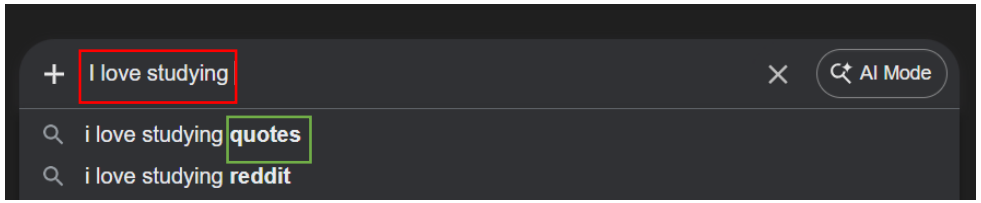
Introduction to Machine Learning

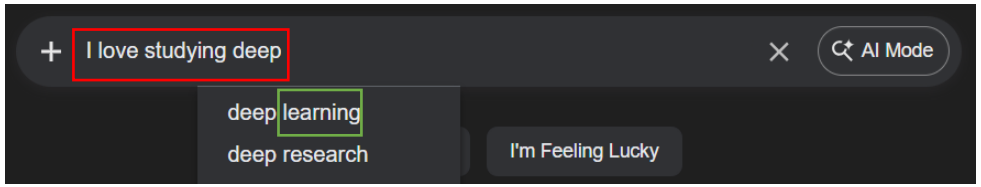
Recurrent Neural Networks (RNNs)

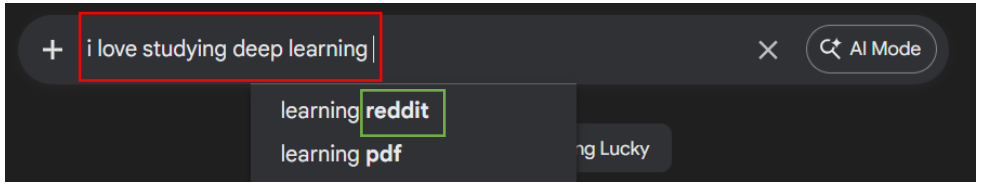
Prof. Subrahmanya Swamy Peruru
Department of Electrical Engineering
IIT Kanpur

Sequential Tasks

Example: Predict the next word







How to feed words as inputs to neural networks?

How to handle inputs of varying length?

Words are symbols; neural networks need numbers

- A neural network cannot directly process words because these are just symbols /tokens, not numerical vectors.

Goal: word → vector of numbers

- So we first choose a large **vocabulary** (corpus) of size V:

$$\mathcal{V} = \{\text{cat}, \text{car}, \text{dog}, \text{sat}, \text{on}, \text{mat}, \dots\}$$

and then represent each word in the vocabulary by a vector.

Two common choices:

- One-hot vector:
  - Sparse
  - large dimension V
  - does not capture similarity: (cat, dog) are as unrelated as (cat, car)

$$\mathbf{x}(\text{cat}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{x}(\text{car}) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{x}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots$$

$$d(cat, dog) = \sqrt{2}$$
$$d(cat, car) = \sqrt{2}$$

$$cosine(cat, dog) = 0$$
$$\text{cosine}\ (cat, car) = 0$$

- Word embedding: dense, lower dimension D

Dense word embeddings: Word2Vec, GloVe, FastText

Instead of using a huge sparse one-hot vector, we learn a dense vector:

$$\mathbf{x}(\text{word}) \in \mathbb{R}^D, \qquad D \ll V$$

Examples:

- **Word2Vec (**word2vec-google-news-300**)**: $D = 300, V \approx 3$ million
- **GloVe (**Common Crawl**)**: $D = 300, V \approx 2.2$ million
- **FastText (**crawl-300d-2M.vec**)**: $D = 300, V \approx 2$ million

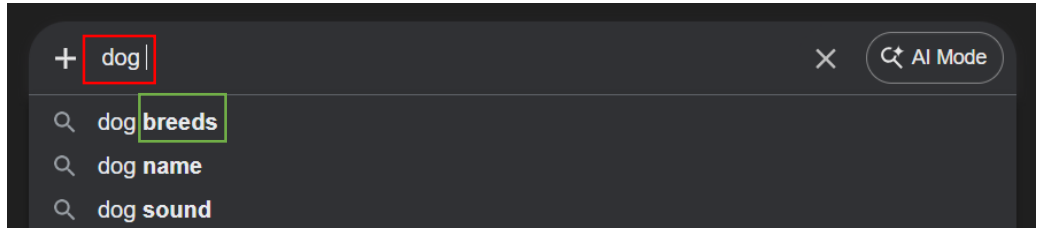
Key advantage: semantically similar words get similar vectors.

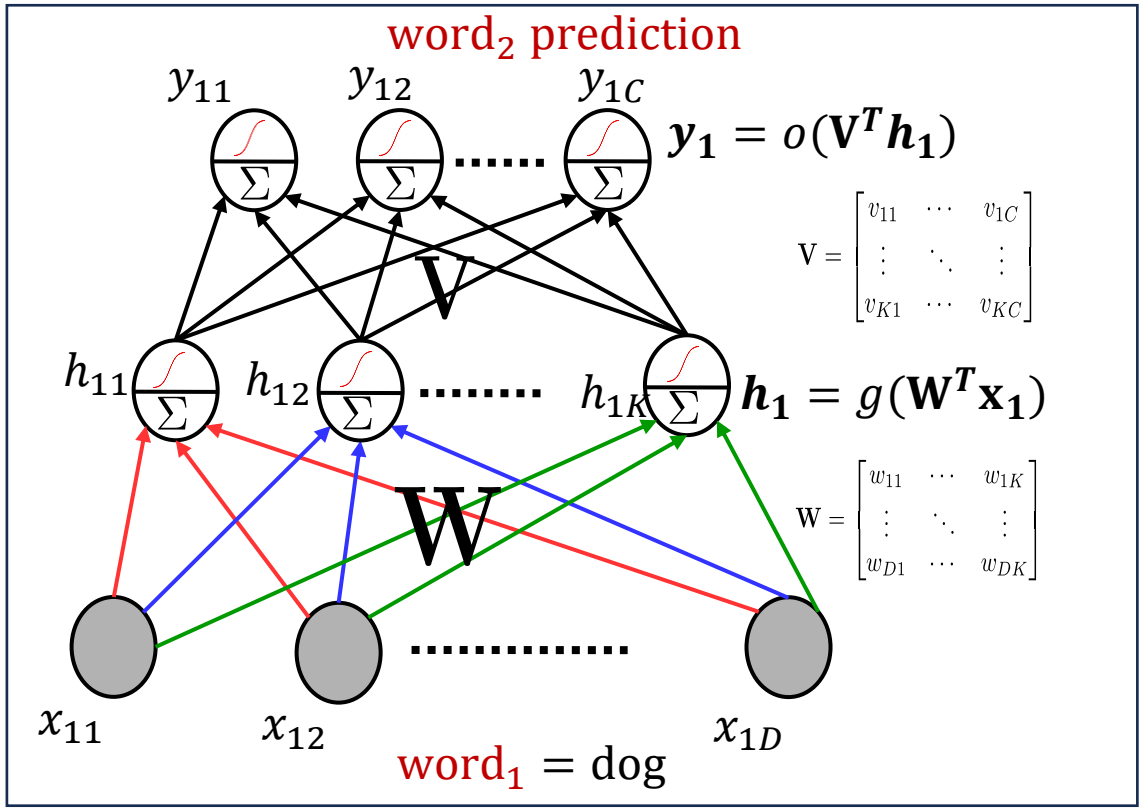
```python
import gensim.downloader as api

# load pretrained model
model = api.load("word2vec-google-news-300")

# compute similarities
print("cat vs dog:", model.similarity("cat", "dog"))
print("cat vs car:", model.similarity("cat", "car"))
```

cat vs dog cosine similarity: 0.76

cat vs car cosine similarity: 0.21

Sequential Tasks





1. Word2vec for input:

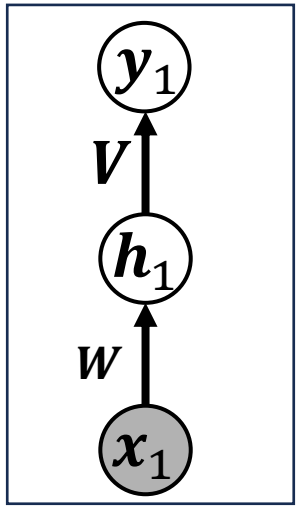
$$\text{word}_1 = \text{dog} \rightarrow \quad \boldsymbol{x_1} = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1D} \end{bmatrix}$$

2. Feed embedding vector $\boldsymbol{x_1}$ to neural network

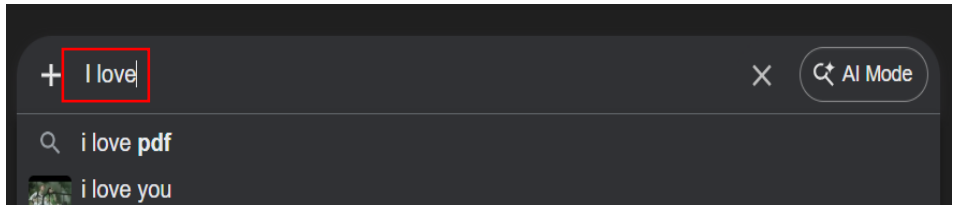
3. Obtain Predictions (probabilities) for next word

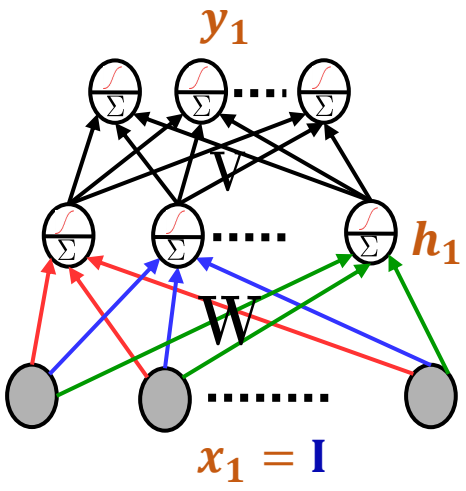
$$\mathbf{y_1} = \begin{bmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1C} \end{bmatrix} = \begin{bmatrix} \text{prob}(\text{word}_2 = \text{breeds}) \\ \text{prob}(\text{word}_2 = \text{name}) \\ \vdots \\ \text{prob}(\text{word}_2 = \text{sound}) \end{bmatrix}$$

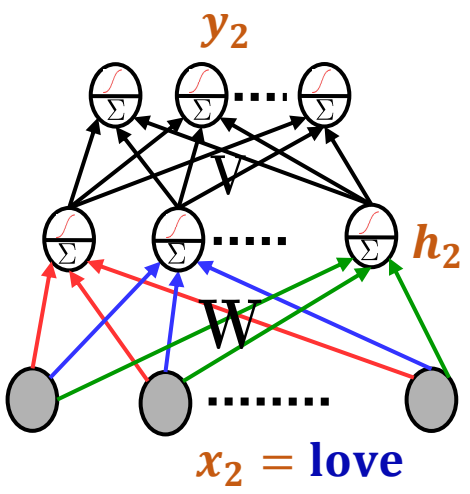
Compact Representation:

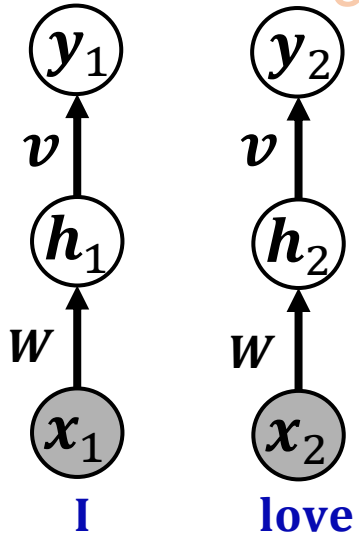


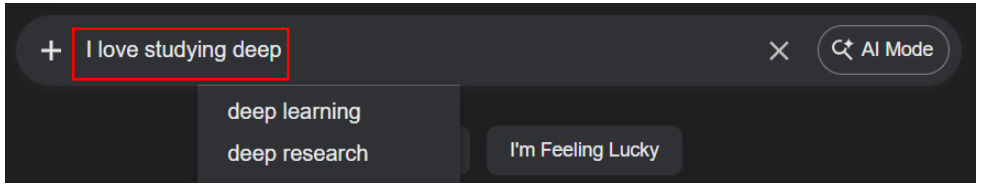
Sequential Tasks

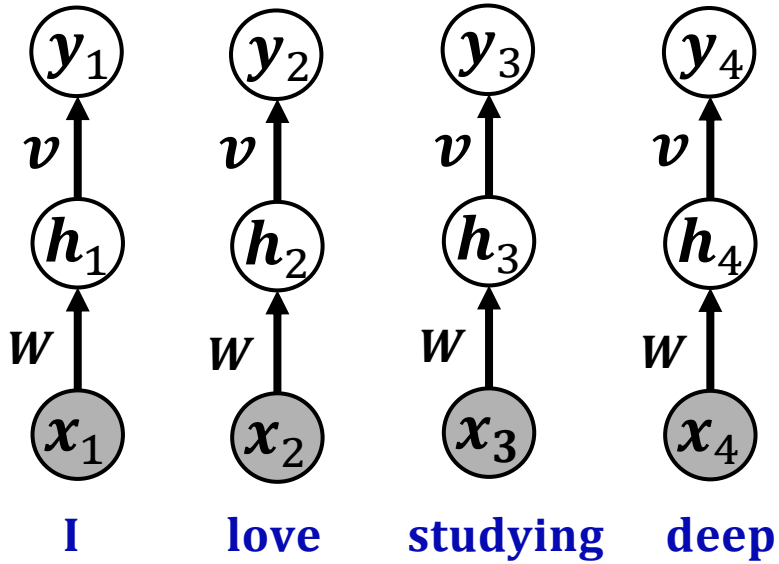






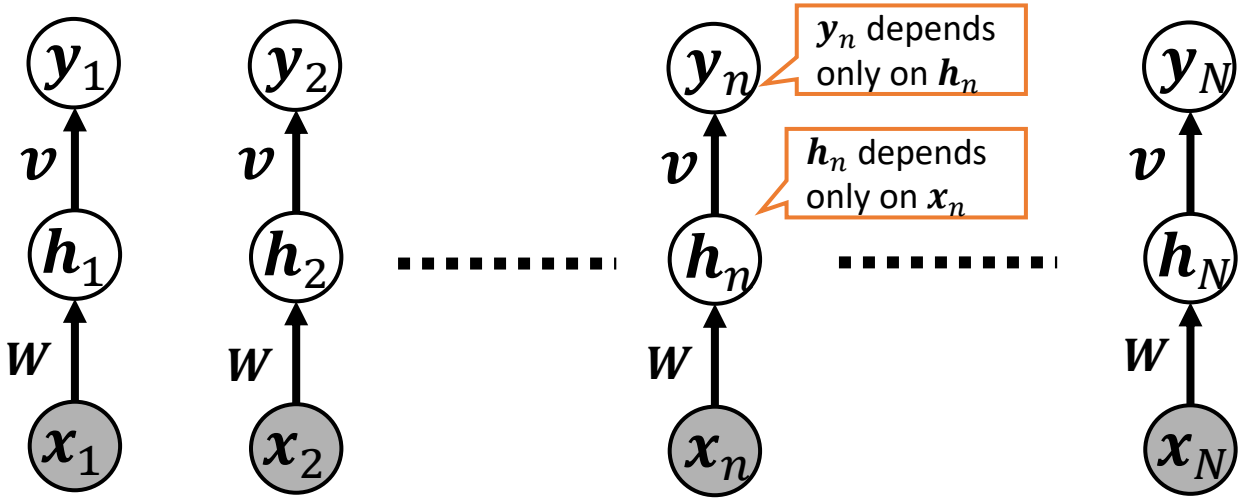






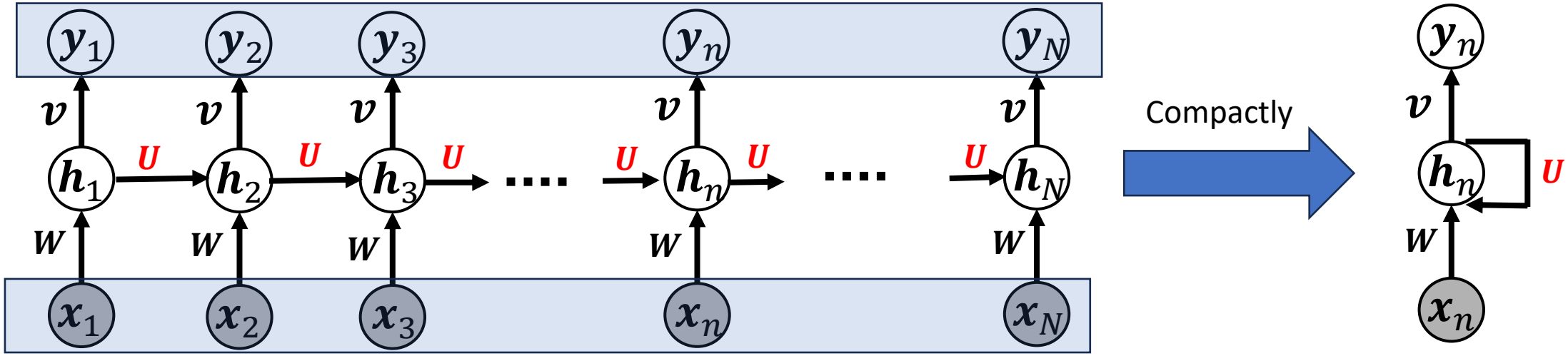
Recurrent Connections in Deep Neural Networks

- Feedforward nets such as MLP and CNN assume independent observations



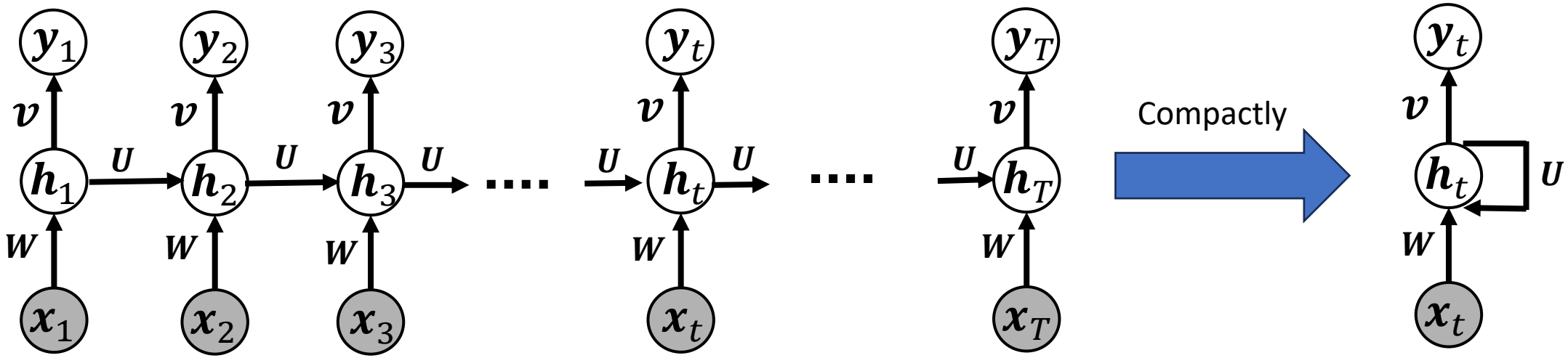
Feedforward neural networks are not ideal when inputs $[x_1, x_2, \ldots, x_N]$ and/or outputs $[y_1, y_2, \ldots, y_N]$ represent sequential data (e.g., sentences)

- 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)



- RNN has three sets of weights $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{v}$
- $\boldsymbol{W}$ 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., $\boldsymbol{y}_t = o(\boldsymbol{v}\boldsymbol{h}_t)$
- Important: Same $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{v}$ are used at all steps of the sequence (weight sharing)

Motivation for Bidirectional RNNs

Example task: Named Entity Recognition (NER)

Given a sentence, label each word as:

- PER – person
- ORG – organization
- LOC – location
- O – none of these

Sentence 1

Gandhi led the freedom movement

Gandhi → PER

Sentence 2

Gandhi International Airport is very busy

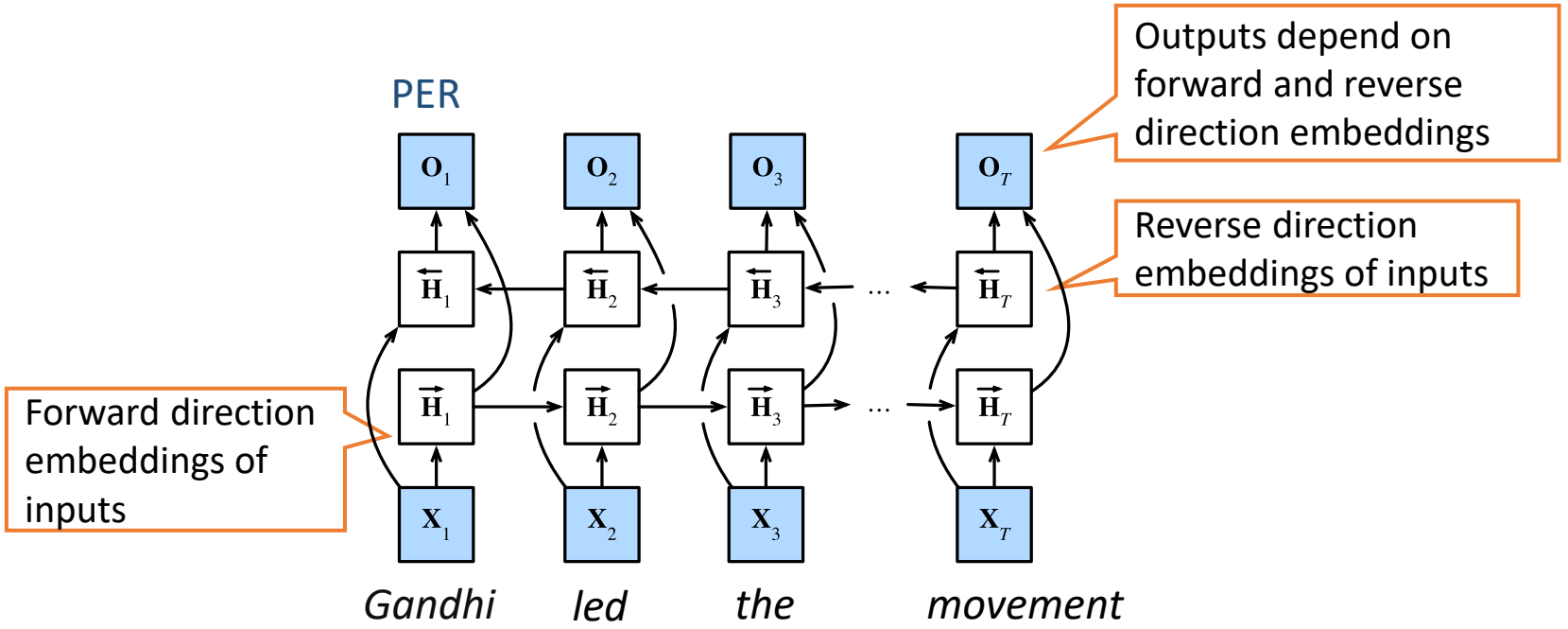
Gandhi *International Airport* → LOC

The correct meaning of the word **Gandhi** depends on the **surrounding words**.

To label words correctly, it helps to use both past and future context.

Bidirectional RNN

- RNNs only remember the information from the previous words
- Bidirectional RNNs can remember information from the past and future words

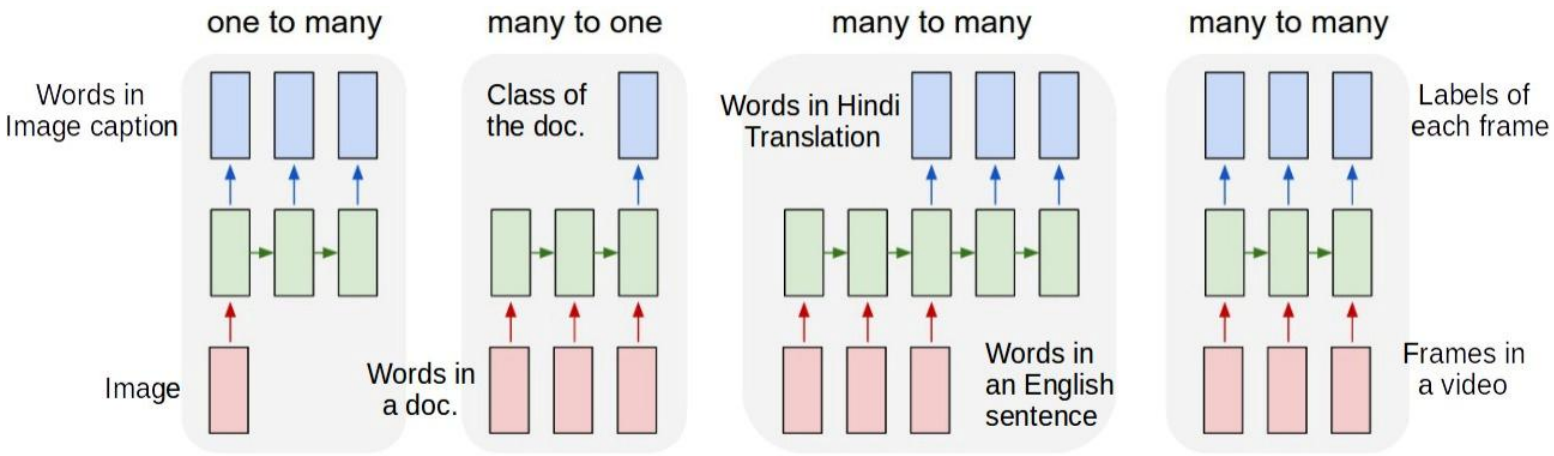


Because computing the reverse direction embedding needs future words, the full **output is available only after the whole sequence is seen**.

Pic source: https://d2l.ai/

Sequential Tasks: More Examples

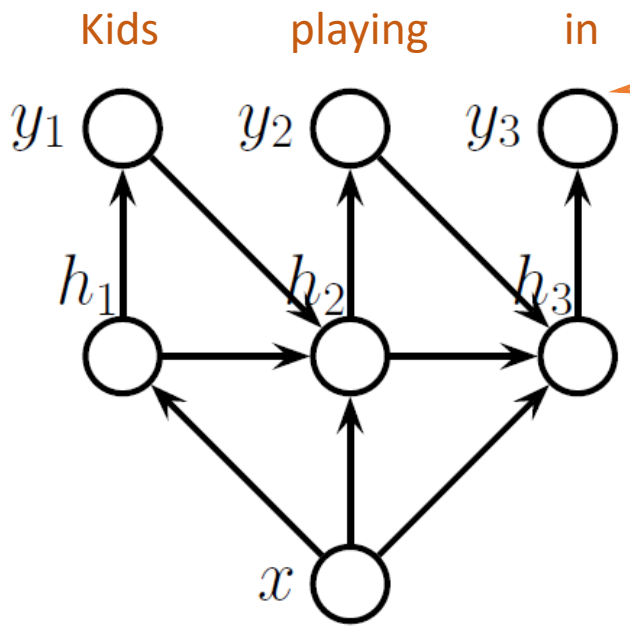
- Different inputs or outputs need not have the same length



- 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 Neural Networks: Some Examples

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



Predicted y_{t-1} also fed into h_t

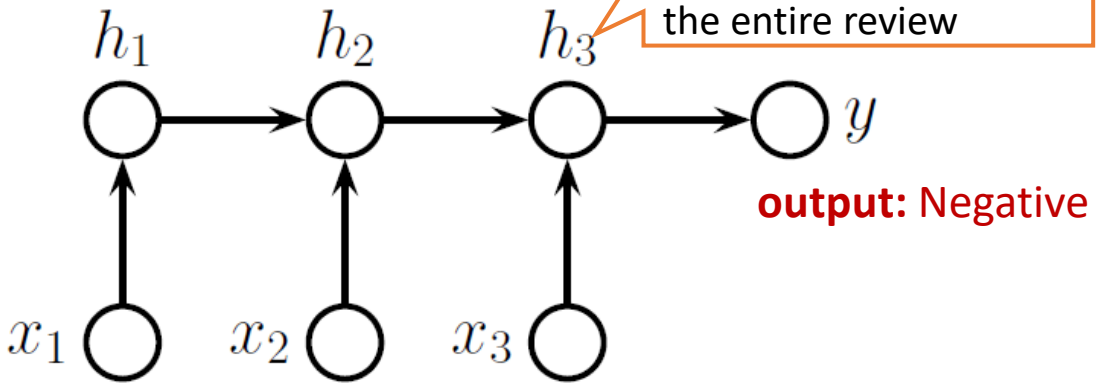
At test time, we can only feed the predicted y_{t-1}

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

Image Embeddings can be generated using CNNs

Recurrent Neural Networks: Some Examples

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?? ☺

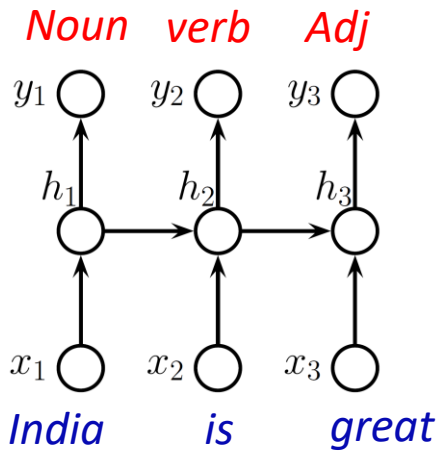
Indeed; this can be an issue with RNNs

output: Negative

Input: The movie started well but the ending was terrible

Recurrent Neural Networks: Some Examples

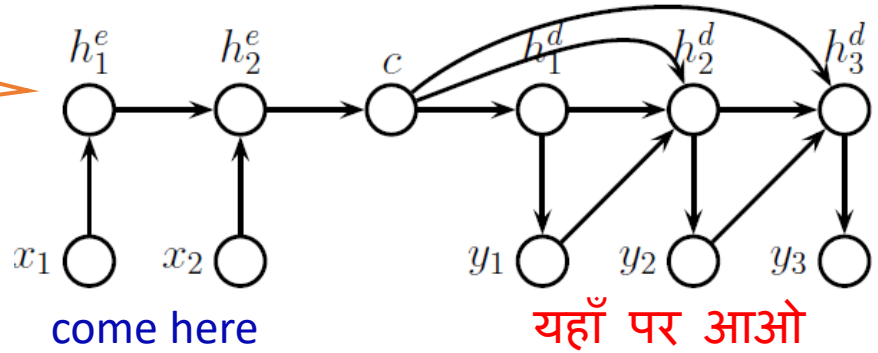
- Parts-of-Speech tagging (or "aligned" translation; input and output have same length)



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

Such problems usually require a sequence encoder- decoder architecture

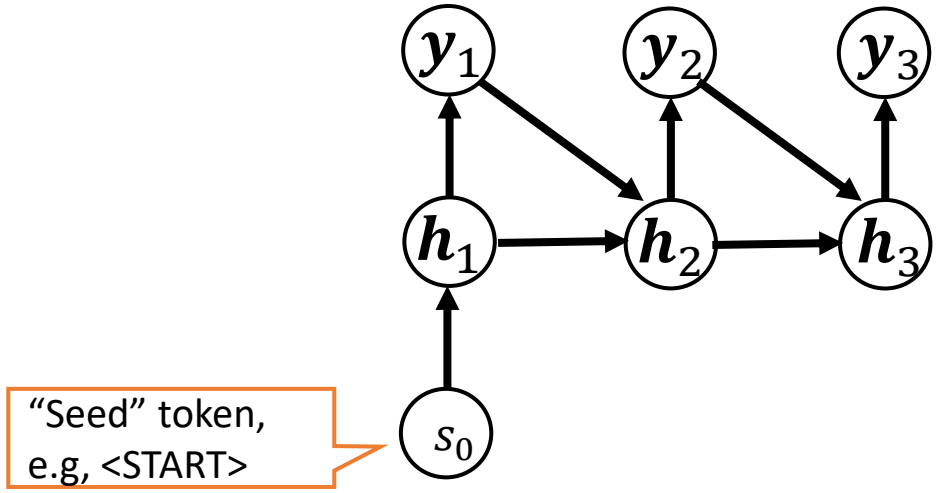
Encode the input sequence (embeddings of tokens) into a single embedding vector *c* and then decode this embedding one output token at a time



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

Recurrent Neural Networks: Some Examples

- Unconditional generation (no input, only an output sequence is generated given a RNN that was trained using some training data containing several sequences)

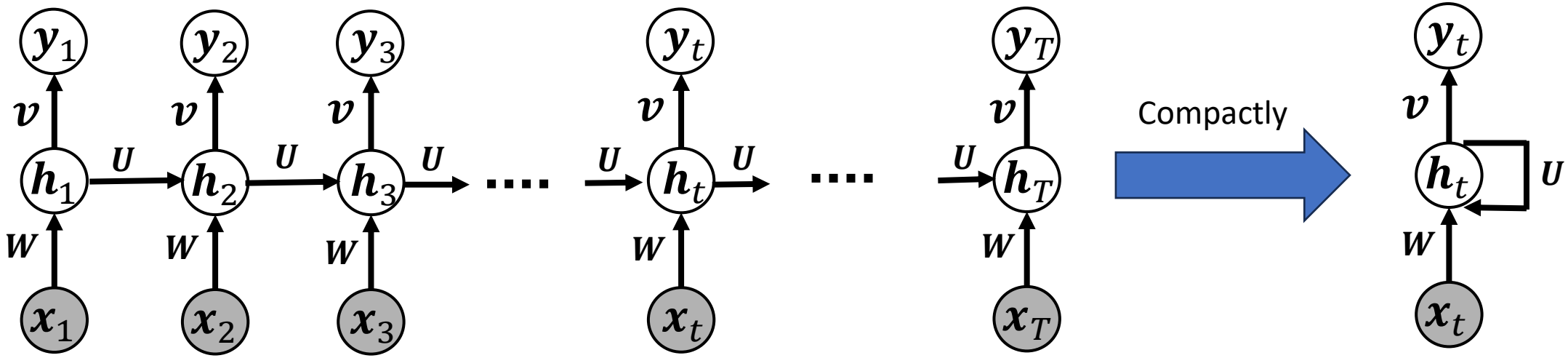


- Each generate word/token is fed to the next step's hidden state

- Generation stops when an "end" token (e.g., <END>) is generated

For RNNs, Long Distant Past is Hard to Remember

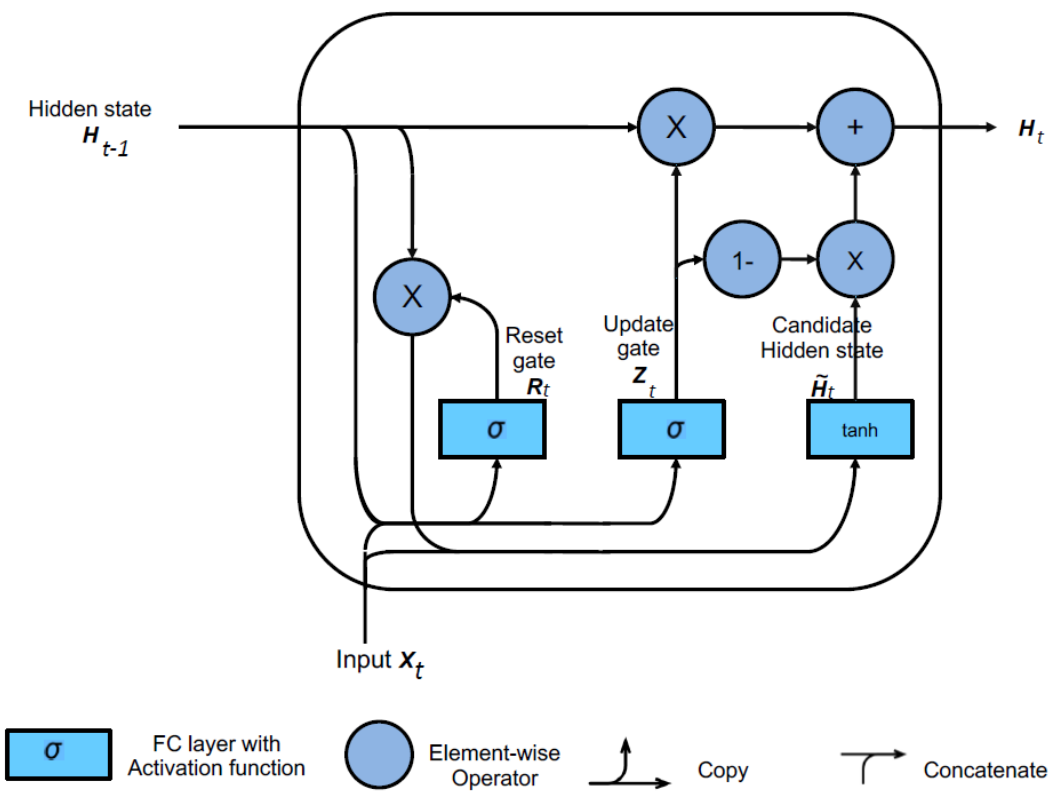
- The hidden layer nodes h_t are supposed to summarize the past up to time $t-1$

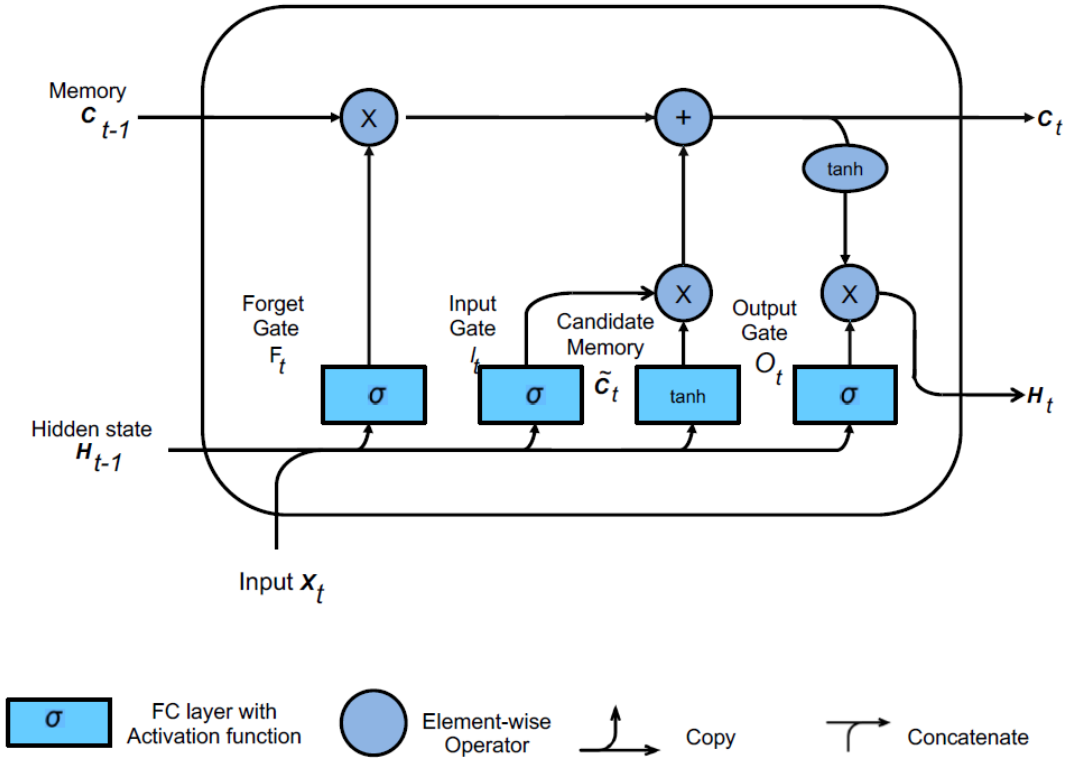


- In theory, they should. In practice, they can't. Some reasons
  - Vanishing gradients along the sequence too – past knowledge gets "diluted"
  - Hidden nodes also have limited capacity because of their finite dimensionality
- Various extensions of RNNs have been proposed to address forgetting
  - Gated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





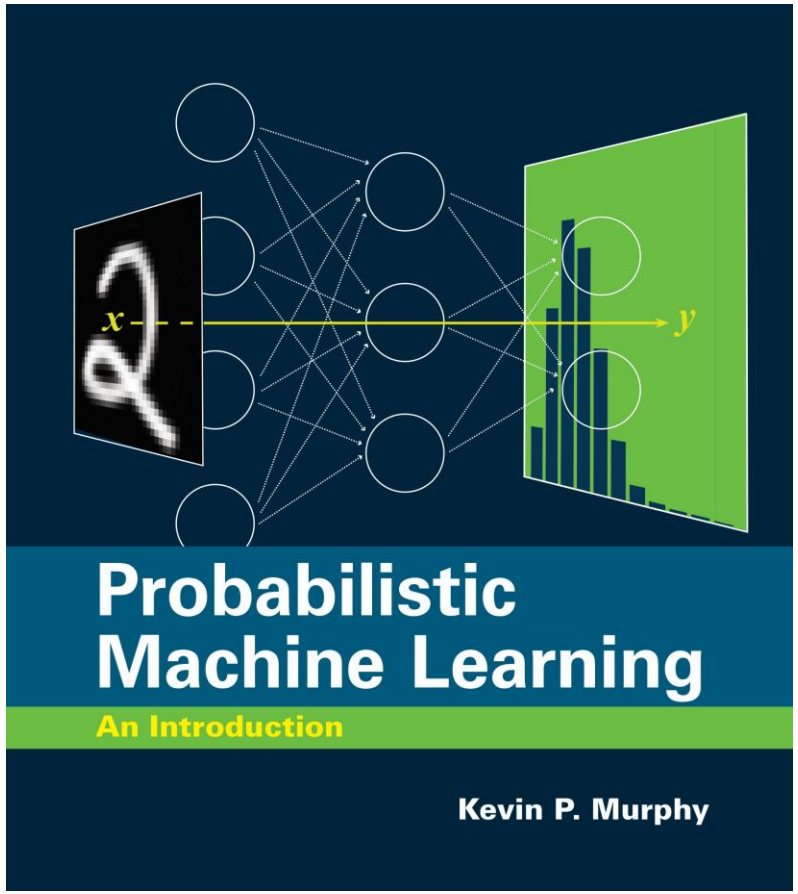
Pic source: https://d2l.ai/

Acknowledgments

- Slides are adapted from
  - Prof. Piyush Rai's course at IITK

References

Chapter 15



Thank you