{f} \in \mathbb{R}^K; \mathbf{f} = [\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}]^T$ be our "7 day rolling average" filter vector
- Filtering $\mathbf{x}$ with $\mathbf{f}$ (written $\mathbf{f} * \mathbf{x}$) yields a new vector $\mathbf{h}$, with:

$$h_j = \sum_{k=1}^{7} f_k x_{(j+k-7)} = \frac{1}{7}(x_{j-6} + \ldots + x_j)$$

- (For now, let's not worry about the edge cases where $j + k - 7 \leq 0$.)

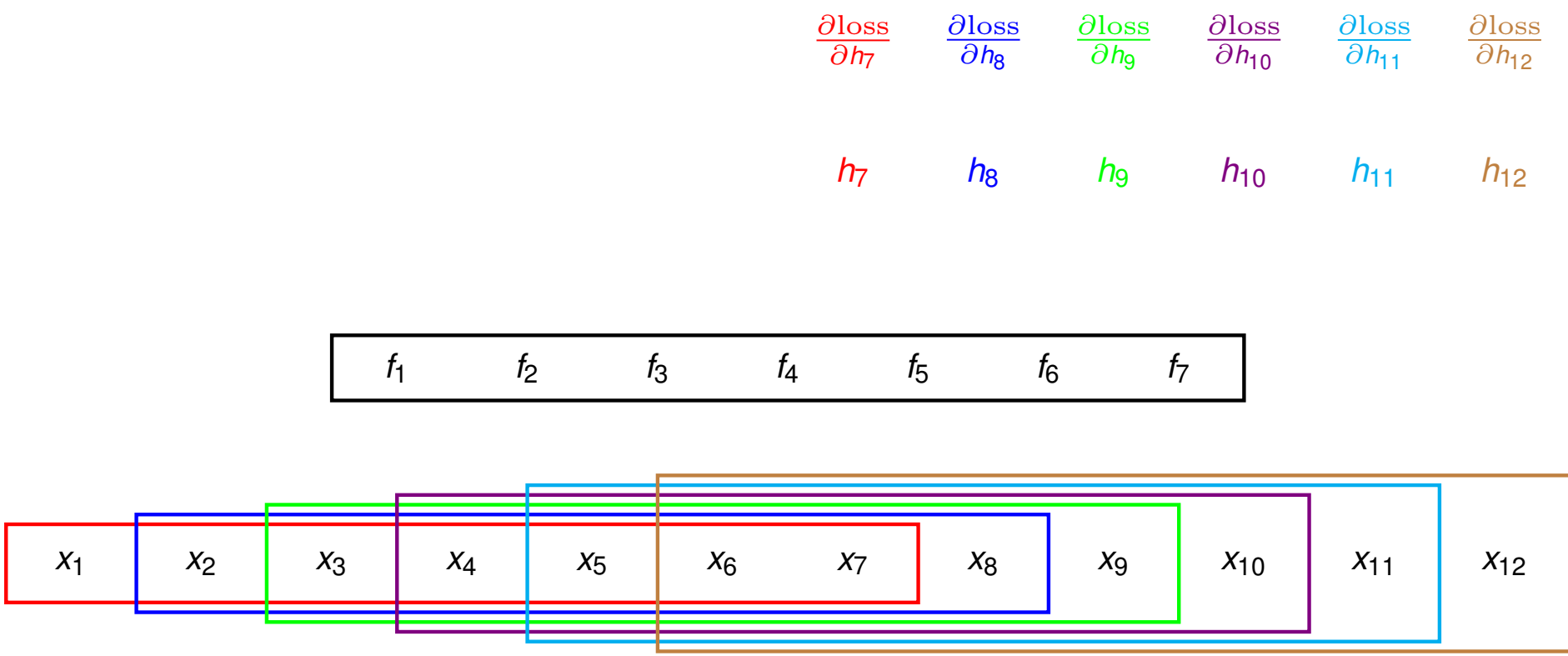
1-D CONVOLUTION

- Let's say that we want to train a linear regression model to predict some variable of interest from each datapoint.
- Since the raw data is noisy, it makes sense to smoothe it with our rolling average filter first:

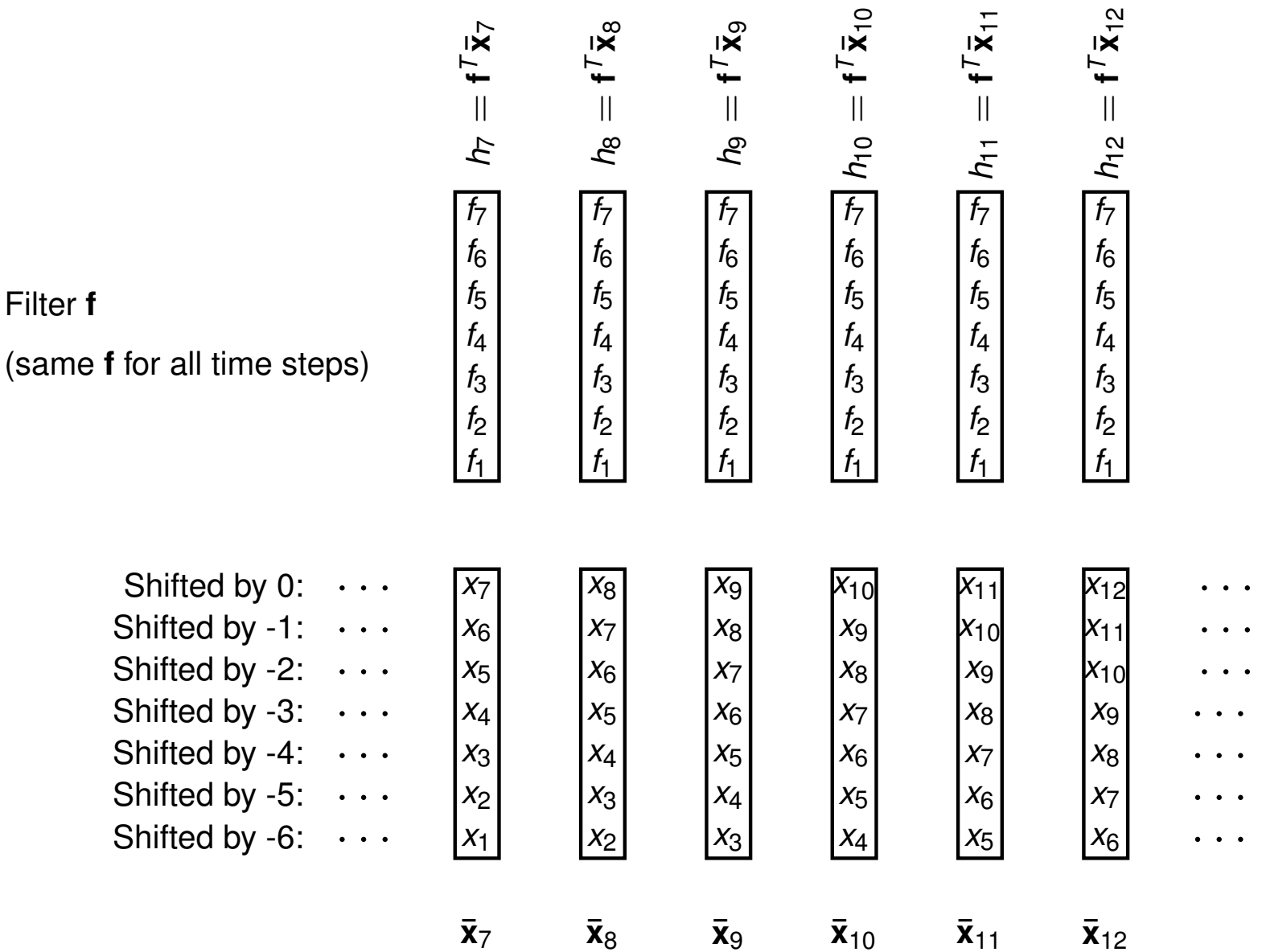
$$\hat{y}_j = wh_j + b; \quad h_j = (\mathbf{f} * \mathbf{x})_j$$

- But maybe we should weight the datapoints in our 7-day window differently? Maybe we should give a higher weight to datapoints close to the current day j?
- We can manually engineer a filter that we think is better, for example:
- $\mathbf{f}' = [\frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}]^T$
- Alternative: Let the *model* choose the optimal filter parameters, based on the training data.
- How? Gradient descent!

BACKPROPAGATION FOR 1-D CONVOLUTION



CONVOLUTION AS DOT PRODUCTS



BACKPROPAGATION FOR 1-D CONVOLUTION

- If we pretend that there is a separate filter parameter $\mathbf{f}_j$ for each $\bar{\mathbf{x}}_j$, then the gradient of the loss w.r.t. that filter would simply be:

$$\nabla_{\mathbf{f}_j} \text{loss} = \frac{\partial h_j}{\partial \mathbf{f}_j} \frac{\partial \text{loss}}{\partial h_j} = \bar{\mathbf{x}}_j \frac{\partial \text{loss}}{\partial h_j}$$

- But of course, there is only one filter. So we add up the gradients for all time steps:

$$\nabla_{\mathbf{f}} \text{loss} = \sum_j \nabla_{\mathbf{f}_j} \text{loss} = \sum_j \bar{\mathbf{x}}_j \frac{\partial \text{loss}}{\partial h_j}$$

BIAS, NONLINEARITIES, PADDING, STRIDE (1)

- **Bias and nonlinearities:**
  - In a real CNN, we usually add a trainable scalar bias b, and we apply a nonlinearity g (such as the rectified linear unit):

$$h_j = g(b + \sum_{k=1}^{K} f_k x_{(j+k-K)})$$

- **Edge cases:**
  - Possibel strategies for when the filter overlaps with the edges (beginning/end) of $\mathbf{x}$:
    - Outputs where filter overlaps with edge are undefined, i.e., $\mathbf{h}$ has fewer dimensions than $\mathbf{x}$ ($K - 1$ fewer, to be exact)
    - Pad $\mathbf{x}$ with zeros before applying the filter
    - Pad $\mathbf{x}$ with some other relevant value, such as the overall average, the first/last value, ...

BIAS, NONLINEARITIES, PADDING, STRIDE (2)

- **Stride:**
  - The stride of a convolutional layer is the “step size” with which the filter is moved over the input
  - We apply **f** to the *j*’th window of **x** only if *j* is divisible by the stride.
  - In NLP, the stride is usually 1.

CONVOLUTION WITH MORE THAN ONE AXIS

- Extending the convolution operation to tensors with more than one axis is straightforward.
- Let $\mathbf{X} \in \mathbb{R}^{J_1 \times \ldots \times J_L}$ be a tensor that has L axes and let $\mathbf{F} \in \mathbb{R}^{K_1 \times \ldots \times K_L}$ be a filter.
  - The dimensionalities of the filter axes are called filter sizes or kernel sizes ("kernel width", "kernel height", etc.)
  - From now on, we assume that the filter is applied to a symmetric window around position j, not just to positions to the left of j. (The rolling average was a special scenario, because we assume that future data points $x_{j'}$ with $j' > j$ are not available on day j.)
- Then the output $\mathbf{H} = \mathbf{F} * \mathbf{X}$ is a tensor with L axes, where:

$$h_{(j_1,\ldots,j_L)} = g\Big(b + \sum_{k_1=1}^{K_1} \cdots \sum_{k_L=1}^{K_L} f_{(k_1,\ldots,k_L)} x_{(j_1+k_1-\left\lceil \frac{K_1}{2} \right\rceil,\ldots,j_L+k_L-\left\lceil \frac{K_L}{2} \right\rceil)}\Big)$$

EXAMPLE: 2D CONVOLUTION

Filter **F** with $K_2 = K_2 = 3$

1	2	3
1	2	3
0	1	2

Input **X** with $J_1 = J_2 = 4$ (padded)

0	0	0	0	0	0
0	3	2	3	2	0
0	2	1	2	1	0
0	2	1	2	1	0
0	1	0	1	0	0
0	0	0	0	0	0

Output **H** with $J_1 = J_2 = 4$
(without bias or nonlinearity)

16	?	?	?
?	?	26	?
?	?	?	?
?	?	?	5

CHANNELS

- So far, we have assumed that each position in the input (each day or each pixel) contains a single scalar.
- Now, we assume that our input has M features ("channels") per position, i.e., there is an additional feature axis: $\mathbf{X} \in \mathbb{R}^{J_1 \times \ldots \times J_L \times M}$
- Example:
  - $M = 3$ channels per pixel in an RGB (red/green/blue) image
  - In NLP: dimensionality of pretrained word embeddings
- Then our filter gets an additional feature axis, which has the same dimensionality as the feature axis of $\mathbf{X}$:

$$\mathbf{F} \in \mathbb{R}^{K_1 \times \ldots \times K_L \times M}$$

- During convolution, we simply sum over this new axis as well:

$$h_{(j_1,\ldots,j_L)} = g\Big(b + \sum_{k_1=1}^{K_1} \ldots \sum_{k_L=1}^{K_L} \sum_{m=1}^{M} f_{(k_1,\ldots,k_L,m)} x_{(j_1+k_1-\left\lceil \frac{K_1}{2} \right\rceil,\ldots,j_L+k_L-\left\lceil \frac{K_L}{2} \right\rceil,m)}\Big)$$

FILTER BANKS

- A filter bank is a tensor that consists of N filters, which each have the same shape.
- The filters of a filter bank are applied to $\mathbf{X}$ independently, and their outputs are stacked (they form a new axis in the output).
- So let this be our input and filter bank:

$$\mathbf{X} \in \mathbb{R}^{J_1 \times \ldots \times J_L \times M}$$

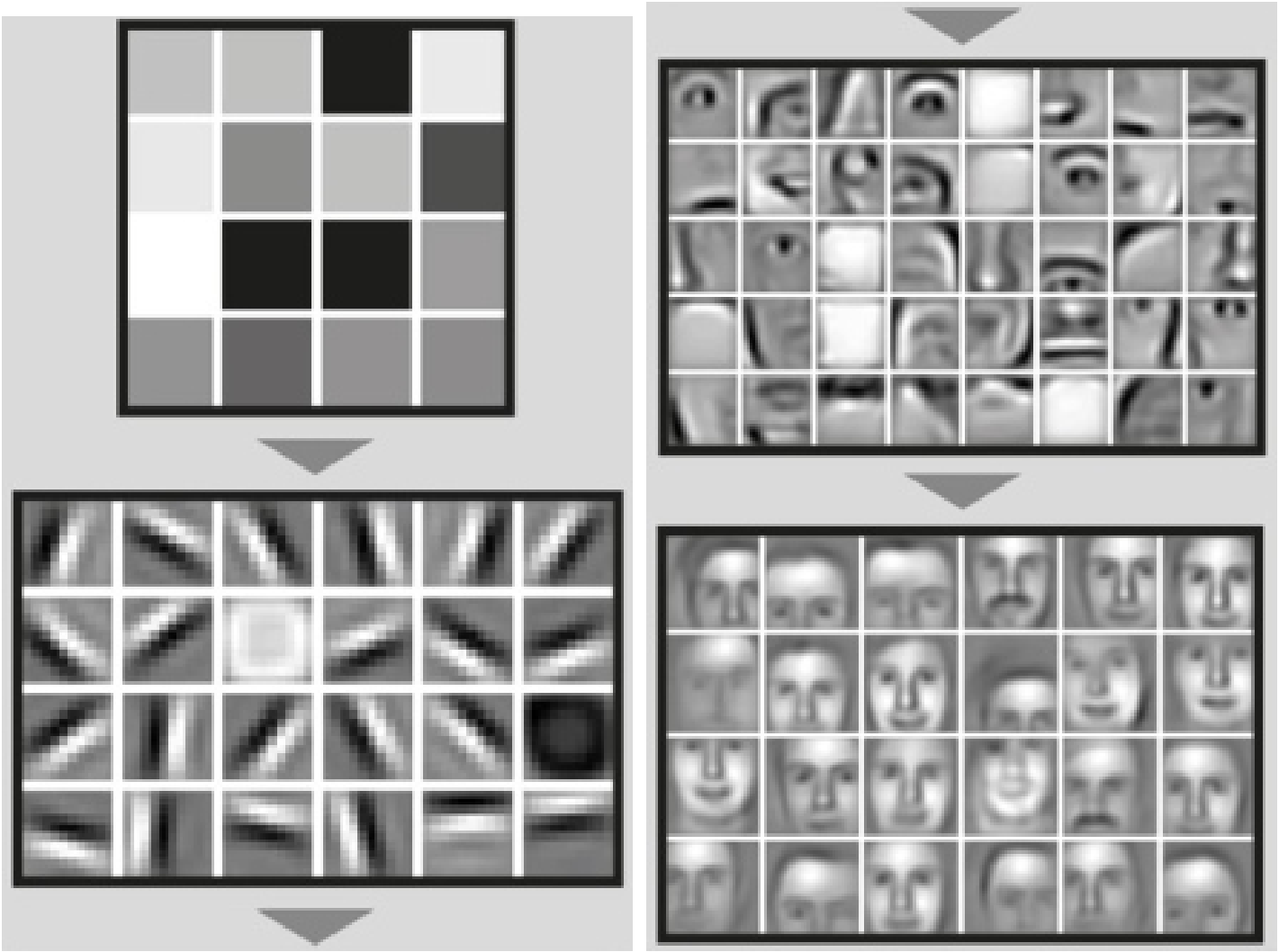
$$\mathbf{F} \in \mathbb{R}^{K_1 \times \ldots \times K_L \times M \times N}$$

- Then our output is a tensor of shape $\mathbf{H} \in \mathbb{R}^{J_1 \times \ldots \times J_L, N}$
  - (assuming that we are padding the first L axes of $\mathbf{X}$ to preserve their dimensionality in $\mathbf{H}$)
- where $h_{(j_1,\ldots,j_L,n)}$ is same as above

ASSUMPTIONS BEHIND CONVOLUTION

- Translation invariance: Same object in different positions.
  - Parameter sharing: Filters are shared between all positions.
- Locality: Meaningful objects form coherent areas
  - In a single convolutional layer, information can travel no further than a few positions (depending on filter size)
- Hierarchy of features from simple to complex:
  - In computer vision, we often apply many convolutional layers one after another
  - With every layer, the information travels further, and the feature vectors become more complex
  - Pixels $\rightarrow$ edges $\rightarrow$ shapes $\rightarrow$ small objects $\rightarrow$ bigger objects

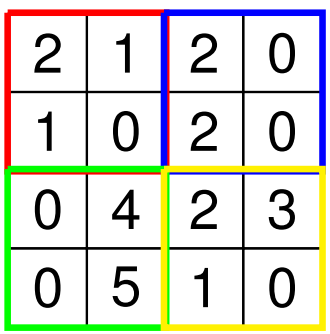
CONVOLUTION



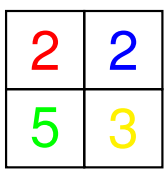
Source: Computer science: The learning machines. Nature (2014).

POOLING LAYERS (1)

- Pooling layers are parameter-less
- Divide axes of **H** (excluding the filter/feature axis) into a coarse-grained “grid”
- Aggregate each grid cell with some operator, such as the average or maximum
- Example (shown without a feature axis):



Max pooled

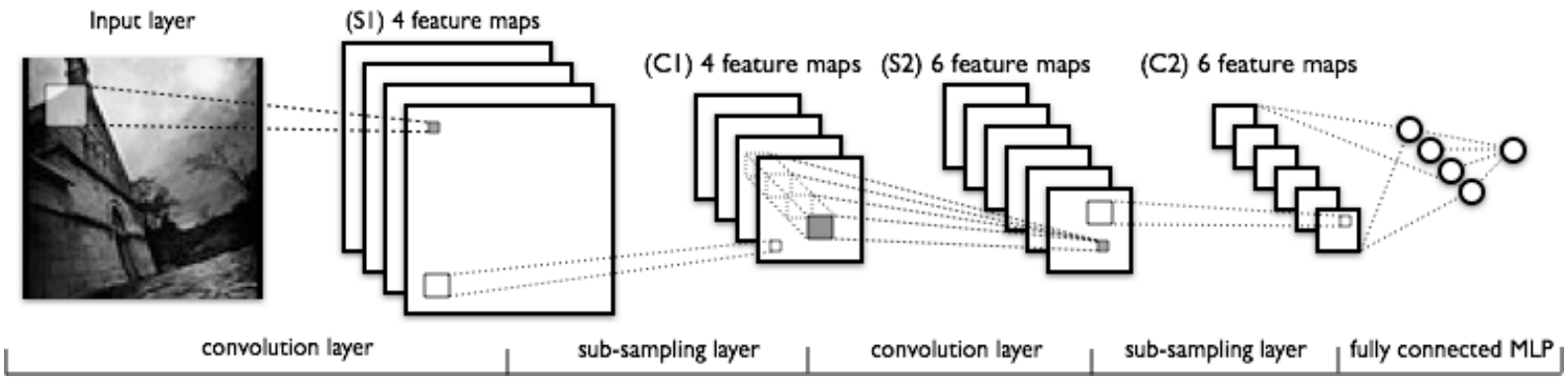


POOLING LAYERS (2)

- When pooling, we lose fine-grained information about exact positions
- In return, pooling allows us to aggregate features from a local neighborhood into a single feature.
    - For example, if we have a neighborhood where many “dog features” were detected (meaning, shapes that look like parts of a dog), we want to aggregate that information into a single “dog neuron”

CONVOLUTION AND POOLING: LENET

- In computer vision, we often apply pooling between convolutional layers.
- Repeated pooling has the effect of reducing tensor sizes.



LeCun et al. (1998). Gradient-based learning applied to document recognition.

CNNs IN NLP

- Images are 2D, but text is a 1D sequence (of words, n-grams, etc).
- Words are usually represented by M-dimensional word embeddings (e.g., from Word2Vec)
- So on text, we do 1-D convolution with M input features:
  - Input matrix: $\mathbf{X} \in \mathbb{R}^{J \times M}$
  - Filter bank of N filters: $\mathbf{F} \in \mathbb{R}^{K \times M \times N}$; $K < J$
  - Output matrix: $\mathbf{H} \in \mathbb{R}^{J \times N}$
    - (assuming that we padded $\mathbf{X}$ with zeros along its first axis)
- Usually, CNNs in NLP are not as deep as CNNs in Computer Vision – often just one convolutional layer.

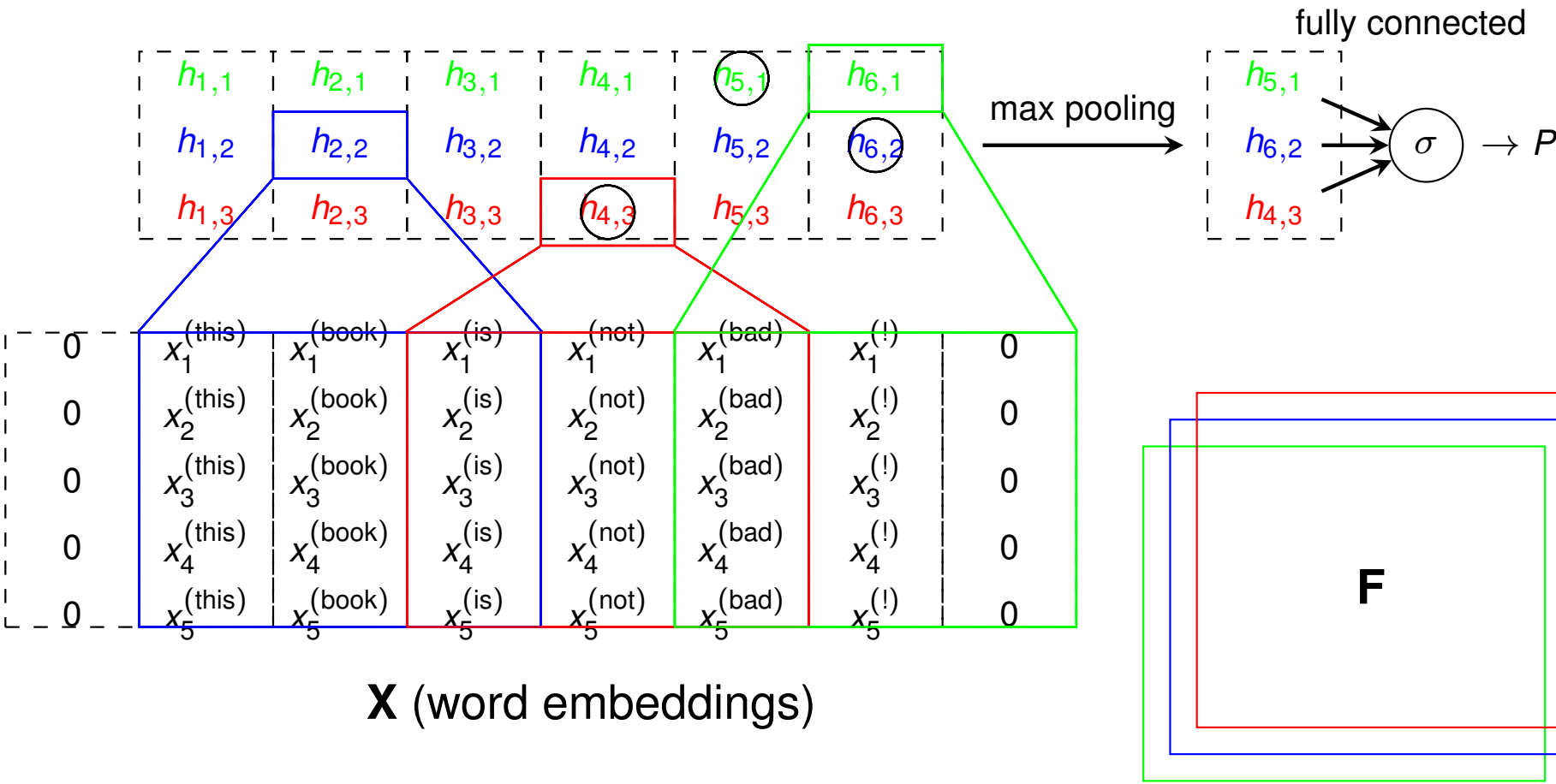
POOLING IN NLP

- Pooling is less frequently used in NLP.
- If the task is a word-level task (i.e., we need one output vector per word), we can simply use the output of the final convolutional layer – no pooling needed.
- If the task is some sentence-level task (i.e., we need a single vector to represent the entire sentence), we usually do a "global" pooling step over the entire sentence *after* the last convolutional layer:

$$\mathbf{h}_{\text{avgpool}} = \frac{1}{J} \sum_{j=1}^{J} \mathbf{h}_j$$

$$\mathbf{h}_{\text{maxpool}} = \begin{bmatrix} \max_j & h_{j,1} \\ & \vdots \\ \max_j & h_{j,N} \end{bmatrix}$$

CNNs IN NLP



X (word embeddings)

X = “this book is not bad !”, Y = “positive”

Parameters in the filter bank? $3 \times 5 \times 3$ (assuming no bias)

CNNs IN NLP, "ROTATED VIEW"

- CNN are used to derive latent (dense) representations of larger portions of text by aggregating representations across **short sequences** (of words)
  - This allows for modeling of local dependencies
  - But cannot capture long-distance semantic interactions