Fundamentals of AI and ML

Lecture: Sequence modelling

by

Rajdip Nayek

Assistant Professor,
Applied Mechanics Department,
IIT Delhi

Instructor email: rajdipn@am.iitd.ac.in

Sequence vs Independent examples

- A normal machine learning data set is a collection of observations (**order of observations DO NOT matter**)

Input (say image)	Output (cat or dog)
Image #1	Cat
Image #2	Dog
…	…
Image #N	Cat

- A time series dataset is different. Time series adds an **explicit order dependence between observations**: **a time dimension**. This additional dimension is both a constraint and a structure that provides a source of additional information

Input (previous day SENSEX variation)	Output (next day SENSEX prediction at 11 am)
History#1 = $[\mathrm{Price}(t_1), \mathrm{Price}(t_2), \dots, \mathrm{Price}(t_M)]$	Price#1
History#1 = $[\mathrm{Price}(t_1), \mathrm{Price}(t_2), \dots, \mathrm{Price}(t_M)]$	Price#2
…	…
History#N = $[\mathrm{Price}(t_1), \mathrm{Price}(t_2), \dots, \mathrm{Price}(t_M)]$	Price#N

Time series (as a sequence) Nomenclature

- The current time is defined as t and an observation at current time is define as $observation$ at time $t \rightarrow obs(t)$

- We are often interested in the observations made at previous times, called **lag times** or just simply **lags**

- Times in the past are negative relative to the current time
  - Previous time $\rightarrow t - 1$ and observation at previous time $obs(t-1)$
  - Time before previous time $\rightarrow t - 2$ and observation as $obs(t-2)$

- Times in future are what we are interested in predicting and are positive relative to current time
  - Next time $\rightarrow t + 1$ and observation at next time $obs(t+1)$
  - Time after next time $\rightarrow t + 2$ and observation as $obs(t+2)$

`t-n`: A prior or lag time (e.g. `t-1` for the previous time).

`t`: A current time and point of reference.

`t+n`: A future or forecast time (e.g. `t+1` for the next time).

Concerns of time-series forecasting

1. **How much data do you have available and are you able to gather it all together**?

More data is often more helpful, offering greater opportunity for exploratory data analysis, model testing and tuning, and model accuracy

2. **What is the time horizon of predictions that is required?**

Short, medium or long term? Shorter time horizons are often easier to predict with higher confidence

3. **Can forecasts be updated frequently over time or must they be made once and remain static?**

Updating forecasts as new information becomes available often results in more accurate predictions

4. **At what temporal frequency are forecasts required?**

Often forecasts can be made at a lower or higher frequencies, allowing you to harness down-sampling, and up-sampling of data, which in turn can offer benefits while modeling

Concerns of time-series forecasting

Time series data often requires **cleaning**, **scaling**, and even **transformation.** For example:

- **Frequency:** Perhaps data is provided at a frequency that is too high to model or is unevenly spaced through time requiring resampling for use in some models
- **Outliers:** Perhaps there are corrupt or extreme outlier values that need to be identified and handled
- **Missing data**: Perhaps there are gaps or missing data that need to be interpolated or imputed

Examples of time-series forecasting

There is almost an endless supply of time series forecasting problems. Below are 10 examples from a range of industries:

- Forecasting crop yield (tons) by state each year
- Forecasting seizure detection (EEG trace in seconds: yes/no)
- Forecasting daily stock closing price
- Forecasting annual birth rate at city hospitals
- Forecasting daily product sales (units) for a store
- Forecasting daily train station passenger volume
- Forecasting quarterly state unemployment
- Forecasting hourly server utilization demand
- Forecasting state rabbit population (breeding season)
- Forecasting daily average gasoline price in a city

Time series forecasting as Supervised Learning

Time series forecasting can be framed as a supervised learning problem

- Input variables (X)
- Output variable (y)
- Learn the mapping function from the input to the output $y = f(X)$
- The goal is to approximate the real underlying mapping so well that when you have new input data (X), you can predict the output variables (y) for that data

```
X,  y
5,  0.9
4,  0.8
5,  1.0
3,  0.7
4,  0.9
```

Regression

Using Sliding window

Time series forecasting can be framed as a supervised learning problem

- Input variables (X)
- Output variable (y)
- We can restructure this time series dataset as a supervised learning problem by using the value at the previous time step to predict the value at the next **time-step**

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

Transform the dataset

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

Sliding window

Time series forecasting can be framed as a supervised learning problem

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

- The previous time step is the input (X) and the next time step is the output (y) in our supervised learning problem
- The order between the observations is preserved, and must continue to be preserved when using this dataset to train a supervised model
- We have no previous value that we can use to predict the first value in the sequence. We will delete this row as we cannot use it
- We can also see that we do not have a known next value to predict for the last value in the sequence. We may want to delete this value while training our supervised model also

The use of prior time steps to predict the next time step is called **the sliding window method**

Sliding window for multivariate time series

The number of observations recorded at a given time in a time series dataset matters

- **Univariate Time Series**: These are datasets where only a single variable is observed at each time

```
time,   measure
1,      100
2,      110
3,      108
4,      115
5,      120
```

- **Multivariate Time Series**: These are datasets where two or more variables are observed at each time

```
time,   measure1,   measure2
1,      0.2,        88
2,      0.5,        89
3,      0.7,        87
4,      0.4,        88
5,      1.0,        90
```

Sliding window for multivariate time series

- Multivariate Time Series: These are datasets where two or more variables are observed at each time

time,	measure1,	measure2
1,	0.2,	88
2,	0.5,	89
3,	0.7,	87
4,	0.4,	88
5,	1.0,	90

- We can reframe it as a supervised ML problem with **sliding window of 1**

Use measure 1 to predict measure 2

X1,	X2,	X3,	y
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

Predict both measure 1 and measure 2 for next time step

X1,	X2,	y1,	y2
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

Sliding window for multiple time steps

- The number of time steps ahead to be forecasted is important!
  - **One-step Forecast**: This is where the next time step $(t + 1)$ is predicted
  - **Multi-step Forecast**: This is where two or more future time steps are to be predicted

```
time, measure
1,    100
2,    110
3,    108
4,    115
5,    120
```

One-step forecast

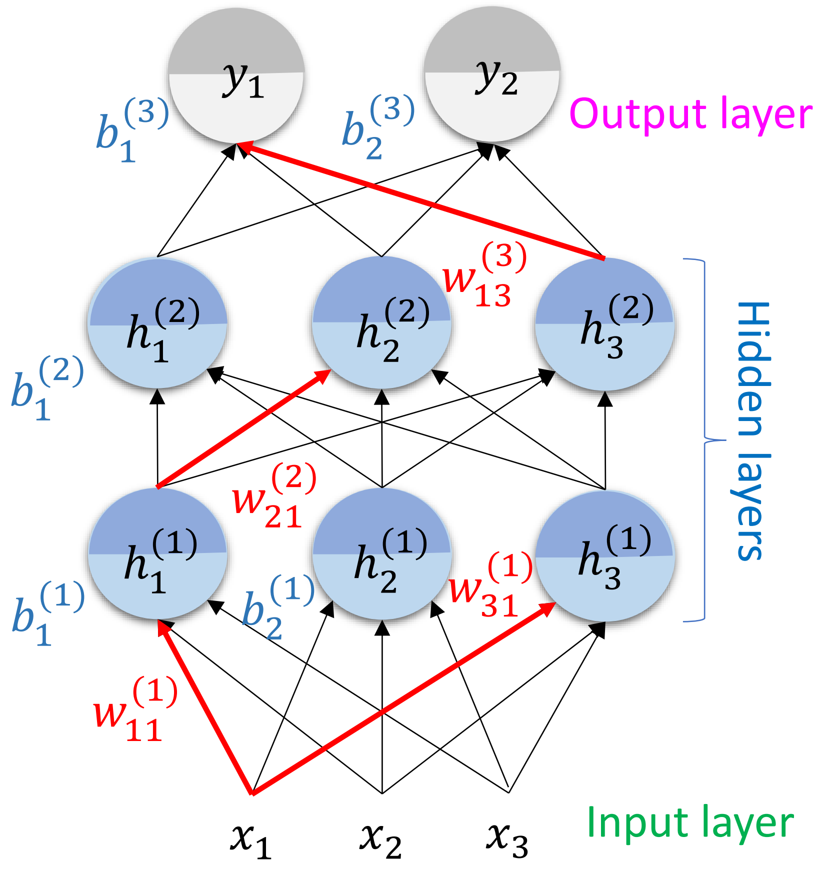
```
X,    y
?,    100
100,  110
110,  108
108,  115
115,  120
120,  ?
```

Two-step forecast

```
X1,   y1,   y2
?     100,  110
100,  110,  108
110,  108,  115
108,  115,  120
115,  120,  ?
120,  ?,    ?
```

ANN for time-series forecasting

- ANNs approximate a mapping function from input variables to output variables
  - **Mapping can be nonlinear**
  - **Multivariate inputs are supported**
  - **Multivariate outputs are supported**
  - **Multi-step forecasting** can also be done using **sliding windows**

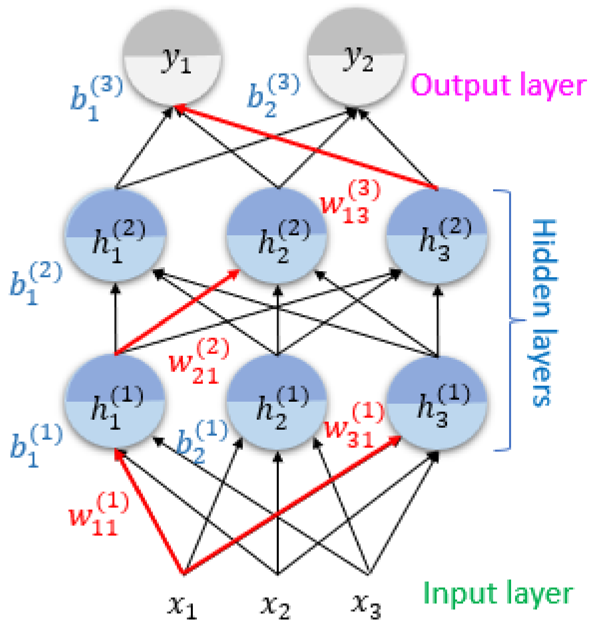


- Limitations for sequence modelling
  - **Fixed Inputs**: The number of input variables is automatically fixed by the number of lags (or lag times)
  - **Fixed Outputs**: The number of output variables is also fixed

Feedforward neural networks do offer great capability but still suffer from this key limitation **of having to specify the temporal dependence upfront** in the design of the model

ANN for time-series forecasting

- ANNs approximate a mapping function from input variables to output variables
  - **Univariate** ANN Models
  - **Multivariate** ANN Models
  - **Multi-Step** ANN Models
  - **Multivariate Multi-Step** ANN Models



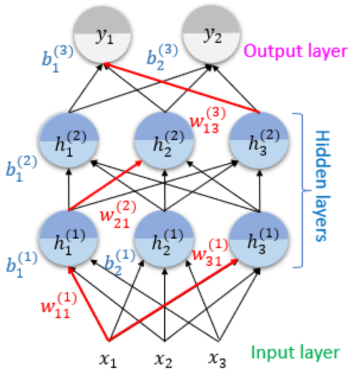
- **Univariate** time-series for fitting ANN model
  - Choose a lag (here lags = 3)

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

```
[10 20 30]  40
[20 30 40]  50
[30 40 50]  60
[40 50 60]  70
[50 60 70]  80
[60 70 80]  90
```

ANN for time-series forecasting

- ANNs approximate a mapping function from input variables to output variables
  - **Univariate** ANN Models
  - **Multivariate** ANN Models
  - **Multi-Step** ANN Models
  - **Multivariate Multi-Step** ANN Models



- **Multivariate** time-series for fitting ANN model
  - Multiple input time-series (choose lags = 3)

```
       X1   X2   y
t1   [[ 10  15  25]
t2    [ 20  25  45]
      [ 30  35  65]
      [ 40  45  85]
      [ 50  55 105]
      [ 60  65 125]
      [ 70  75 145]
      [ 80  85 165]
t9    [ 90  95 185]]
```

```
Input      Output
[[10 15]
 [20 25]
 [30 35]]  65
[[20 25]
 [30 35]
 [40 45]]  85
[[30 35]
 [40 45]
 [50 55]]  105
[[40 45]
 [50 55]
 [60 65]]  125
```

Input is a matrix!!

ANNs can only take input as vector, so *flatten* it

```
[10, 15, 20, 25, 30, 35]
```

ANN for time-series forecasting

- ANNs approximate a mapping function from input variables to output variables
  - **Univariate** ANN Models
  - **Multivariate** ANN Models
  - **Multi-Step** ANN Models
  - **Multivariate Multi-Step** ANN Models

- **Multi-Step** time-series for fitting ANN model
  - Showing an example with univariate time-series
  - Predict for **two (or more time)** steps

[10, 20, 30, 40, 50, 60, 70, 80, 90] ➡

Input	*Output*
[10 20 30]	[40 50]
[20 30 40]	[50 60]
[30 40 50]	[60 70]
[40 50 60]	[70 80]
[50 60 70]	[80 90]

ANN for time-series forecasting

- ANNs approximate a mapping function from input variables to output variables
  - **Univariate** ANN Models
  - **Multivariate** ANN Models
  - **Multi-Step** ANN Models
  - **Multivariate Multi-Step** ANN Models

- **Multivariate Multi-Step** time-series for fitting ANN model
  - Multivariate input and multi-step output

	X_1	X_2	y
t_1	[[10	15	25]
t_2	[20	25	45]
	[30	35	65]
	[40	45	85]
	[50	55	105]
	[60	65	125]
	[70	75	145]
	[80	85	165]
t_9	[90	95	185]]

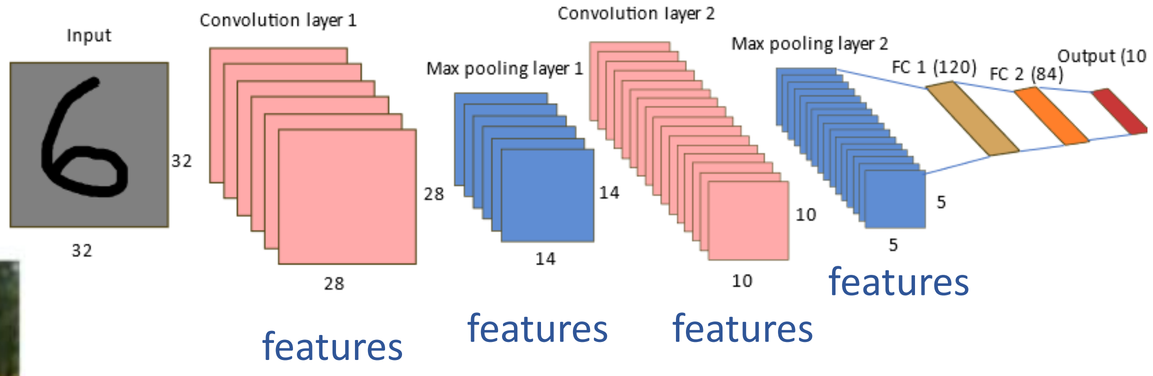
→

Input	*Output*
[[10 15] [20 25] [30 35]]	[65 85]
[[20 25] [30 35] [40 45]]	[85 105]
[[30 35] [40 45] [50 55]]	[105 125]

CNN for time-series forecasting

- CNNs were designed to efficiently handle image data (very effective for computer vision problems)

Learns to automatically extract features from the image data



  - **Image classification**
  - **Object localization**

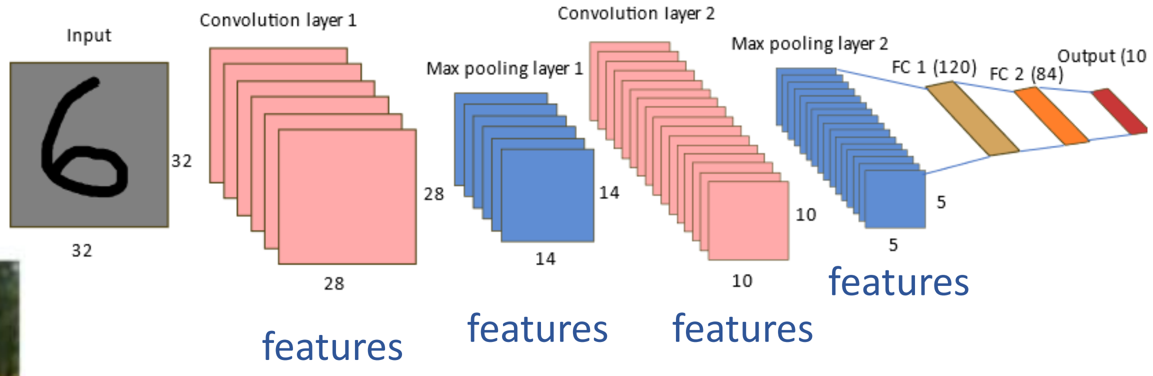
- **Image captioning:** Can CNN do this?
  - We will see later towards the end of this lecture

CNN for time-series forecasting

- CNNs were designed to efficiently handle image data (very effective for computer vision problems)

Learns to automatically extract features from the image data

  - **Image classification**
  - **Object localization**



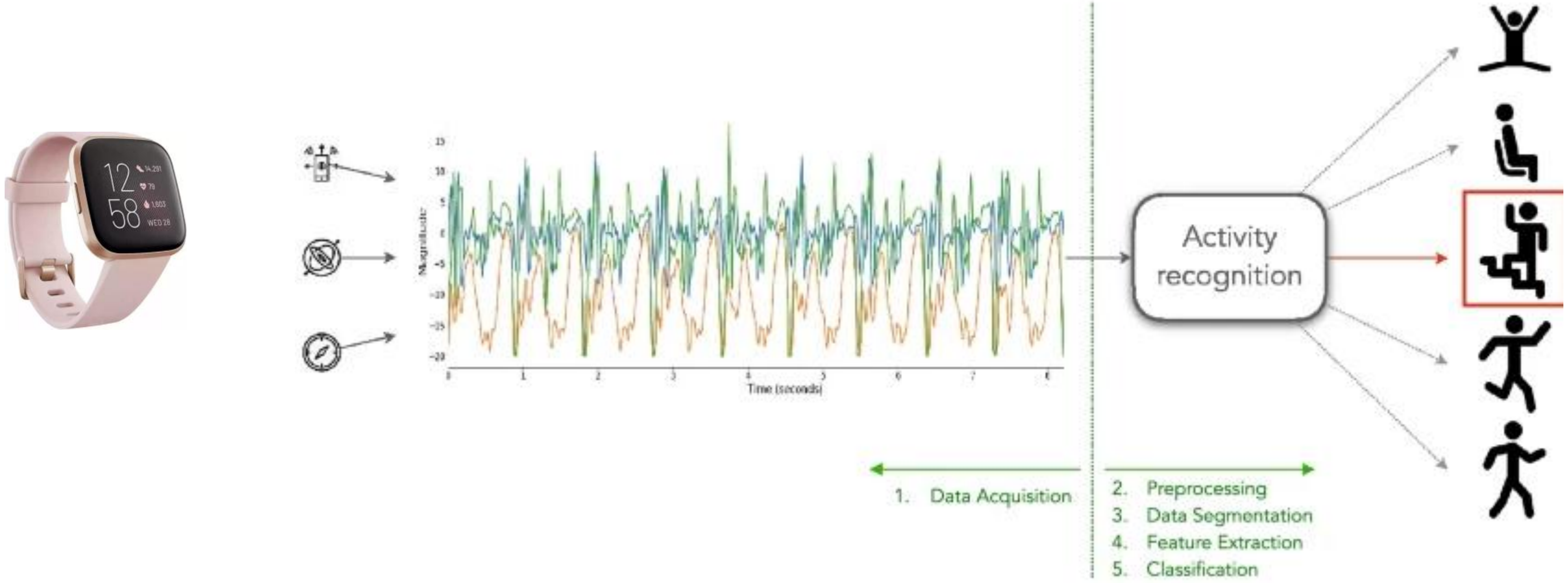
- CNN can learn and automatically extract features from raw time-series data which can **serve as input to time-series forecasting problems**
  - A sequence of observations can be treated like a one-dimensional image that a CNN can read and distill features

CNN for time-series forecasting

- CNNs were designed to efficiently handle image data (very effective for computer vision problems)

  Learns to automatically extract features from the image data

- CNN can learn and automatically extract features from raw time-series data which can serve as input to time-series forecasting problems
  - A sequence of observations can be treated like a **one-dimensional image** that a CNN can read and distill features
  - Thus, CNN can be used for **time-series classification**, e.g. automatically detecting human activities based on raw acceleration sensor data from fitness devices and smartphones



CNN for time-series forecasting

- CNNs were designed to efficiently handle image data (very effective for computer vision problems)

  Learns to automatically extract features from the image data

- CNN can learn and automatically extract features from raw time-series data which can serve as input to time-series forecasting problems
  - A sequence of observations can be treated like a one-dimensional image that a CNN can read and distill features
  - Thus, CNN can be used for **time-series classification**, e.g. automatically detecting human activities based on raw acceleration sensor data from fitness devices and smartphones

- **Feature Learning**: Automatic identification, extraction of salient features from raw input data can be done using CNN, but can a CNN do time-series forecasting:
  - Yes, they can do better than ANNs but are still limited in capabilities as the inputs have to be manually designed

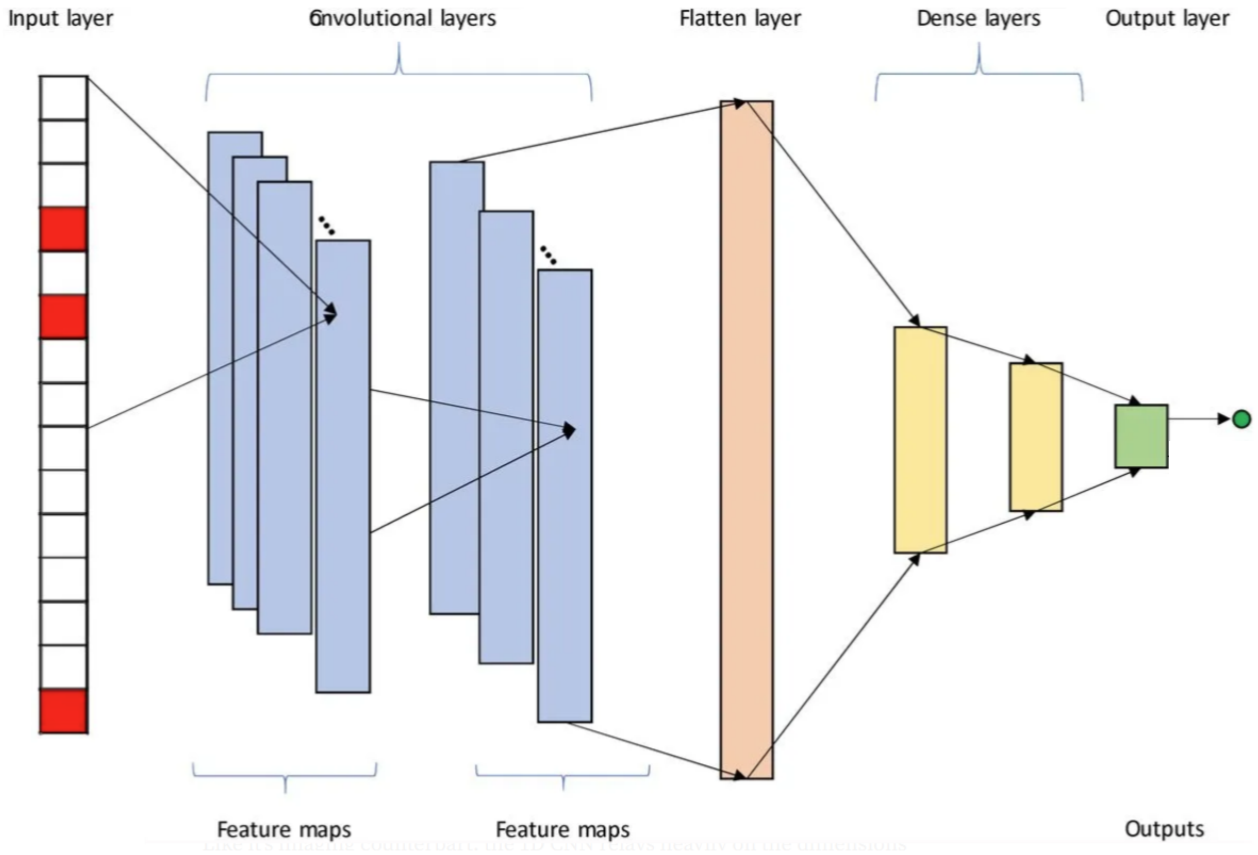
CNN for time-series forecasting

- **Univariate** time-series for fitting ANN model
  - Choose a lag (here lags = 3)

[10, 20, 30, 40, 50, 60, 70, 80, 90]

↓

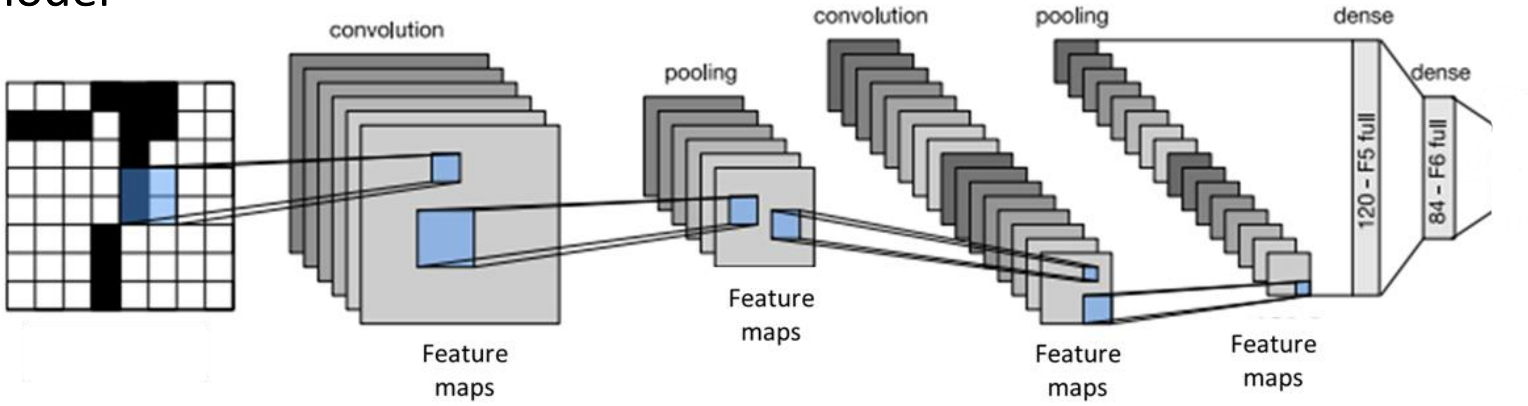
```
[10 20 30] 40
[20 30 40] 50
[30 40 50] 60
[40 50 60] 70
[50 60 70] 80
[60 70 80] 90
```



Use 1D CNN

CNN for time-series forecasting

- **Multivariate** time-series for fitting CNN model
  - Choose a lag (here lags = 3)



	X_1	X_2	y
t_1	[[10	15	25]
t_2	[20	25	45]
	[30	35	65]
	[40	45	85]
	[50	55	105]
	[60	65	125]
	[70	75	145]
	[80	85	165]
t_9	[90	95	185]]

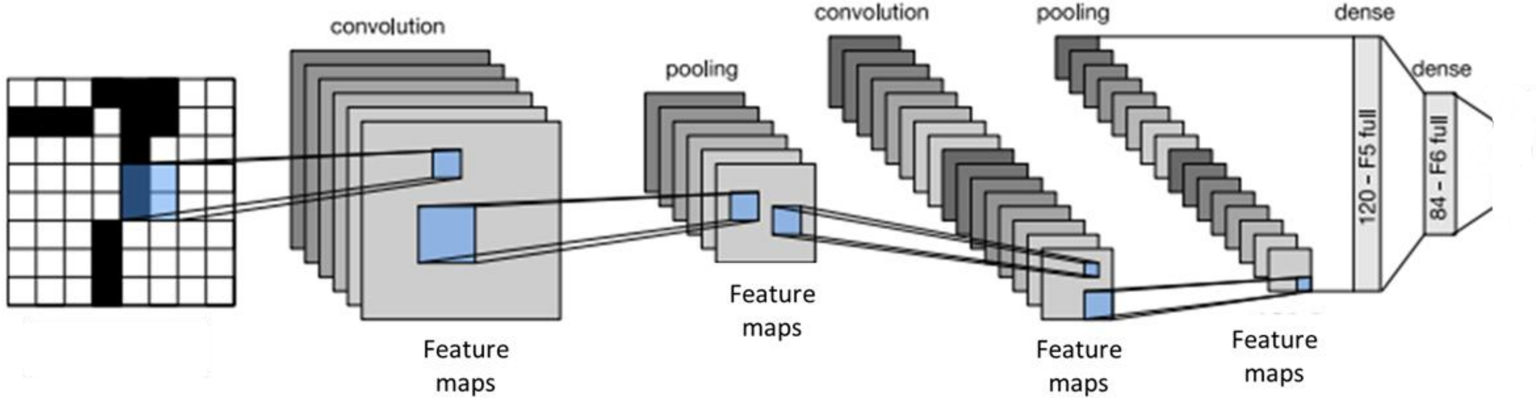
→

Input	*Output*
[[10 15] [20 25] [30 35]]	65
[[20 25] [30 35] [40 45]]	85
[[30 35] [40 45] [50 55]]	105
[[40 45] [50 55] [60 65]]	125

Use 2D CNN with 1 output and 2x3 matrix as input

CNN for time-series forecasting

- **Multivariate Multi-Step** time-series for fitting CNN model
  - Multivariate input and multi-step output



$$\begin{array}{ll} & \quad X_1 \quad X_2 \quad y \\ t_1 & \\ t_2 & \\ t_9 & \end{array}$$

```
t1  [[ 10  15  25]
t2   [ 20  25  45]
     [ 30  35  65]
     [ 40  45  85]
     [ 50  55 105]
     [ 60  65 125]
     [ 70  75 145]
     [ 80  85 165]
t9   [ 90  95 185]]
```

Input / *Output*

```
[[10 15]
 [20 25]
 [30 35]] [65 85]
[[20 25]
 [30 35]
 [40 45]] [ 85 105]
[[30 35]
 [40 45]
 [50 55]] [105 125]
```

Use 2D CNN with 2 outputs and 2x3 matrix as input

Recurrent Neural Networks for Sequence modelling

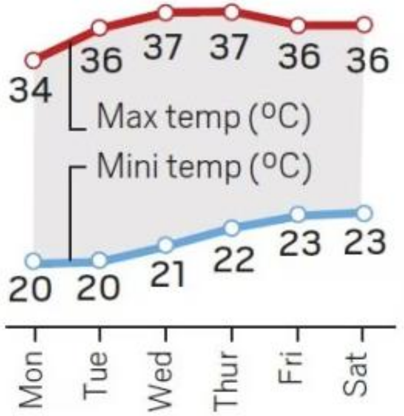
- RNNs add a way of explicit handling of ordered relationship between observations when learning to map from input to outputs, which is not offered by ANN or CNN
  - **Inputs**: Historical data provided to the model to make a single prediction
  - **Outputs**: Prediction for a future time step beyond the data provided as input
- Defining the inputs and outputs forces you to think about what exactly is required to make a prediction
- RNNs have **native support for inputs to be sequences**
- What kind of problems are RNNs used for?
  - Sequence Prediction
  - Sequence Classification
  - Sequence Generation
  - Sequence-to-Sequence Prediction

Recurrent Neural Networks for Sequence Prediction

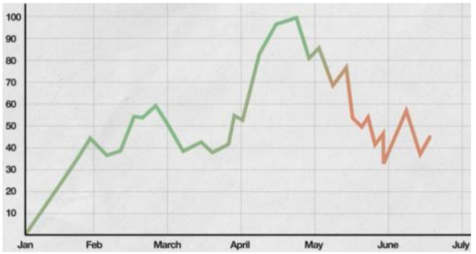
Sequence prediction involves predicting the next value for a given input sequence

[1, 2, 3, 4, 5] → Sequence Prediction Model → [6]

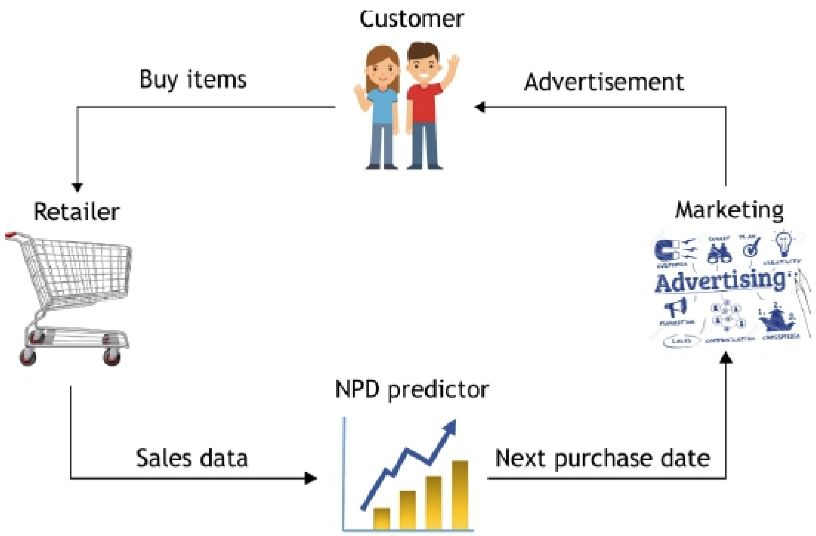
- **Weather Forecasting**: Given a sequence of observations about the weather over time, predict the expected weather tomorrow



- **Stock Market Prediction**: Given a sequence of movements of a security over time, predict the next movement of the security



- **Product Recommendation**: Given a sequence of past purchases for a customer, predict the next purchase for a customer

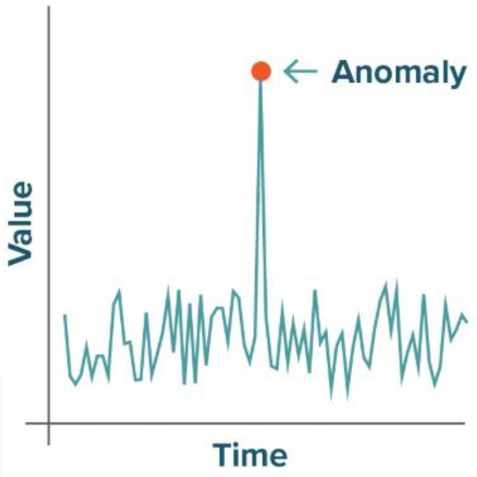


Recurrent Neural Networks for Sequence Classification

Sequence classification involves predicting a class label for a given input sequence

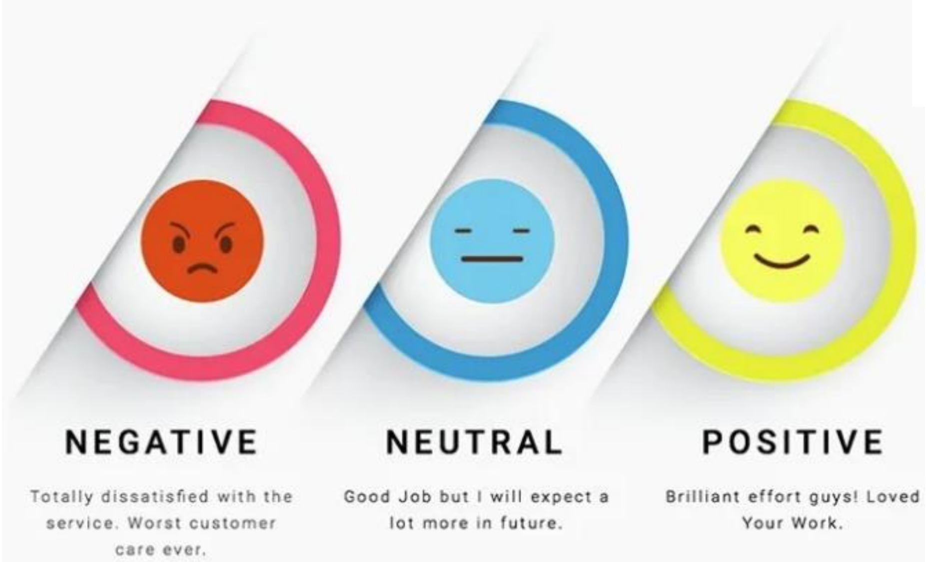
[1, 2, 3, 4, 5] → Sequence Prediction Model → [“good”]

- **DNA Sequence Classification:** Given a DNA sequence of A, C, G, and T values, predict whether the sequence is for a coding or non-coding region



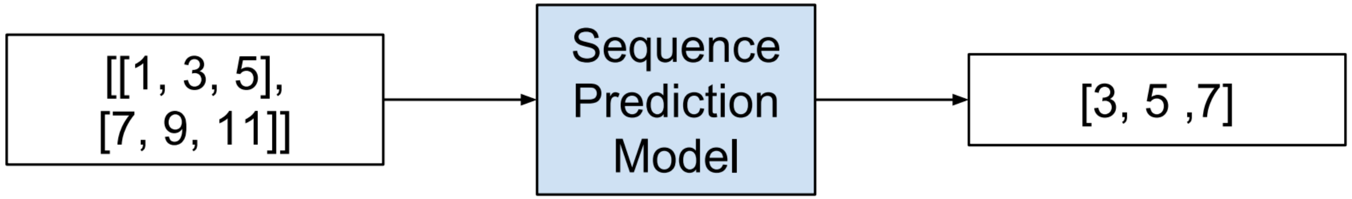
- **Anomaly Detection**: Given a sequence of observations, predict whether the sequence is anomalous or not

- **Sentiment Analysis:** Given a sequence of text such as a review or a tweet, predict whether the sentiment of the text is positive or negative



Recurrent Neural Networks for Sequence Generation

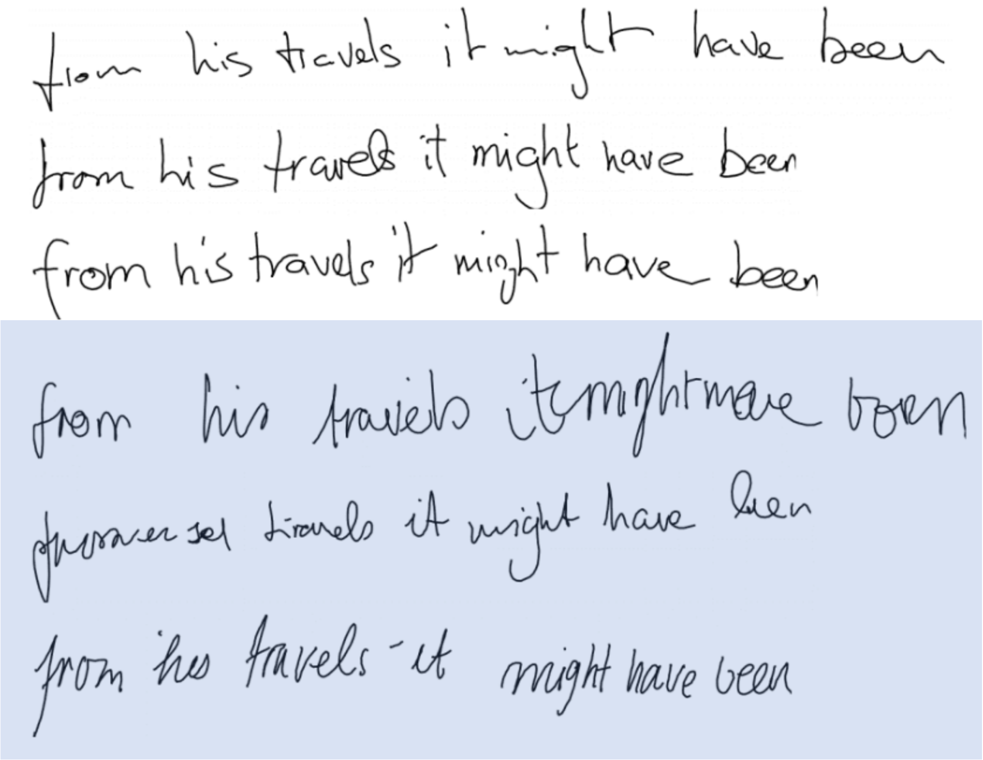
Sequence generation involves generating a new output sequence that has the same general characteristics as other sequences in the corpus



- **Text Generation**: Given a corpus of text, such as the works of Shakespeare, generate new sentences or paragraphs of text that read they could have been drawn from the corpus

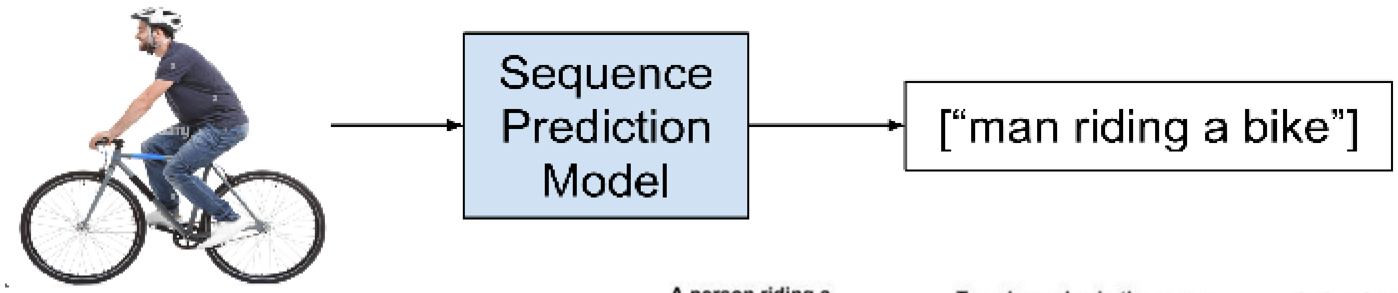
- **Handwriting Prediction**: Given a corpus of handwriting examples, generate handwriting for new phrases that has the properties of handwriting in the corpus

- **Music Generation**. Given a corpus of examples of music, generate new musical pieces that have the properties of the corpus



Recurrent Neural Networks for Sequence Generation

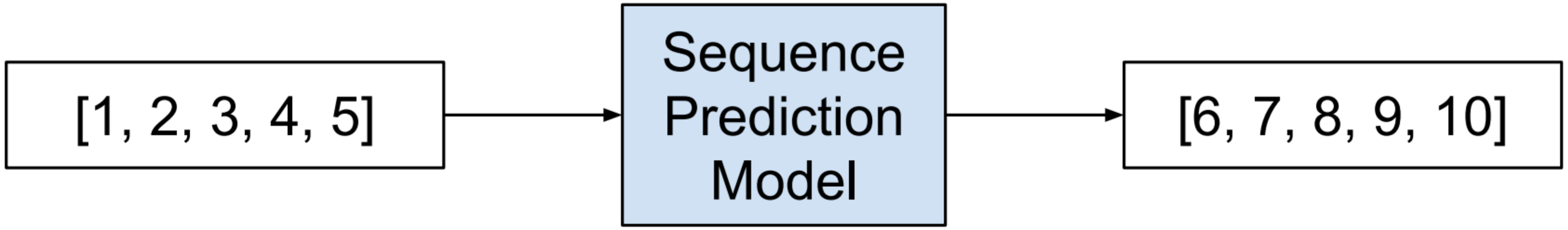
Sequence generation involves generating a new output sequence that has the same general characteristics as other sequences in the corpus



- **Image Caption Generation**: Given an image as input, generate a sequence of words that describe an image
- Here the words are first generated as numbers or numeric vectors, which are then converted to words
- Image caption generation typically also uses a CNN model with RNN

Recurrent Neural Networks for Sequence-to-sequence prediction

Sequence-to-sequence prediction involves predicting an output sequence given an input sequence

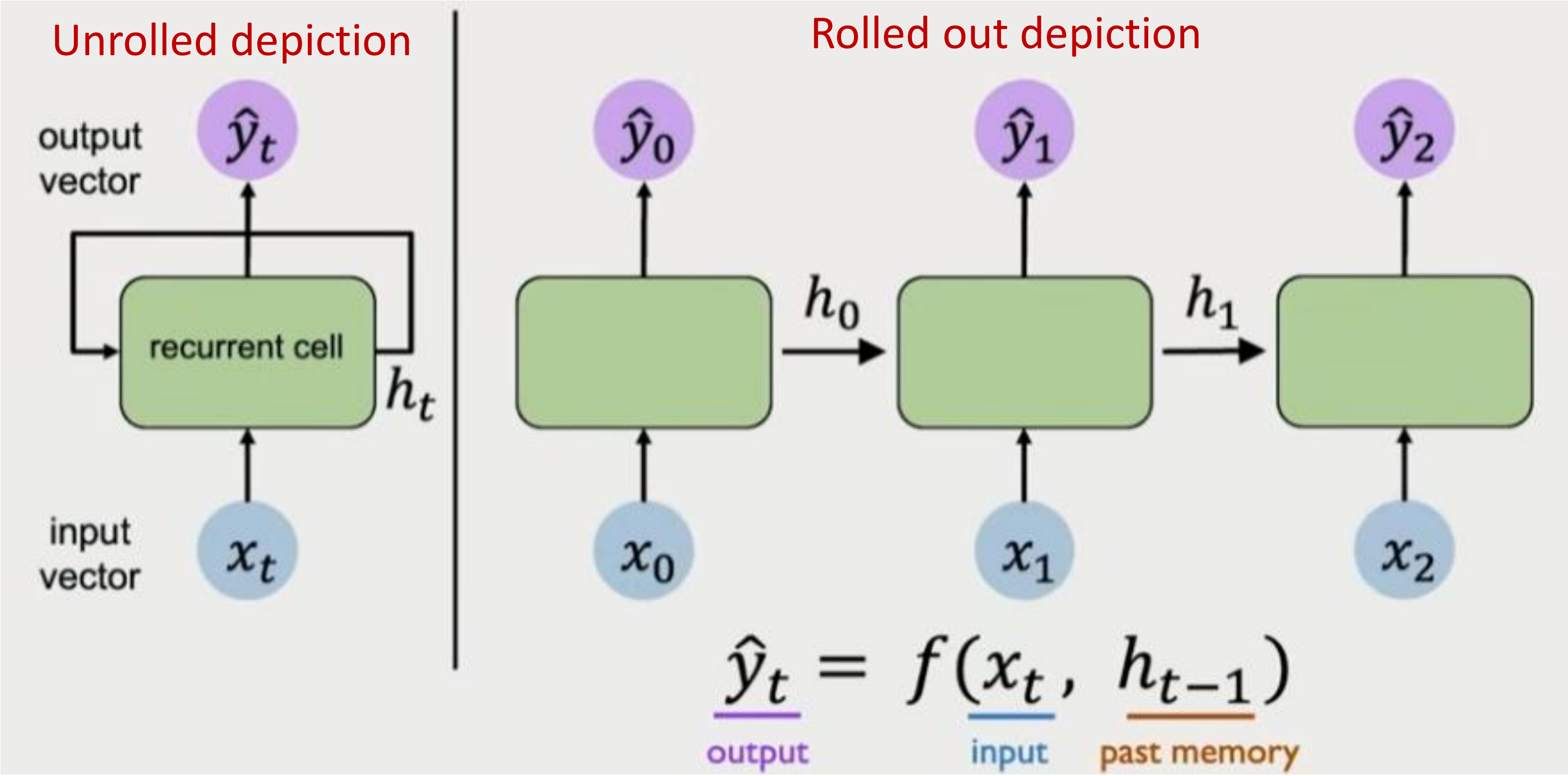


- **Multi-Step Time Series Forecasting:** Given a time-series of observations, predict a sequence of observations for a range of future time steps

- **Text Summarization:** Given a document of text, predict a shorter sequence of text that describes the salient parts of the source document (also a part of Natural Language Processing (NLP))

Recurrent Neural Networks (RNN)

- **Model structure of RNN**



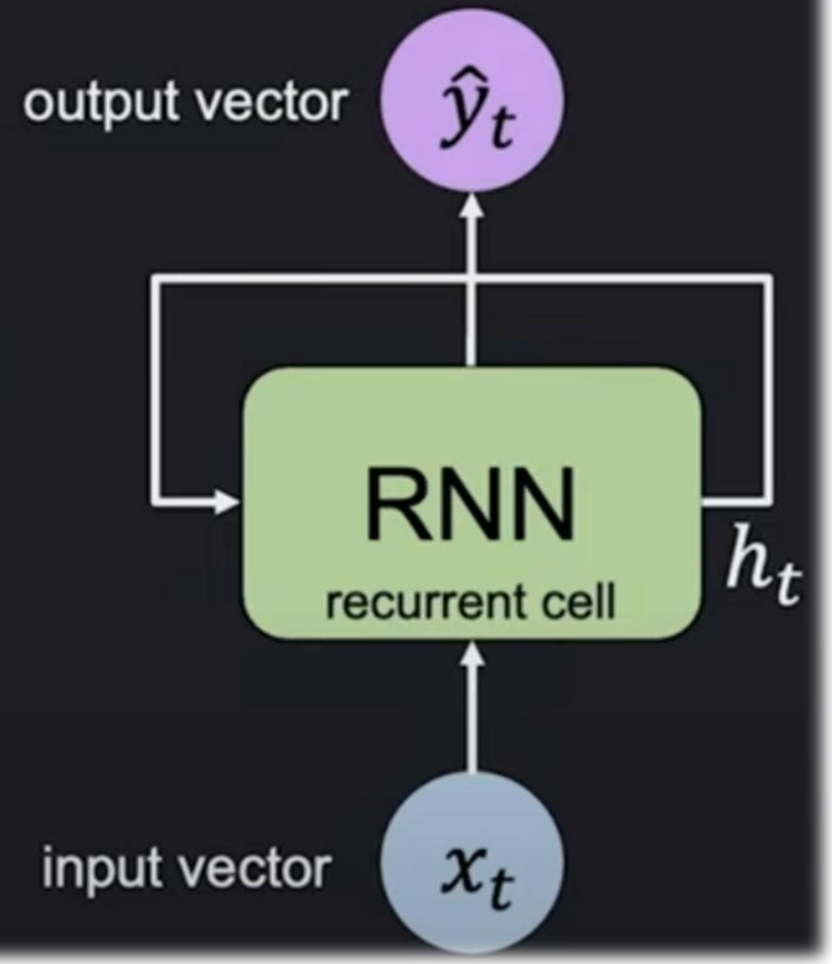
h_t: Also called hidden cell states

RNNs use the concept of **hidden states** to store information from past

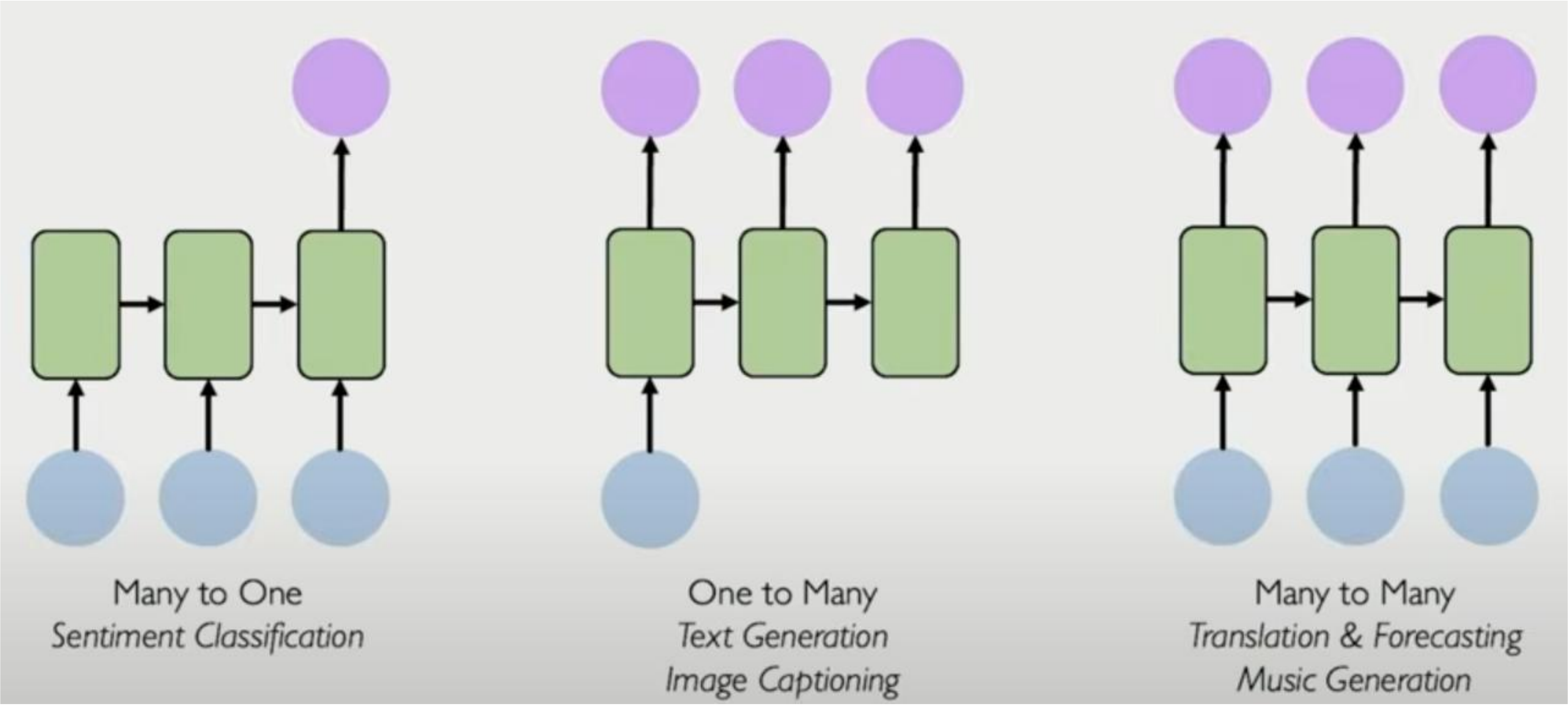
Recurrent Neural Networks (RNN) Intuition

Pseudo code

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



Structure of RNNs for different tasks



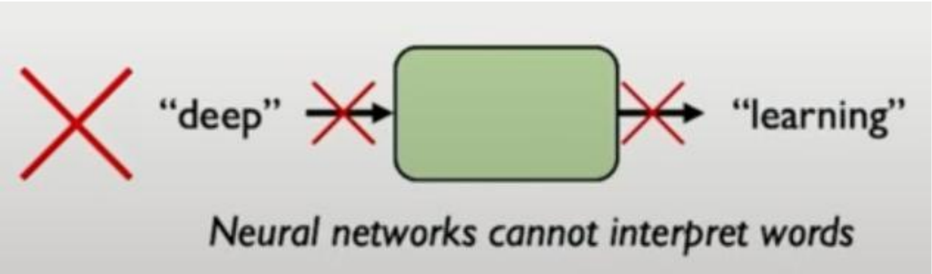
RNNs for text prediction

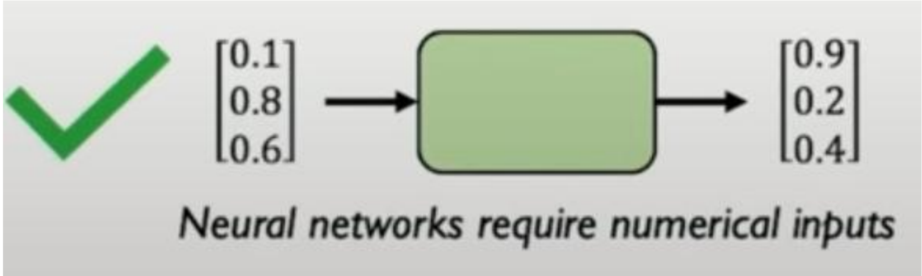
Let's try to predict the next word:

"This morning I took my cat for a walk."

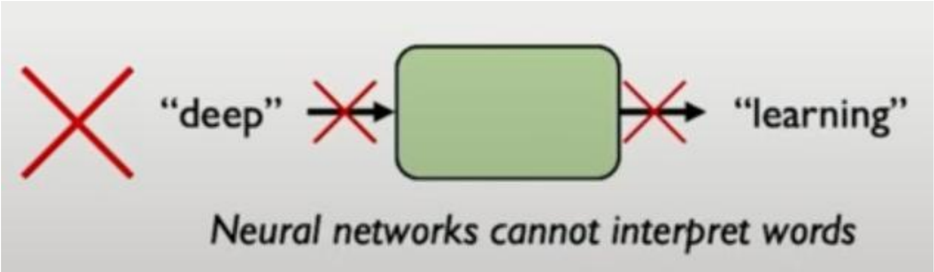
given these words **predict the next word**

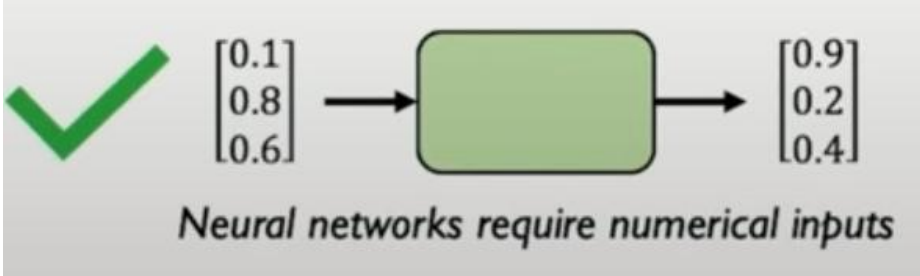
Representing language for a neural network





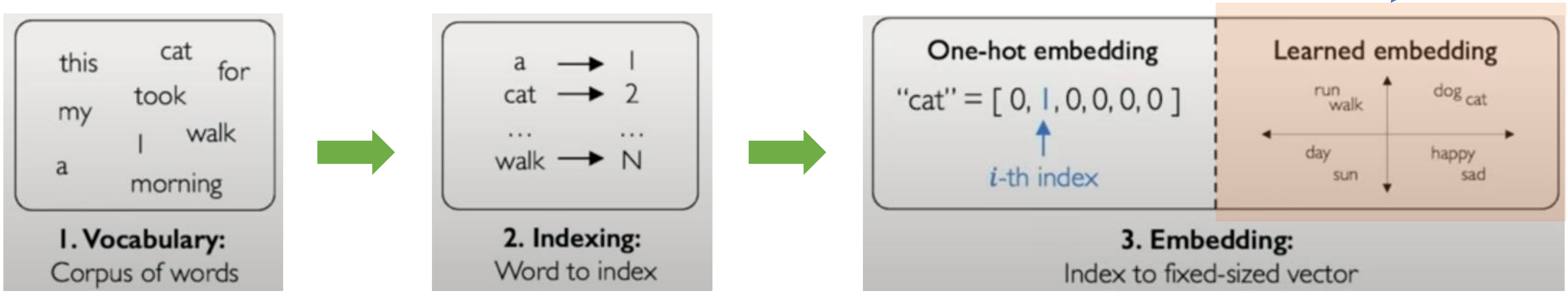
Encoding language for a neural network





Word embedding using one-hot encoding: Transform words to vectors of fixed size

better embedding



Advantages of RNNs

1. Handling variable length sequence data is much easier with RNNs

- A short sentence

The food was great.”

- Medium-long sentence

We visited a restaurant for lunch.”

- Long sentence

We were hungry but we cleaned the house before eating.”

In all the RNN models, we want to be able to give variable length sequence and still be able to get good predictions

Advantages of RNNs

2. It can model long-term dependencies well

"**Kolkata** is where I grew up, but I now live in Delhi. I speak fluent ________ ."

We need to information from **distant past** to be able to understand the context and then accurately predict the correct word

3. Can capture difference in sequence order

"The food was good, not bad at all."

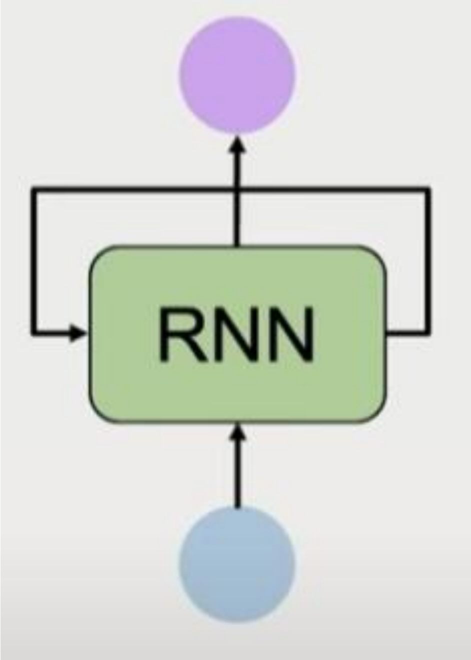
vs

"The food was bad, not good at all."

Summary of requirements for sequence modelling

To model sequences, a neural network must be able to

1. Handle **variable-length** sequences
2. Track **long-term dependencies**
3. Maintain information about **order**
4. **Share parameters** across the sequence

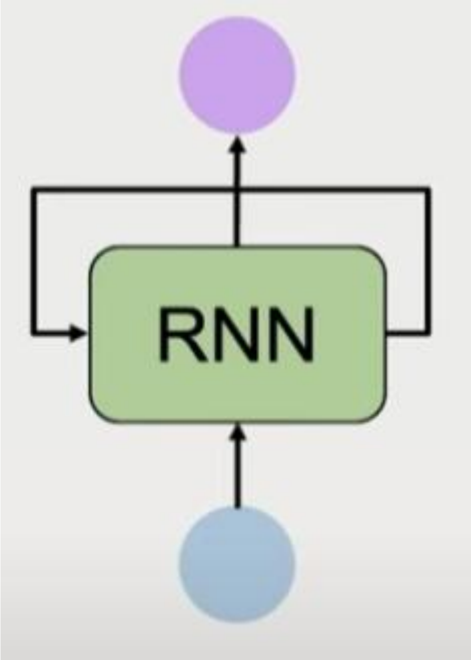


Recurrent Neural Networks (RNNs) meet these sequence modeling **design criteria**

Summary of requirements for sequence modelling

To model sequences, a neural network must be able to

1. Handle **variable-length** sequences
2. Track **long-term dependencies**
3. Maintain information about **order**
4. **Share parameters** across the sequence



Recurrent Neural Networks (RNNs) meet these sequence modeling **design criteria**

However, training (optimizing the weights and biases) of RNNs causes a problem of **exploding and vanishing** gradients

- To prevent the issue, certain **information gates** are introduced

Gating mechanisms to improve the issues of RNN training

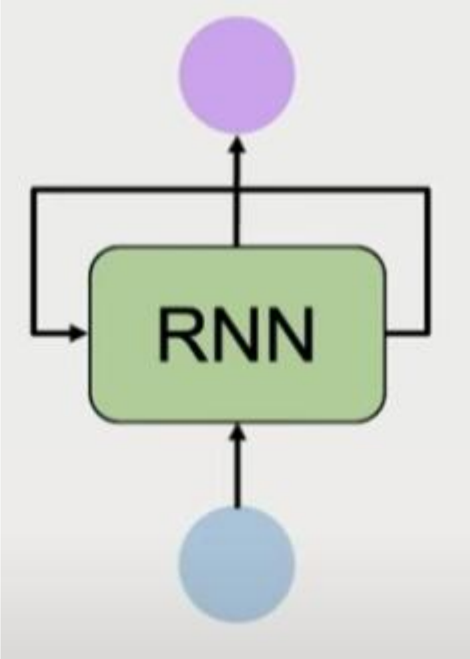
- **Idea**: Use gates to selectively control (add or remove) information within each recurrent unit to be passed on the next unit

- There are three basic types of gates:
  - ***Forget Gate***: Decides what information to discard from the previous cell hidden state
  - ***Input Gate***: Decides which values from the input to update the memory state
  - ***Output Gate***: Decides what to output based on input and the memory of the cell

- The forget gate and input gate are used in the updating of the hidden cell state

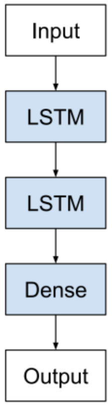
- The output gate decides what the cell actually outputs

- With different gating mechanisms, have two prominent and widely used RNNs
  - Long-short-term-memory (LSTM)
  - Gated-recurrent-units (GRU)



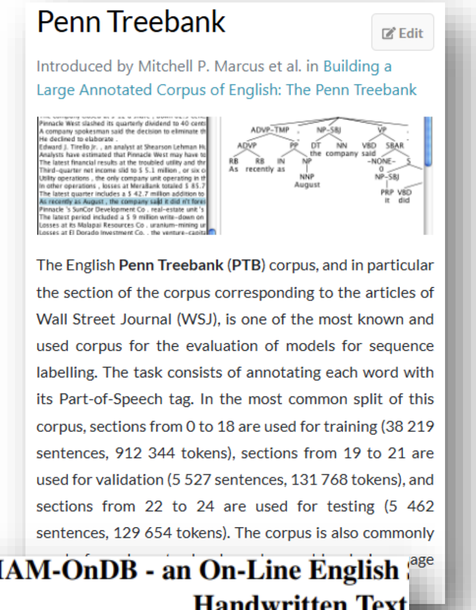
Stacked LSTMs

Stacked LSTMs were introduced in 2013 in an application to speech recognition, beating a benchmark on a challenging standard problem



1. **Penn Treebank Experiments**

- **Task:** Predict the next word or character in text from the Penn Treebank dataset
- **Approach:** Compared word-level and character-level LSTM predictors
- **Key Evaluation Metrics:** Perplexity (how surprised the model is on encountering new data)
- **Outcome:** Demonstrated competitive language modeling ability



Penn Treebank

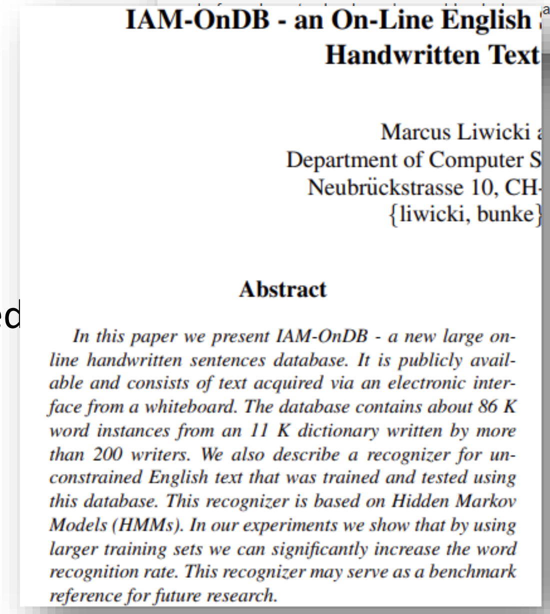
Edit

Introduced by Mitchell P. Marcus et al. in Building a Large Annotated Corpus of English: The Penn Treebank

The English **Penn Treebank** (PTB) corpus, and in particular the section of the corpus corresponding to the articles of Wall Street Journal (WSJ), is one of the most known and used corpus for the evaluation of models for sequence labelling. The task consists of annotating each word with its Part-of-Speech tag. In the most common split of this corpus, sections from 0 to 18 are used for training (38 219 sentences, 912 344 tokens), sections from 19 to 21 are used for validation (5 527 sentences, 131 768 tokens), and sections from 22 to 24 are used for testing (5 462 sentences, 129 654 tokens). The corpus is also commonly

2. **Handwriting Prediction (IAM-OnDB Dataset)**

- **Task:** Used the IAM online handwriting dataset, where pen positions were recorded as time series
- **Approach:** The network learned to generate realistic-looking handwriting strokes
- **Key Evaluation Metrics:** Introduced a probabilistic mixture density output layer to model real-valued handwriting data
- **Outcome:** Evaluated using log-likelihood loss and sum-squared error



IAM-OnDB - an On-Line English
Handwritten Text

Marcus Liwicki
Department of Computer S
Neubrückstrasse 10, CH-
{liwicki, bunke}

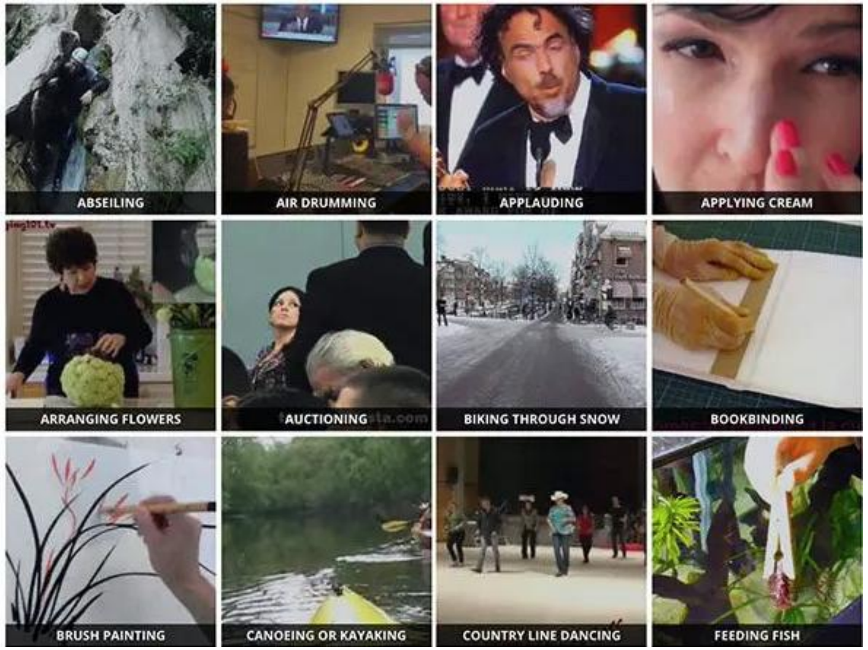
Abstract

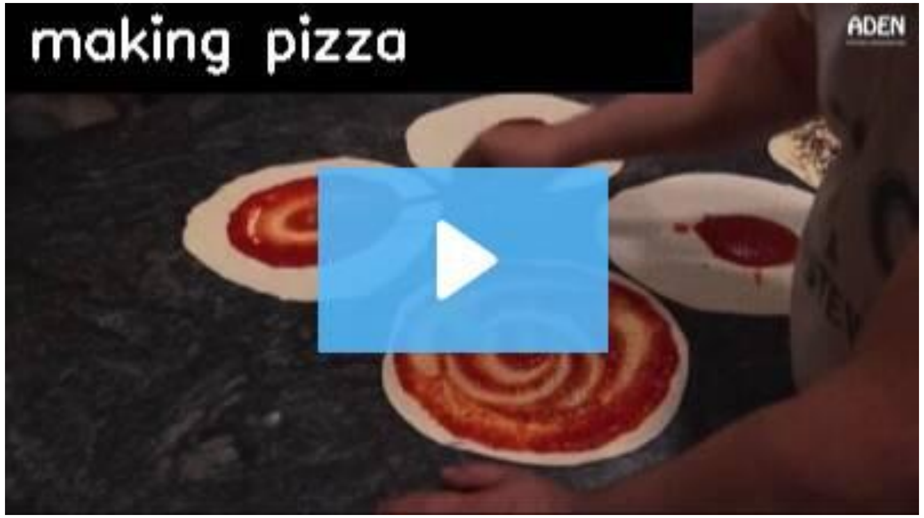
In this paper we present IAM-OnDB - a new large on-line handwritten sentences database. It is publicly available and consists of text acquired via an electronic interface from a whiteboard. The database contains about 86 K word instances from an 11 K dictionary written by more than 200 writers. We also describe a recognizer for unconstrained English text that was trained and tested using this database. This recognizer is based on Hidden Markov Models (HMMs). In our experiments we show that by using larger training sets we can significantly increase the word recognition rate. This recognizer may serve as a benchmark reference for future research.

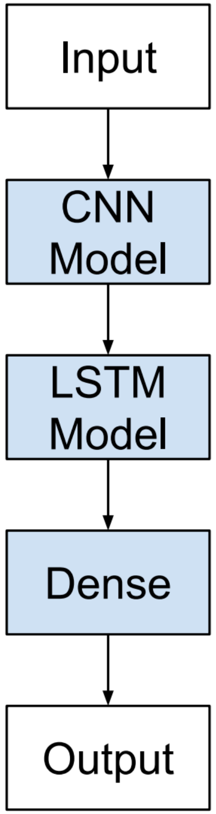
CNN LSTMs

CNN LSTM architecture involves using CNN layers for feature extraction on input data combined with LSTMs to support sequence prediction

- **Activity Recognition**: generating a textual description of an activity demonstrated in a sequence of images
- **Image Description**: generating a textual description of a single image
- **Video Description**: generating a textual description of a sequence of images

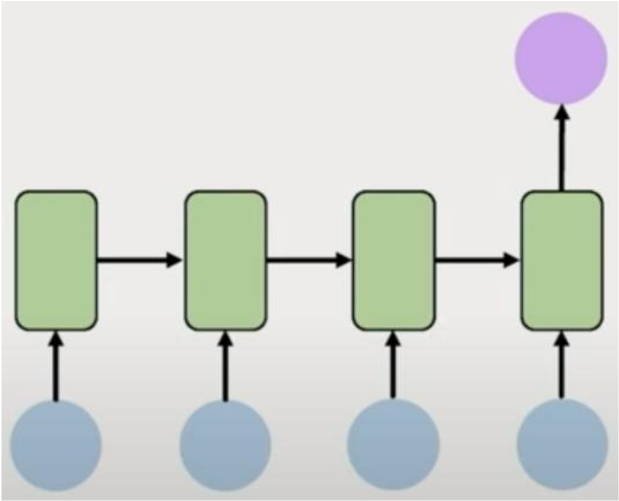






Limitations of RNNs for sequence modeling

- **RNNs** offer tremendous capabilities for modelling sequences, but like any technology, it has some limitations



- **Information encoding bottleneck:** The amount of information stored by RNNs depend on the fixed size of the hidden cell state $\boldsymbol{h}_t$. There is only a certain amount of limited information that can be stored in a finite-sized vector

- **Training time is slow, cannot be parallelized**: RNNs process information time-step by time-step , they can be parallelized, and are slow to train

- **Not long memory:** The encoding bottleneck of the hidden state can limit the capacity of long-term memory

Goal of sequence modeling

Let’s now think of how to overcome these limitations. Think of our fundamental goal of sequence modeling:

Take a sequence of inputs

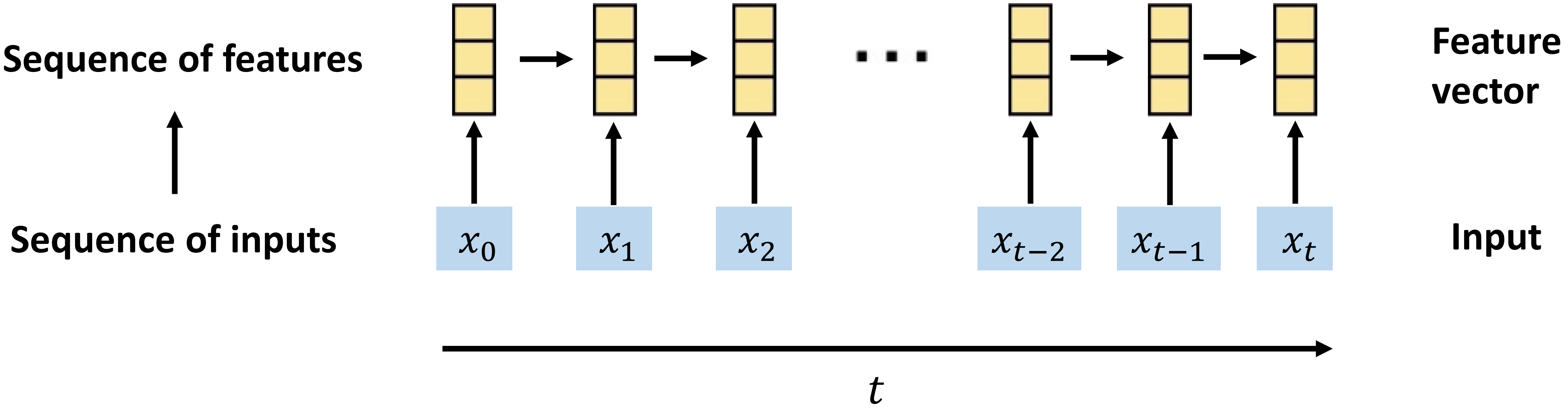
Sequence of inputs x_0 x_1 x_2 x_{t-2} x_{t-1} x_t **Input**

t

Goal of sequence modeling

Let’s now think of how to overcome these limitations. Think of our fundamental goal of sequence modeling:

Take a sequence of inputs → Use a neural network to compute features and states representing the inputs

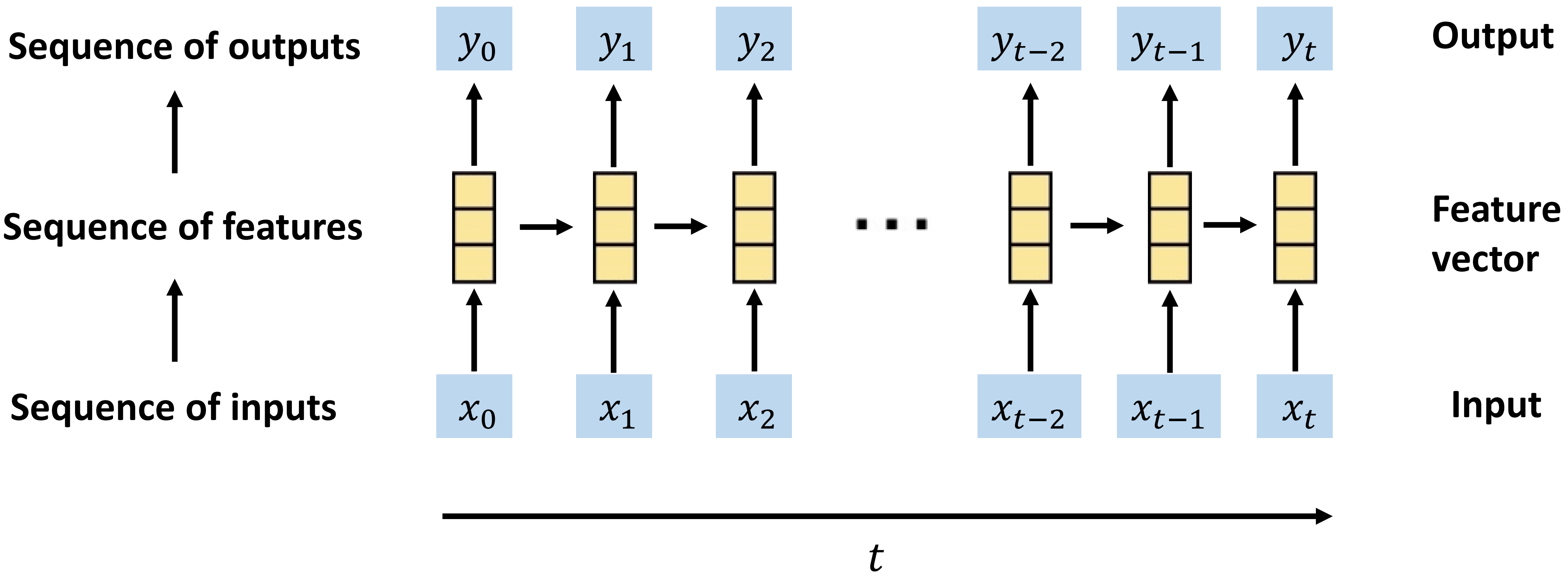


Goal of sequence modeling

Let's now think of how to overcome these limitations. Think of our fundamental goal of sequence modeling:

Take a sequence of inputs → Use a neural network to compute features and states representing the inputs → and then be able to generate predictions of the output according to that sequence

With RNNs we process sequence time-step by time-step, and use recurrence

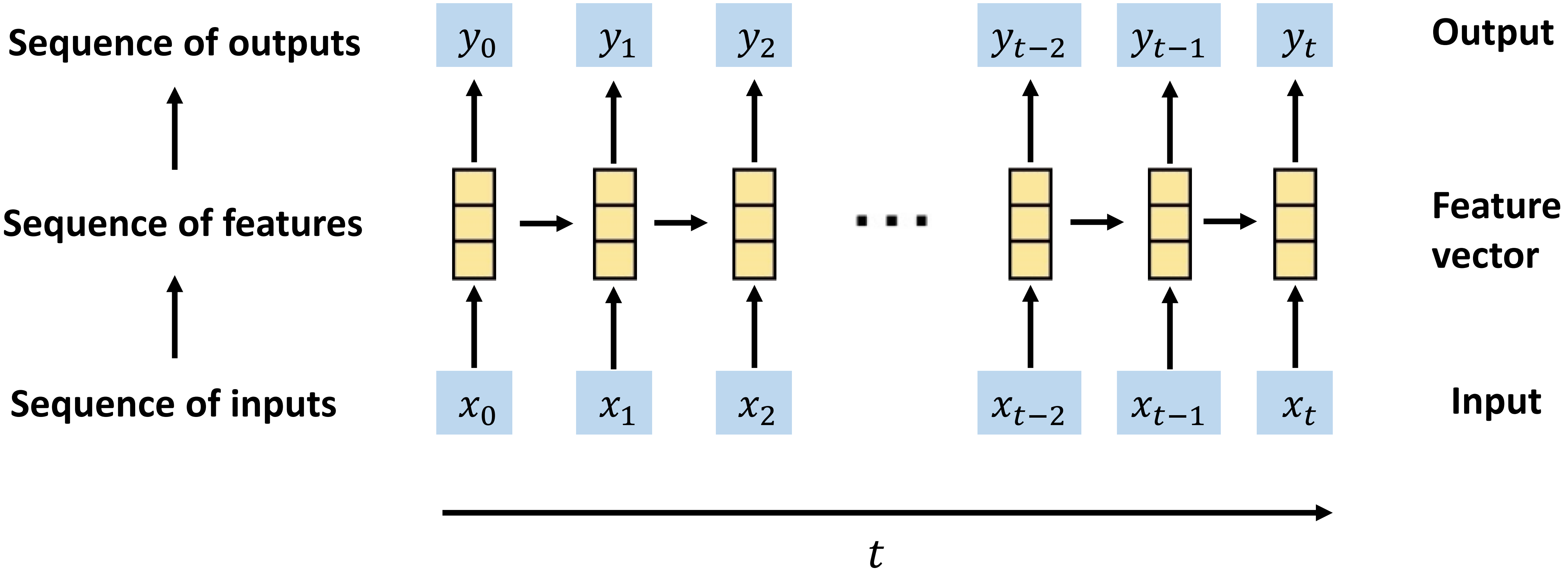


Goal of sequence modeling

Let's now think of how to overcome these limitations. Think of our fundamental goal of sequence modeling:

Take a sequence of inputs → Use a neural network to compute features and states representing the inputs → and then be able to generate predictions of the output according to that sequence

With RNNs we process sequence time-step by time-step, and use recurrence



Goal of sequence modeling

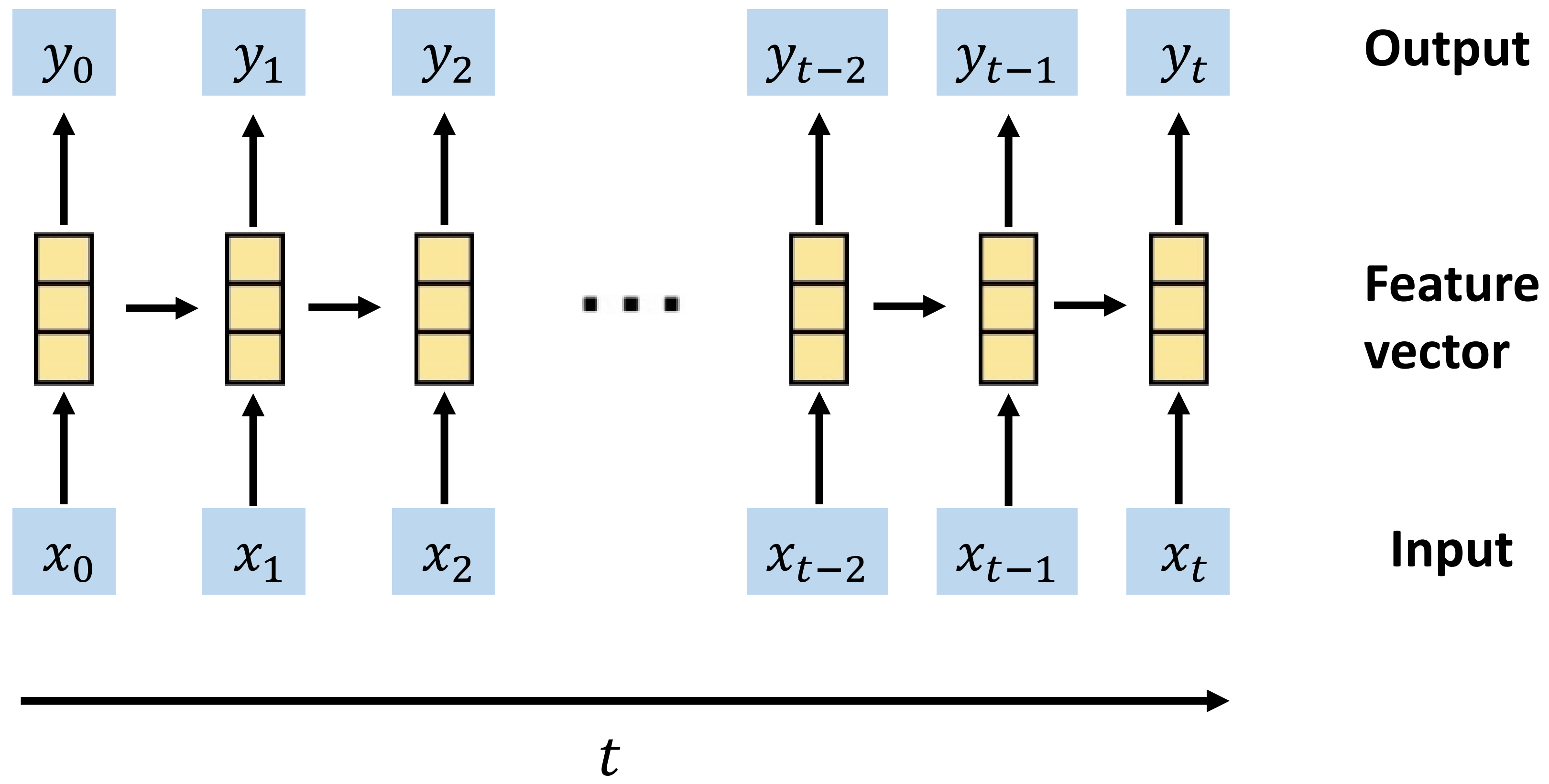
Let's now think of how to overcome these limitations. Think of our fundamental goal of sequence modeling:

Take a sequence of inputs → Use a neural network to compute features and states representing the inputs → and then be able to generate predictions of the output according to that sequence

With RNNs we process sequence time-step by time-step, and use recurrence

Limitations of RNNs

- Encoding bottleneck
- Slow, no parallelization
- Not long memory



Goal of sequence modeling

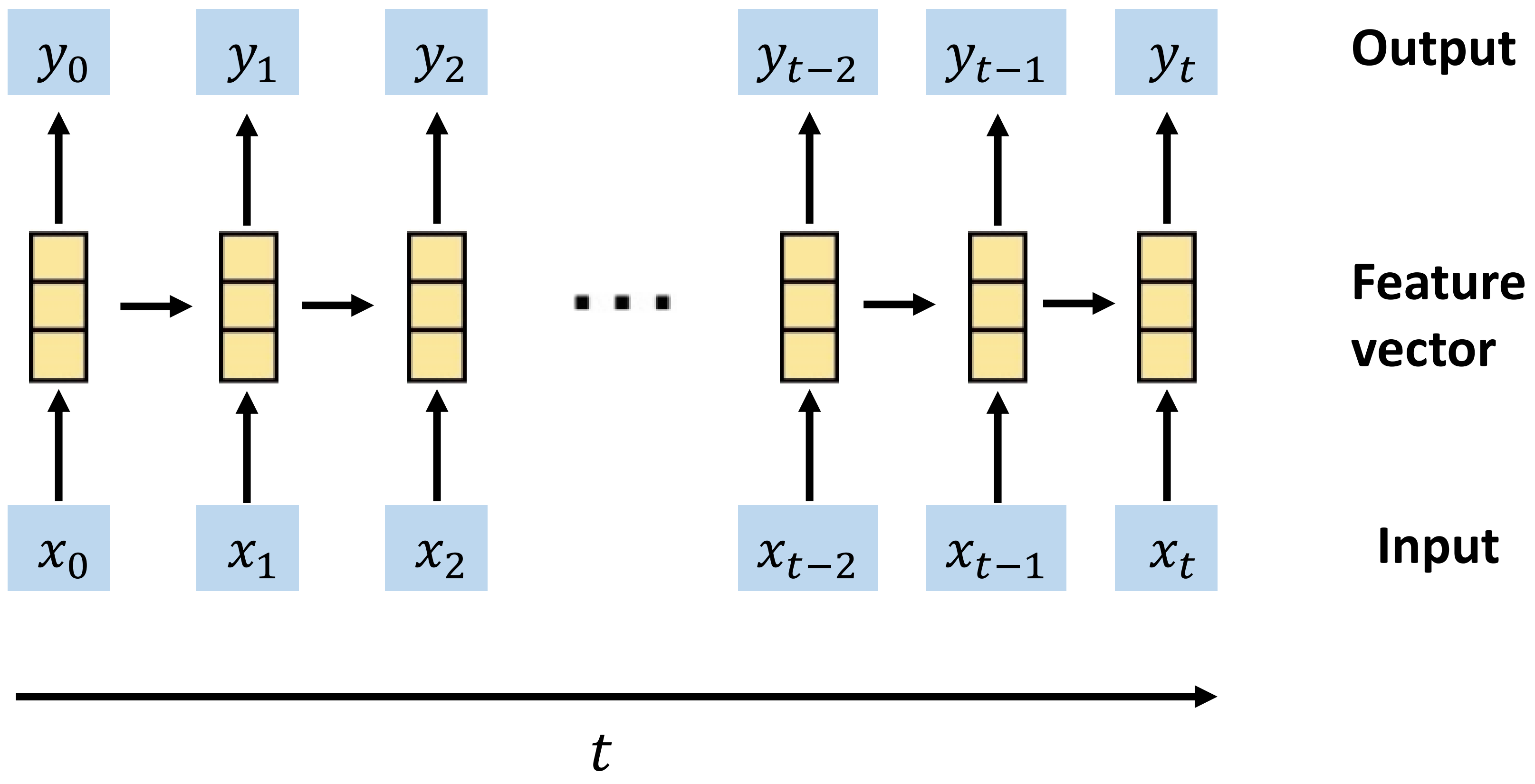
Let's now think of how to overcome these limitations. Think of our fundamental goal of sequence modeling:

Take a sequence of inputs → Use a neural network to compute features and states representing the inputs → and then be able to generate predictions of the output according to that sequence

Can we eliminate the need for recurrence entirely?

Desired Capabilities

- Efficient processing of sequence (removing the idea of time, if possible)
- Parallelization
- Long memory



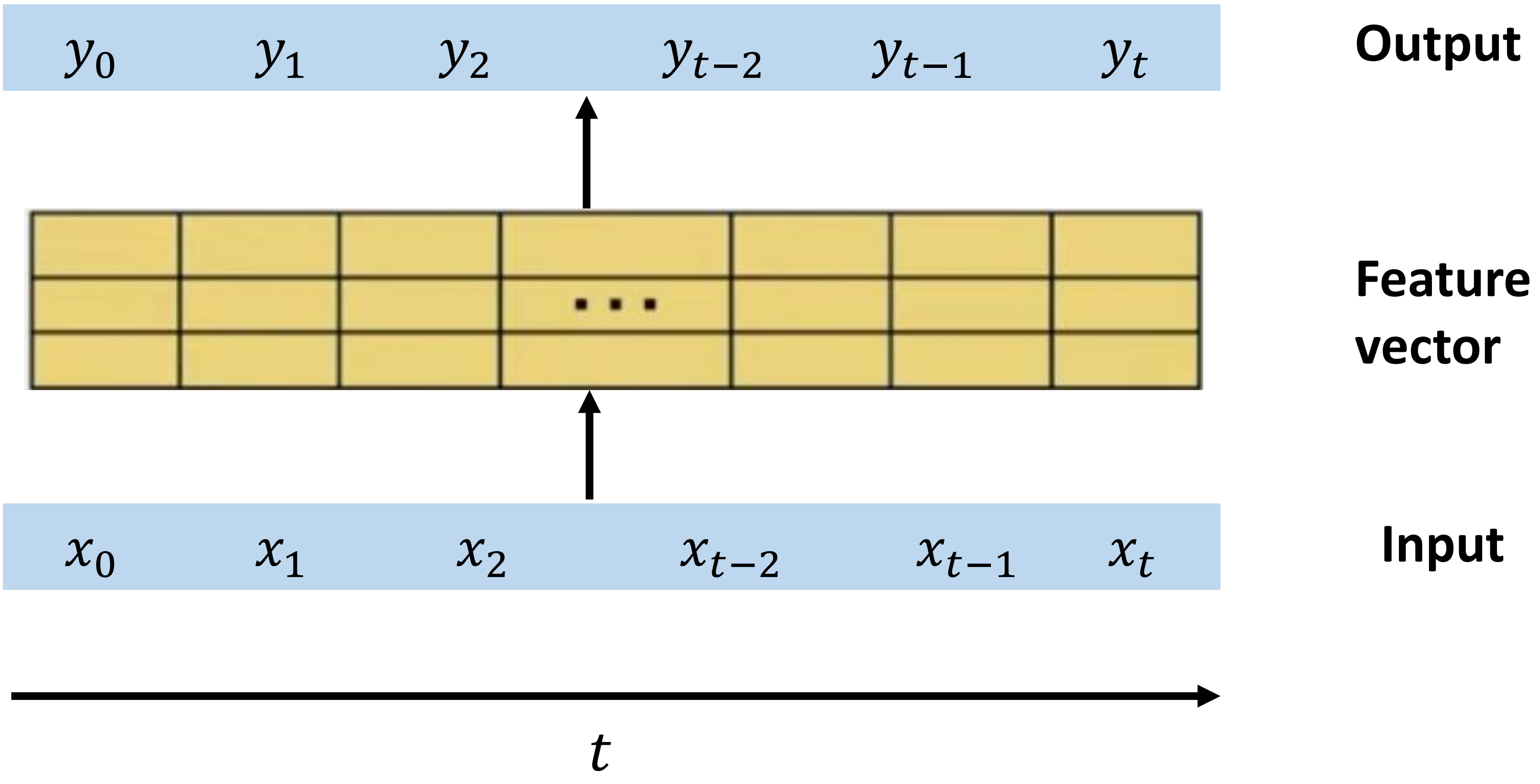
Goal of sequence modeling

Can we ignore the notion of individual time-steps, and put everything together into one large input vector and one large output vector and use an ANN?

Can we eliminate the need for recurrence entirely?

Desired Capabilities

- Efficient processing of sequence (removing the idea of time, if possible)
- Parallelization
- Long memory



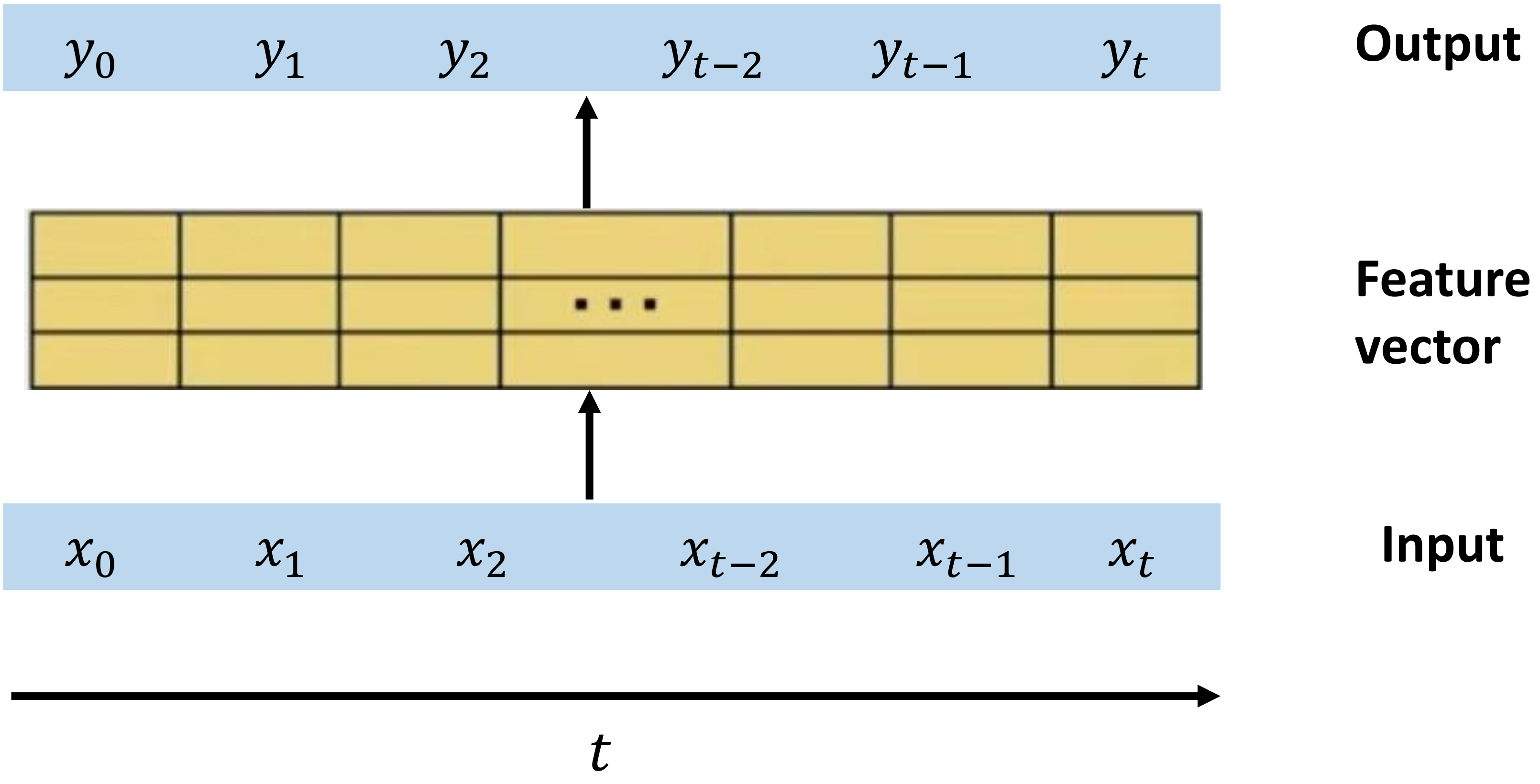
Goal of sequence modeling

Can we ignore the notion of individual time-steps, and put everything together into one large input vector and one large output vector and use an ANN?

Idea I: Feed everything into an ANN

Can we eliminate the need for recurrence entirely?

- ✓ No recurrence
- × Not scalable
- × No order
- × No memory



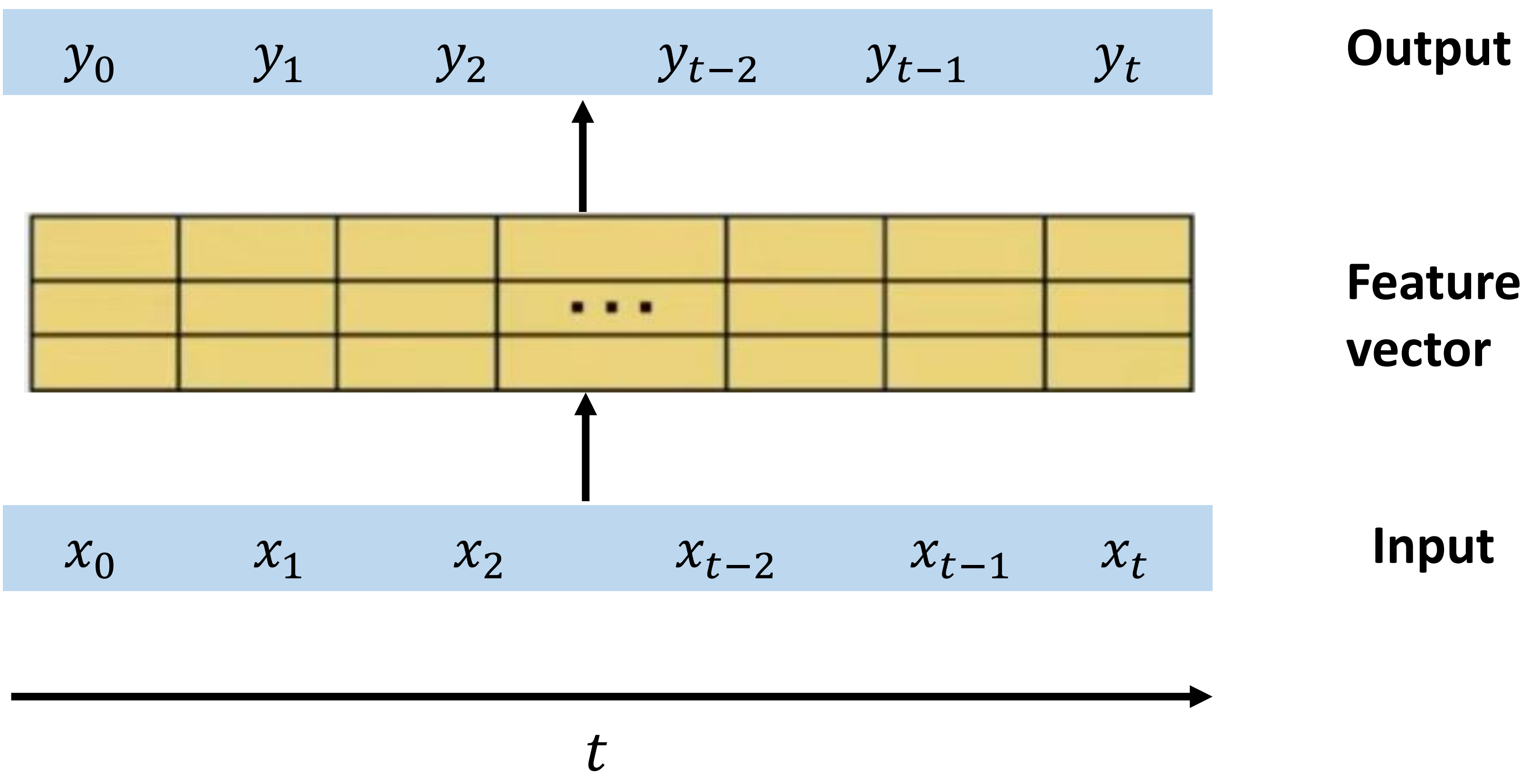
Goal of sequence modeling

Can we ignore the notion of individual time-steps, and put everything together into one large input vector and one large output vector and use an ANN?

Idea I: Feed everything into an ANN

Can we eliminate the need for recurrence entirely?

- ✓ No recurrence
- × Not scalable
- × No order
- × No memory



Idea II: Identify and attend to what's important

Goal of sequence modeling

Idea II: Identify and attend to what's important

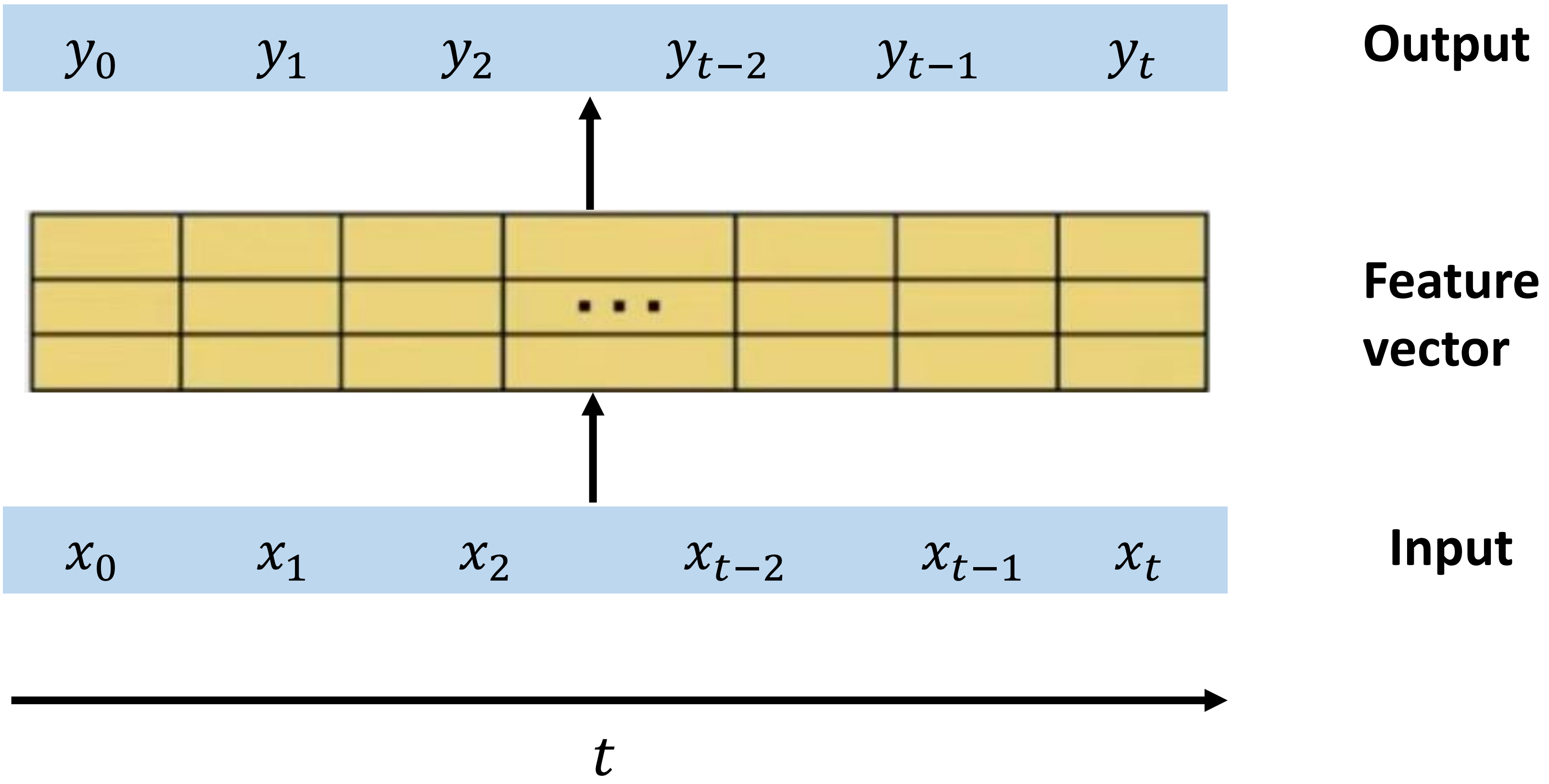
Take a sequential data and define a mechanism that by its own can pick out and look at the parts of the information that are important relative to other parts

How to define a way to do this?

- Identify important parts of a sequence
- Model the dependencies of each part to other parts of the sequence that relate to each other

Attention Is All You Need

(2017)



Attention is All You Need

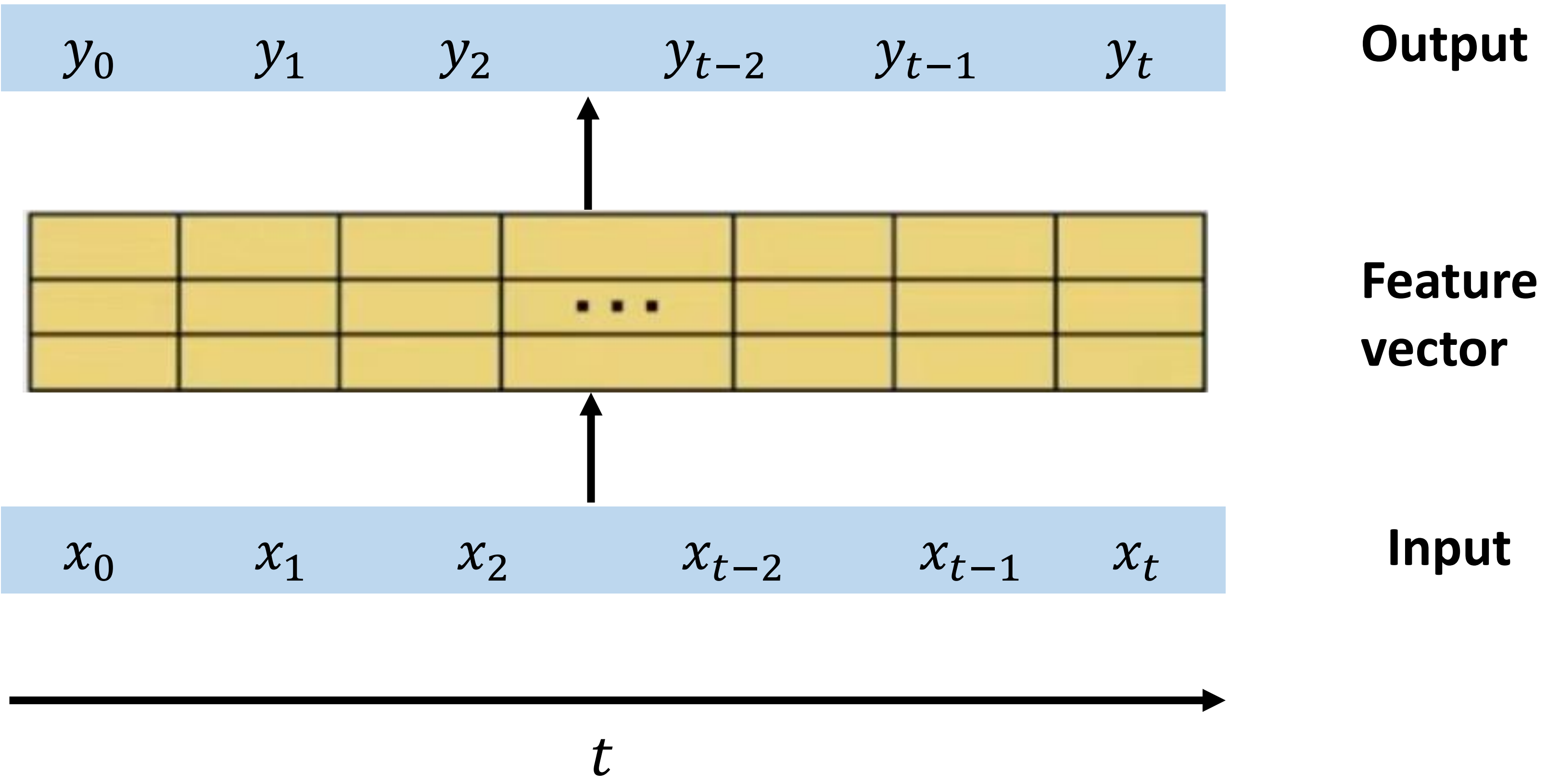
ChatGPT → the last T stands for a Transformer which is a neural network model which is used for sequential data whose inner foundational mechanism is based on Attention

Let's now talk about the attention mechanism

- Identify important parts of a sequence
- Model the dependencies of each part to other parts of the sequence that relate to each other

Attention Is All You Need

(2017)



Intuition behind Self-Attention

Attending to the most important parts of an input

- As humans, we have this inherent ability to look into an input and find **the important features**
- Let's start with an image and see what is more important
- One way is to scan the entire image back and forth for important things pixel by pixel and determine how important these individual pixels are, but **our brains don't operate like that**
- We are able to automatically look into it and get the most important part

Intuition behind Self-Attention

Attending to the most important parts of an input

- We are able to automatically look into it and get the most important part

1. Identify which parts to attend to
2. Extract the features with high attention values

Intuition behind Self-Attention

Attending to the most important parts of an input

- We are able to automatically look into it and get the most important part

1. Identify which parts to attend to
2. Extract the features with high attention values

Similar to a search problem

In a **search** problem:

- You ask a question or a query
- And you are trying to get answer

A simple search example

Let’s say you have a question and you search up on the internet

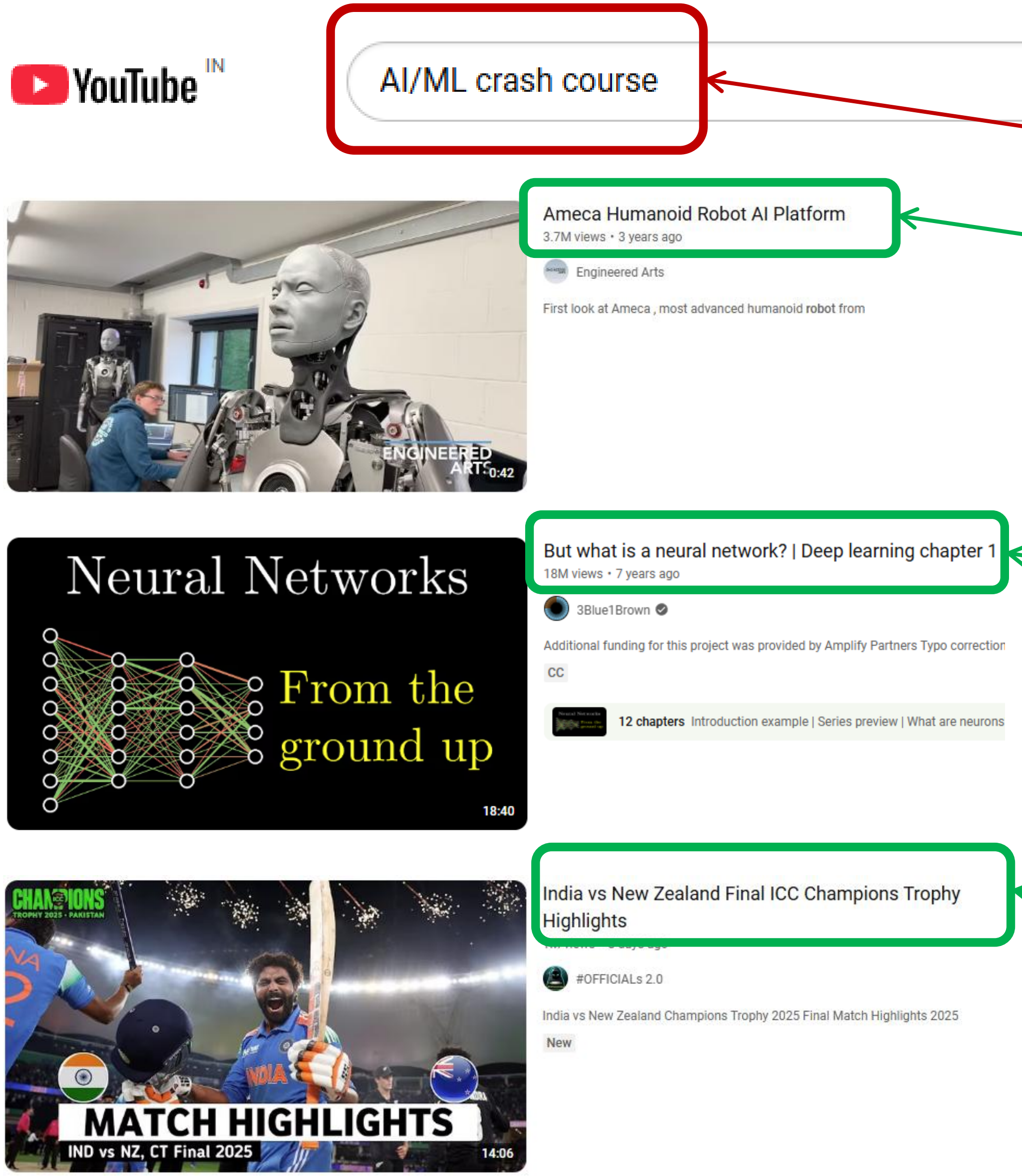
How can I know more about AI/ML?

A simple search example

Let’s say you have a question and you search up on the internet (say Youtube)

How can I
know more
about AI/ML?

Understanding Attention with Search



Query (Q)

Key (K_1)

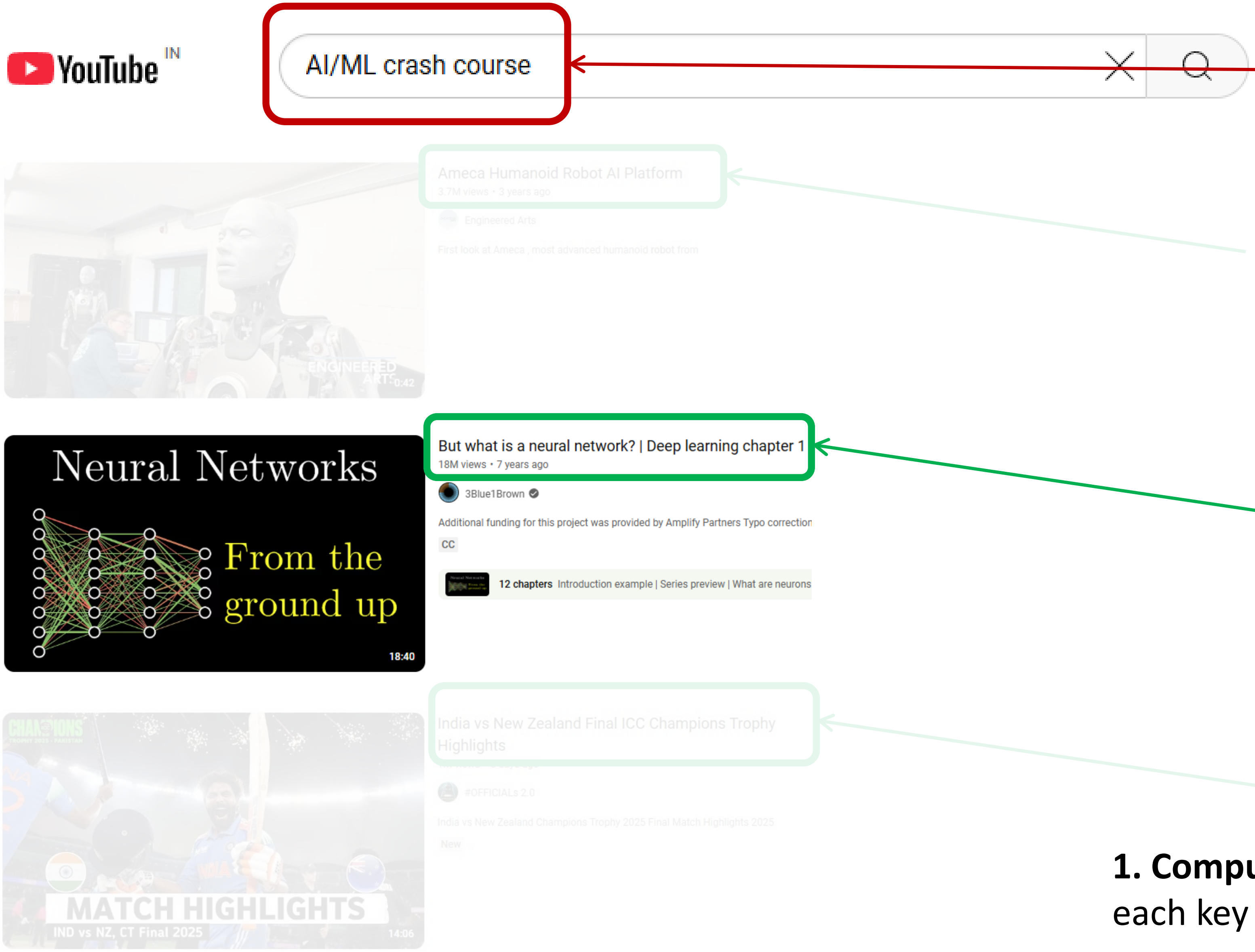
Keys are descriptors that capture/summarize the information contained in the videos

Key (K_2)

Key (K_3)

Search operates by comparing the closeness of the query with keys

Understanding Attention with Search



Query (Q)

Key (K_1)

Key (K_2)

Key (K_3)

How similar is the key to the query?

1. Compute attention mask: how similar is each key to the desired query?

Intuition behind Self-Attention

Attending to the most important parts of an input

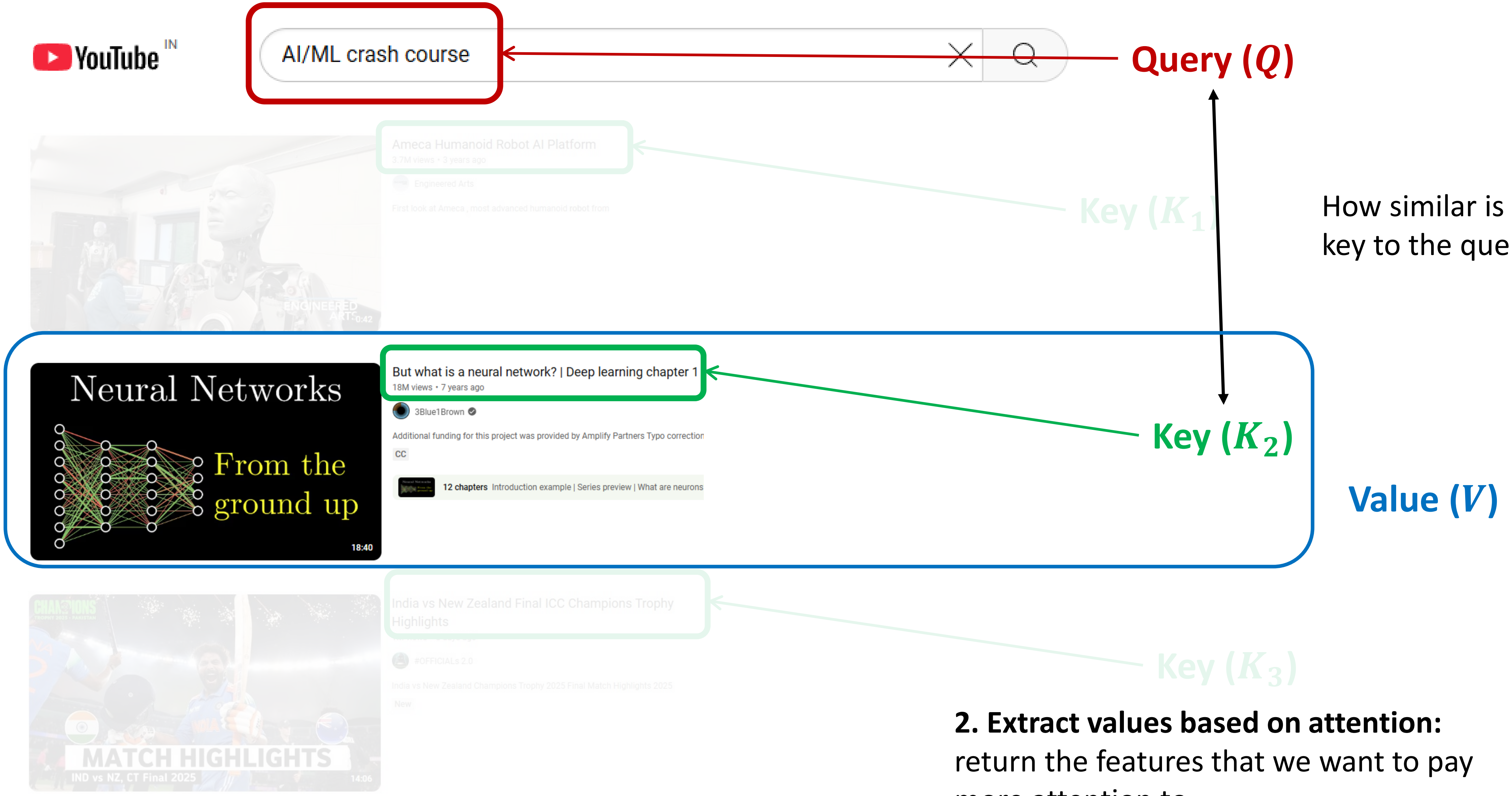
- We are able to automatically look into it and get the most important part

1. Identify which parts to attend to
2. Extract the features with high attention values

In a **search** problem:

- You ask a question or a query
- And you are trying to get answer

Understanding Attention with Search



How similar is the key to the query?

2. Extract values based on attention: return the features that we want to pay more attention to

Learning Self-Attention using Neural Networks

Let's come back to our sequence modeling problem where we have a series of words and we want to predict the next word (and we don't want to process this information time-step by time-step and we are going to feed in the data all at once)

Goal: Identify and attend the most important parts of an input

$\boldsymbol{x}$

He tossed the tennis ball to serve

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

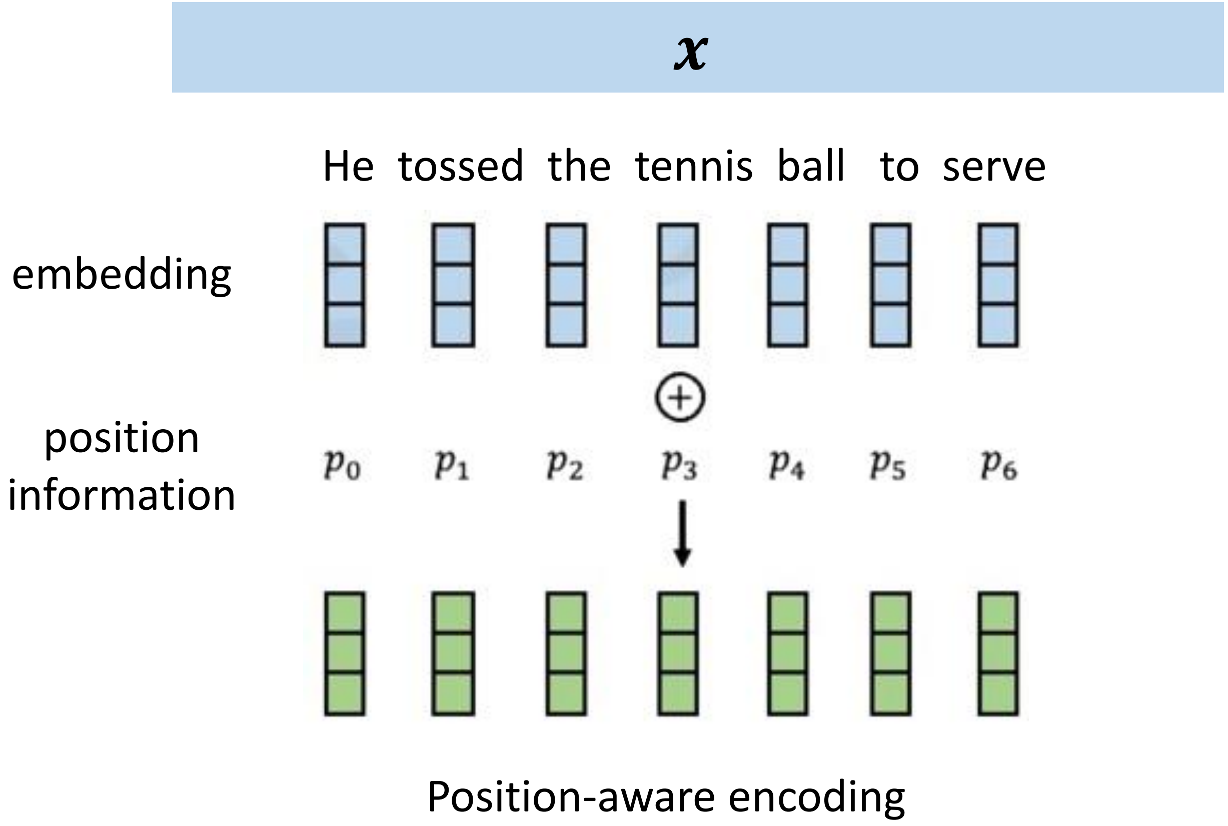
Data is fed all at once! Need to **encode position information** to understand order

Learning Self-Attention using Neural Networks

Let's come back to our sequence modeling problem where we have a series of words and we want to predict the next word (and we don't want to process this information time-step by time-step and we are going to feed in the data all at once)

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**



Data is fed all at once! Need to **encode position information** to understand order

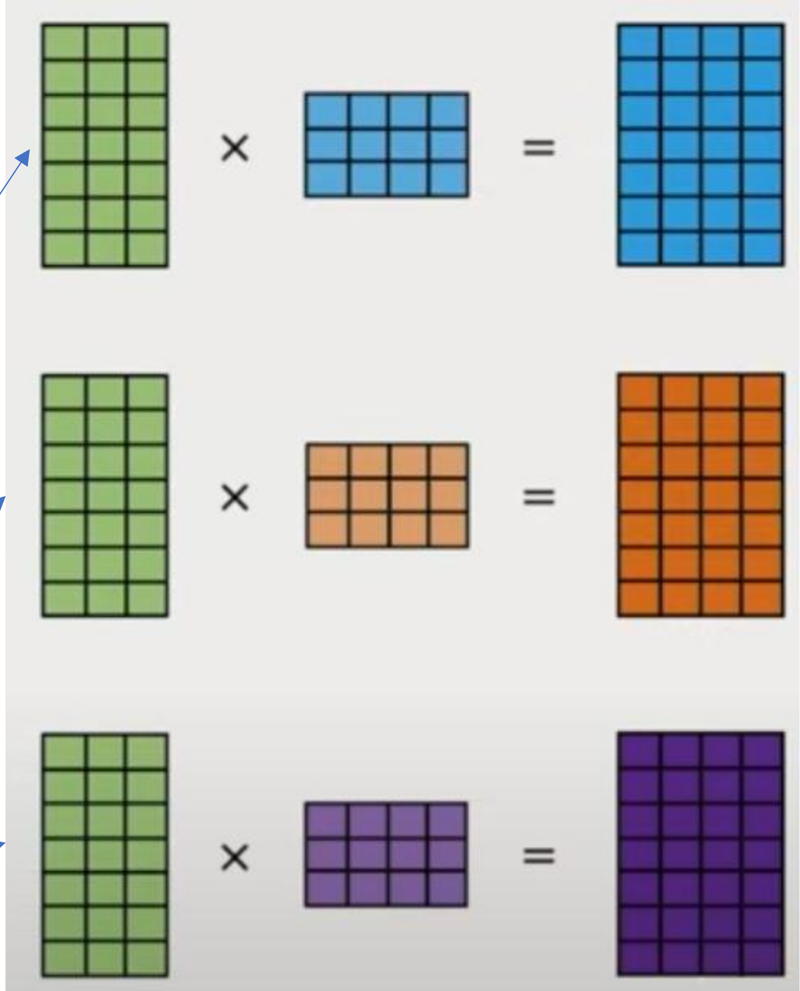
Learning Self-Attention using Neural Networks

Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Same positional encoding repeated



Query (Q)

Key (K)

Value (V)

Positional encoding

(Different) Linear layer

Output

Learning Self-Attention using Neural Networks

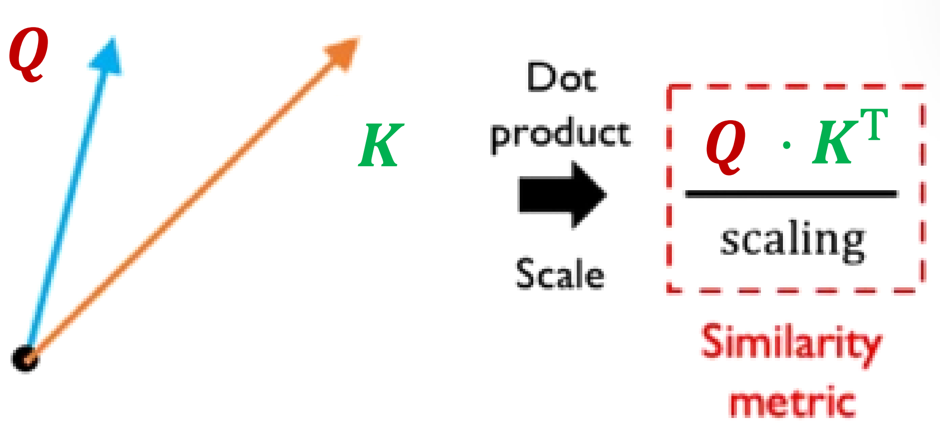
Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Attention Score: compute pairwise similarity between each **query (Q)** and **key (K)**

How to compute similarity between two sets of features?



Also known as the "cosine similarity"

Learning Self-Attention using Neural Networks

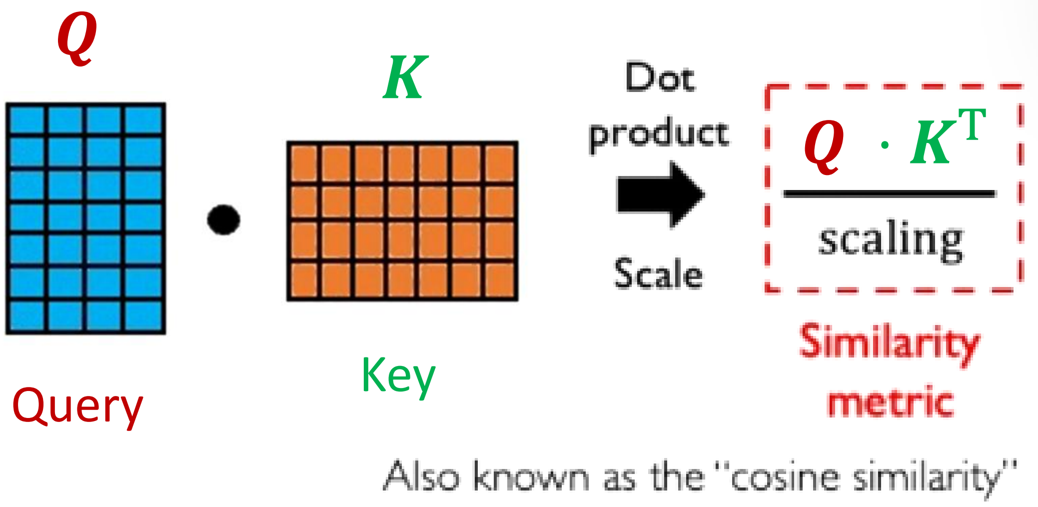
Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Attention Score: compute pairwise similarity between each **query (Q)** and **key (K)**

How to compute similarity between two sets of features?



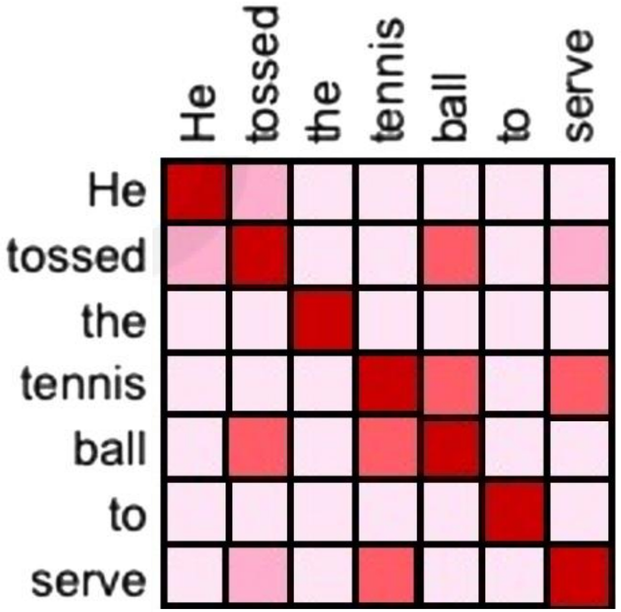
Learning Self-Attention using Neural Networks

Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Attention Weighting: where to focus the attention to?
How similar is the key to the query?



$$\textbf{softmax}\left(\frac{Q \cdot K^{T}}{\text{scaling}}\right)$$

Attention weighting

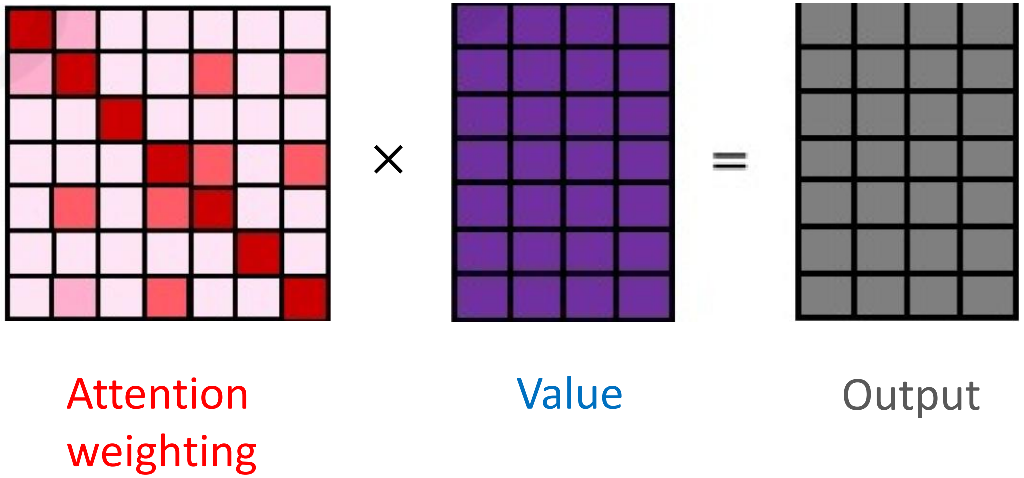
Learning Self-Attention using Neural Networks

Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Last step: self-attend to extra features



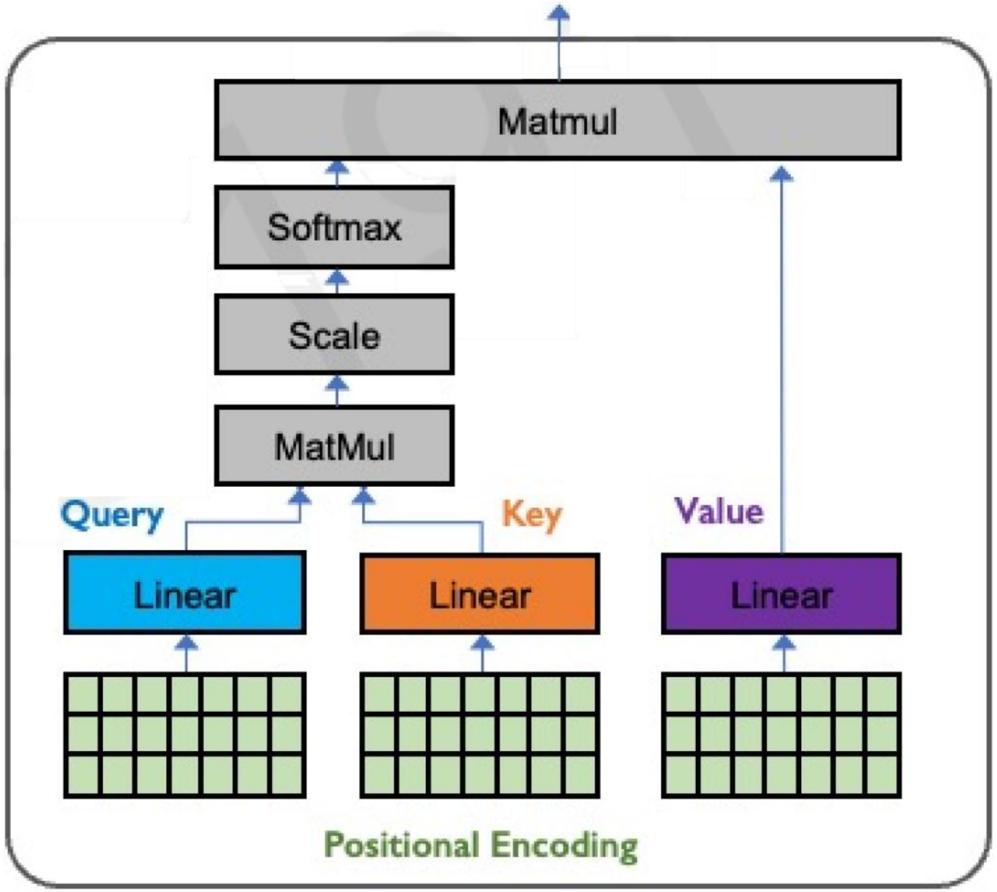
$$\textbf{softmax}\left(\frac{\boldsymbol{Q}\cdot\boldsymbol{K}^{\mathrm{T}}}{\text{scaling}}\right)\cdot\boldsymbol{V} = A(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})$$

Learning Self-Attention using Neural Networks

Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**



These operations form a self-attention head that can plug into a larger network

Each head attends to a different part of input

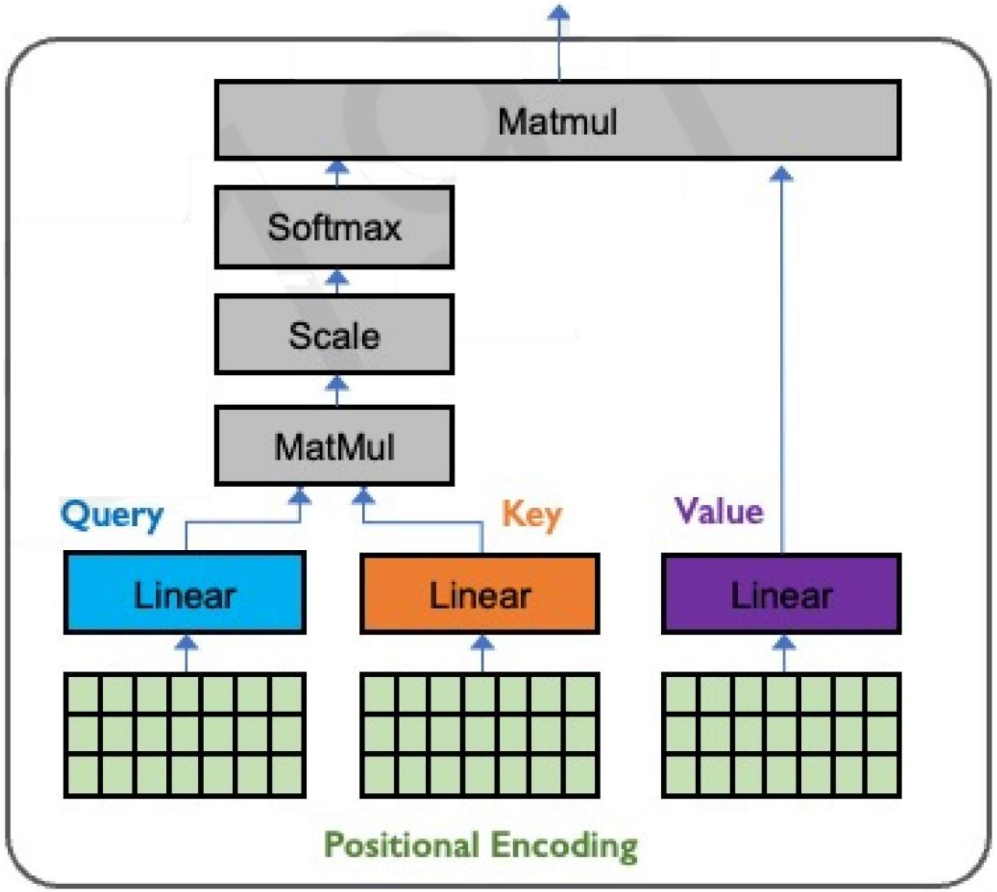
$$\textbf{softmax}\left(\frac{\boldsymbol{Q}\cdot\boldsymbol{K}^{\mathrm{T}}}{\mathrm{scaling}}\right)\cdot\boldsymbol{V}$$

Learning Self-Attention using Neural Networks

Next step: Do all kinds of search operation automatically using neural networks to extract three matrices that we call the query, key, and value matrices

Goal: Identify and attend the most important parts of an input

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**



$$\textbf{softmax}\left(\frac{Q \cdot K^{T}}{\text{scaling}}\right) \cdot V$$

Attention is the foundation building block of the Transformer architecture

Applying Multiple Self-Attention Heads

×

=

Attention Weighting

Value

Output

Output of attention head 1

Output of attention head 2

Output of attention head 3

Applications of self-attentions

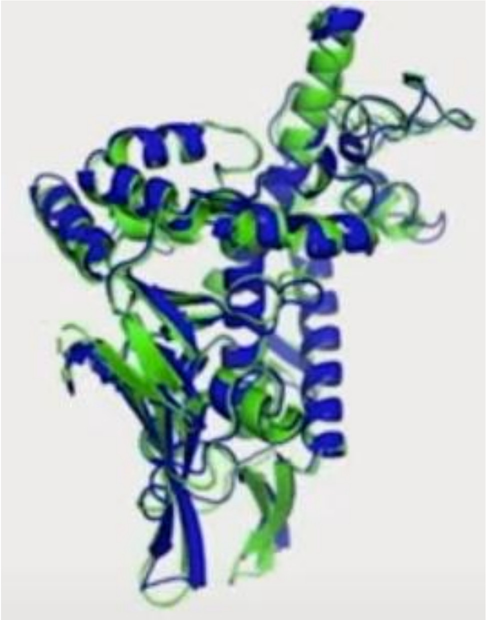
Language processing

An armchair in the shape of an avocado

Transformers: GPT, BERT

Devlin et al., *NAACL* 2019
Brown et al., *NeurIPS* 2020

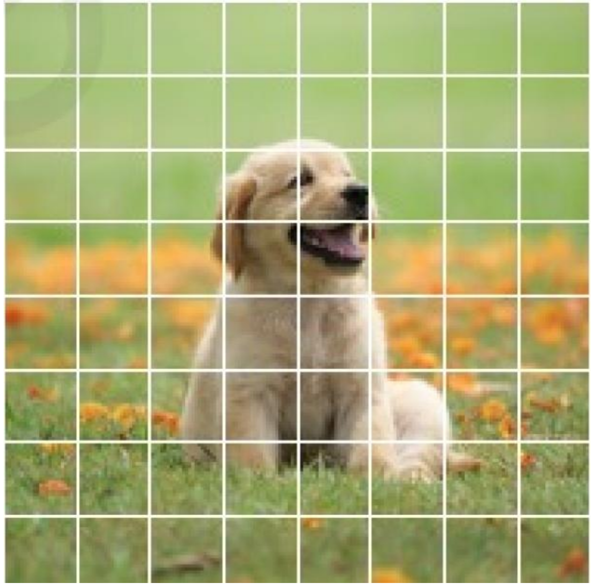
Biological Sequences



Protein Structure Models

Jumper et al., *Nature* 2021
Lin et al., *Science* 2023

Computer Vision



Vision Transformers

Dosovitskiy et al., *ICLR* 2020