Deep Neural Networks for Structured Inputs (contd)

CS771: Introduction to Machine Learning

Convolutional Neural Networks (CNN)

- CNN consists of a sequence of operations to transform an input to output
  - Convolution (a linear transformation but more "local" than the one in MLP)
  - Nonlinearity (e.g., sigmoid, ReLU, etc) after the convolution operation
  - Pooling (aggregates local features into global features and reduce representation size)

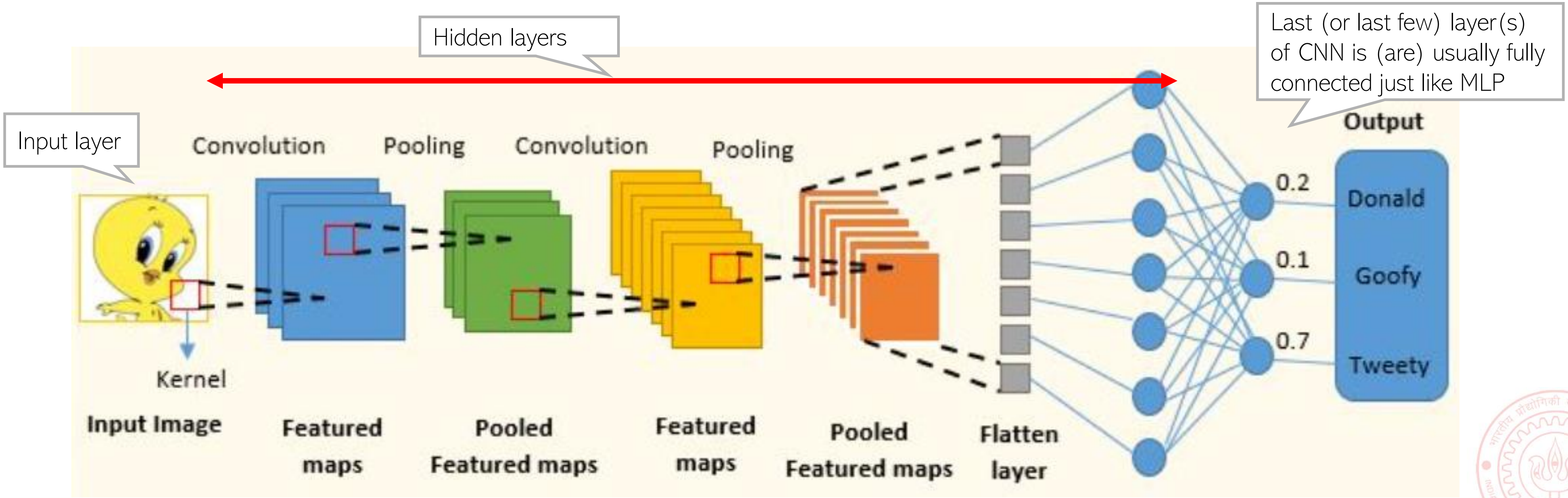
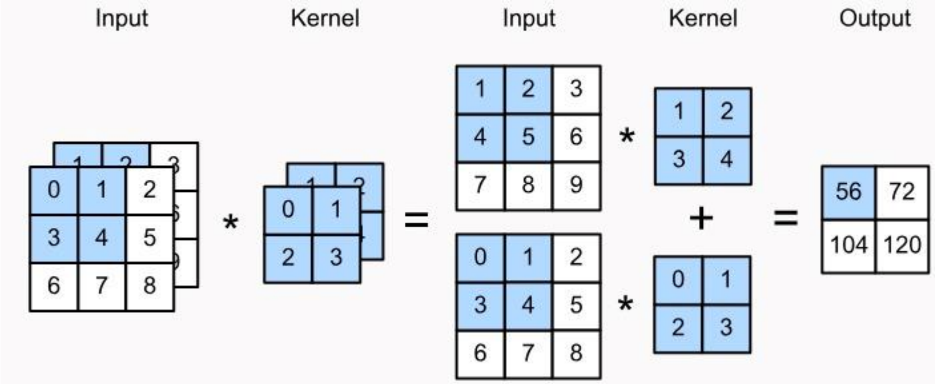
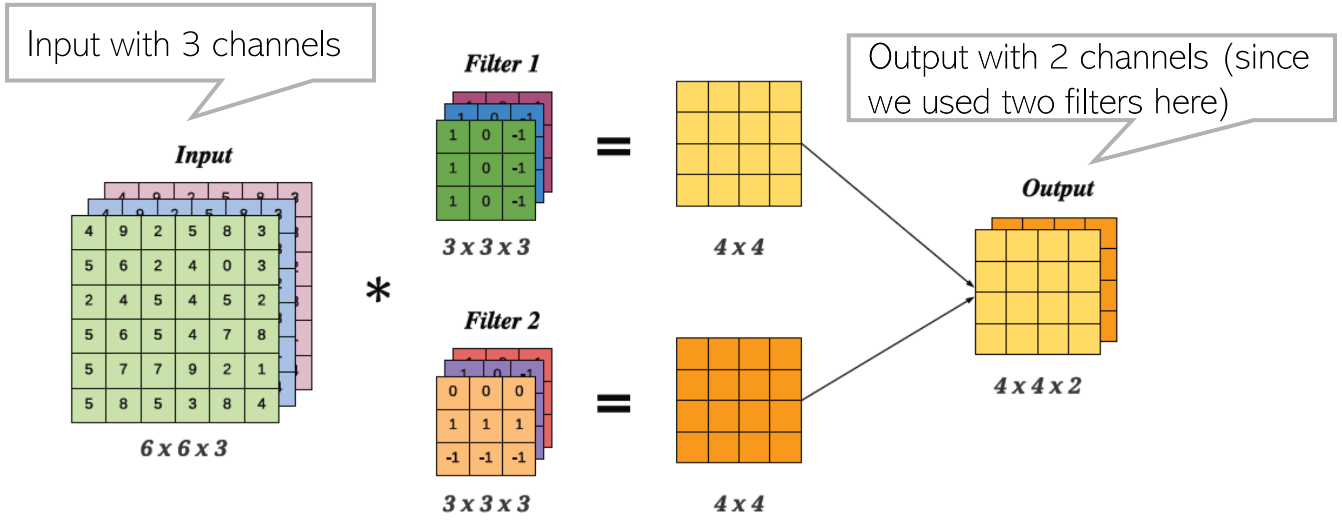


Figure credit: https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/

Multiple Input Channels

- If the input has multiple channels (e.g., images with R,G,B channels), then each filter/kernel also needs to have multiple channels, as shown below (left figure)
- We perform per-channel convolution followed by an aggregation (sum across channels)



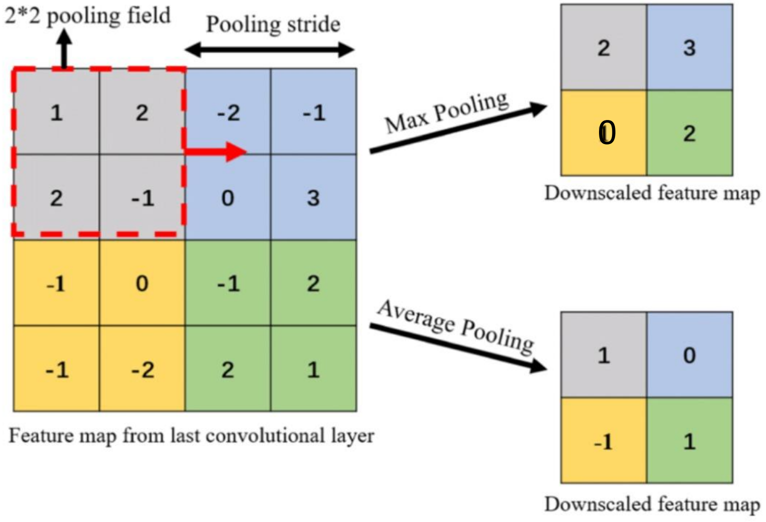


- Note that (right figure above) we typically also have multiple such filters (each with multiple channels) which will give us multiple such feature maps

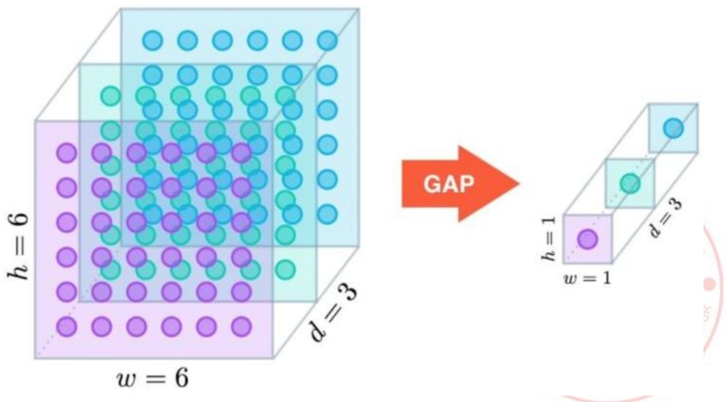
Figure credit: PML-1 (Murphy, 2022),

Pooling

- CNNs also consist of a pooling operation after each conv layer
- Pooling plays two important roles
  - Reducing the size of the feature maps
  - Combining local features to make global features
- Need to specify the size of group to pool, and pooling stride

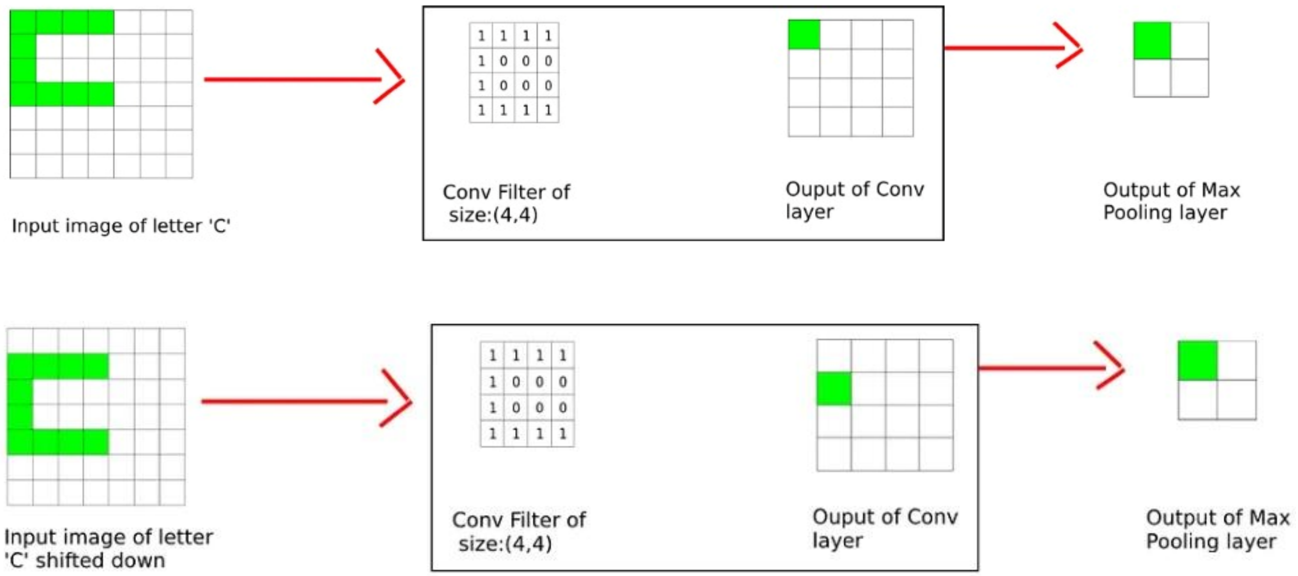


- Max pooling and average pooling are popular pooling methods
- "Global average pooling" (GAP) is another option
  - Given feature map of size $h \times w \times d$ (e.g, if there are d channels), it averages all $h \times w$ locations to give a $1 \times d$ feature map
  - Reduces the number of features significantly and also allows handling feature maps of different heights and widths



CNNs have Translation Invariance!

- Even if the object of interest has shifted/translated, CNN don't face a problem (it will be detected regardless of its location in the image)
- The simple example below shows how (max) pooling helps with this



- CNNs use a combination of conv + pooling operations in several hidden layers so CNNs remain invariant to even more significant translations

Pic credit: https://divsoni2012.medium.com/translation-invariance-in-convolutional-neural-networks-61d9b6fa03df

CNN: Summary of the overall architecture

- The overall structure of a CNN looks something like this

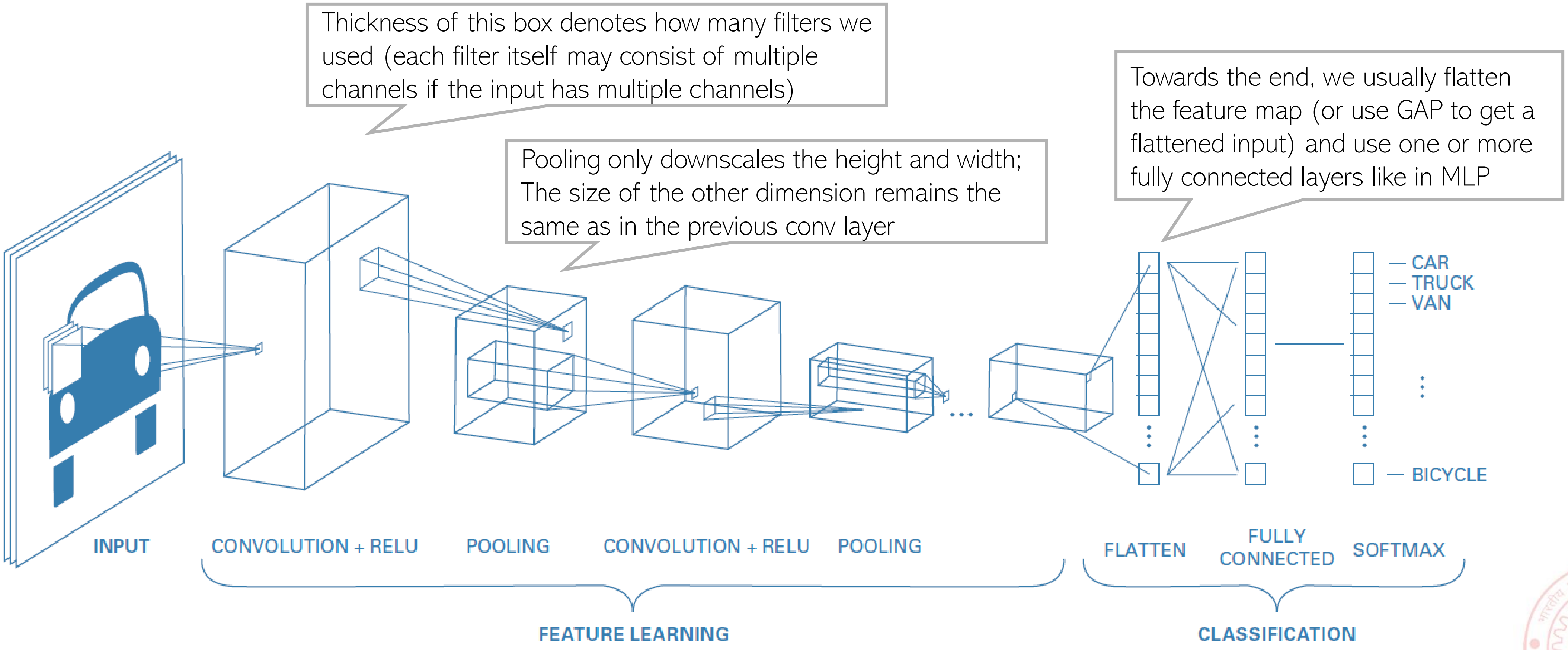
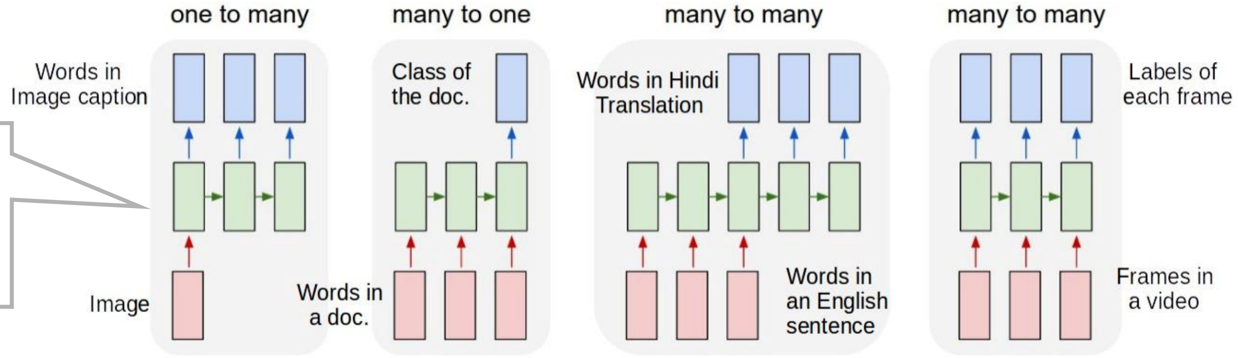


Figure credit: PML-1 (Murphy, 2022),

Sequential Data

- In many problems, each input, each output, or both may be in form of sequences

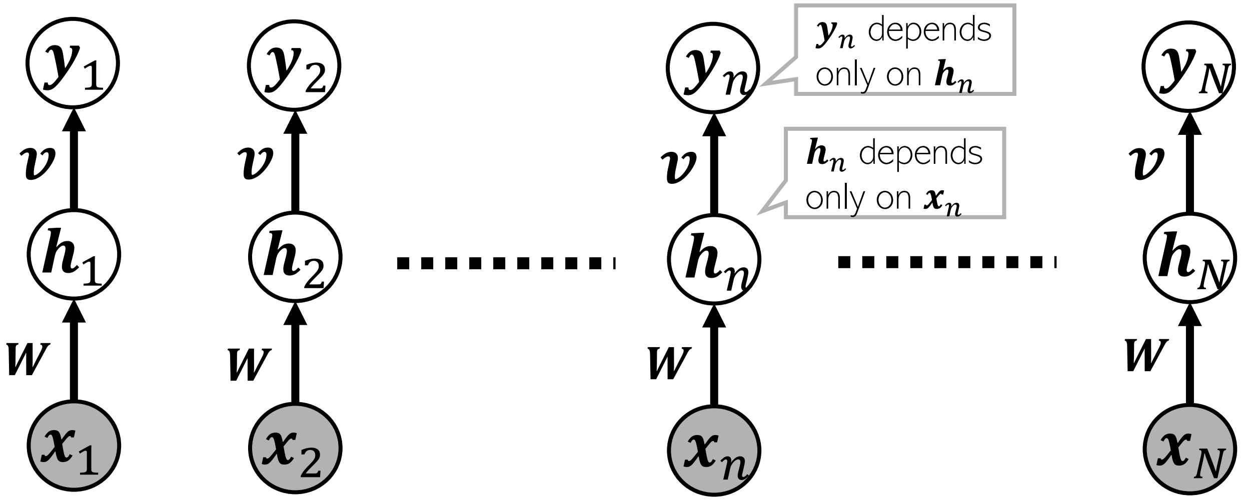
These green cells are some feature representation (e.g., hidden layer of a deep neural network) of the inputs



- Different inputs or outputs need not have the same length
- Some examples of prediction tasks in such problems
  - Image captioning: Input is image (not a sequence), output is the caption (word sequence)
  - Document classification: Input is a word sequence, output is a categorical label
  - Machine translation: Input is a word sequence, output is a word sequence (in different language)
  - Stock price prediction: Input is a sequence of stock prices, output is its predicted price tomorrow
  - No input – just output (e.g., generation of random but plausible-looking text)

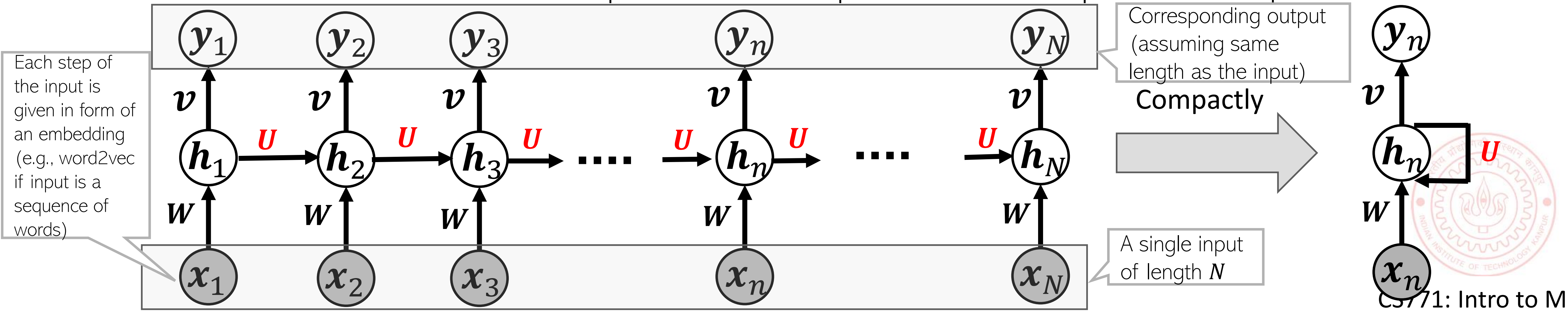
Recurrent Connections in Deep Neural Networks

- Feedforward nets such as MLP and CNN assume independent observations



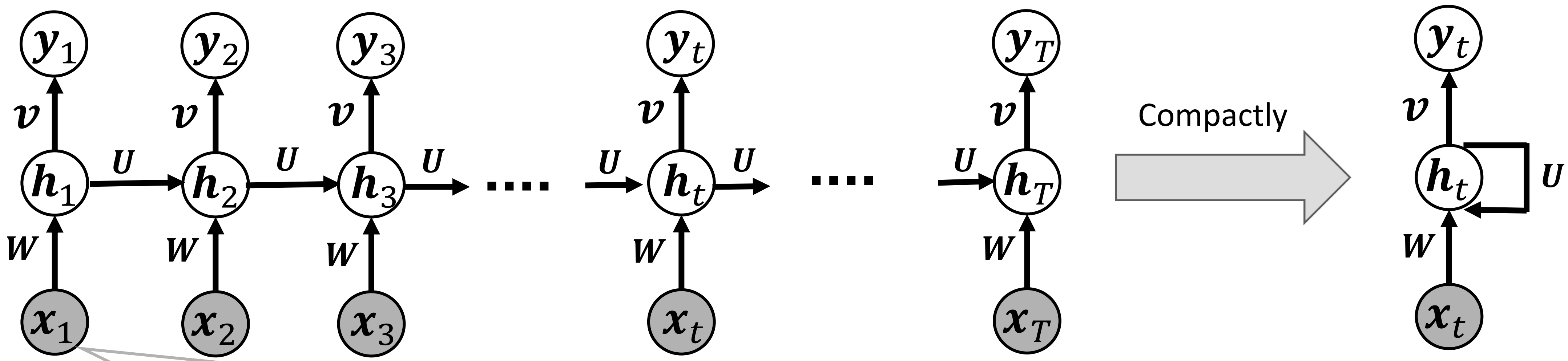
Feedforward neural networks are not ideal when inputs $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ and/or outputs $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$ represent sequential data (e.g., sentences)

- A recurrent structure can be helpful if each input and/or output is a sequence



Recurrent Neural Networks

- A basic RNN's architecture (assuming input and output sequence have same lengths)



Given in form of an embedding (e.g., word embedding if x_1 is a word

- RNN has three sets of weights $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{v}$

g is some activation function like ReLU

- $\boldsymbol{W}$ and $\boldsymbol{U}$ model how h_t at step t is computed: $\boldsymbol{h}_t = g(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1})$
- $\boldsymbol{v}$ models the hidden layer to output mapping, e.g., $\boldsymbol{y}_t = o(\boldsymbol{v}\boldsymbol{h}_t)$

o depends on the nature of y_t.If it is categorical then o can be softmax

- Important: Same $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{v}$ are used at all steps of the sequence (weight sharing)

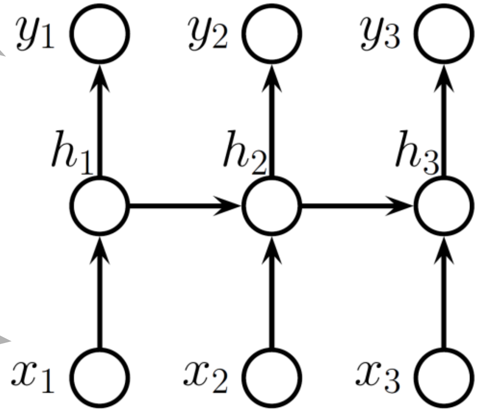
Recurrent Neural Networks: Some Examples

- Parts of speech tagging (or "aligned" translation; input and output have same length)

Parts of speech tag for each word

Each node denotes an embedding of the word

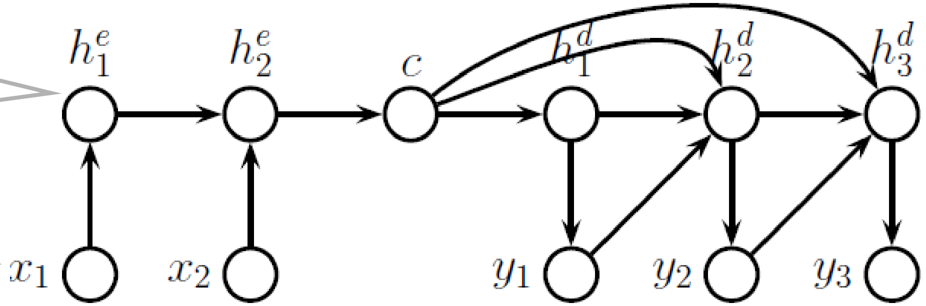
Words in a sentence



- "Unaligned" translation (input and output can have different lengths)

Such problems usually require a sequence encoder- sequence decoder architecture

Encode the input sequence (embeddings of tokens) into a single embedding vector $\boldsymbol{c}$ and then decode this embedding one output token at a time



- In the unaligned case, generation stops when an "end" token (e.g., <END>) is generated on the output side

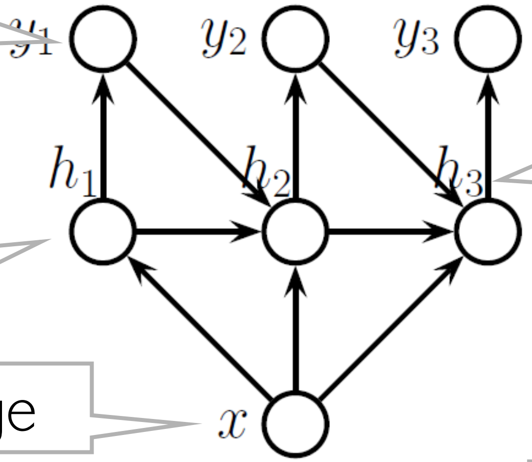
Recurrent Neural Networks: Some Examples

- Consider generating a sequence $y_1, y_2, \ldots, y_T$ given an input x

Words in the generated caption

Hidden states at each step of the sequence

An image



Predicted y_{t-1} also fed into h_t

At test time, we can only feed the predicted y_{t-1}

During training, if the true y_{t-1} is fed, we call it "teacher forcing"

- Predicting the sentiment of a movie review

This final hidden state is supposed to contain the information about the entire review

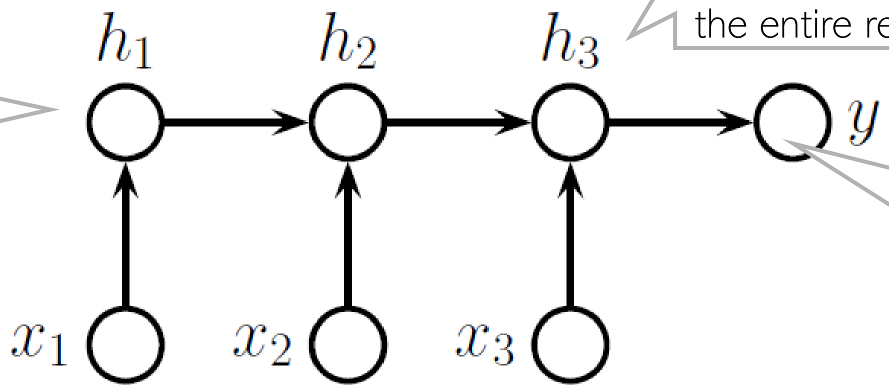
Isn't this too much to expect?? ☺

Indeed; this can be an issue with RNNs

Hidden states

Each node denotes an embedding of the word

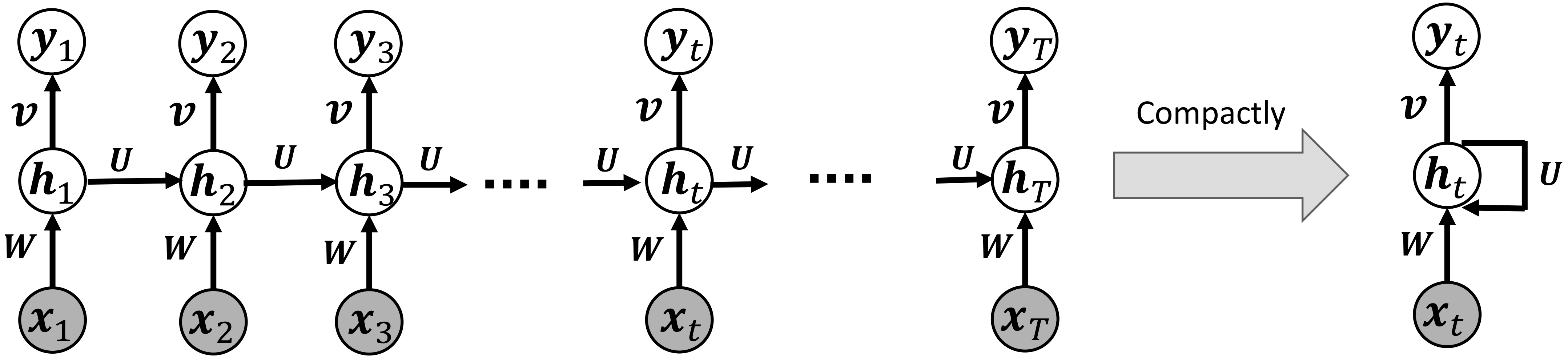
Words in the review



Predicted sentiment

For RNNs, Long Distant Past is Hard to Remember

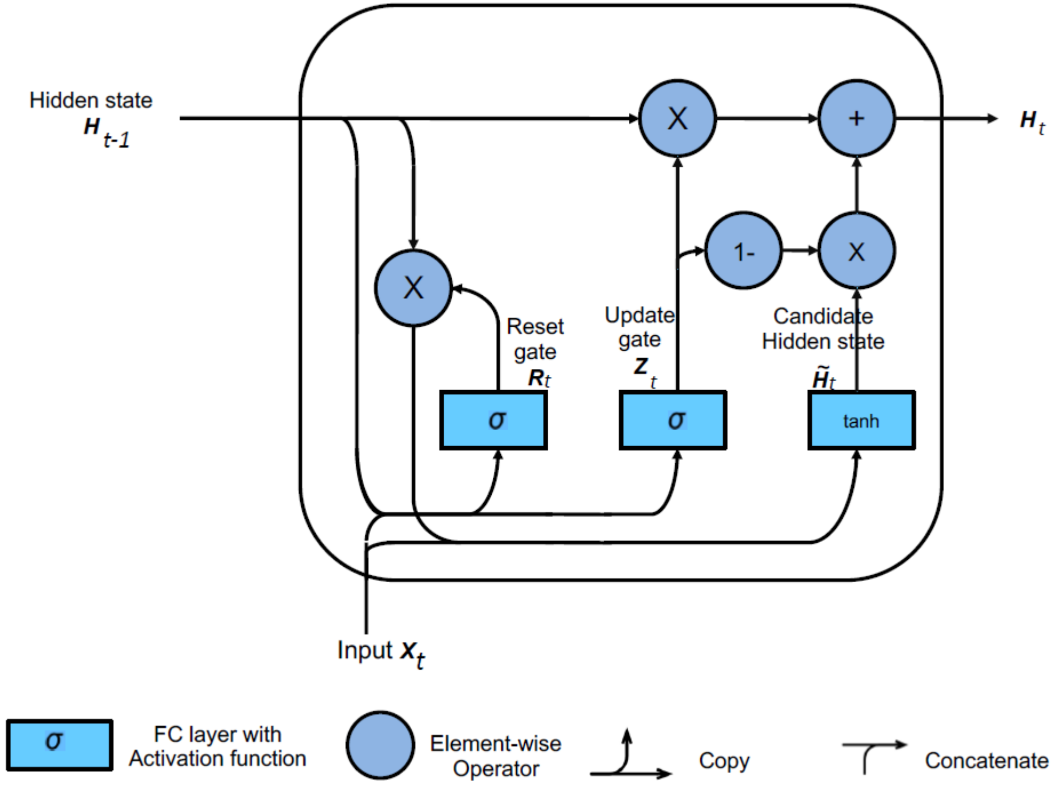
- The hidden layer nodes $\boldsymbol{h}_t$ are supposed to summarize the past up to time $\boldsymbol{t-1}$

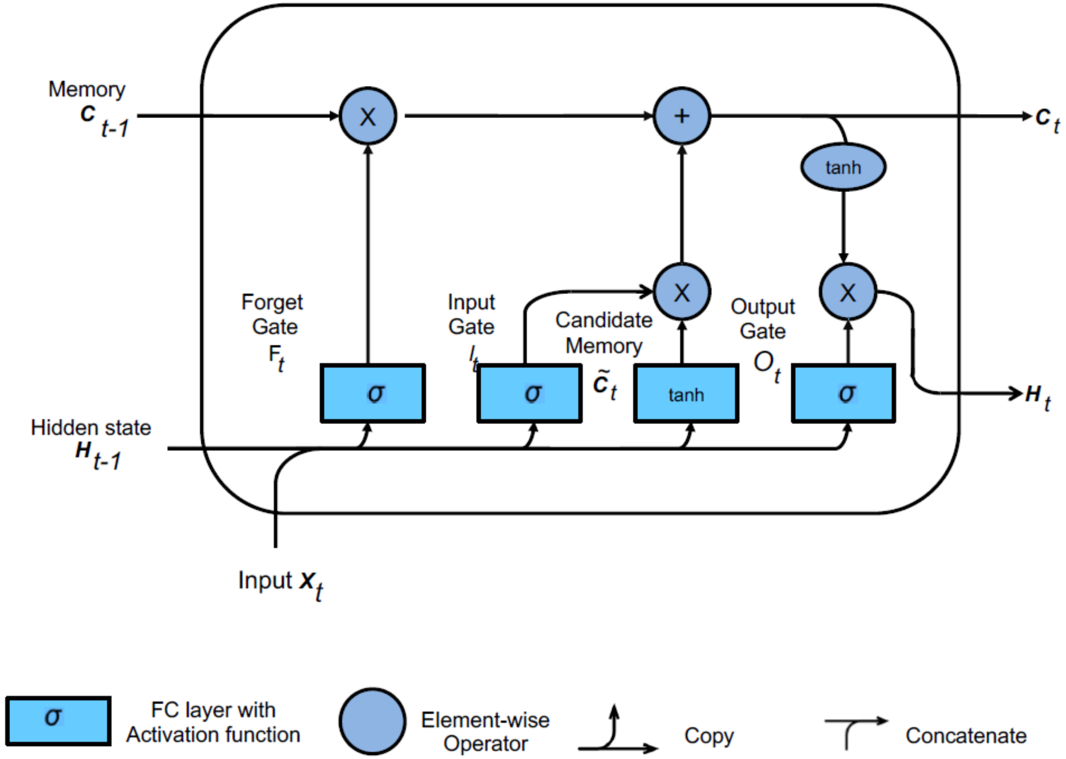


- In theory, they should. In practice, they can't. Some reasons
  - Vanishing gradients along the sequence too – past knowledge gets "diluted"
  - Hidden nodes also have limited capacity because of their finite dimensionality
- Various extensions of RNNs have been proposed to address forgetting
  - Gated Recurrent Units (GRU), Long Short Term Memory (LSTM)

GRU and LSTM

- GRU and LSTM are variants of RNNs. These contain specialized units and "memory" which modulate what/how much information from the past to retain/forget

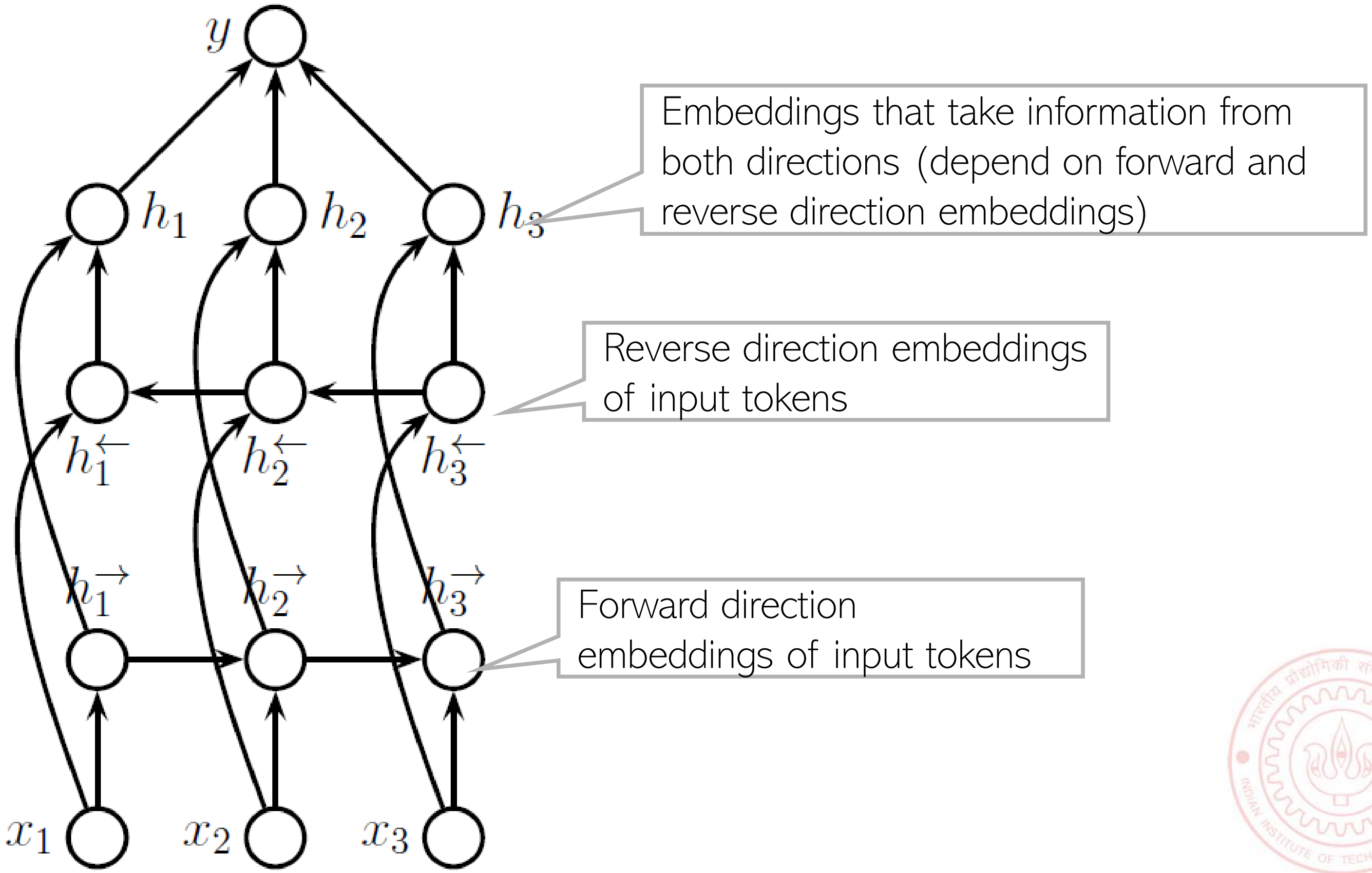




Pic source: https://d2l.ai/

Bidirectional RNN

- RNNs and GRU and LSTM only remember the information from the previous tokens
- Bidirectional RNNs can remember information from the past and future tokens



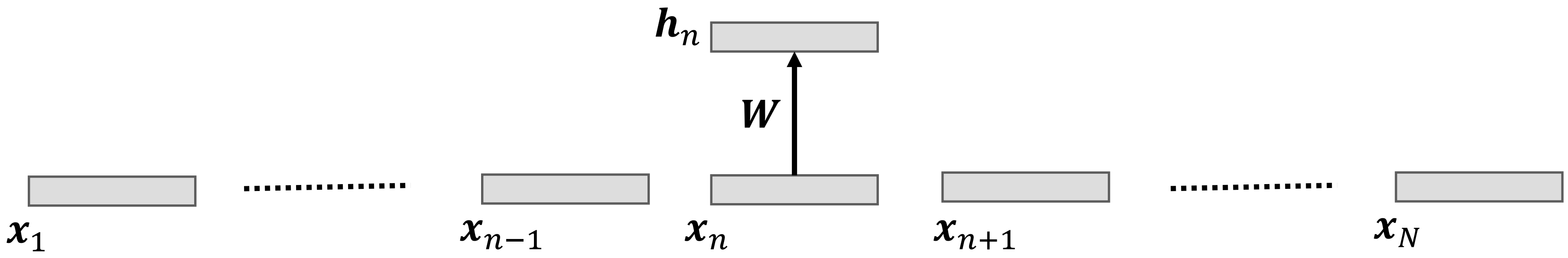
Attention

- Each layer in standard deep neural nets computes a linear transform + nonlinearity
- For N inputs, collectively denoting inputs as $\boldsymbol{X} \in \mathbb{R}^{N \times K_1}$ and outputs as $\boldsymbol{H} \in \mathbb{R}^{N \times K_2}$

$$\boldsymbol{H} = g(\boldsymbol{X}\boldsymbol{W})$$

Notation alert: Input $\boldsymbol{X}$ can be data (if $\boldsymbol{H}$ denotes first hidden layer) or the $\boldsymbol{H}$ of the previous hidden layer

- Here the weights $\boldsymbol{W} \in \mathbb{R}^{K_1 \times K_2}$ do not depend on the inputs $\boldsymbol{X}$
  - Output $\boldsymbol{h}_n = g(\boldsymbol{W}^\top \boldsymbol{x}_n) \in \mathbb{R}^{K_2}$ only depends on $\boldsymbol{x}_n \in \mathbb{R}^{K_1}$ and pays no attention to $\boldsymbol{x}_m$, $m \neq n$

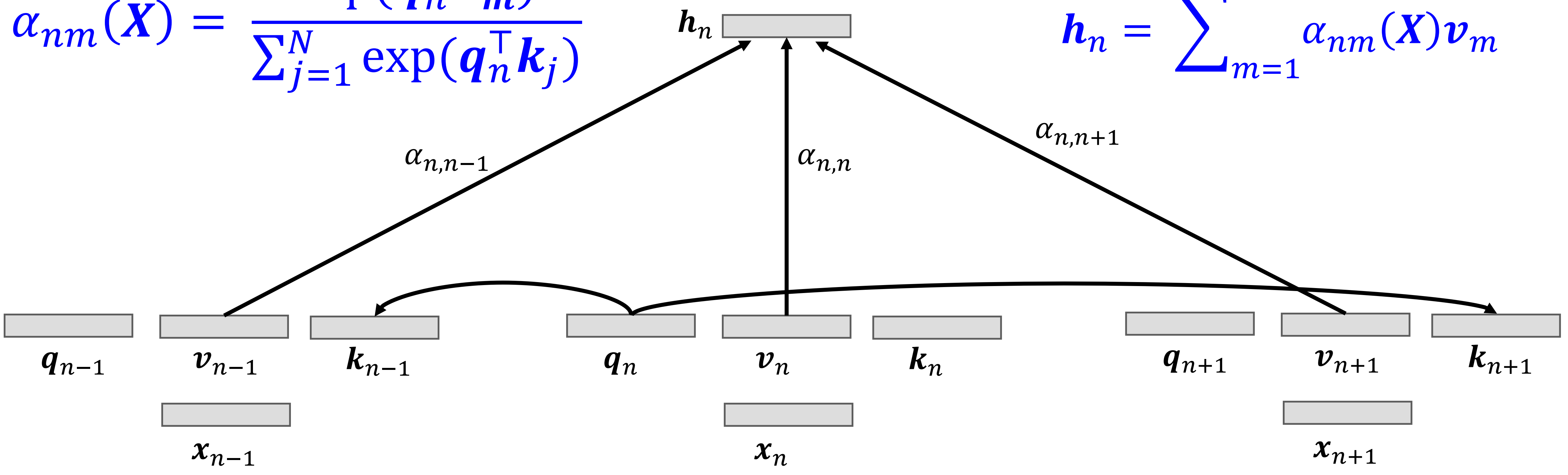


  - When different inputs outputs have inter-dependencies (e.g., they denote representations of words in a sentence, or patches in an image), paying attention to other inputs is helpful/needed

The Attention Mechanism

$$\alpha_{nm}(\boldsymbol{X}) = \frac{\exp(\boldsymbol{q}_n^\top \boldsymbol{k}_m)}{\sum_{j=1}^{N} \exp(\boldsymbol{q}_n^\top \boldsymbol{k}_j)}$$

$$\boldsymbol{h}_n = \sum_{m=1}^{N} \alpha_{nm}(\boldsymbol{X}) \boldsymbol{v}_m$$



$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$ $\quad$ $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K$ $\quad$ $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V$