ARTIFICIAL INTELLIGENCE NEEDS REAL INTELLIGENCE

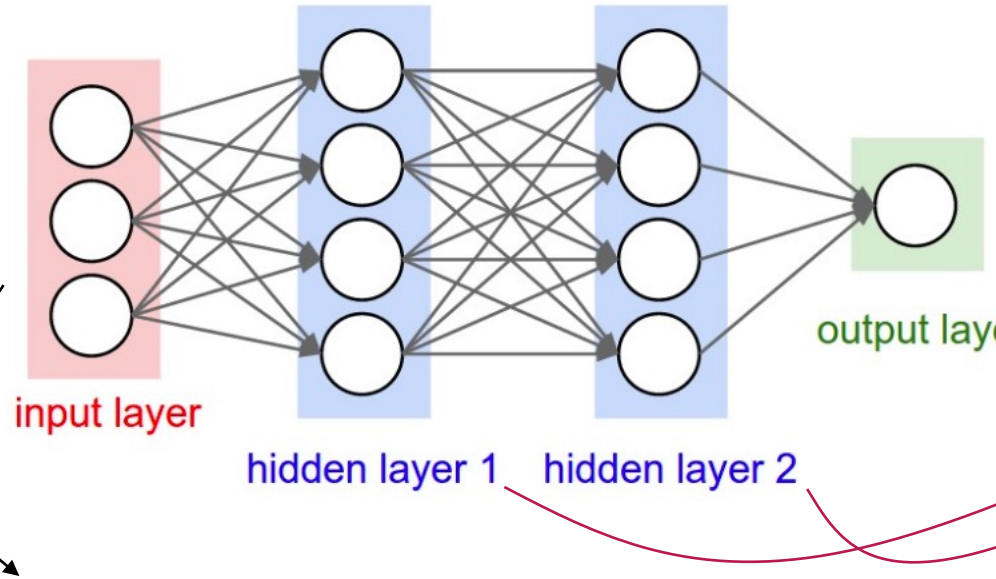
Recurrent Neural Network & Natural Language Processing

Professor: Tianyuan ZHANG
tianyuan.zhang@kedgebs.com

KEDGE BUSINESS SCHOOL

Recap of Previous Session

- **Artificial neural network**




```
# define a custom neural network class
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(3, 4), nn.ReLU(),
            nn.Linear(4, 4), nn.ReLU(),
            nn.Linear(4, 1)
        )
    def forward(self.x):
        return self.net(x)
```

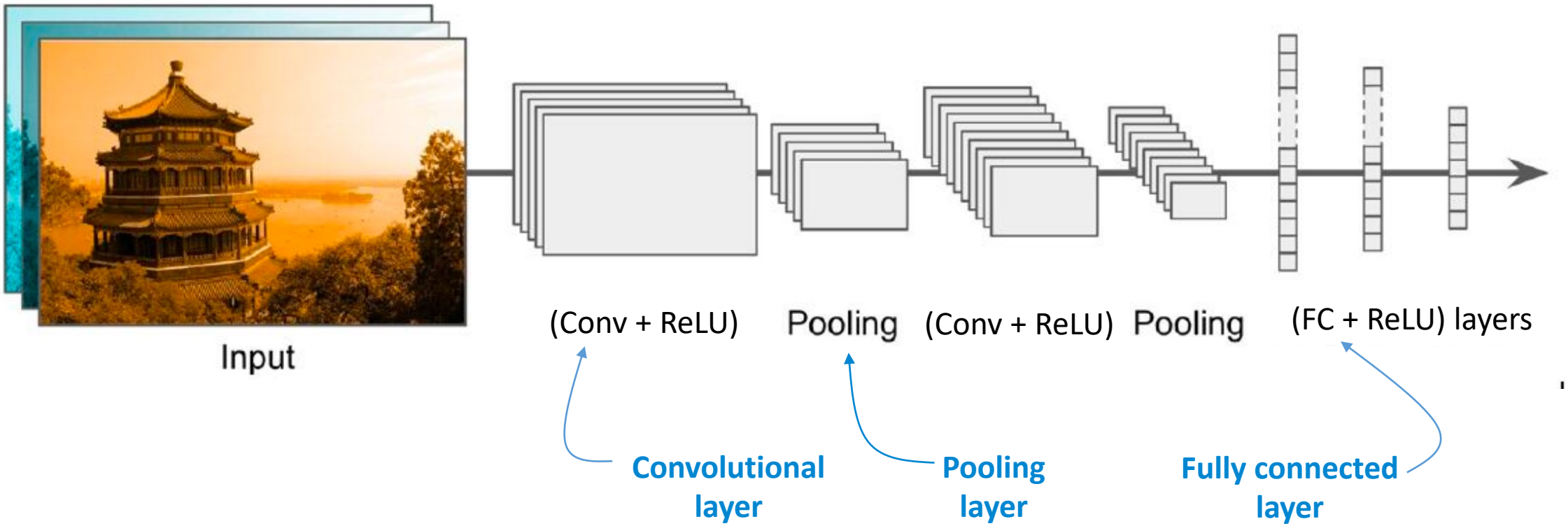
Activation function

Fully connected layer

- *Input is a batch of 1D vectors.*
- *The length of the 1D vector is the number of input features.*

Recap of Previous Session

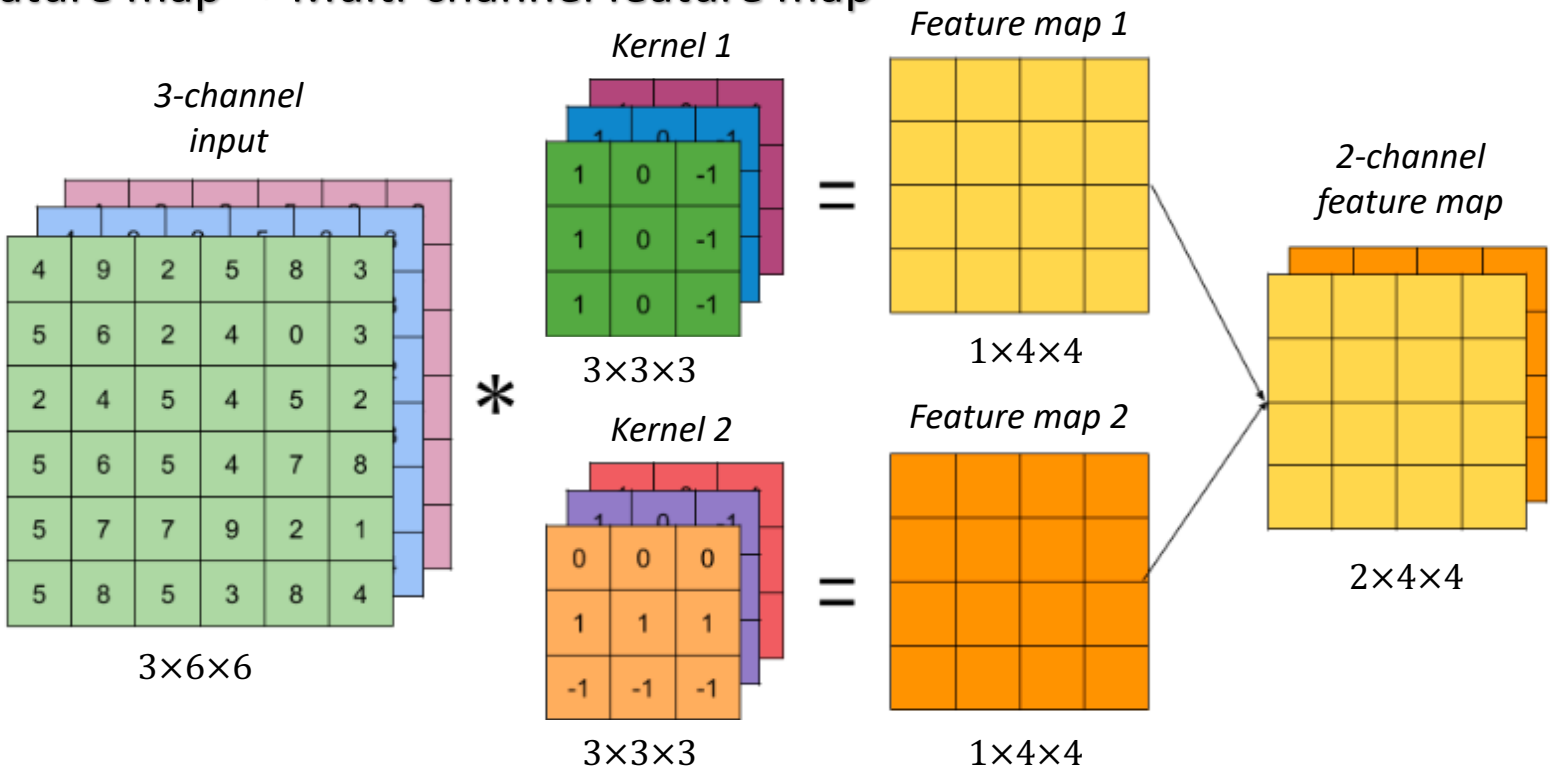
- **Convolutional Neural Network**



Recap of Previous Session

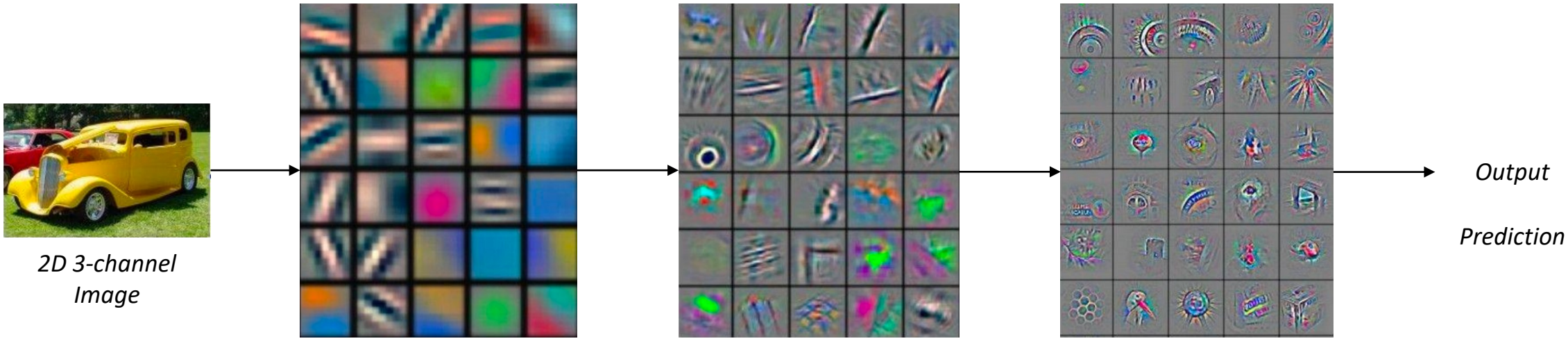
- **Convolutional Neural Network**
  - Based on convolution operation
    - Multi-channel image / feature map → Multi-channel feature map

1. *For multi-channel input feature map, perform convolution operation per input channel.*
2. *Add up the channel-wise results to obtain the single-channel output feature map*
3. *If there are multiple. convolutional kernels, stack the single-channel outputs of all kernels to get the multi-channel output feature map*



Recap of Previous Session

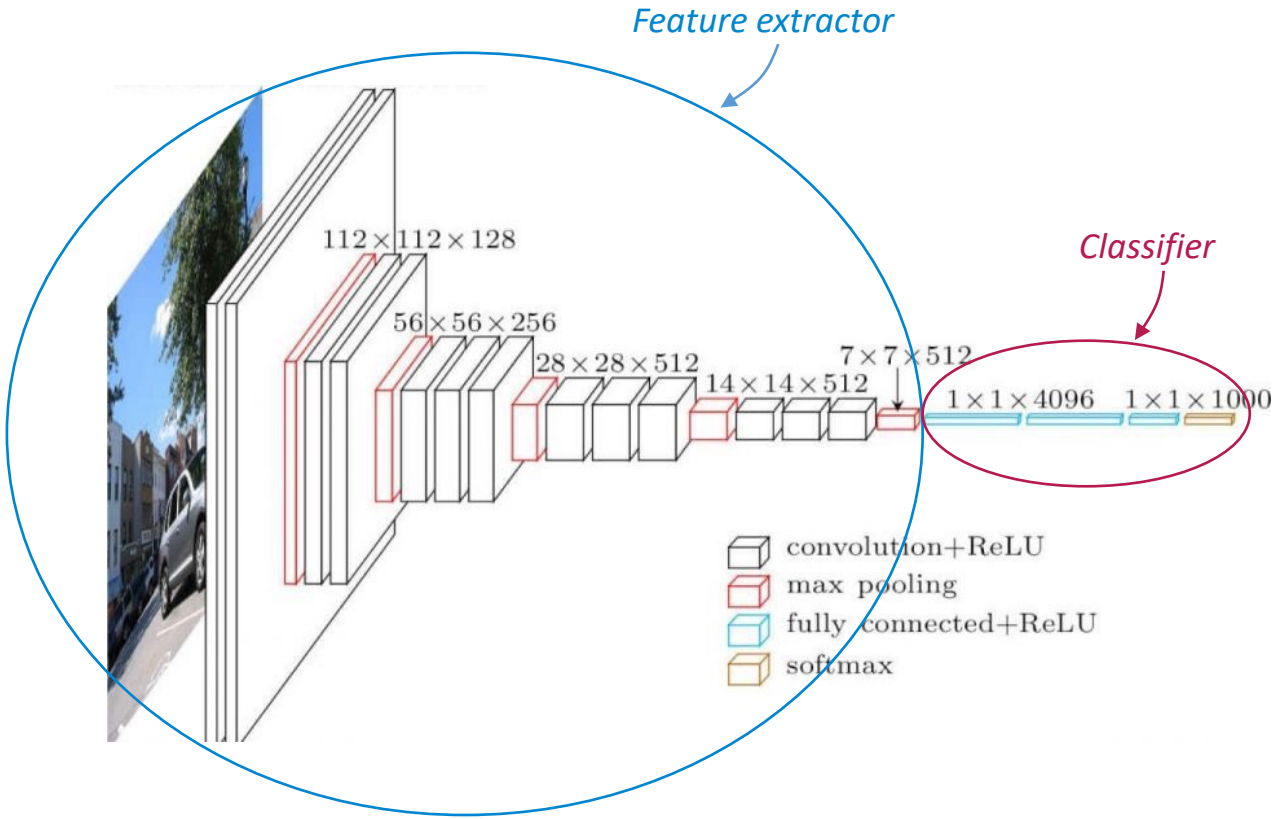
- **Convolutional neural network**



- Detect local spatial patterns and extract them as feature maps by convolution
- Stack multiple convolutional layers to aggregate features hierarchically

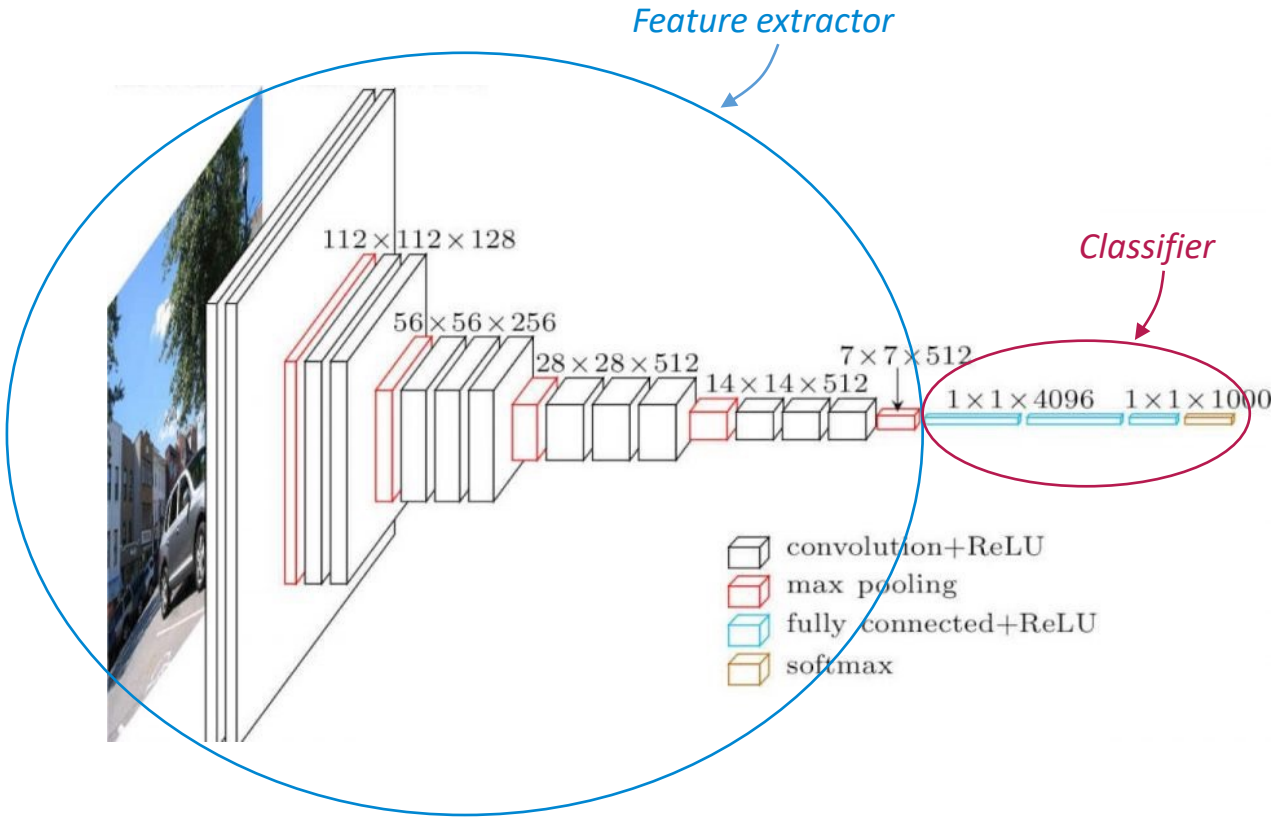
Recap of Previous Session

- Transfer learning
  - Train a deep neural network on a large dataset can be time-consuming
  - Use a pre-trained neural network to extract features and build a new classification / regression model based on that
  - For CNN:
    - Before flattening → Feature extractor
    - After flattening → Classifier



Recap of Previous Session

- Transfer learning
  1. Load a pre-trained neural network
  2. Freeze the feature extractor
  3. Replace the original classifier with new layers
  4. Train the new classifier with a relatively large learning rate
     - The feature extractor remains unchanged
  5. (Optional) Fine-tune the entire neural network with a relatively small learning rate

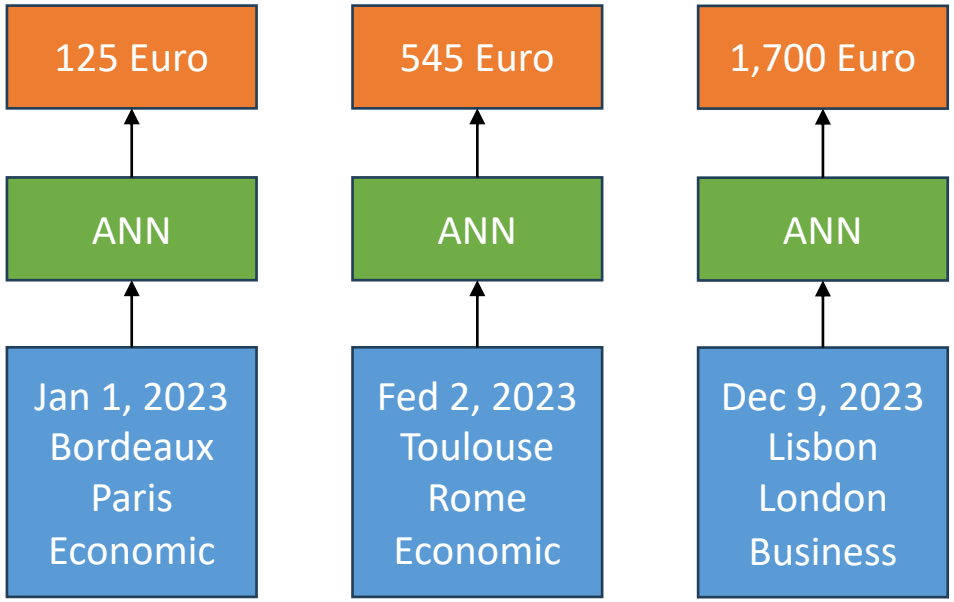


Outline

- **Recurrent neural network**
- Natural language processing

Recurrent neural network

- **Limitation** of traditional ANN and CNN:



Flight price prediction
ANN-based

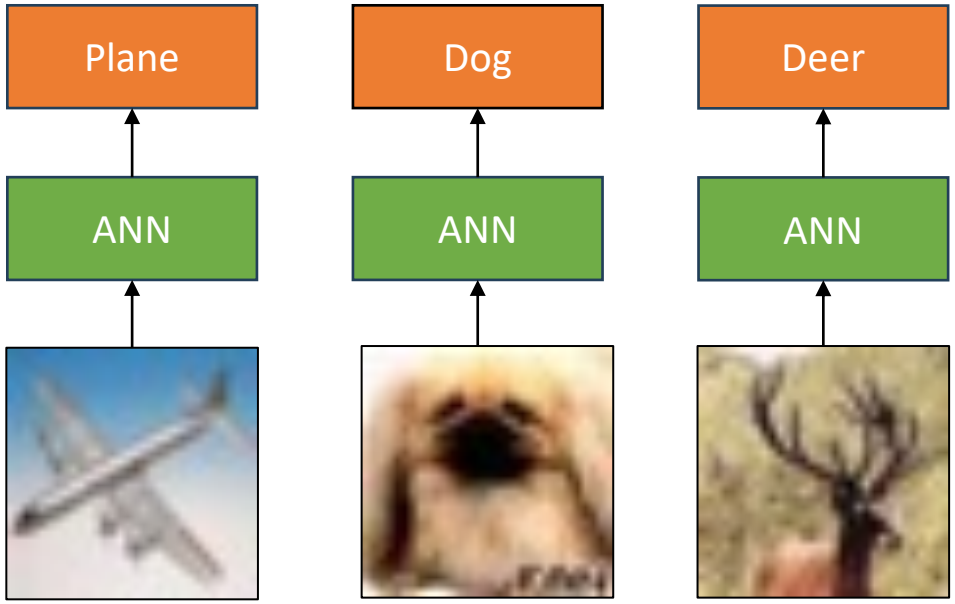


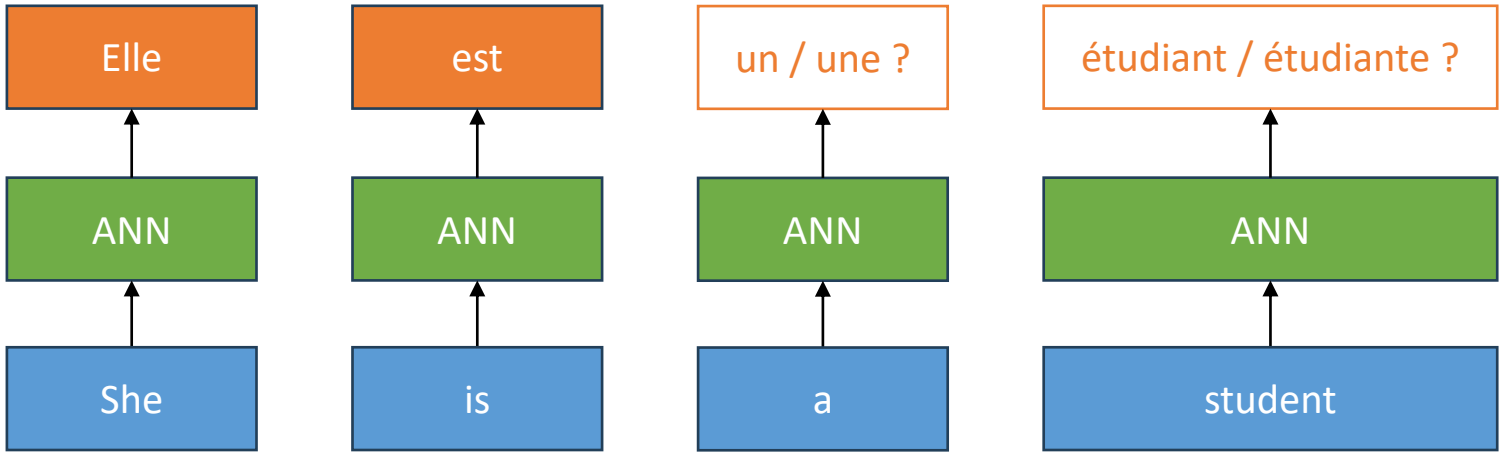
Image classification
CNN-based

Recurrent neural network

- **Limitation** of traditional ANN and CNN:
  - When making predictions, each sample is treated independently
  - The network has no memory when making predictions
    - The neural network can't remember the predictions it made in the past
    - The neural network can't remember the samples it saw in the past
  - Given the same sample, the network always make the same prediction
  - Suppose there is a sequence of samples to predict, the order of samples won't influence the network's predictions
    - The network can't use the order of samples in the sequence as information

Recurrent neural network

- **Limitation** of traditional ANN and CNN:
  - Suppose we have an ANN-based language translation model
    - English to French
    - The model takes an English word as input and outputs the corresponding French word
  - Suppose we want to translate this sentence: She is a student.

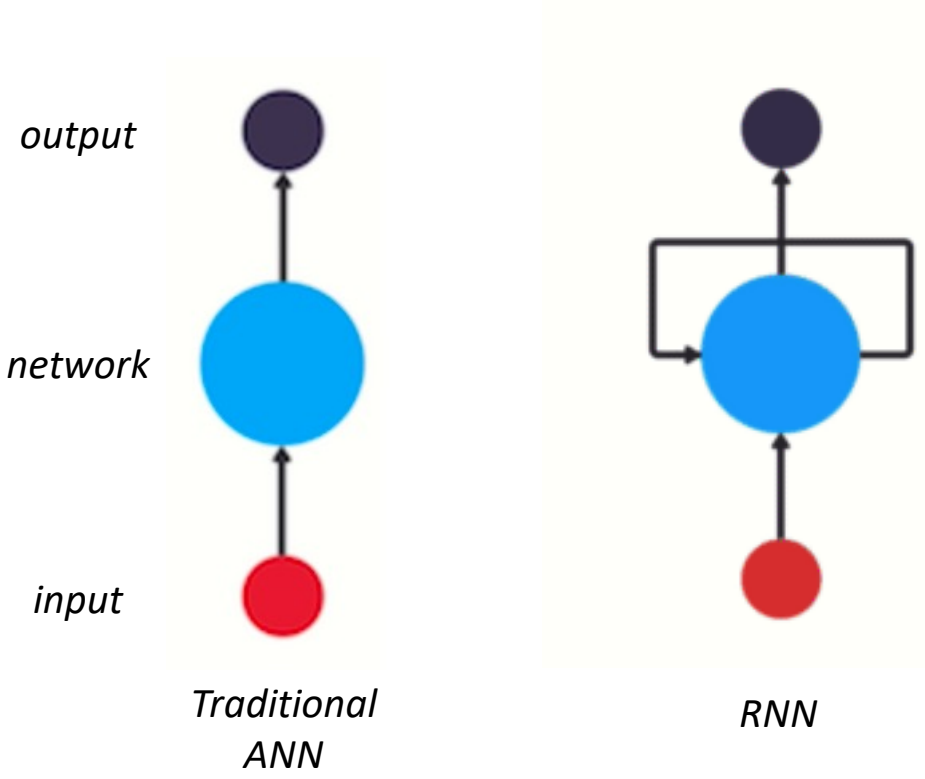


Recurrent neural network

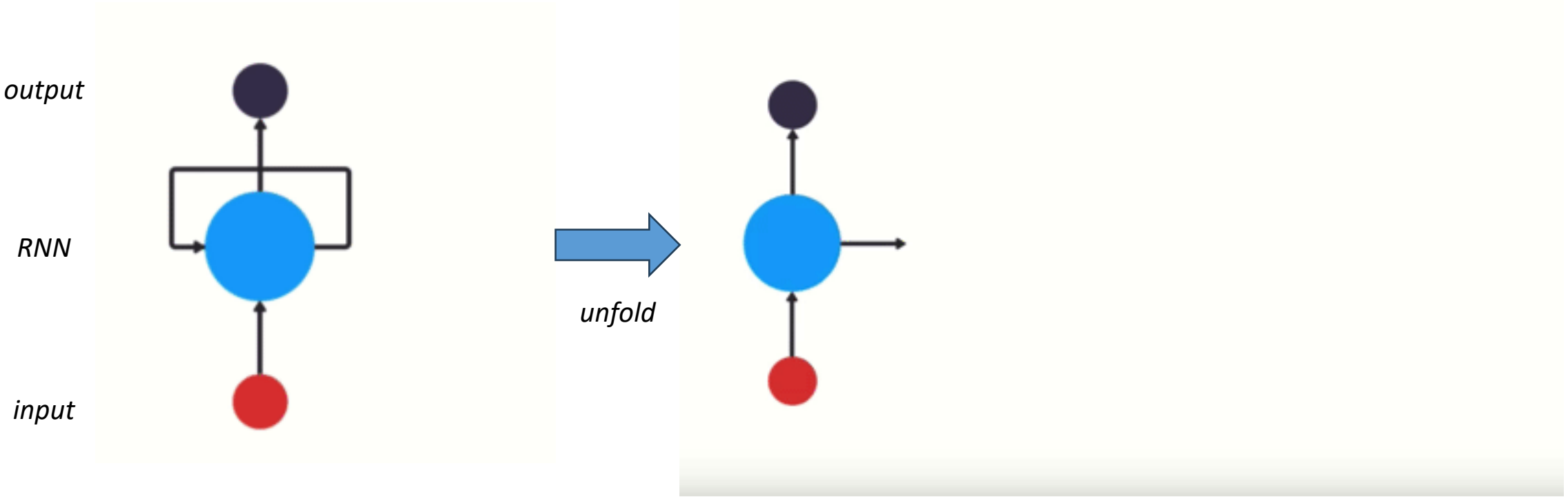
- **Sequence data**
  - A sequence of samples
  - The samples in the sequence is dependent with each other
  - The order of samples also carries information for our task
    - For example, recognize the emotional of the sentence
      - She was not happy, merely pretending → Negative
      - She was happy, not merely pretending → Positive
    - Chatbot needs to remember the previous conversation to understand the context
  - The neural network needs to have memory

Recurrent neural network

- **Recurrent neural network**
  - RNN allows the previous output to be used as the input while having hidden states.
  - The hidden state can be seen as the memory of the RNN
    - It carries the useful information extracted by RNN from the previous input samples
    - The hidden state will change when an input sample is fed into the RNN
    - Then, the new hidden state will be fed into the RNN again along with the next input sample to generate the output

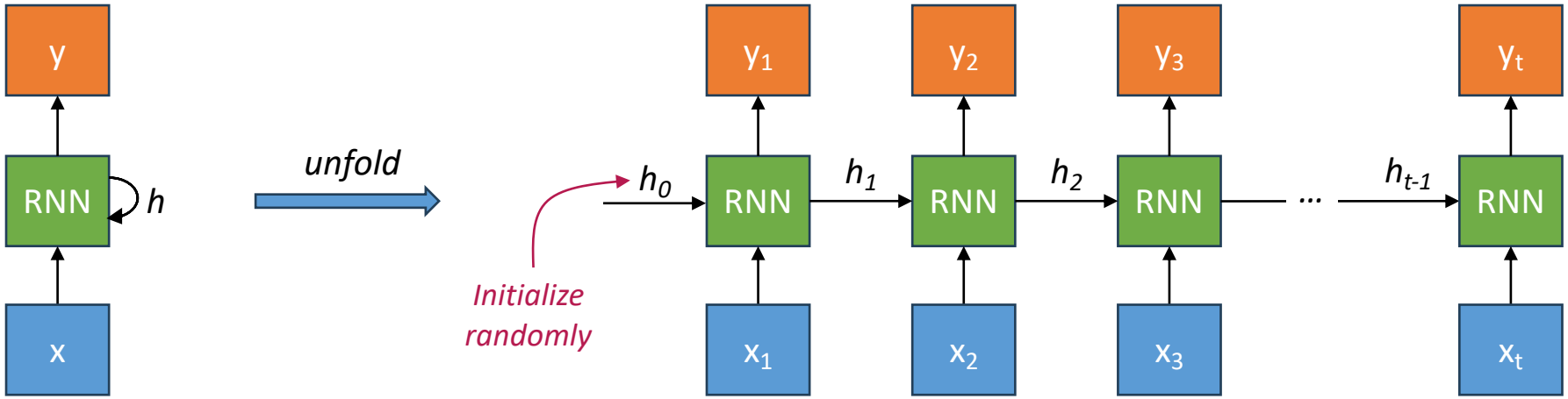


Recurrent neural network



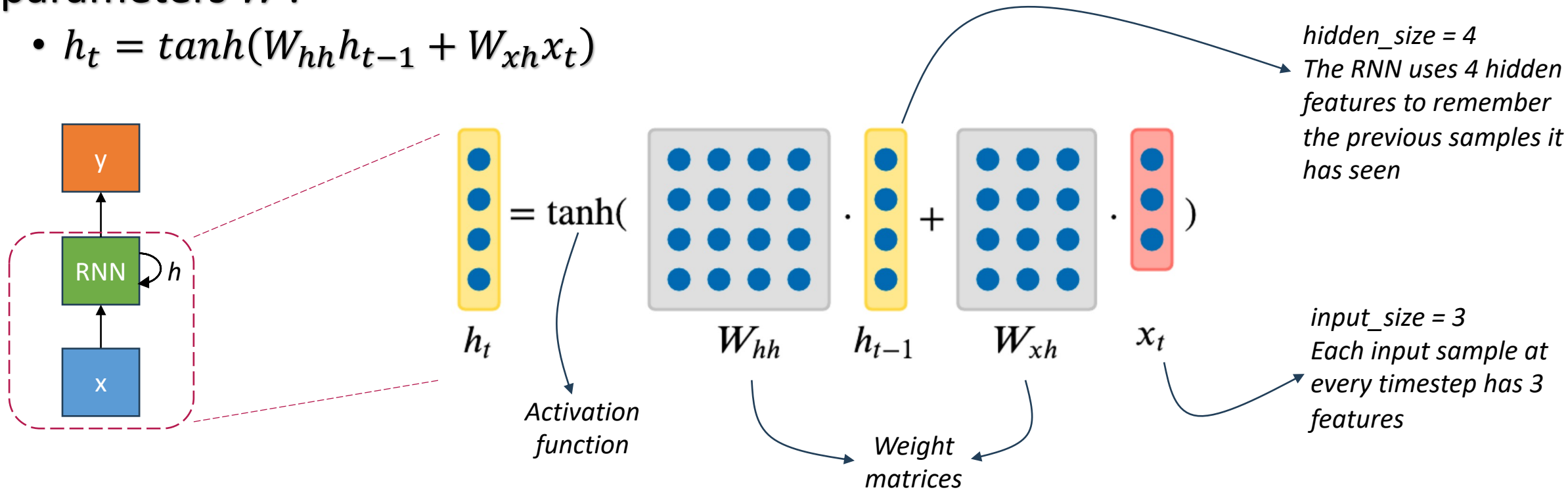
Recurrent neural network

- RNN can be represented as a recurrence formula of function f_W with parameters W:
  - At the current timestep t, the hidden state is $h_t = f_W(h_{t-1}, x_t)$
  - x_t is the input at the current timestep
  - h_{t-1} is the hidden state of the previous timestep



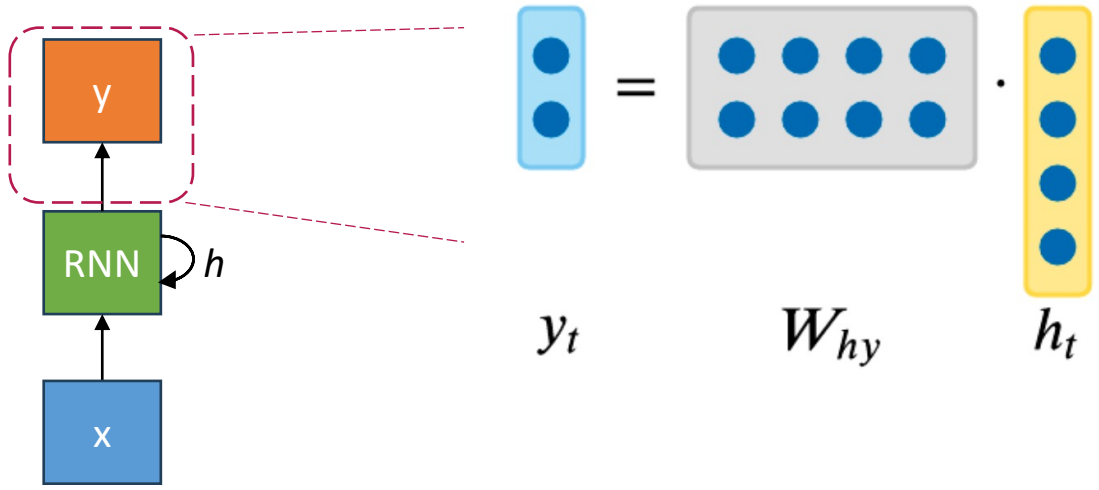
Recurrent neural network

- RNN can be represented as a recurrence formula of function f_W with parameters W:
  - $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$



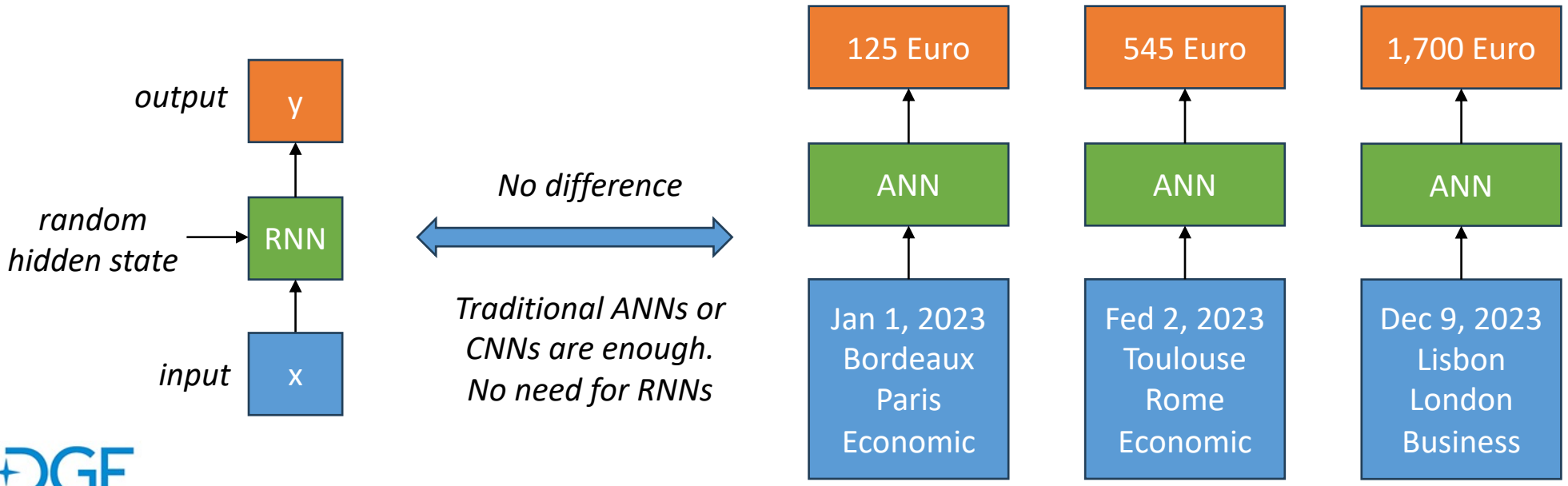
Recurrent neural network

- RNN can be represented as a recurrence formula of function f_W with parameters W:
  - $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
  - h_t can be used as the prediction, but in most cases, RNN will generate prediction y_t on top of h_t
    - $y_t = W_{hy}h_t$



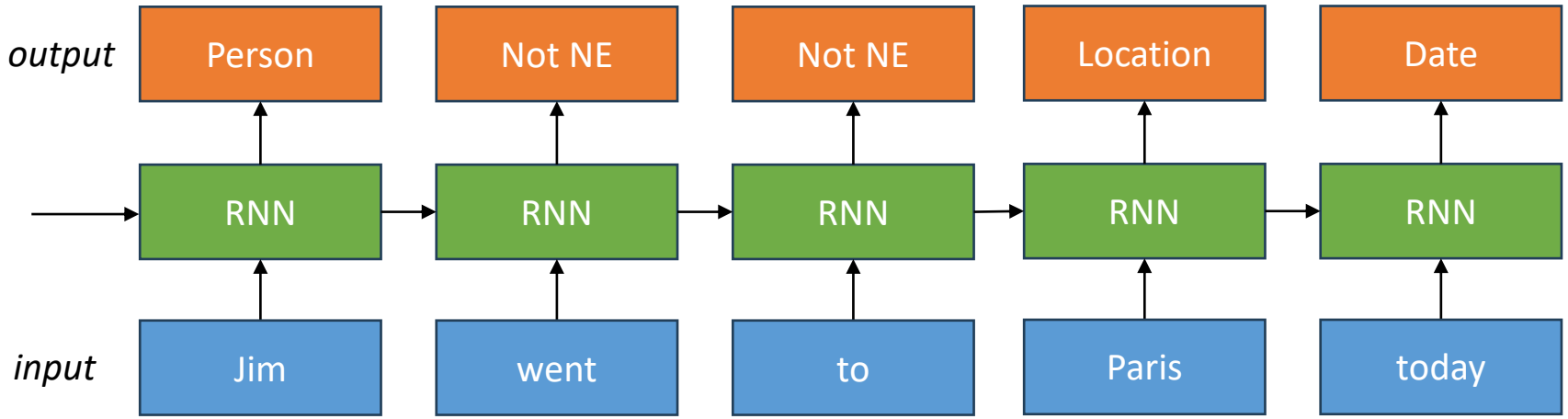
Recurrent neural network

- Different architectures of RNN:
  - The useless architecture: one-to-one
    - The RNN takes a single input sample with a randomly initialized hidden state and makes a single prediction
    - There is no sequence, so RNN works in the same way as traditional ANNs



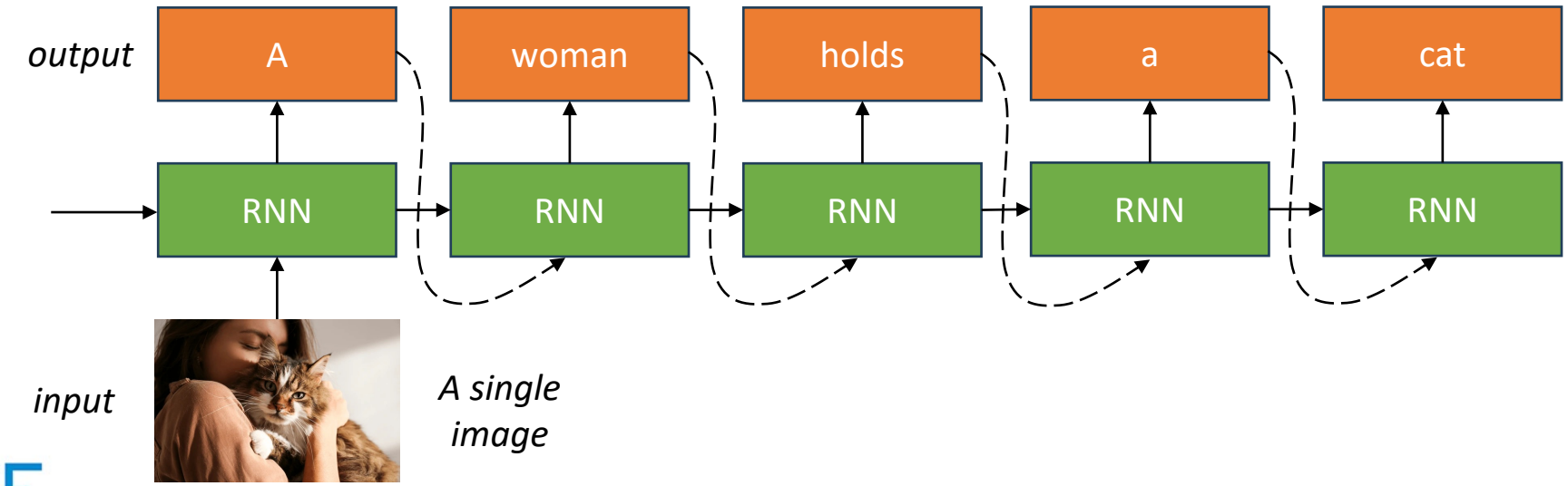
Recurrent neural network

- Different architectures of RNN:
  - The default architecture: many-to-many
    - Given a sequence of input samples, the RNN will make a prediction at each timestep
    - The output is also a sequence, and its length is the same as the length of input sequence
    - Example application: named-entity (NE) recognition



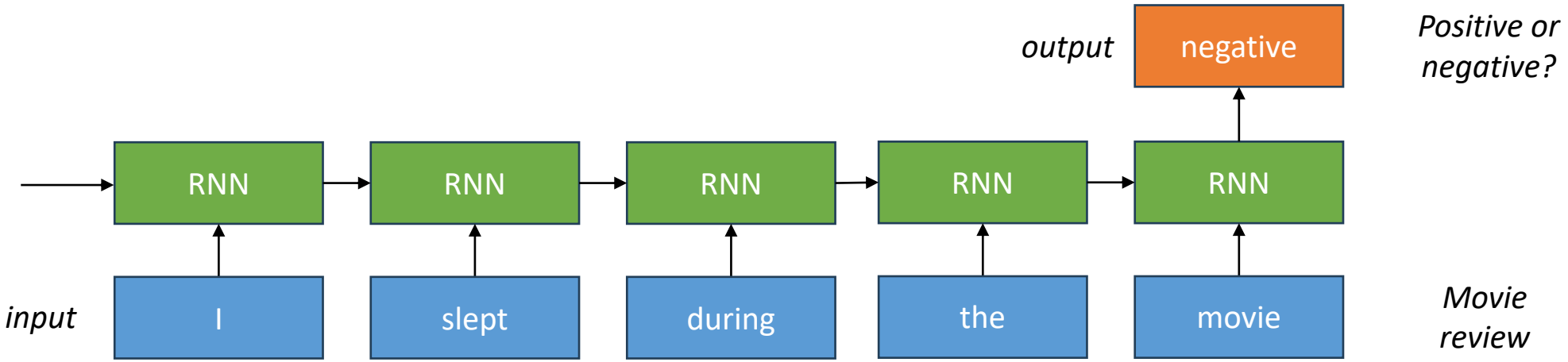
Recurrent neural network

- Different architectures of RNN:
  - one-to-many
    - Given a single input samples, the RNN will make a sequence of predictions
    - The previous prediction can be fed back to RNN as the new input at the current timestep
    - Example application: image caption



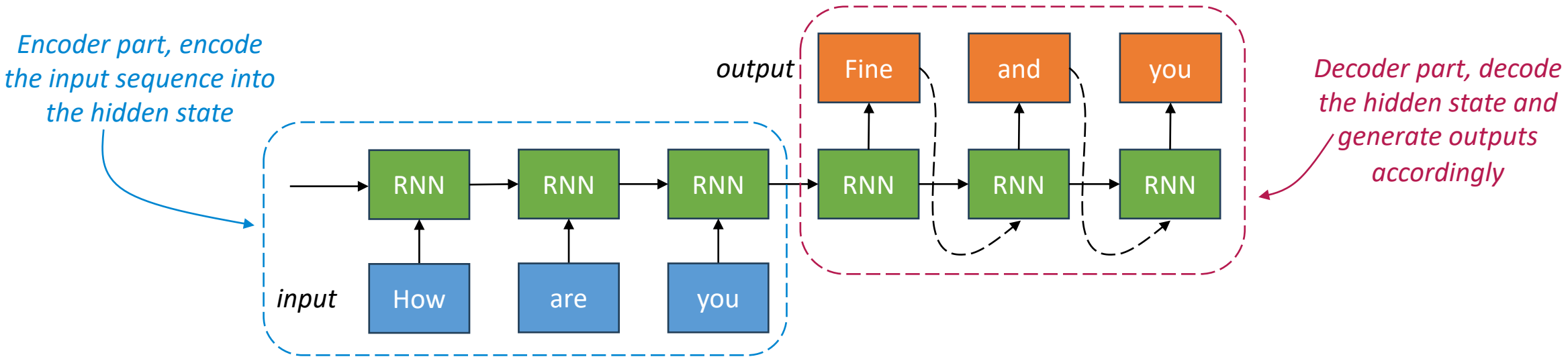
Recurrent neural network

- Different architectures of RNN:
  - many-to-one
    - Given a sequence of input samples, the RNN will make a single prediction at the end
    - The single prediction is determined by all input samples in the input sequence
    - Example application: text classification



Recurrent neural network

- Different architectures of RNN:
  - seq-to-seq
    - A variant of many-to-many, the encoder-decoder architecture
    - The RNN first takes a sequence of inputs, and then generate a sequence of outputs
    - Example application: chatbot



Recurrent neural network

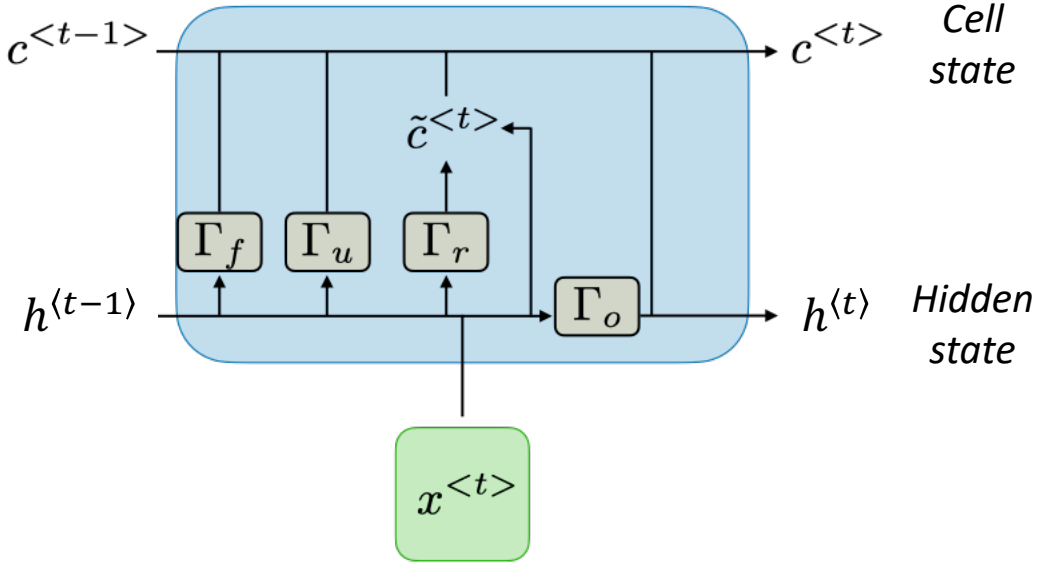
- PyTorch implementation
  - Gated Recurrent Unit (GRU)
    - Update gate
      - How much past should matter now?
    - Relevance gate
      - Drop previous information?
  - Long Short-Term Memory (LSTM)
    - A generalization of GRU

Recurrent Layers

`nn.RNNBase`	Base class for RNN modules (RNN, LSTM, GRU).
`nn.RNN`	Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence.
`nn.LSTM`	Apply a multi-layer long short-term memory (LSTM) RNN to an input sequence.
`nn.GRU`	Apply a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

Recurrent neural network

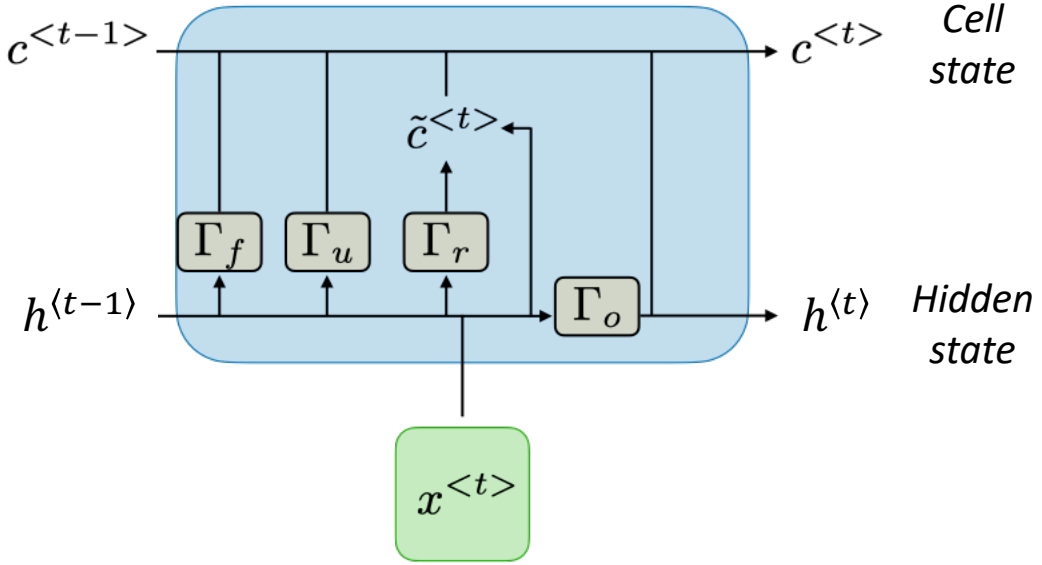
- Long Short-Term Memory (LSTM)
  - A generalization of GRU
  - Forgetting is as important as remembering
  - Update gate Γ_u:
    - How much past should matter now?
  - Relevance gate Γ_r :
    - Drop previous information or not?
  - Forget gate Γ_f :
    - How much previous information to forget?
  - Output gate Γ_o :
    - How much information to be shown in output?



Long Short-Term Memory

Recurrent neural network

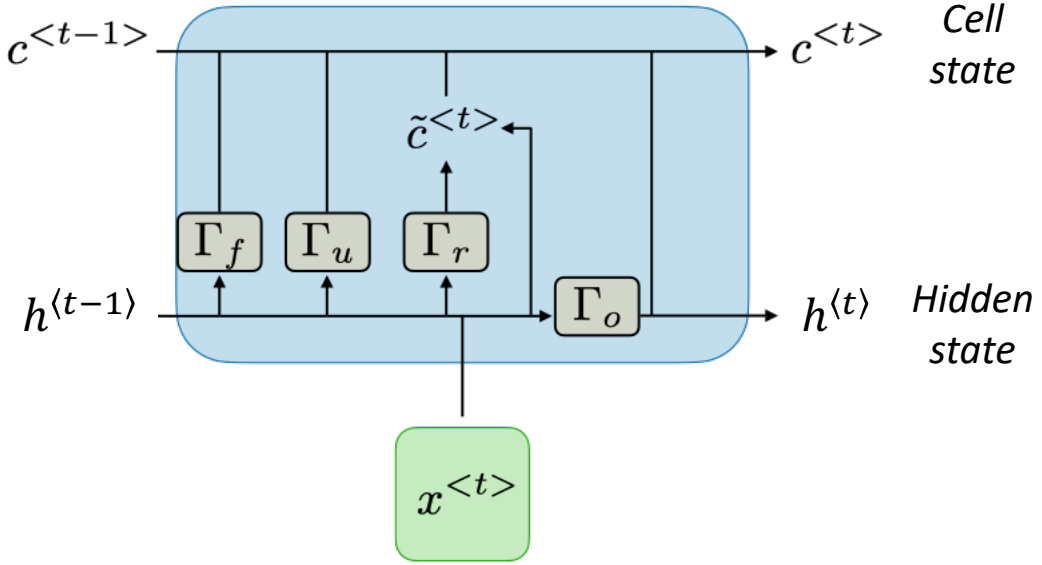
- PyTorch implementation of LSTM
  - torch.nn.LSTM()
    - input_size: the number of input features at each timestep
    - hidden_size: then number of hidden features
    - batch_first: if true, the input should be provided as the shape of (batch, seq, feature)
      - For example, (512, 10, 2) means
        - batch_size is 512, each batch has 512 sequences
        - seq_length is 10, each sequence has 10 samples
        - input_size is 2, each sample has 2 features



Long Short-Term Memory

Recurrent neural network

- PyTorch implementation of LSTM
  - torch.nn.LSTM()
    - Outputs: output, (h_n, c_n)
    - If batch_first = True:
      - output:
        - Shape: (batch_size, seq_length, hidden_size)
        - The sequence of all hidden states
      - h_n:
        - Shape: (1, batch_size, hidden_size)
        - The final hidden state
      - c_n:
        - Shape: (1, batch_size, hidden_size)
        - The final cell state



Long Short-Term Memory

Outline

- Recurrent neural network
- **Natural language processing**

Natural language processing

- General definition:
  - An interdisciplinary field of computer science and information retrieval
  - Give computers the ability to support and manipulate human language
- Common NLP tasks
  - Text classification: classify e-mail as spam or no-spam
  - Sentiment analysis: predict how positive/negative the meaning of a sentence is
  - Named-Entity recognition: extract certain entities like date, location, etc.
  - Machine translation: bilingual translation, English to French
  - Text generation: chatbot, ChatGPT → Artificial general intelligence
  - …

Natural language processing

- To solve NLP tasks with neural networks, we need to represent text as tensors (numbers)
  - Character-level representation:
    - One-hot encoding
      - For example, for the sentence “Hello!”
      - There are 5 different characters, each can be represented by a one-hot encoded vector

Hello! →

H	0	1	0	0	0
e	1	0	0	0	0
l	0	0	1	0	0
l	0	0	1	0	0
o	0	0	0	1	0
!	0	0	0	0	1

- *(seq_len, vector_len)*
- *A sequence of vectors*
- *The sequence length is the number of total characters.*
- *The vector length is the number of unique characters.*

Natural language processing

- To solve NLP tasks with neural networks, we need to represent text as tensors (numbers)
  - Character-level representation:
    - One-hot encoding
    - For English, if we only consider the letters, there are 26 unique letters
      - Each letter will be represented by a vector the length of which is 26.
    - Problem:
      - Each character by itself doesn't have much meaning, unless characters are aggregated as words.
  - Word-level representation

Natural language processing

- To solve NLP tasks with neural networks, we need to represent text as tensors (numbers)
  - Character-level representation
  - Word-level representation:
    - Word-level representation actually works at the token level, not the word level
    - Token: A word or any atomic parse element in the text
      - For example, “^_^” is treated as a token.
      - Tokenization: Convert the text into a sequence of tokens.

```
tokenizer = torchtext.data.utils.get_tokenizer('basic_english')
tokenizer('He said: hello')

>> ['he', 'said', 'hello']
```

Natural language processing

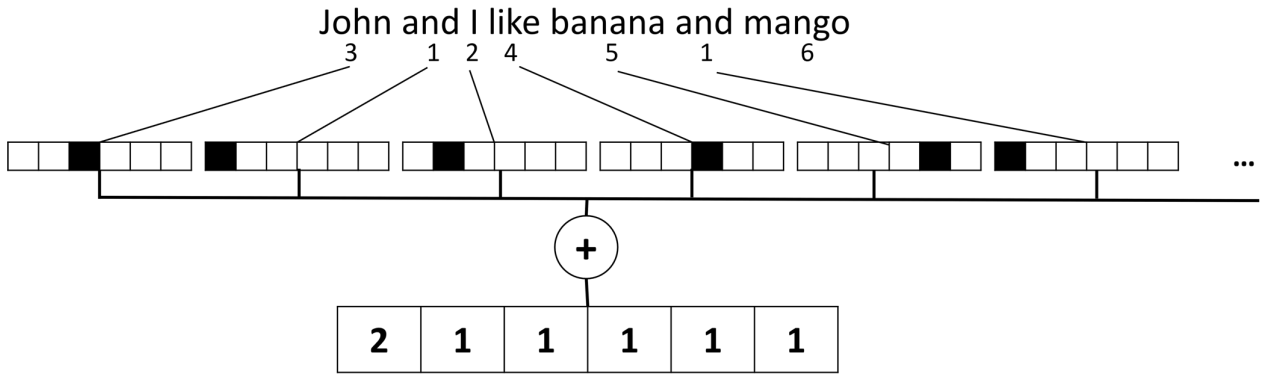
- Represent text as tensors
  - Character-level representation
  - Word-level representation:
    - Step 1. Tokenization: Convert the input text into a sequence of tokens
      - “He said: hello” → [‘he’, ‘said’, ‘hello’]
    - Step 2. Create a vocabulary:
      - Assign each unique token a unique number
      - Convert the sequence of tokens to a sequence of numbers
      - [‘he’, ‘said’, ‘hello’] → [0, 1, 2]
    - Step 3. One-hot encoding
      - Convert each number to a vector the length of which is the number of unique tokens in the text
      - [0, 1, 2] → [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

Natural language processing

- Represent text as tensors
  - Character-level representation
  - Word-level representation:
    - Problems:
      - The text will be represented by a sequence of vectors.
        - The sequence length is the number of total tokens.
        - The vector length is the number of unique tokens.
      - Suppose we have multiple sentences with different words.
        - The vector length is fixed.
        - The sequence length for each sentence is different.
  - Bag-of-words representation:
    - To represent text by one fixed-length vector instead of a sequence of vectors.

Natural language processing

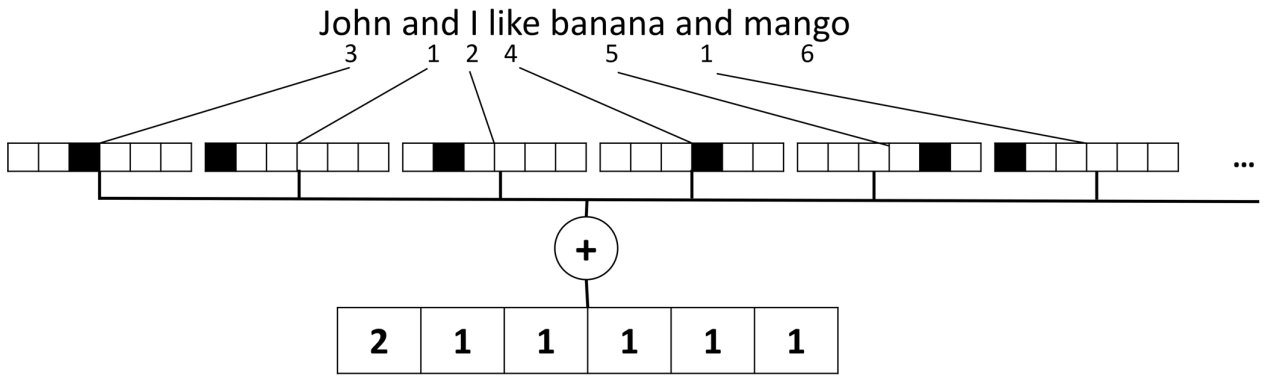
- Represent text as tensors
  - Character-level representation
  - Word-level representation
  - Bag-of-words representation:
    - To represent text by one fixed-length vector instead of a sequence of vectors.



- *There are 6 unique tokens in the text.*
- *Each token is represented by a vector*
  - *The vector size is 6*
- *The text is represented by the sum of all vectors instead of a sequence of vectors*
  - *The vector size is 6*
  - *The element in the vector reflects the frequency of one token in this text*

Natural language processing

- Represent text as tensors
  - Character-level representation
  - Word-level representation
  - Bag-of-words representation:
    - To represent text by one fixed-length vector instead of a sequence of vectors.



- *Advantage:*
  - *Each text is represented by a fixed-size vector which can reflect the context and the meaning of the text*
  - *For example, a politics article can have many 'president' tokens*
- *Disadvantage:*
  - *Certain common words, such as 'and', have the highest frequencies, masking out the tokens that are really important.*

Natural language processing

- Represent text as tensors
  - Character-level representation
  - Word-level representation:
    - Problems:
      - The text will be represented by a sequence of vectors.
        - The sequence length is the number of total tokens.
        - The vector length is the number of unique tokens.
      - Suppose we have 3,000 unique tokens in the text.
        - The vector length is 3,000.
        - We are dealing with high-dimensional data
  - Word embedding:
    - To represent tokens by lower-dimensional data

Natural language processing

- Represent text as tensors
  - Word embedding vs. One-hot encoding

John and I like banana and mango

John	0	0	1	0	0	0
and	1	0	0	0	0	0
I	0	1	0	0	0	0
like	0	0	0	1	0	0
banana	0	0	0	0	1	0
and	1	0	0	0	0	0
mango	0	0	0	0	0	1

- *One-hot encoding*
  - *The sequence length is the number of total tokens*
  - *The vector length is the number of unique tokens*
  - *The resulting matrix is too sparse*
    - *There are only 0 and 1*
    - *There are much more 0 than 1*
  - *High-dimensional data*

Natural language processing

- Represent text as tensors
  - Word embedding vs. One-hot encoding

John and I like banana and mango

John	0.1	0.2
and	0.3	0.1
I	0.2	0.1
like	0.5	0.7
banana	0.2	0.3
and	0.3	0.1
mango	0.1	0.7

- *Word embedding*
  - *The sequence length is the number of total tokens*
  - *The vector length is determined by us*
  - *The resulting matrix is dense*
    - *There can be float numbers*
  - *Lower-dimensional data*

Natural language processing

- Represent text as tensors
  - Word embedding vs. One-hot encoding

John and I like banana and mango

John	0	0	1	0	0	0
and	1	0	0	0	0	0
I	0	1	0	0	0	0
like	0	0	0	1	0	0
banana	0	0	0	0	1	0
and	1	0	0	0	0	0
mango	0	0	0	0	0	1

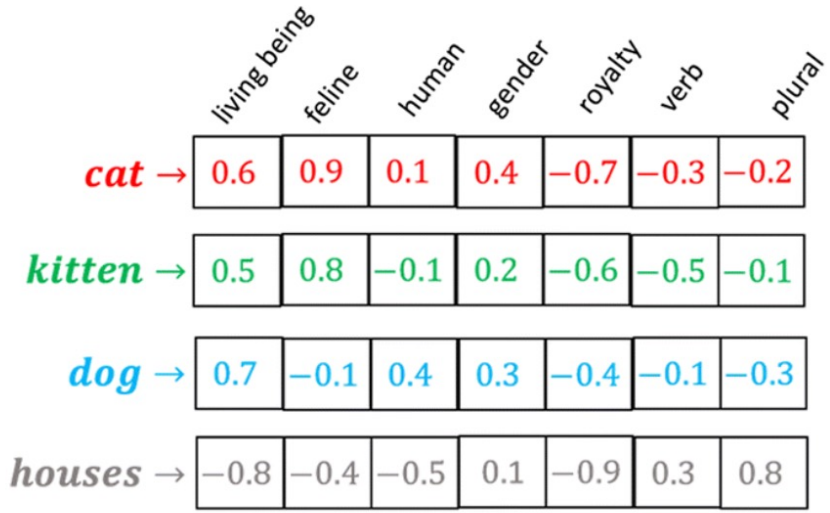
vs.

John	0.1	0.2
and	0.3	0.1
I	0.2	0.1
like	0.5	0.7
banana	0.2	0.3
and	0.3	0.1
mango	0.1	0.7

- Word embedding can be seen as the dimensionality reduction process
  - Use low-dimensional data to represent a lot of different tokens

Natural language processing

- Represent text as tensors
  - Word embedding
    - Embedding can be seen as the dimensionality reduction process
      - Use low-dimensional data to represent a lot of different tokens
    - How to perform the word embedding?
      - Manually defined embedding rules
        - Not a wise way

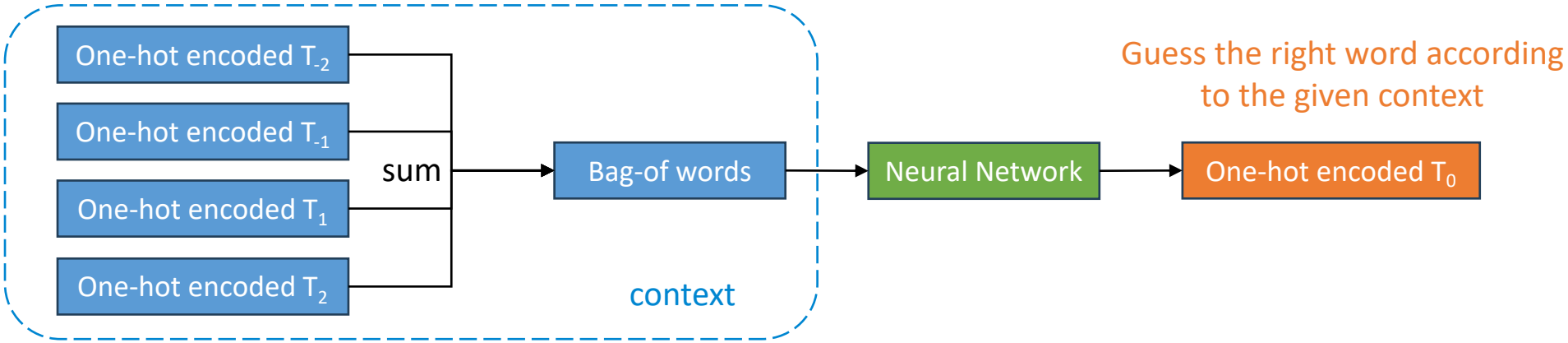


Natural language processing

- Represent text as tensors
  - Word embedding
    - Embedding can be seen as the dimensionality reduction process
      - Use low-dimensional data to represent a lot of different tokens
    - How to perform the word embedding?
      - Manually defined embedding rules
        - Not a wise way
      - Learn from data
        - Treat the word embedding as a layer of the neural network
        - Train the neural network on a particular task, thus learning the word embedding from data

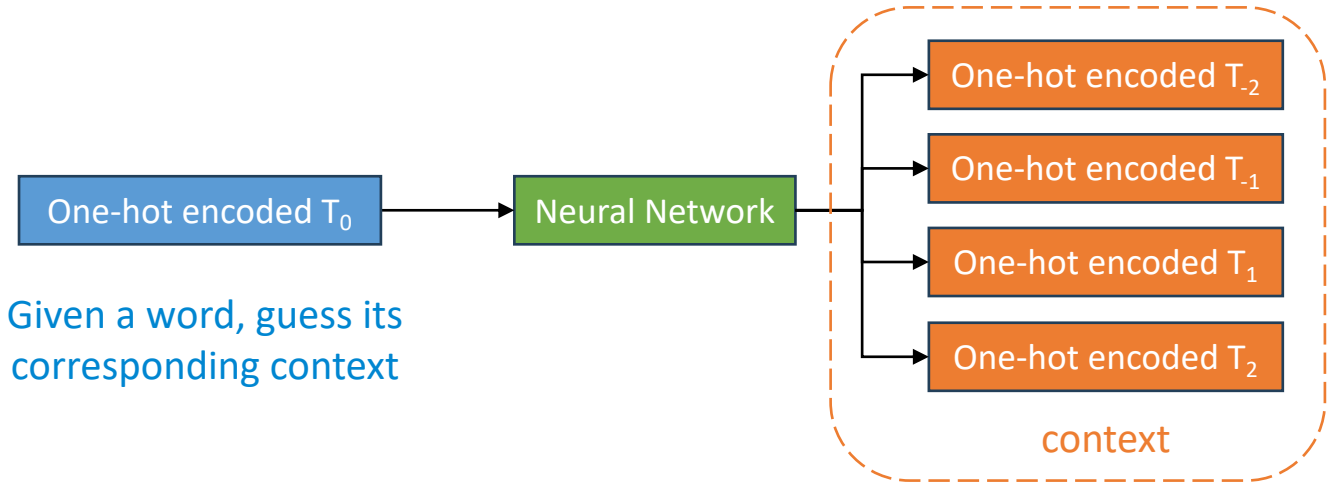
Natural language processing

- Represent text as tensors
  - Word embedding
    - Two common tasks to train a neural network for learning the word embedding:
      - Given a sequence of tokens [T_{-2}, T_{-1}, T_0, T_1, T_2]
      - CBoW (Continuous bag-of-words) task:
        - Use the bag-of-words representation of the surrounding tokens [T_{-2}, T_{-1}, T_1, T_2]
        - To predict the central token T_0



Natural language processing

- Represent text as tensors
  - Word embedding
    - Two common tasks to train a neural network for learning the word embedding:
      - Given a sequence of tokens [T_{-2}, T_{-1}, T_0, T_1, T_2]
      - CBoW (Continuous bag-of-words) task
      - Continuous skip-gram task:
        - Use the the central token T_0 to predict the surrounding tokens [T_{-2}, T_{-1}, T_1, T_2]

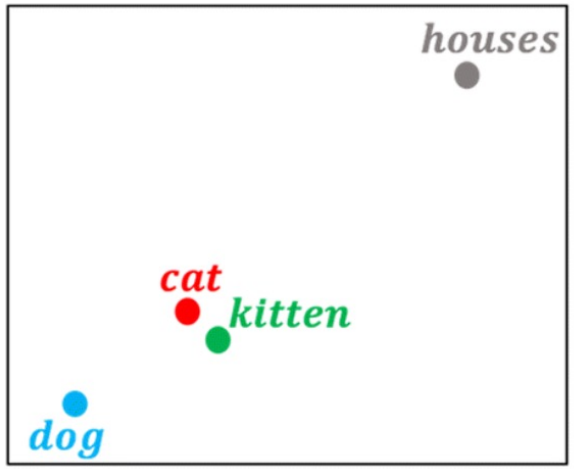


Natural language processing

- Represent text as tensors
  - Word embedding
    - Two common tasks to train a neural network for learning the word embedding:
      - CBoW (Continuous bag-of-words) task
      - Continuous skip-gram task
    - To learn a good word embedding, we usually need to train the network on large-scale data
      - Too time-consuming
      - Use pre-trained word embedding
        - Word2Vec by Google, 2013
          - A pre-trained version using the google news dataset
            - A vocabulary of 3 million unique tokens
            - 100 billion words in the dataset
            - Model size: 1.5GB

Natural language processing

- Represent text as tensors
  - Word embedding
    - Pre-trained Word2Vec
      - If we visualize different tokens in the feature space, we can find some interesting properties:
        - Similar tokens are close to each other

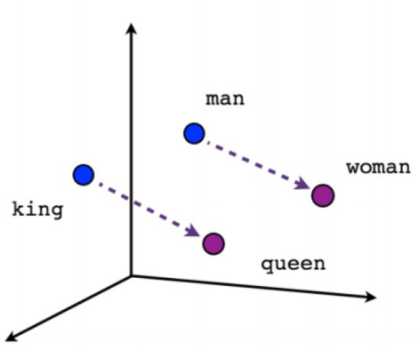


The embedding model learned synonyms, and represent them using similar vectors.

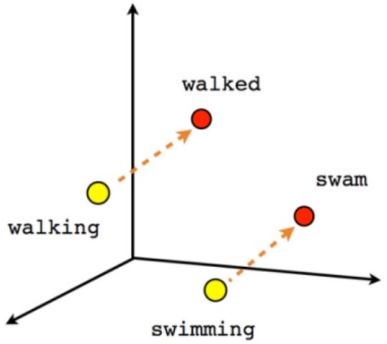
Natural language processing

- Represent text as tensors
  - Word embedding
    - Pre-trained Word2Vec
      - If we visualize different tokens in the feature space, we can find some interesting properties:
        - Similar tokens are close to each other
        - The relative position between two tokens represent similar relationships.

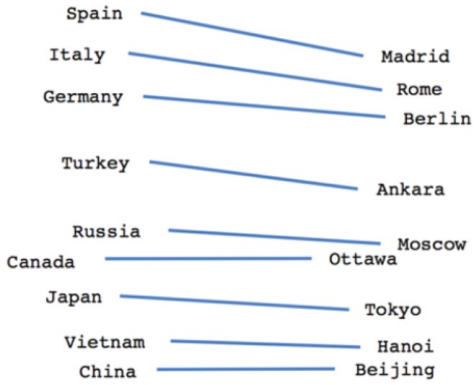
$Vec_{king} - Vec_{man} + Vec_{woman} \sim Vec_{queen}$



Male-Female



Verb tense



Country-Capital

Natural language processing

- Represent text as tensors
  - Word-level representation:
    - Step 1. Tokenization: Convert the input text into a sequence of tokens
      - “He said: hello” → [‘he’, ‘said’, ‘hello’]
    - Step 2. Create a vocabulary:
      - Assign each unique token a unique number
      - Convert the sequence of tokens to a sequence of numbers
      - [‘he’, ‘said’, ‘hello’] → [0, 1, 2]
    - Step 3. ~~One-hot encoding~~ Word embedding
      - Convert each number to a fixed-length vector using word-embedding
      - The vector length is determined by us
        - More unique tokens in the vocabulary requires longer vectors

Natural language processing

- Represent text as tensors
  - Word-level representation:
    - Step 1. Tokenization: Convert the input text into a sequence of tokens
      - "He said: hello" → ['he', 'said', 'hello']
    - Step 2. Create a vocabulary:
      - Assign each unique token a unique number
      - Convert the sequence of tokens to a sequence of numbers
      - ['he', 'said', 'hello'] → [0, 1, 2]
    - Step 3. ~~One-hot encoding~~ Word embedding

Data pre-processing when loading the text dataset and building the data pipeline

Put word embedding as the first layer in the neural network

`nn.Embedding`	A simple lookup table that stores embeddings of a fixed dictionary and size.
`nn.EmbeddingBag`	Compute sums or means of 'bags' of embeddings, without instantiating the intermediate embeddings.

Hands-on Exercise

- **Exercise 08 RNN & NLP - Instruction**
- Exercise 08 RNN & NLP - Assignment