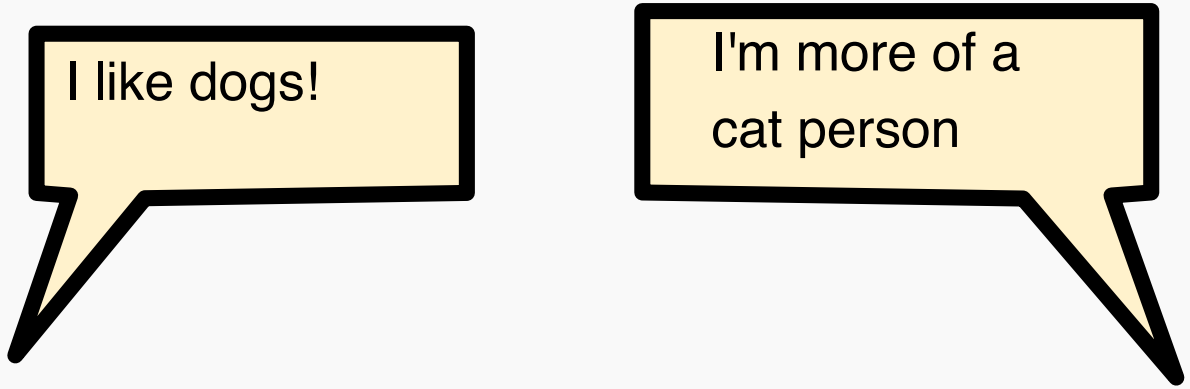
Intro to Language Model and Traditional Language Modeling

Natural Language Processing

Natural Language Processing is the design and analysis of computational algorithms and representations for processing natural human language.



Language Modelling

Today, we heavily focus on Language Modelling (LM) because:

1. It's foundational for nearly all NLP tasks.
2. Since we're ultimately modelling a sequence, LM approaches are generalizable to any type of data, not just text.
3. The data is readily available in huge quantities.

Language Modelling: **starting point**

We need to split the text into smaller units, or **tokens**.

The text will be a sequence of tokens, each containing a part of the information. The entire set of tokens is called **vocabulary**.

The bigger the vocabulary the bigger training set required.

Language Modelling: **tokenization**

We need to define the basic unit (token) of a sentence.

First Approach: whitespace

Split the words on whitespaces only.

Language Modelling: **tokenization**

Second Approach: sub-word tokenization

Split the words on statistically significant fragments.

The token loses their direct interpretability but makes a more flexible approach.

Tokenization is an area of research on itself.

Language Modelling: **Evaluation**

How can we evaluate a language model?

Base Approach: Transform everything into a classification problem.

In training, predict a word in the vocabulary or a class.

This is an extrinsic evaluation, since depends on some external task.

Language Modelling: **Evaluation**

How can we evaluate a language model?

Base Approach: Transform everything into a classification problem

$$P(x \mid \text{the}, \text{award-winning}, \text{actor}, \text{arrived}) = \begin{bmatrix} \text{action} : 0.01 \\ \text{boston} : 0.05 \\ \text{false} : 0.01 \\ \text{today} : 0.31 \\ \text{yesterday}: 0.25 \end{bmatrix}$$

Language Modelling: **Evaluation**

How can we evaluate a language model?

Base Approach: Transform everything into a classification problem

$$P(x \mid \text{the, movie ,could ,have, been, better}) = \begin{cases} \text{Good: 0.3} \\ \text{Bad: 0.7} \end{cases}$$

Language Modelling: **Evaluation**

For more general language models, an entire ecosystem of benchmarks exists. They include a variety of different NLP tasks such as sentiment analysis, question answering, NER, among others.

BLEU: **B**i**L**ingual **E**valuation **U**nderstudy

SQuAD : **S**tanford **Qu**estion **A**nswering **D**ataset

MS MACRO: **MA**chine **R**eading **CO**mprehension **D**ataset

GLUE and SuperGLUE: **G**eneral **L**anguage **U**nderstanding **E**valuation

Language Modelling: **Unigrams**

How can we build a language model?

Naive Approach: Unigram model

Assume each word is independent of all others

Count how often each word occurs (in the training data).

Language Modelling: **Bigrams**

How can we build a language model that uses context?

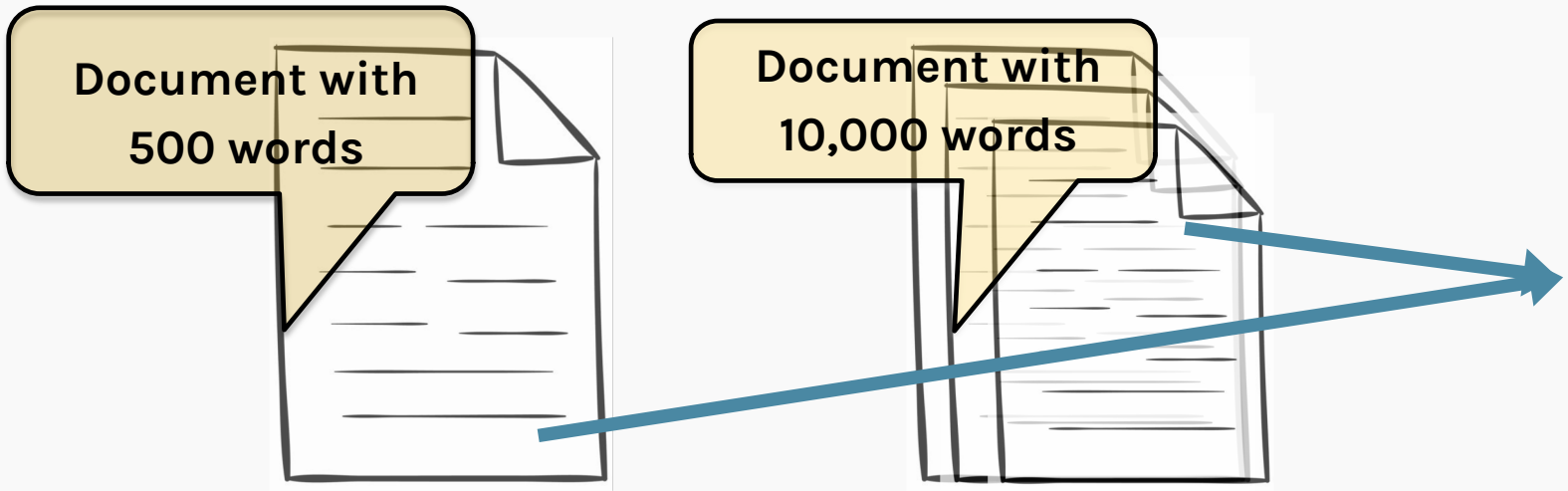
Easiest Approach: bigram model

Look at *pairs* of consecutive words

Language Modelling: **TF-IDF**

- **Term Frequency :** This measures the frequency of a word in a document. This highly depends on the length of the document and the generality of word.

Solution

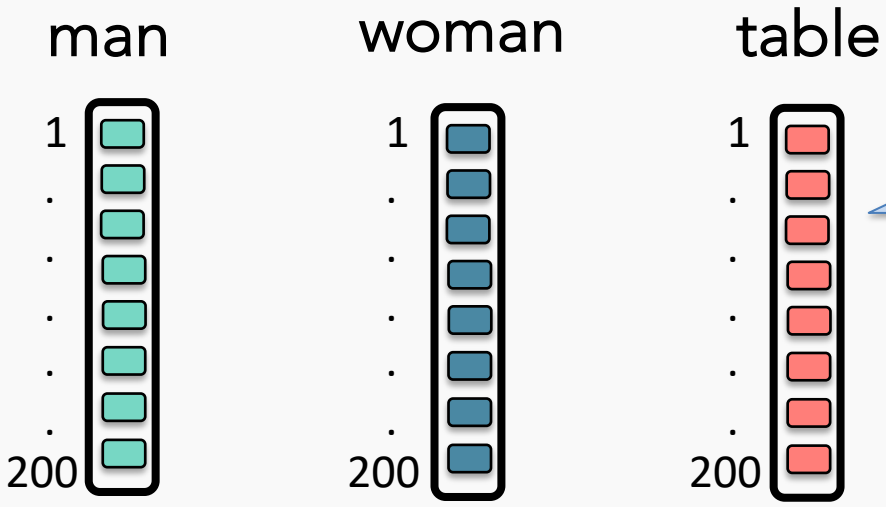


Normalize the count value - Divide the word count with the total number of words in the document.

Language Modelling: **Neural Networks**

IDEA: Let's use a neural network!

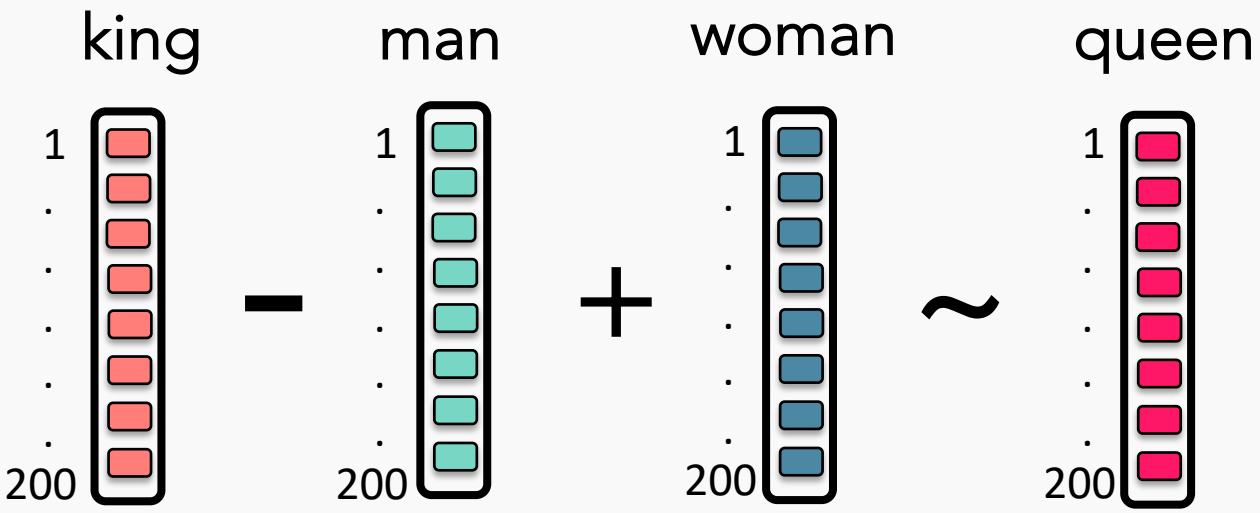
First, each word is represented by a word embedding (e.g., vector of length 200)



- Each rectangle is a *floating-point* scalar
- Words that are more *semantically similar* to one another will have **embeddings** that are also proportionally similar
- We can *use pre-existing* word embeddings that have been trained on gigantic corpora

Language Modelling: **Neural Networks**

These word embeddings are so rich that you get nice properties:

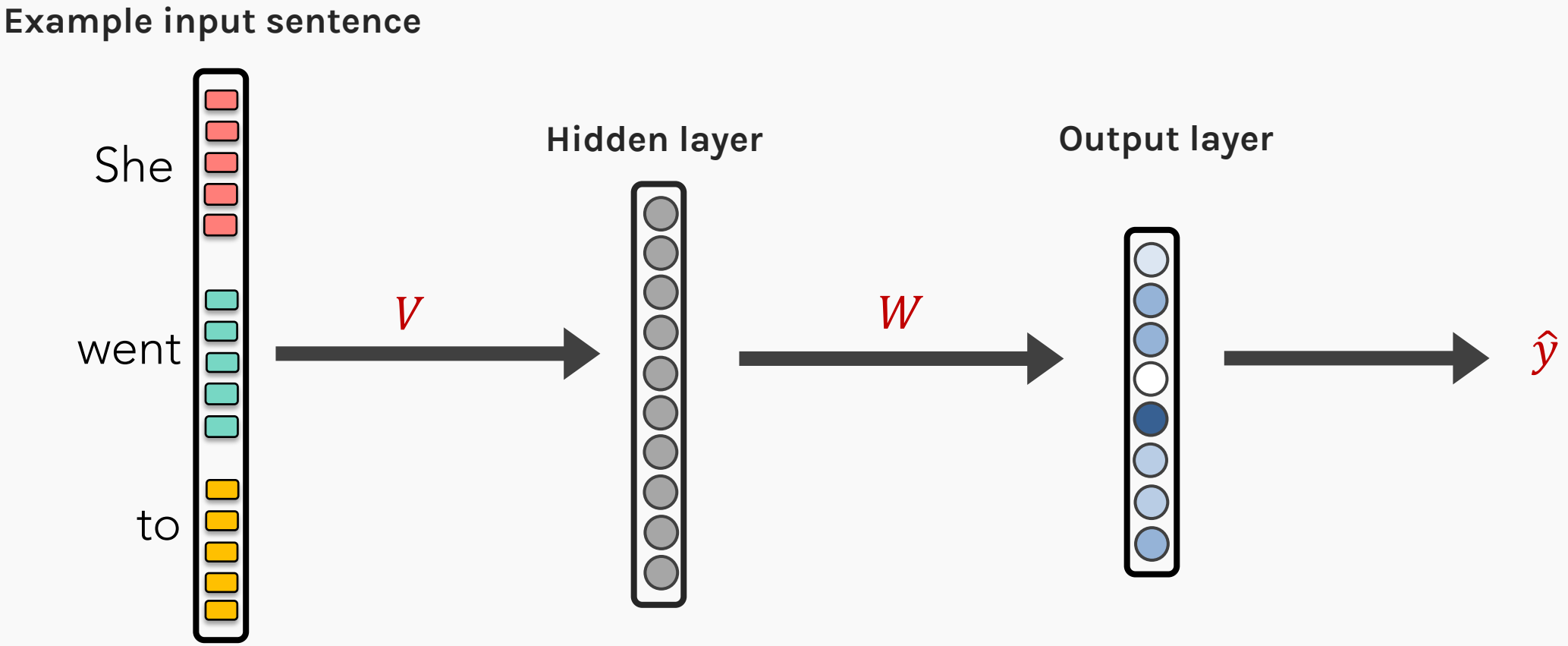


Word2vec: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
GloVe: https://www.aclweb.org/anthology/D14-1162.pdf

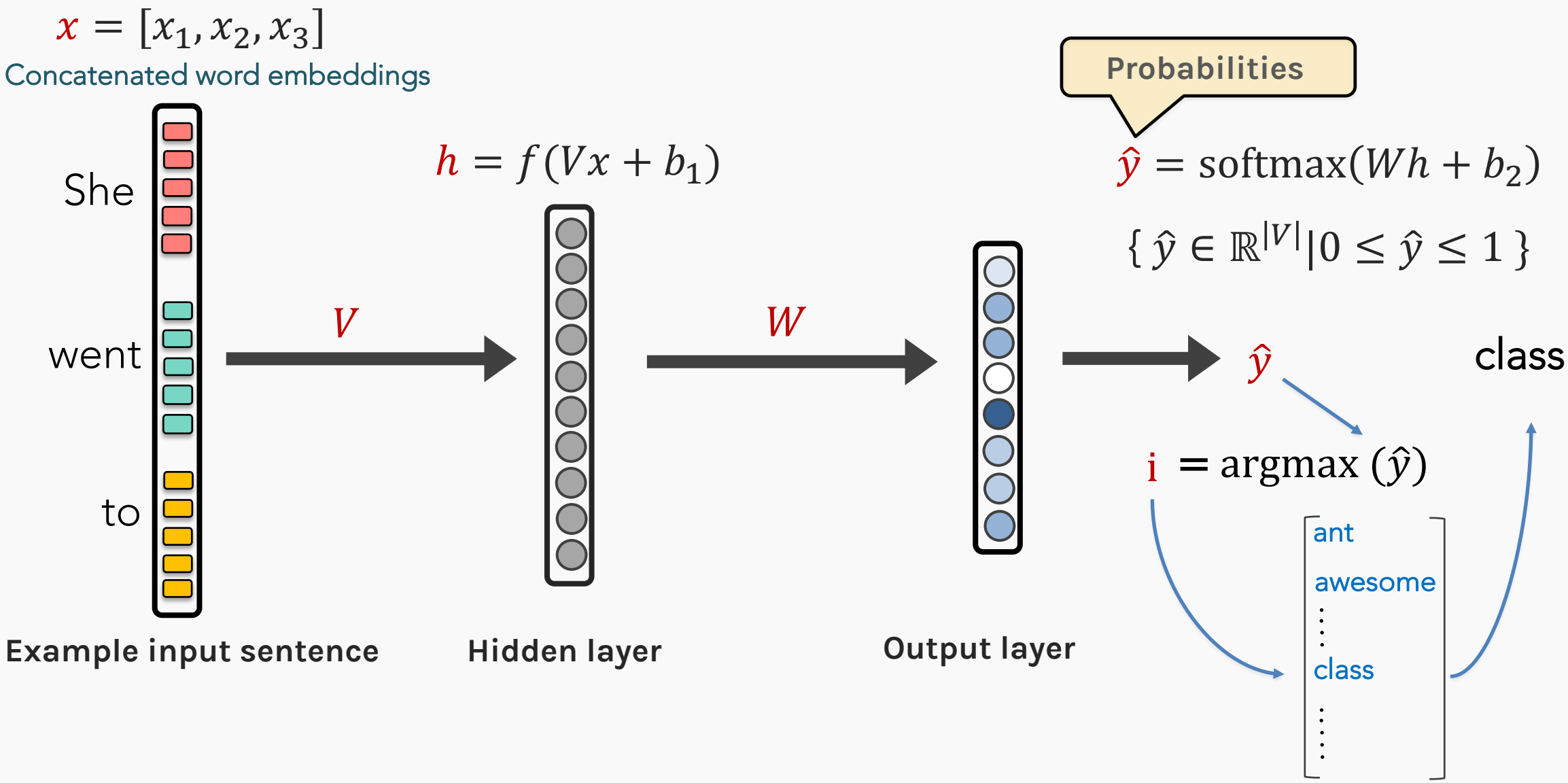
Language Modelling: **Feed-Forward Neural Network**

Neural Approach #1: Feed-forward neural net

General Idea: using *windows* of words, predict the next word



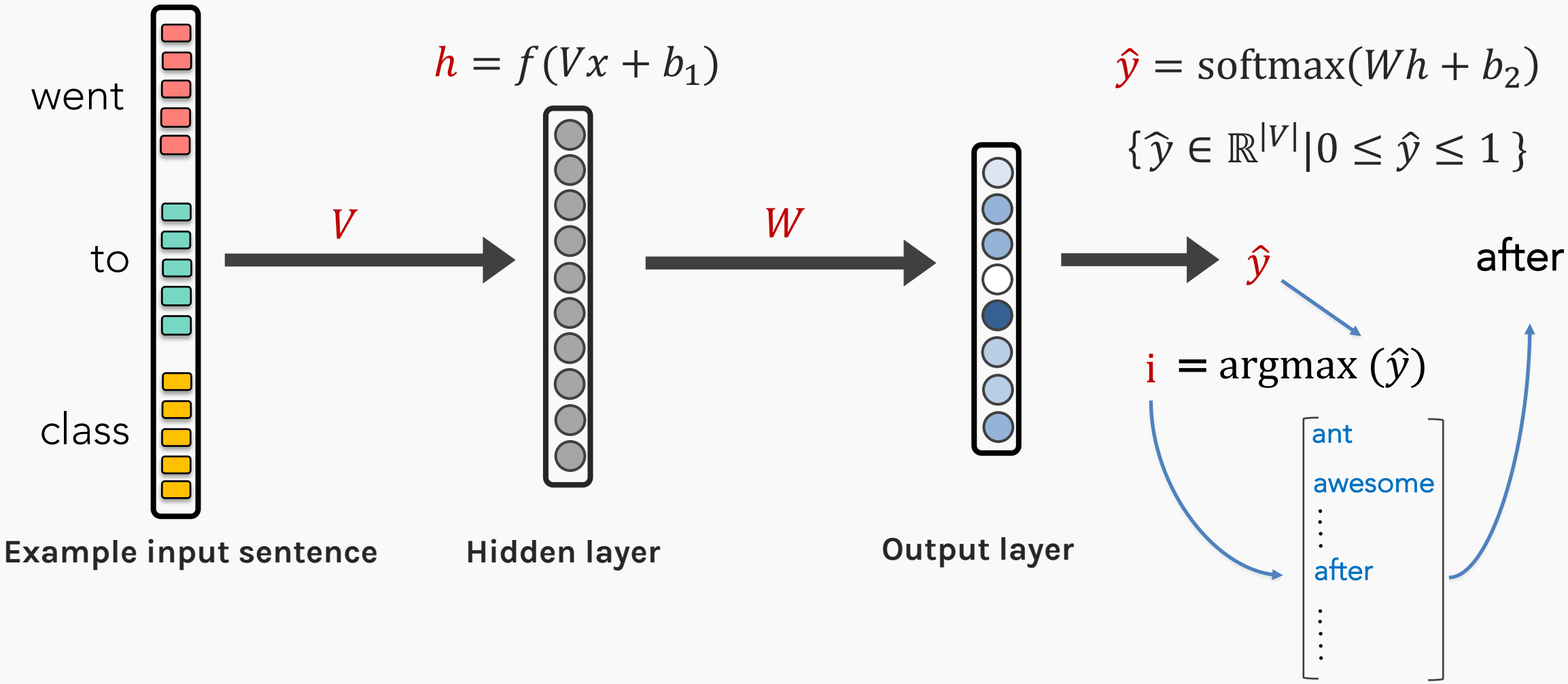
Language Modelling: **Feed-Forward Neural Network**



Language Modelling: **Feed-Forward Neural Network**

$x = [x_2, x_3, x_4]$

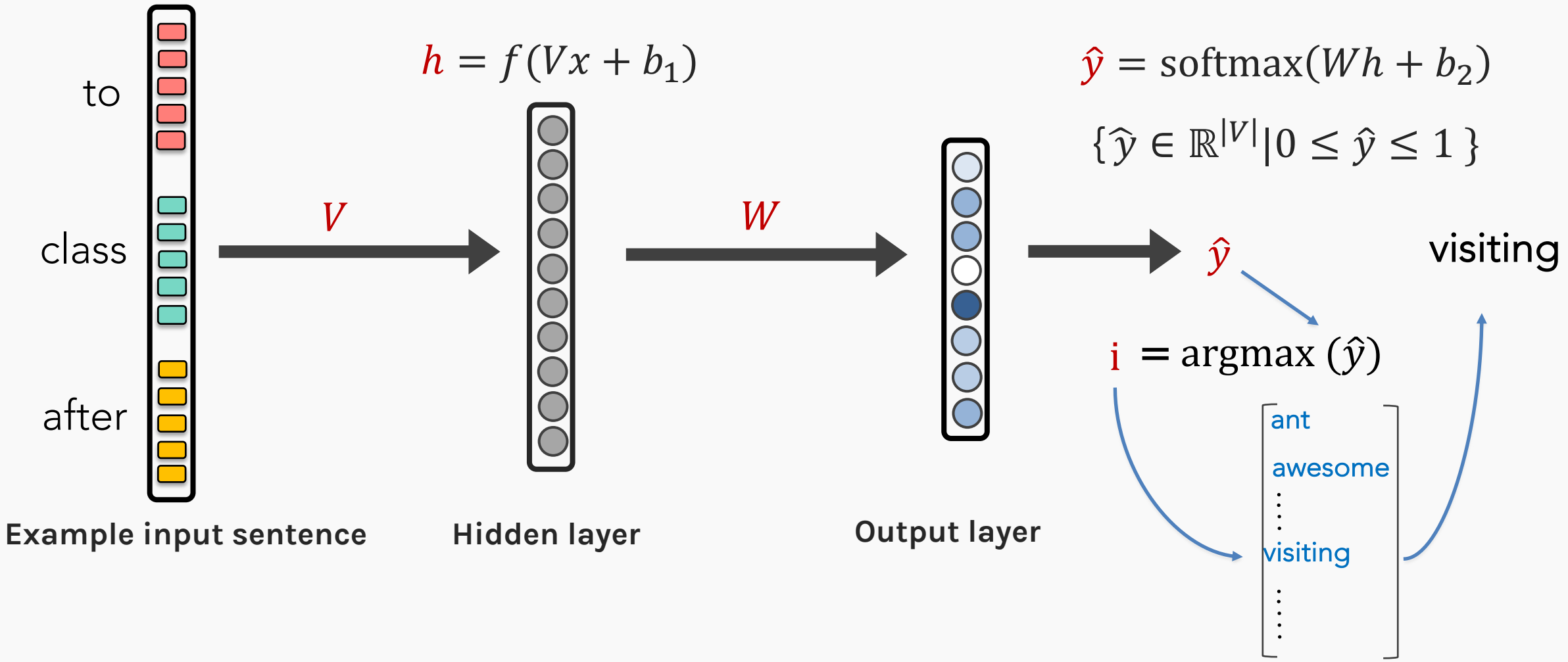
Concatenated word embeddings



Language Modelling: **Feed-Forward Neural Network**

$x = [x_3, x_4, x_5]$

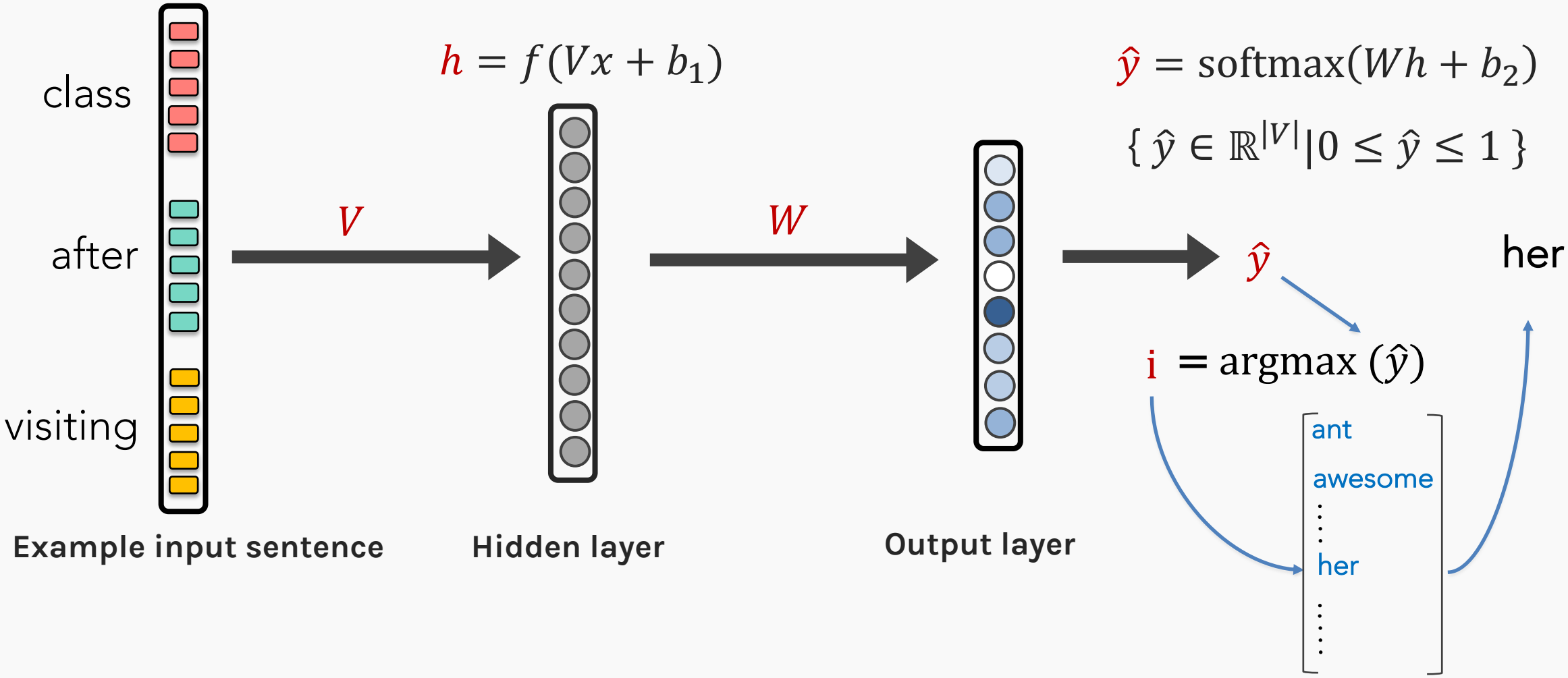
Concatenated word embeddings



Language Modelling: **Feed-Forward Neural Network**

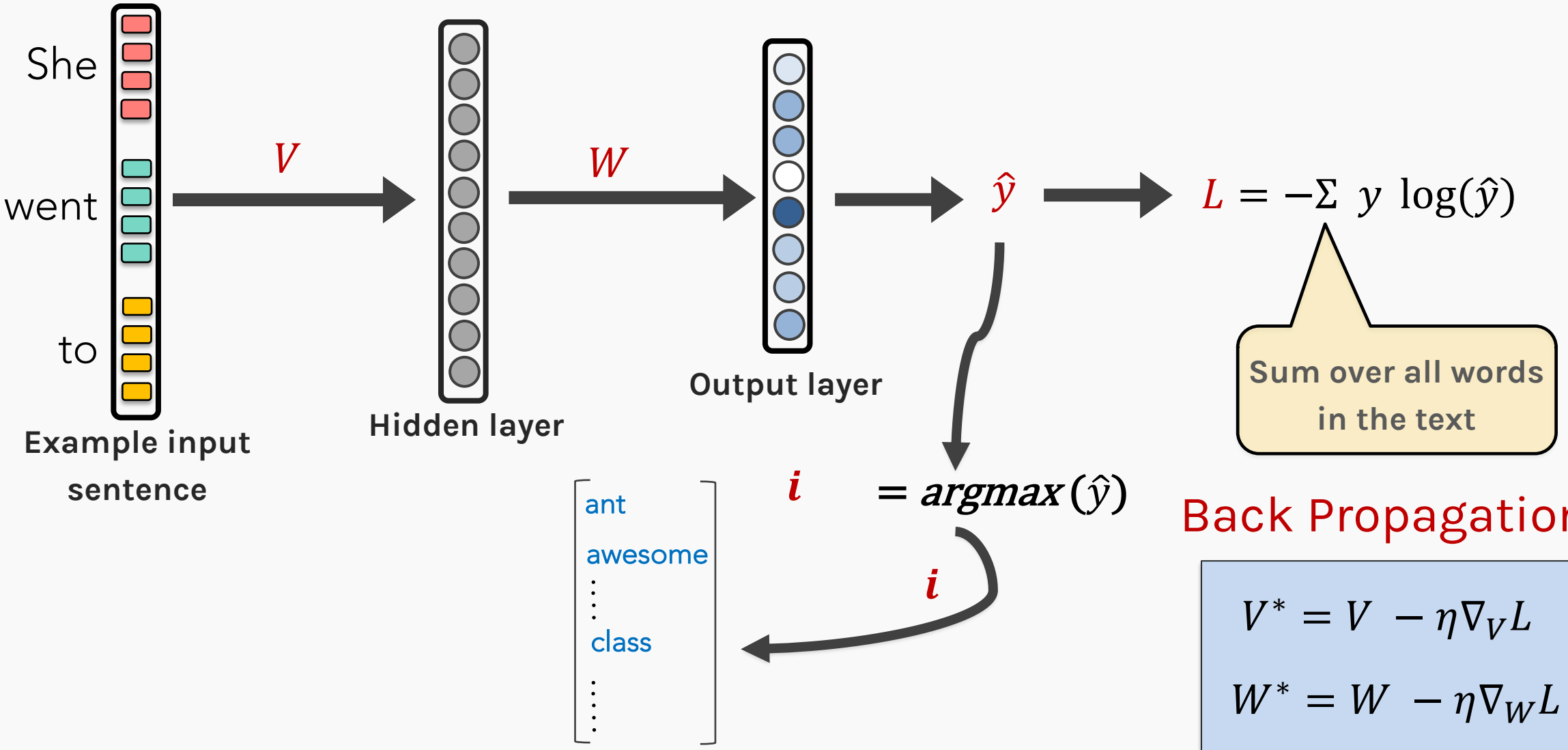
$x = [x_4, x_5, x_6]$

Concatenated word embeddings



Language Modelling: **Feed-Forward Neural Network** Training

$x = [x_1, x_2, x_3]$ $h = f(Vx + b_1)$ $\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$



Back Propagation

$$V^* = V - \eta \nabla_V L$$

$$W^* = W - \eta \nabla_W L$$

Language Modelling : **Feed-Forward Neural Network**

FFNN Strength

- No sparsity issues (it's okay if we've never seen a word)
- No storage issues (we never store counts)

FFNN Issues

- Fixed-window size can never be big enough. Need more context
  - Requires inputting entire context just to predict one word
  - Increasing window size adds many more weights
- The weights awkwardly handle word position
- No concept of time

Language Modelling

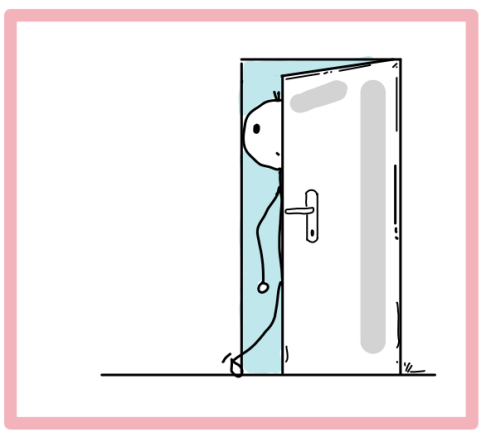
We especially need a system that:

- Has a concept of an “infinite” past, not just a fixed window
- For each new input, output the most likely next event (e.g., word)

Recurrent Neural Networks

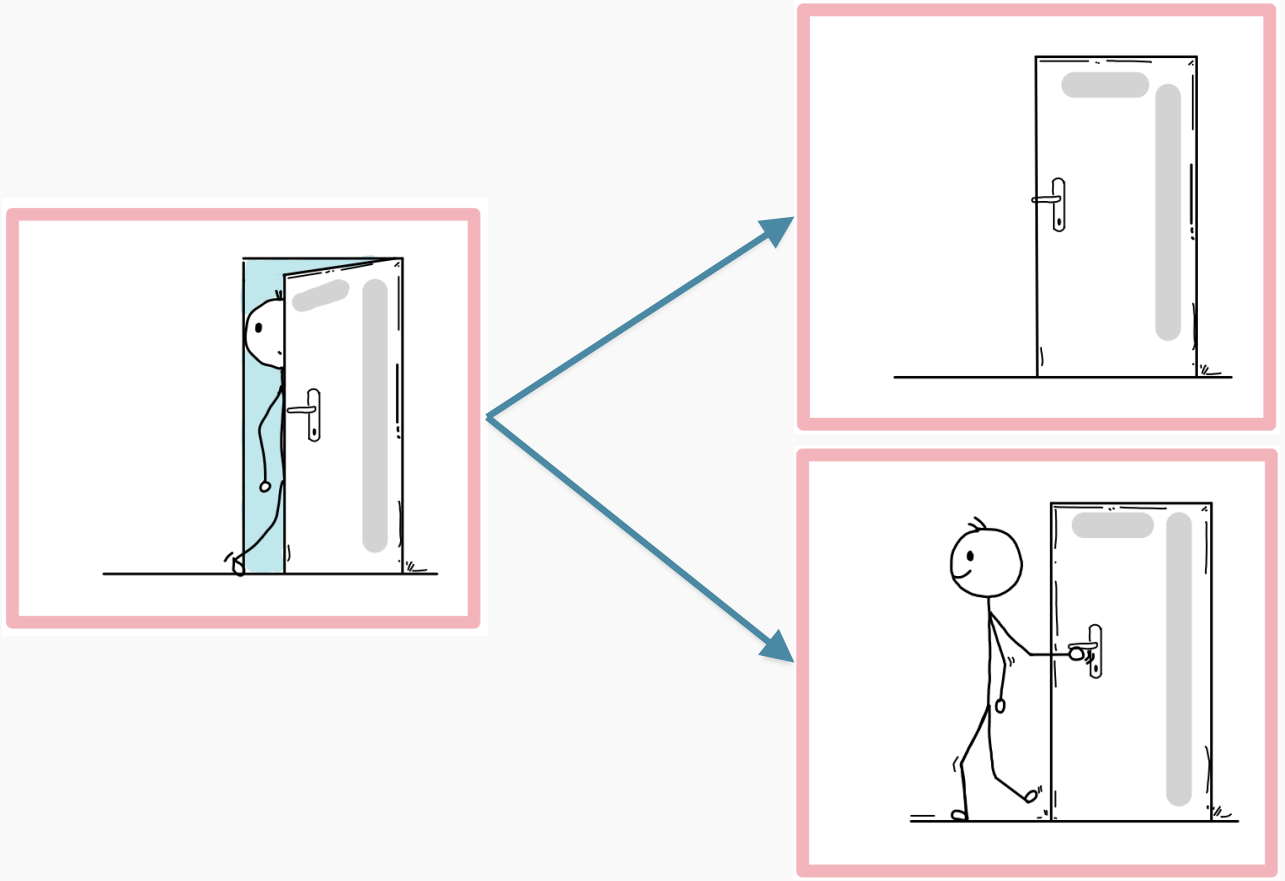
Motivation behind RNNs : **Sequences**

Given this frame, what do you think is the next likely frame?



Motivation behind RNNs : **Sequences**

Given this frame, what do you think is the next likely frame?

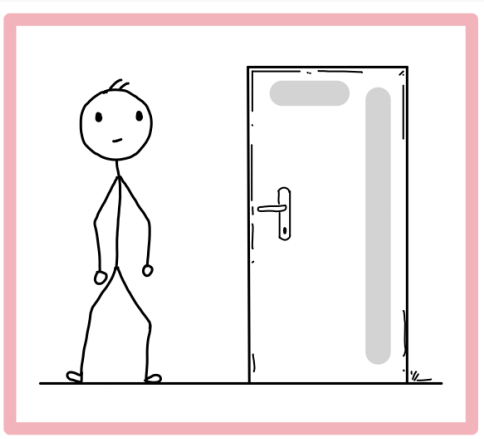


There are 2 options, either the person is going in or is coming out?

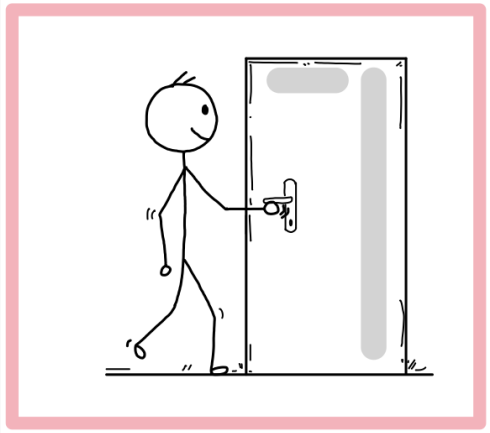
Based on only the first frame, it would be difficult to predict the next frame.

Motivation behind RNNs : **Sequences**

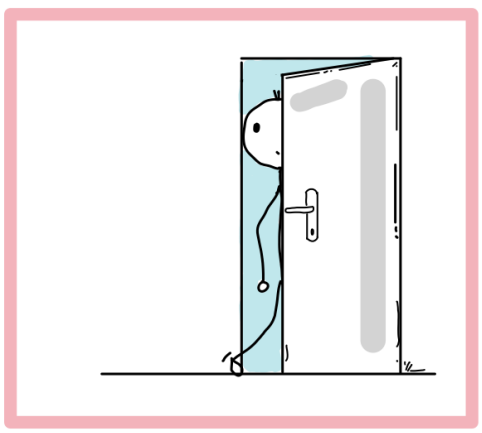
Frame 1



Frame 2



Frame 3

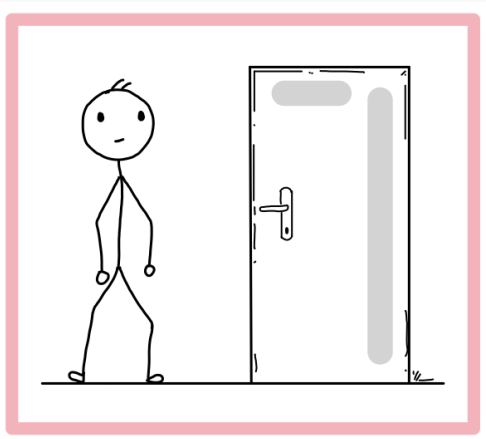


?

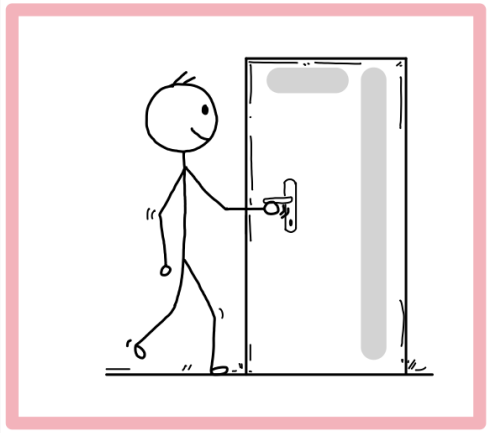
However, if we were to give the model the previous frames it would be easy to predict the next

Motivation behind RNNs : **Sequences**

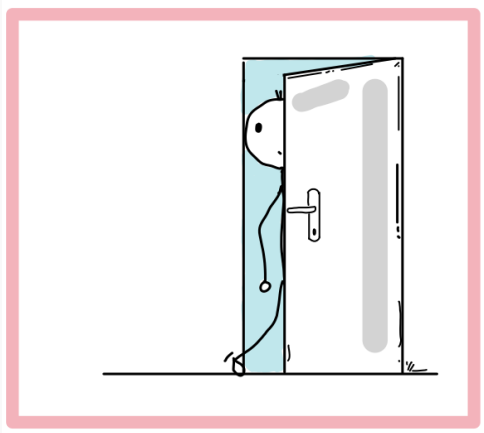
Frame 1



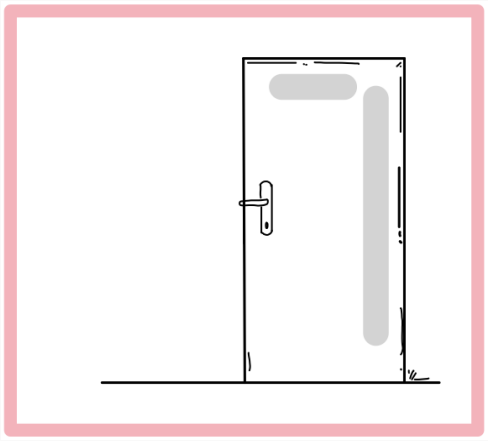
Frame 2



Frame 3



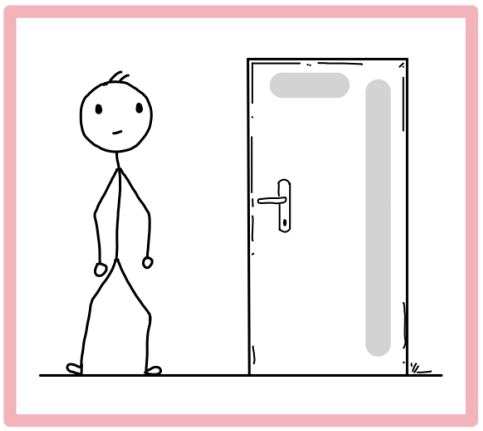
Frame 4



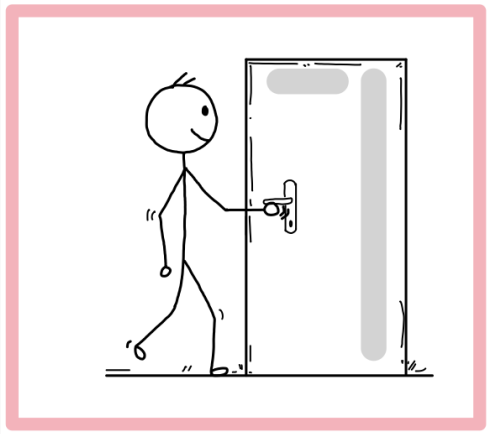
However, if we were to give the model the previous frames it would be easy to predict the next

Motivation behind RNNs : **Sequences**

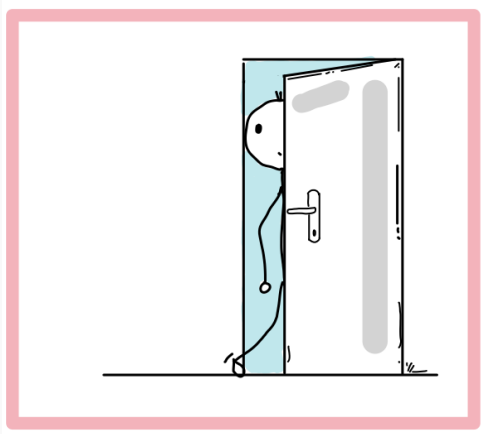
Frame 1



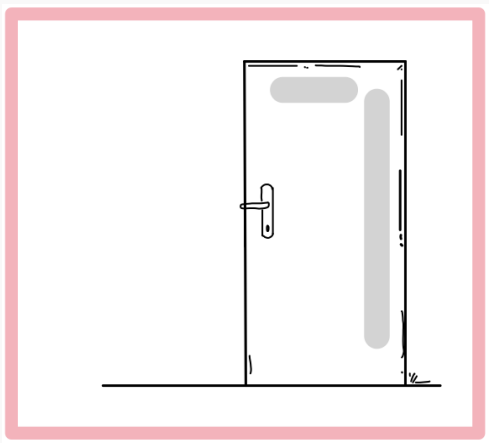
Frame 2



Frame 3



Frame 4

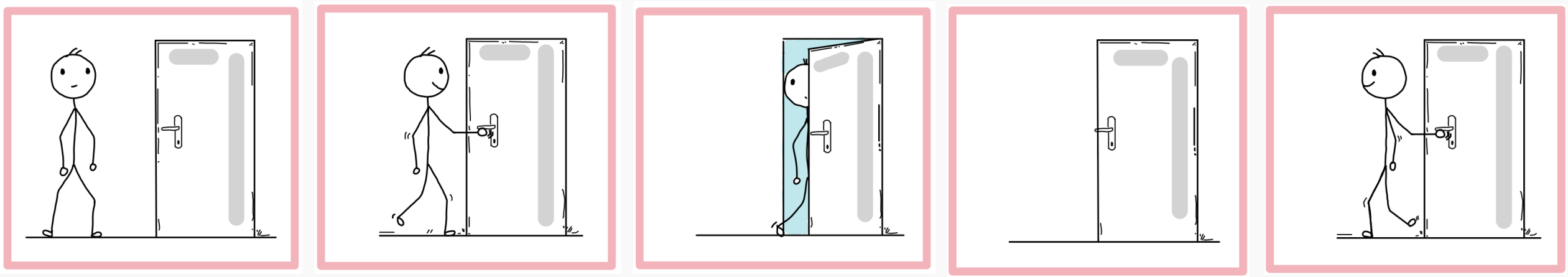


?

However, if we were to give the model the previous frames it would be easy to predict the next

Motivation behind RNNs : **Sequences**

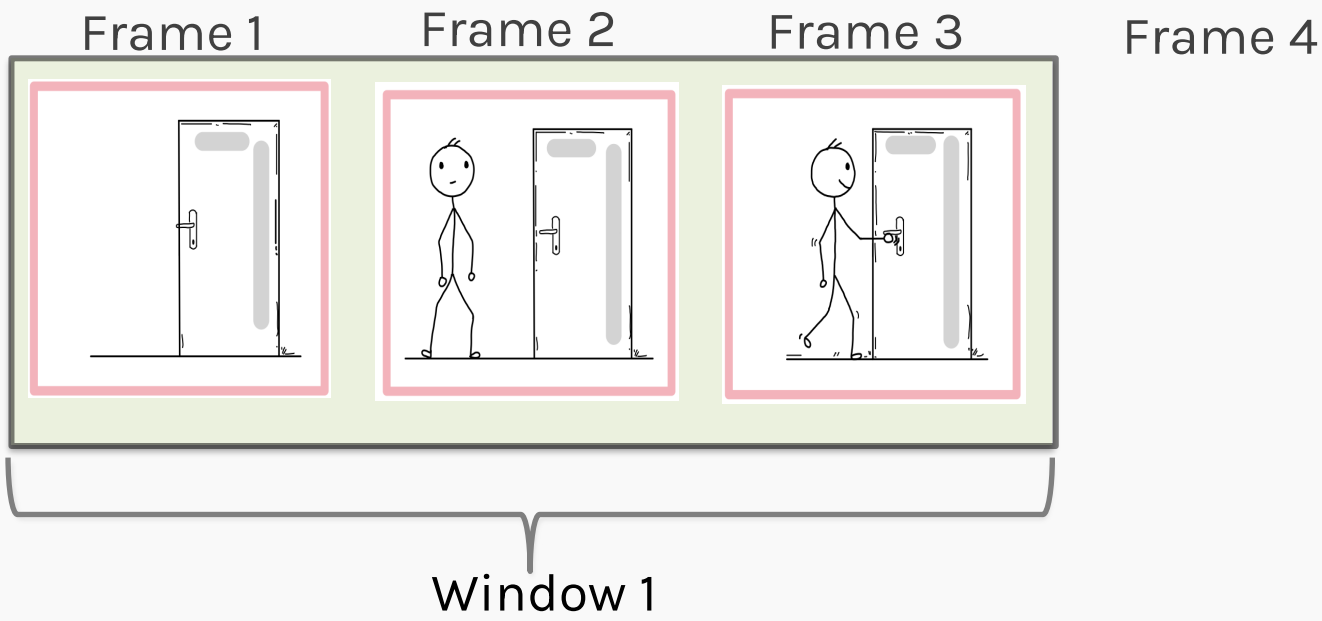
Frame 1 | Frame 2 | Frame 3 | Frame 4 | Frame 5



It would be easy for a model to predict the next frame if the previous **sequence** is provided.

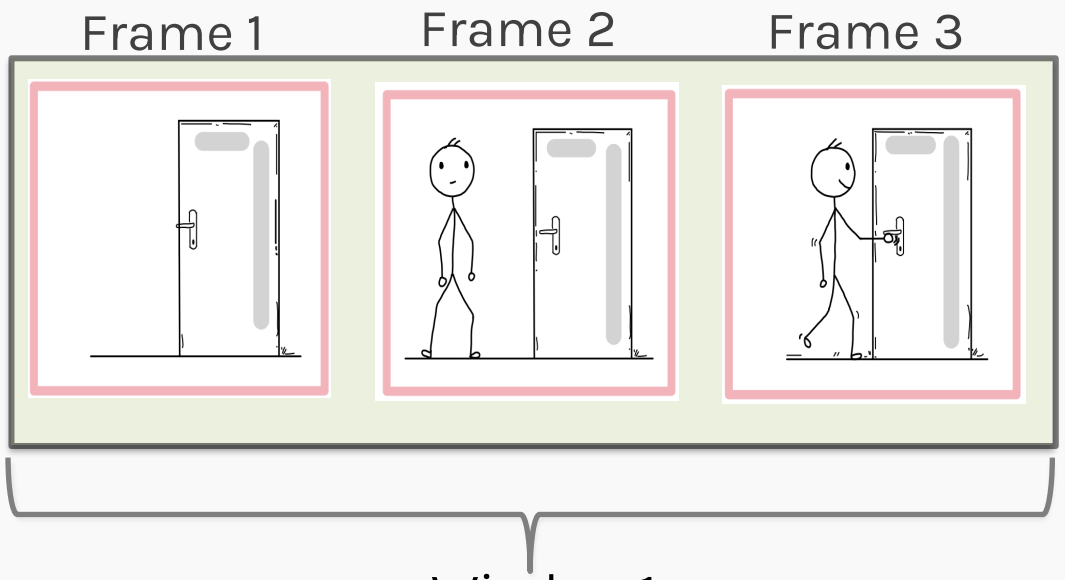
Sequences play an important role for forecasting and predictions.

Motivation behind RNNs : **Windowing**



If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

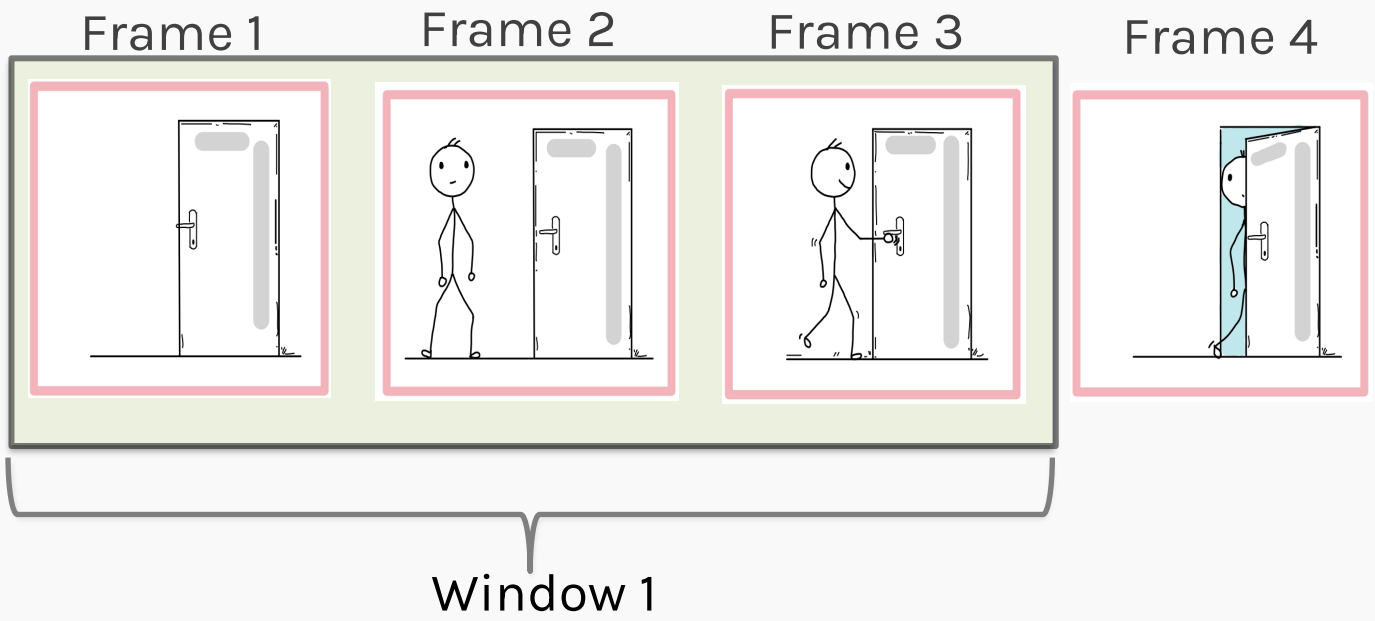
Motivation behind RNNs : **Windowing**



FFNN: Feed Forward NN
FCNN: Fully Connected NN
MLP: Multi Layer Perceptron
FCN: Fully Connected Network
Dense Network

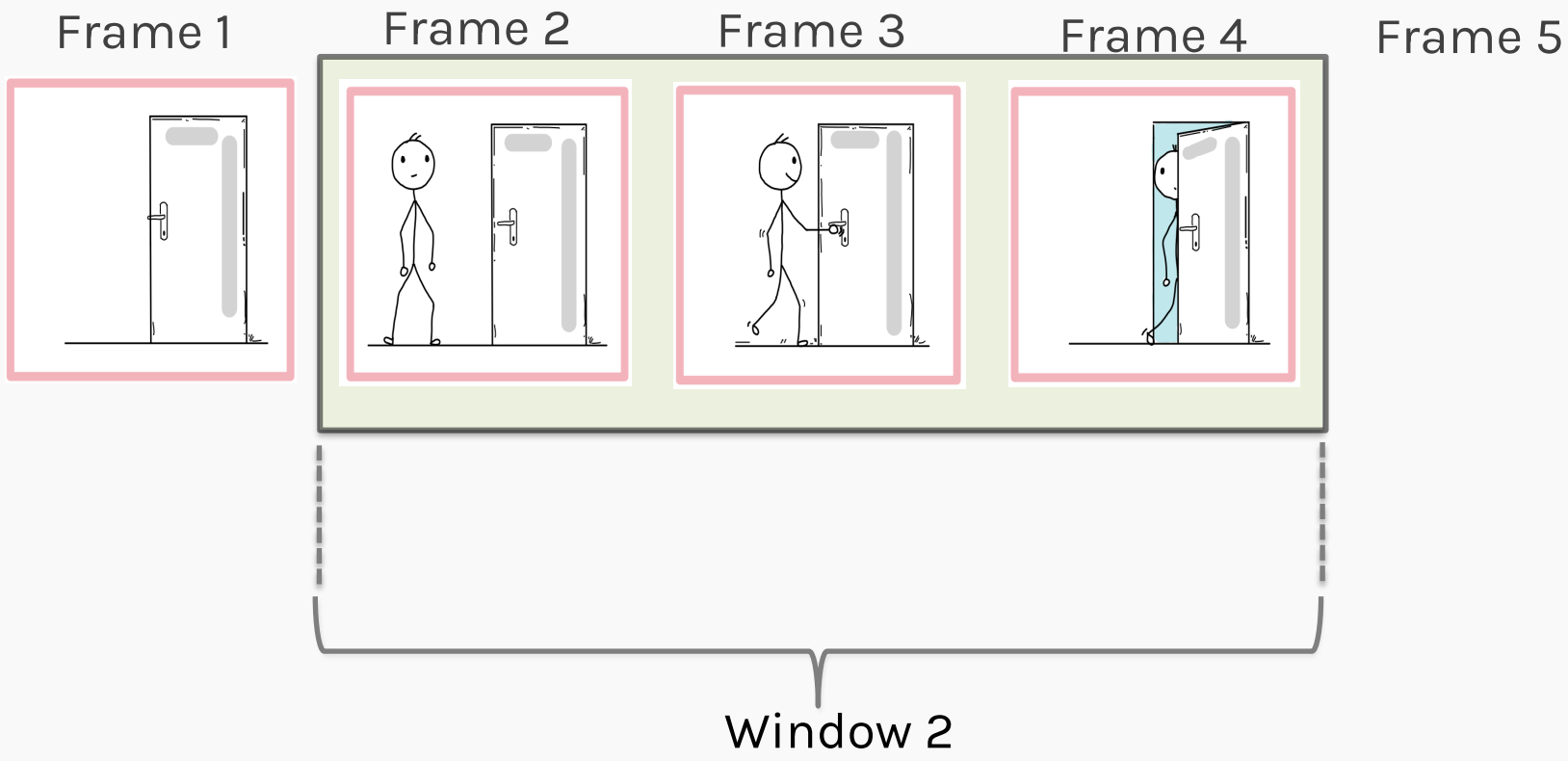
If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**



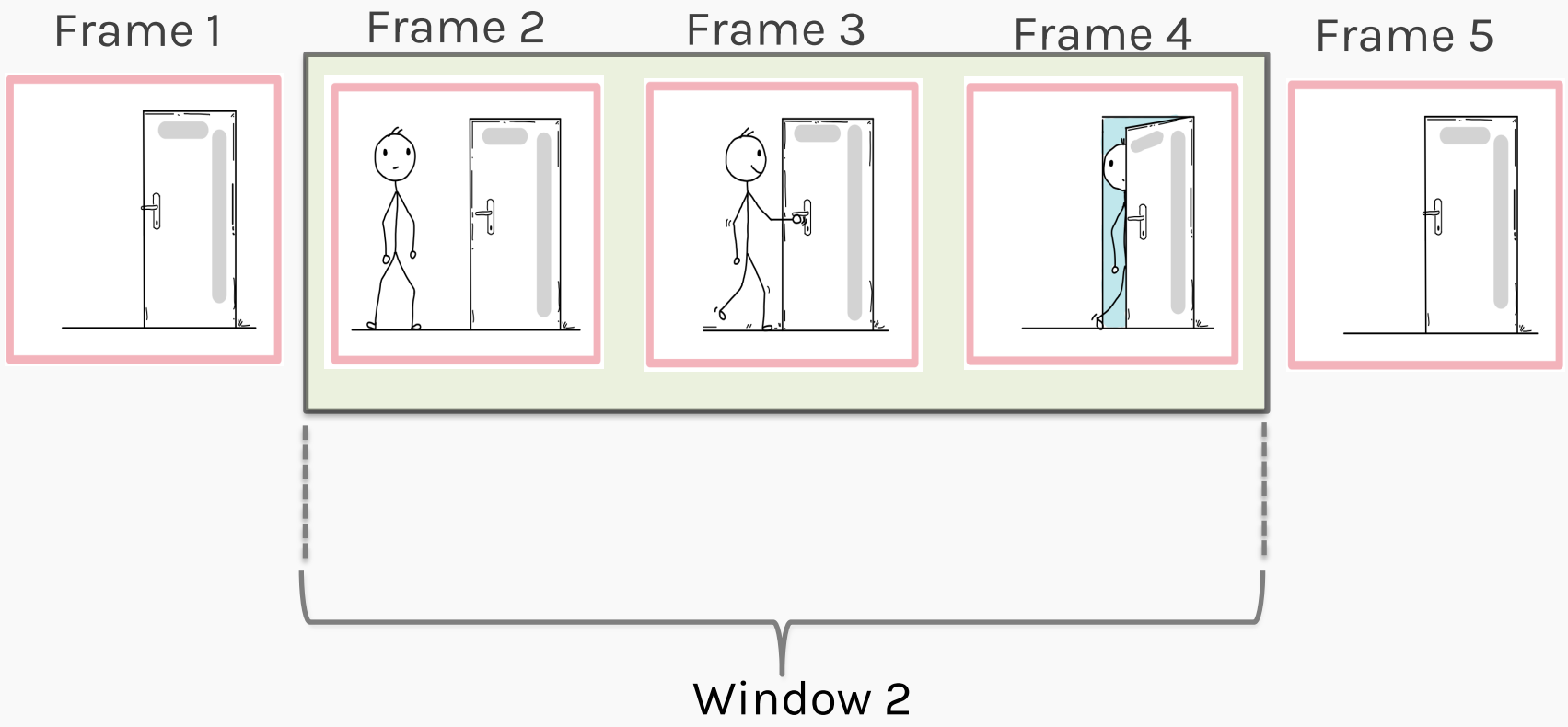
If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**



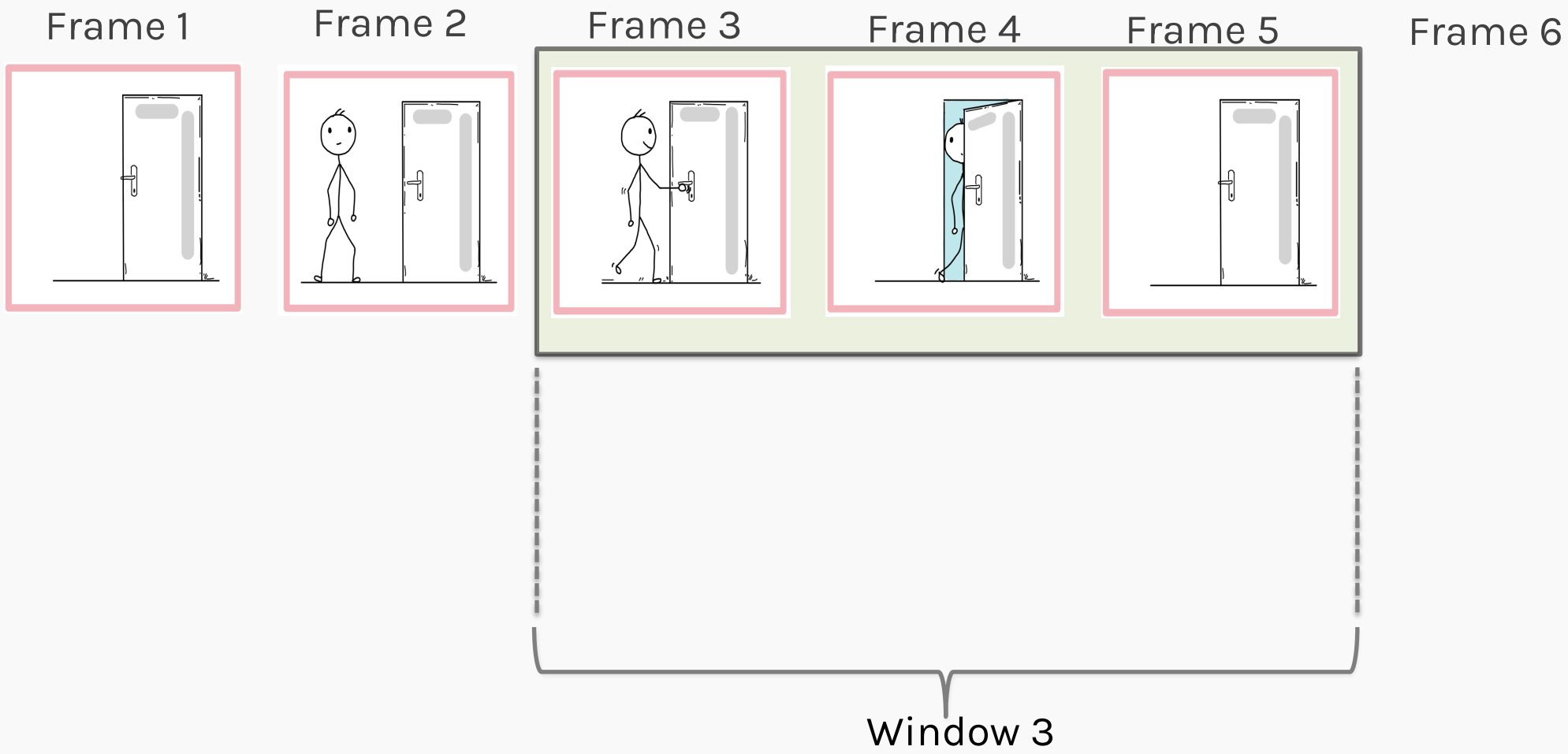
If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**



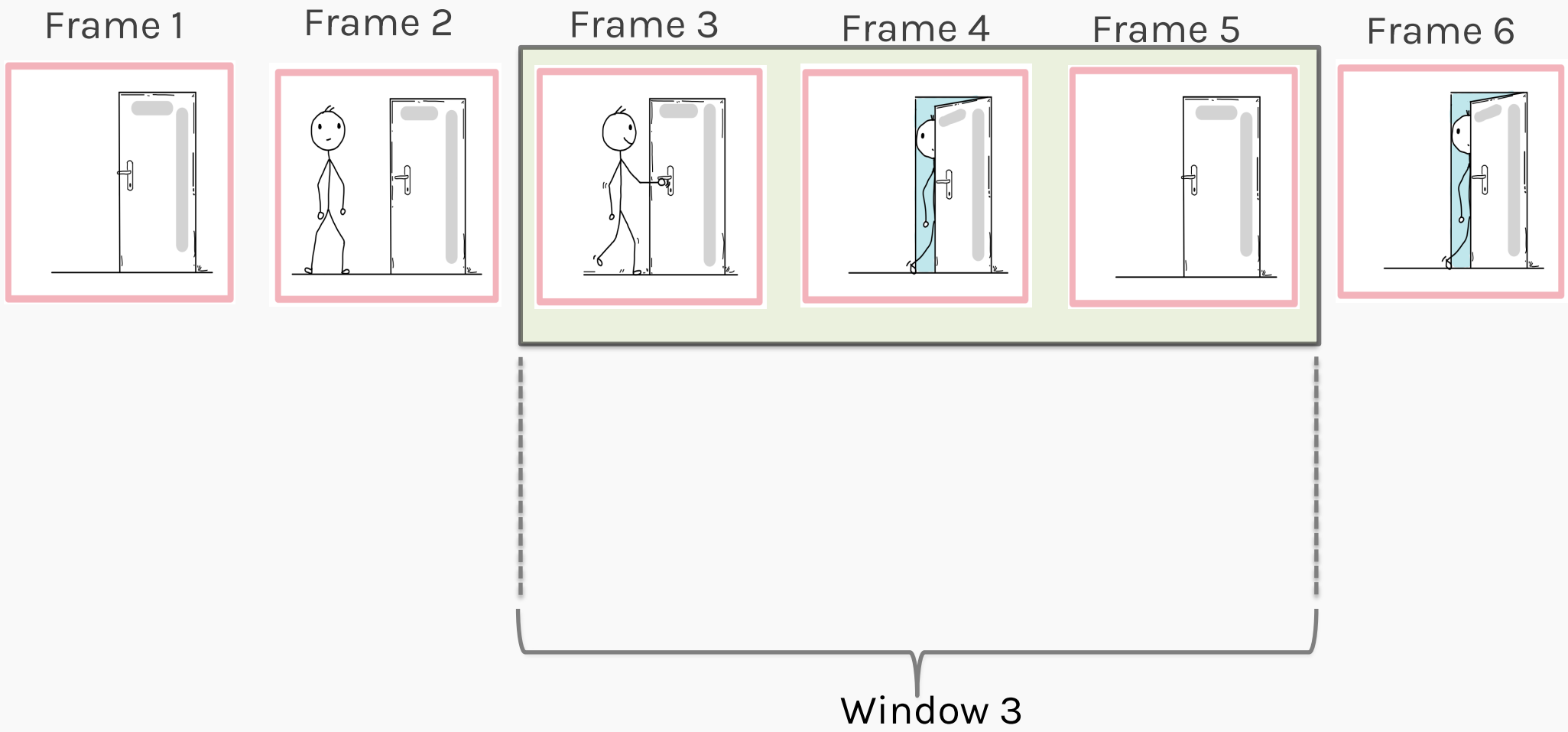
If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**



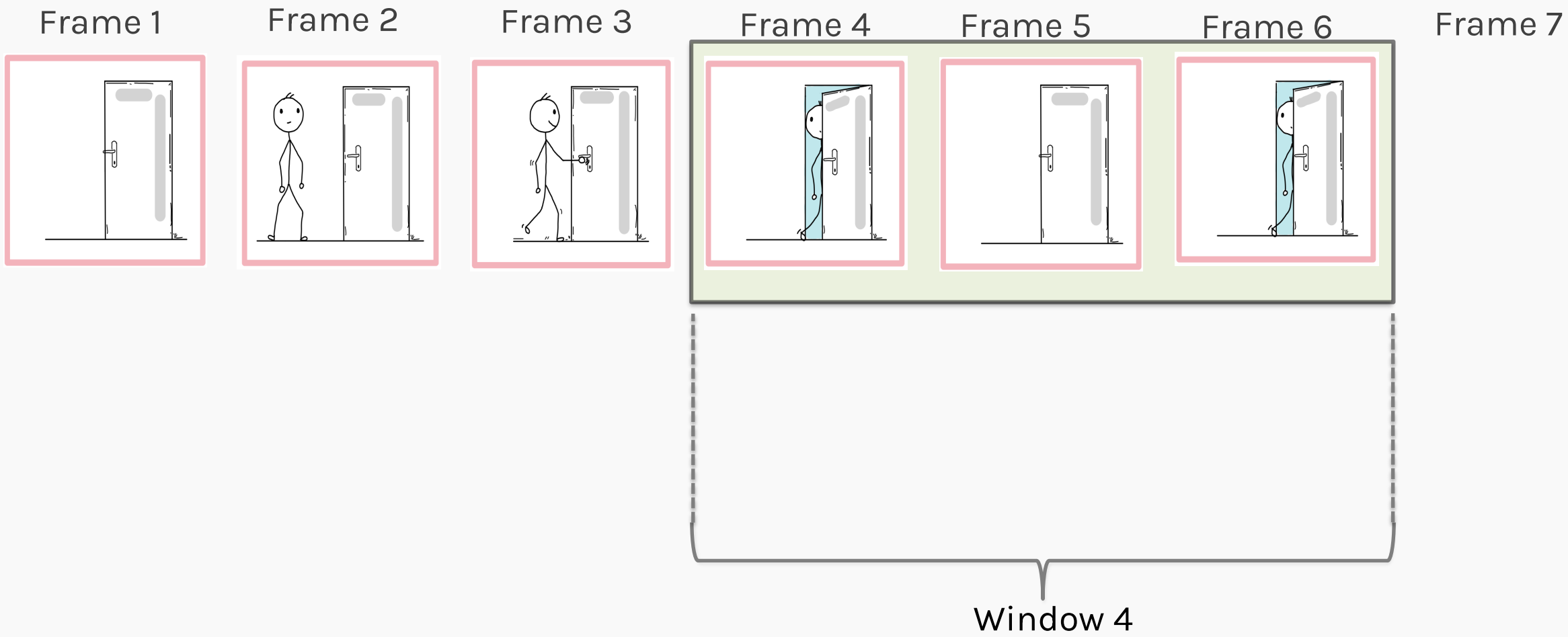
If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**



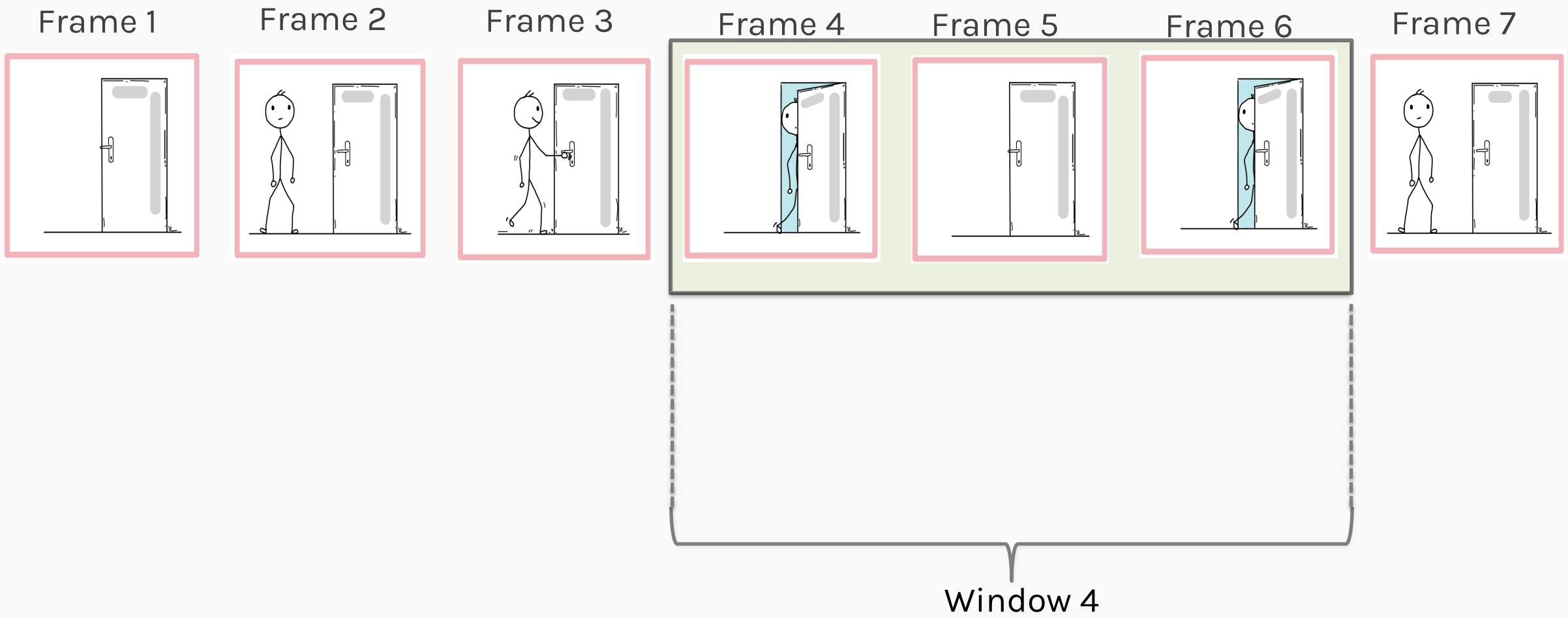
If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**



If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

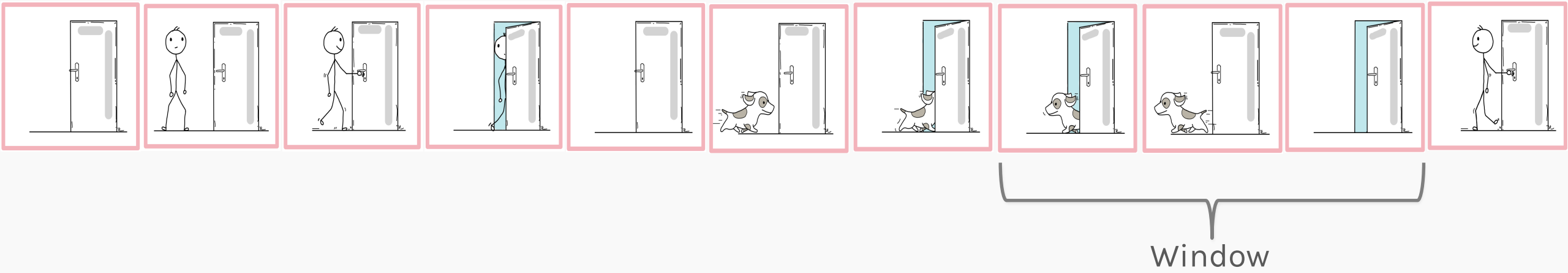
Motivation behind RNNs : **Windowing**



If we window a fixed number of frames (for example 3 here) as input to a neural network like FFNN or CNN, then the prediction will work.

Motivation behind RNNs : **Windowing**

However, consider the following sequence of frames:



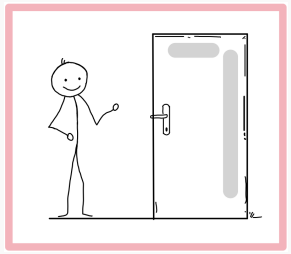
To predict the next frame, the window size is not enough.
Thus, a longer memory is needed.

RNNs introduce a concept of memory and carry forward the context history.

Motivation behind RNNs : **Ordering**

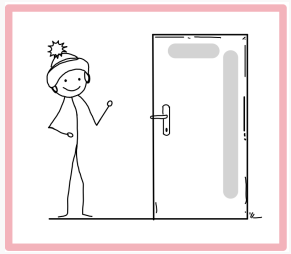
Consider the following sequence of frames given as input to a FFNN:

Frame 1



(wakes up)

Frame 2



(wears hat)

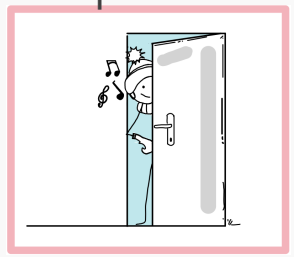
Frame 3

(wears gloves)

Frame 4

(listens to music)

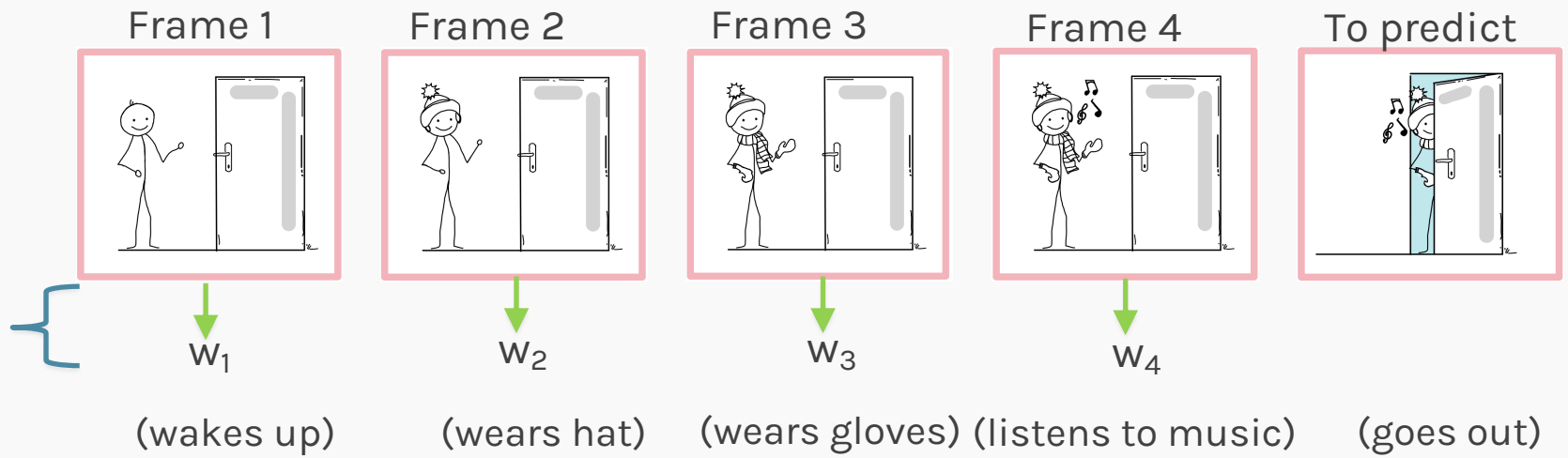
To predict



(goes out)

Motivation behind RNNs : **Ordering**

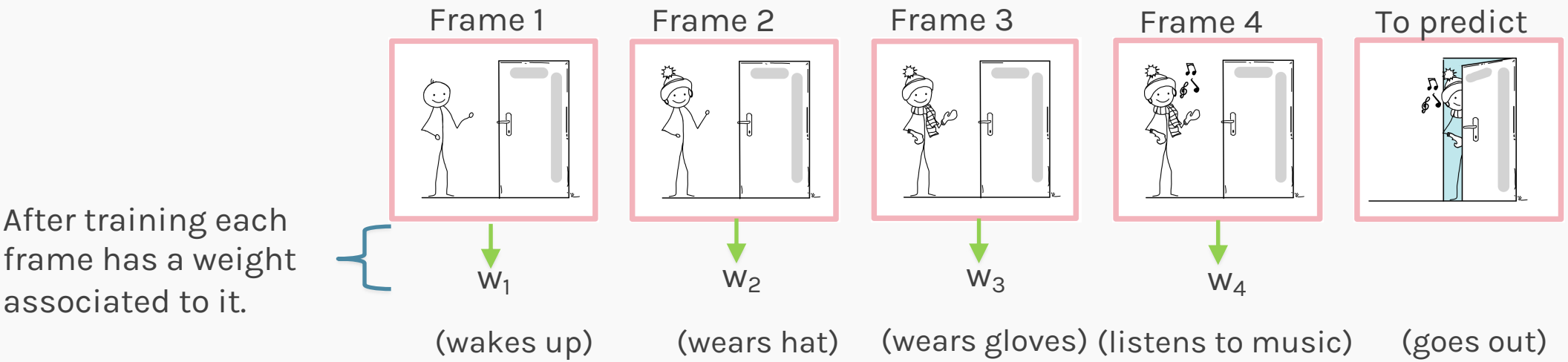
Consider the following sequence of frames given as input to a FFNN:



After training each frame has a weight associated to it.

Motivation behind RNNs : **Ordering**

Consider the following sequence of frames given as input to a FFNN:

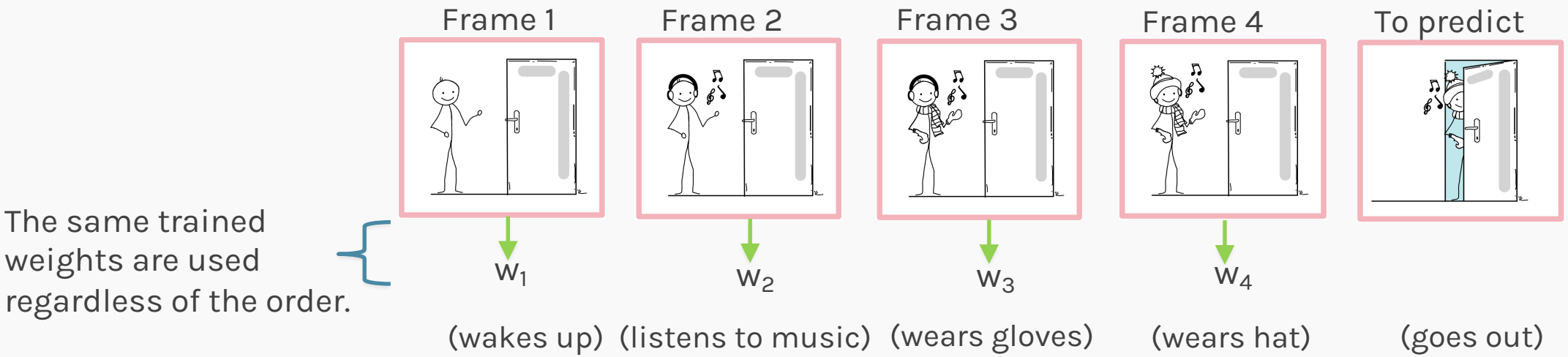


Here w_4 (listens to music) has the lowest weight ~ 0 i.e., it is the least important to predict the next frame as it has nothing to do with going out.

w_2 (wears hat) has the highest value i.e., it is the most important to predict the next frame, followed by w_3 (wears gloves).

Motivation behind RNNs : **Ordering**

The **order** of the frames is now slightly changed and given as input to the FFNN:

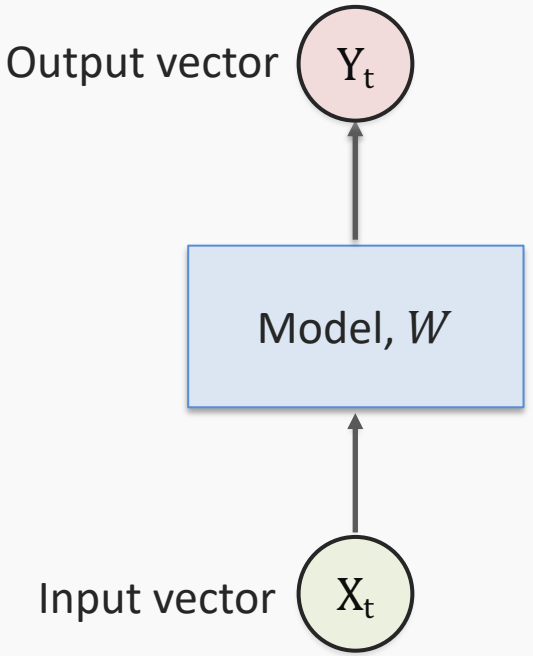


Though this order also makes sense for predicting going out, the prediction will be different this time as wears hat which was most responsible for prediction is multiplied with w_4, which is ~ 0.

However, listening to music that is not relevant for prediction is highly important by multiplying with w_2.

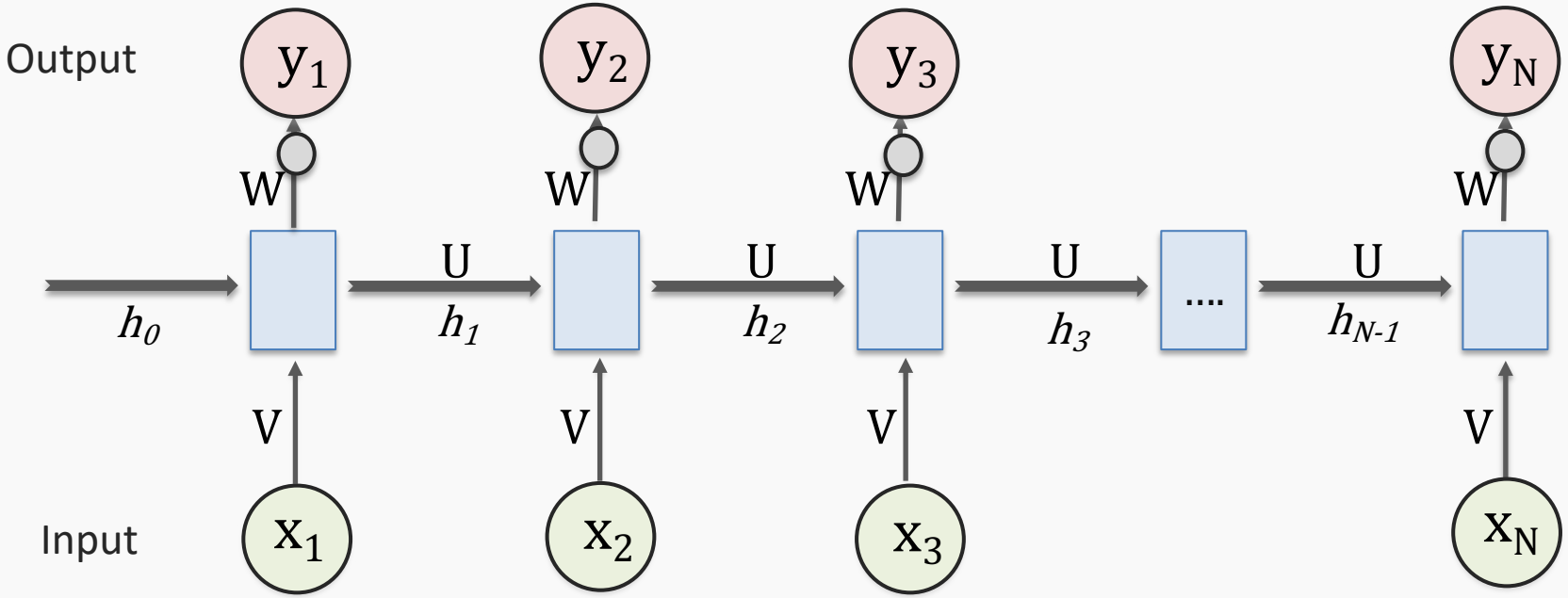
Introduction to RNNs

FEED FORWARD NEURAL NETWORK



- Cannot maintain previous information

RECURRENT NEURAL NETWORK

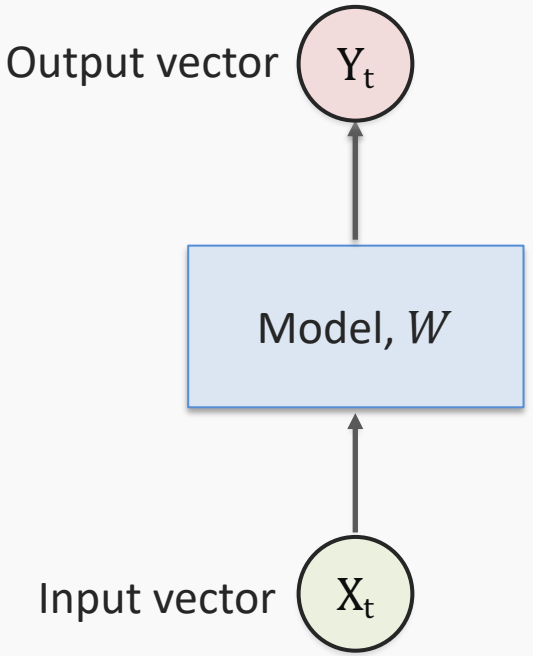


The term **recurrent** comes from the fact that information is being passed from one time step to the next internally within the network.

Network has loops for information to persist over time

Introduction to RNNs

FEED FORWARD NEURAL NETWORK

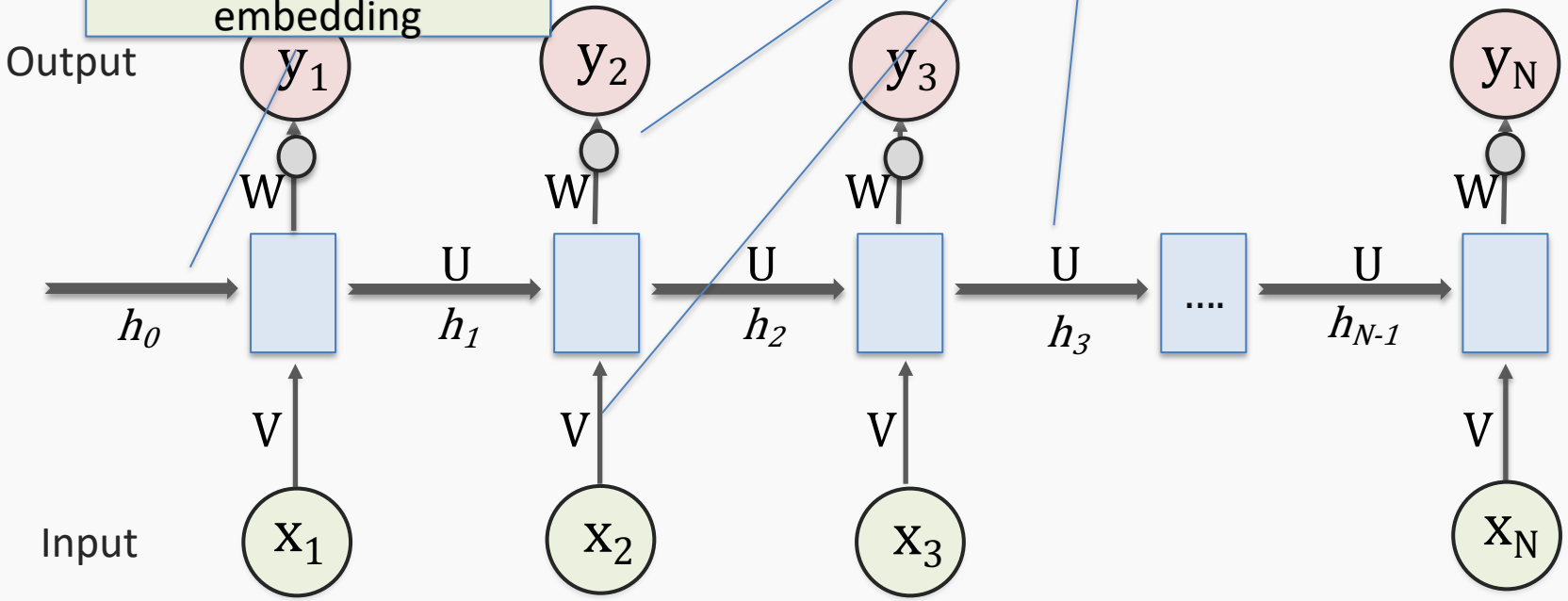


- Cannot maintain previous information

h_i:
memory
state
internal state
hidden state
latent variable
embedding

V, U, W: model parameters

CURRENT NEURAL NETWORK

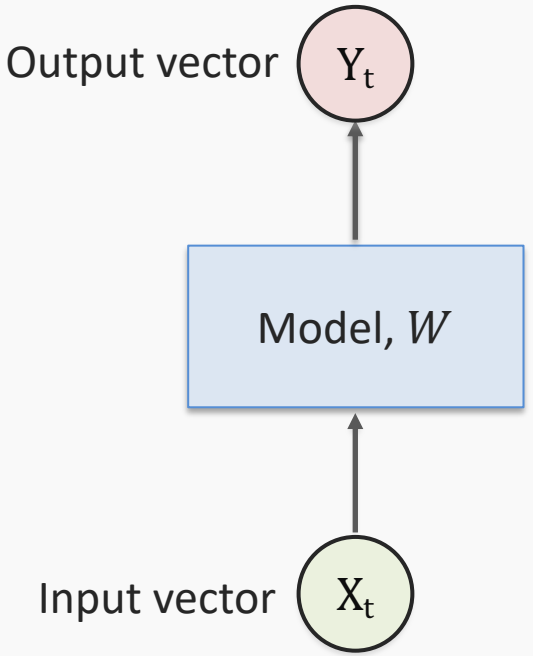


The term **recurrent** comes from the fact that information is being passed from one time step to the next internally within the network.

Network has loops for information to persist over time

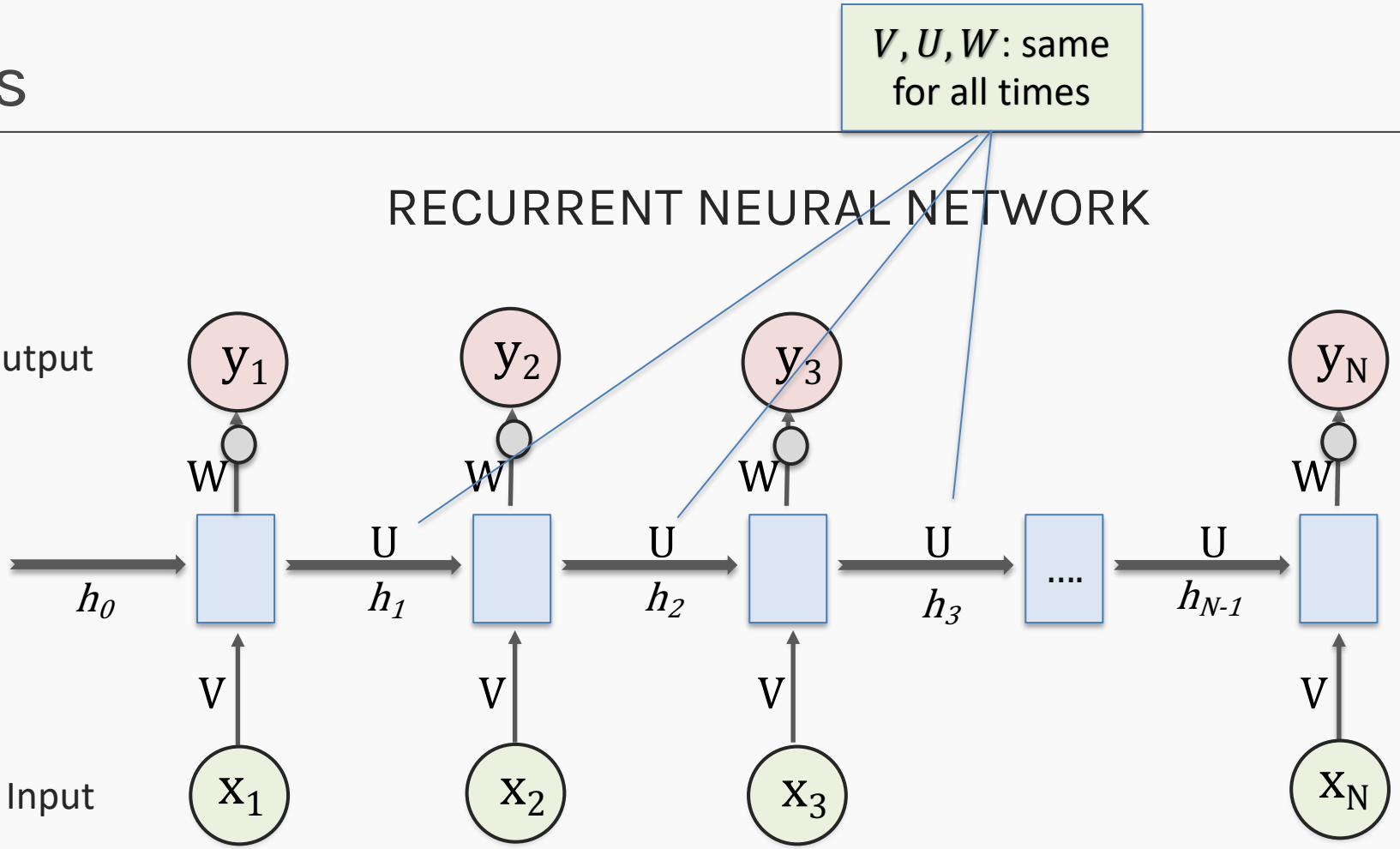
Introduction to RNNs

FEED FORWARD NEURAL NETWORK



- Cannot maintain previous information

RECURRENT NEURAL NETWORK

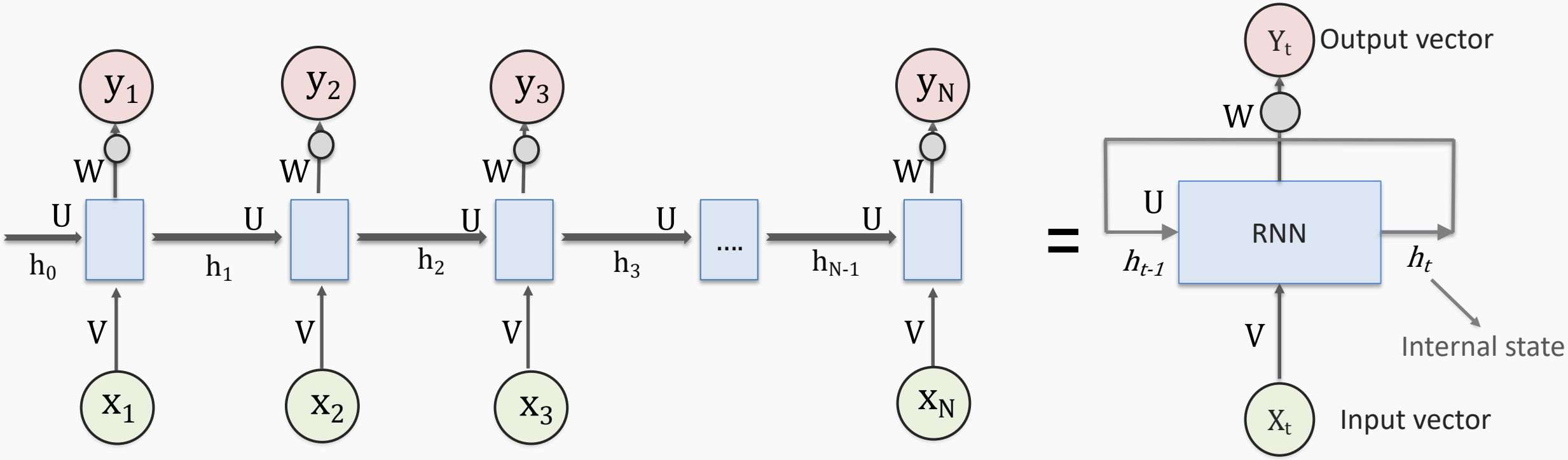


The term **recurrent** comes from the fact that information is being passed from one time step to the next internally within the network

Network has loops for information to persist over time

Introduction to RNNs

Alternative short representation:

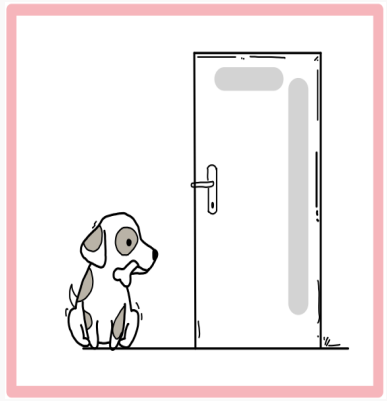


Bidirectional RNNs : **Motivation**

Given only Frame t-1 and t-2, it is difficult to predict the next frame t.

Frame t-2

Frame t-1



Frame t

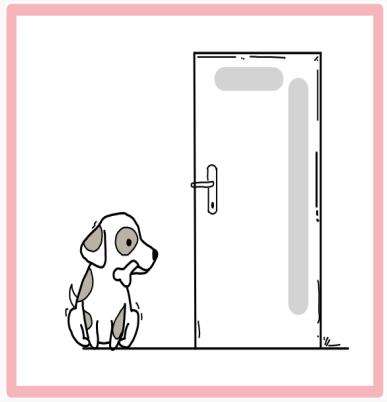
?

Bidirectional RNNs : **Motivation**

However, if we are to give some future frames it would be easier for the model to predict.

Frame t-2

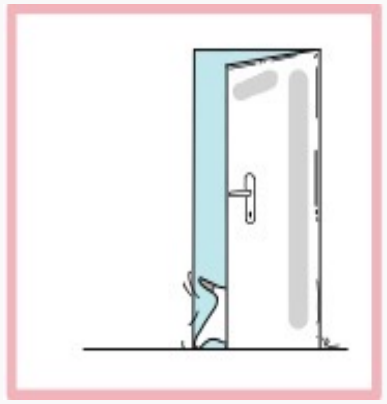
Frame t-1



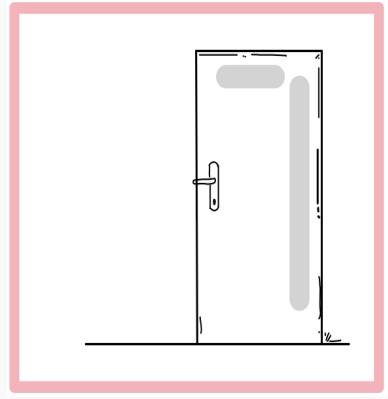
Frame t

?

Frame t+1



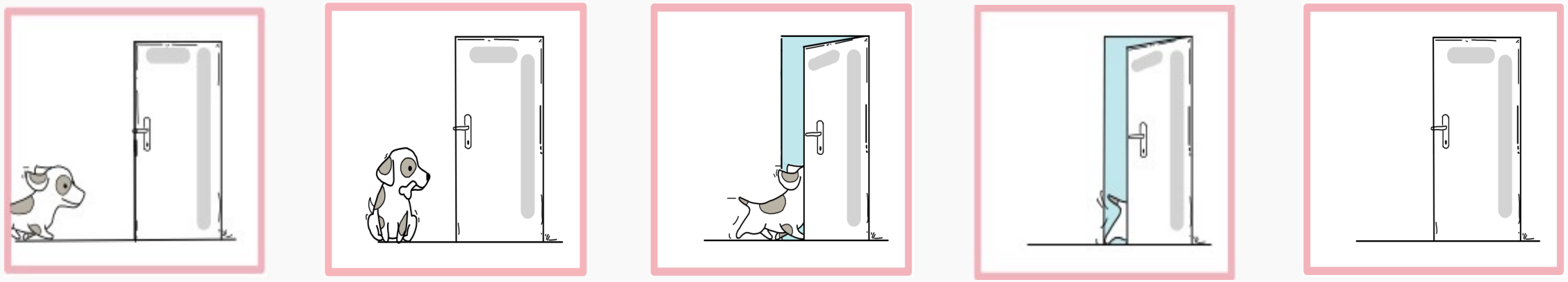
Frame t+2



Bidirectional RNNs : **Motivation**

Thus, sequences after the one to be predicted play an important role to provide context for prediction.

Frame t-2 | Frame t-1 | Frame t | Frame t+1 | Frame t+2



However, simple RNNs process sequences only from left to right. They cannot look ahead into future sequences.

Bidirectional RNNs solve this problem by processing the sequence in both directions.

Bidirectional RNNs

In a Bidirectional RNN there are two separate RNNs used: one for forward direction and one for reverse direction.

This results in a hidden state from each RNN, which are concatenated to form a single hidden state.

Hidden state of Forward RNN

Hidden state of Reverse RNN

$$h_t^F = \tanh(\ V^F X_t\ + U^F\, h_{t-1}^F + \ \beta_1^F)$$

$$h_t^R = \tanh(\ V^R X_t\ + U^R\, h_{t+1}^R + \ \beta_1^R)$$

$$h_t^T = h_t^F + h_t^R$$

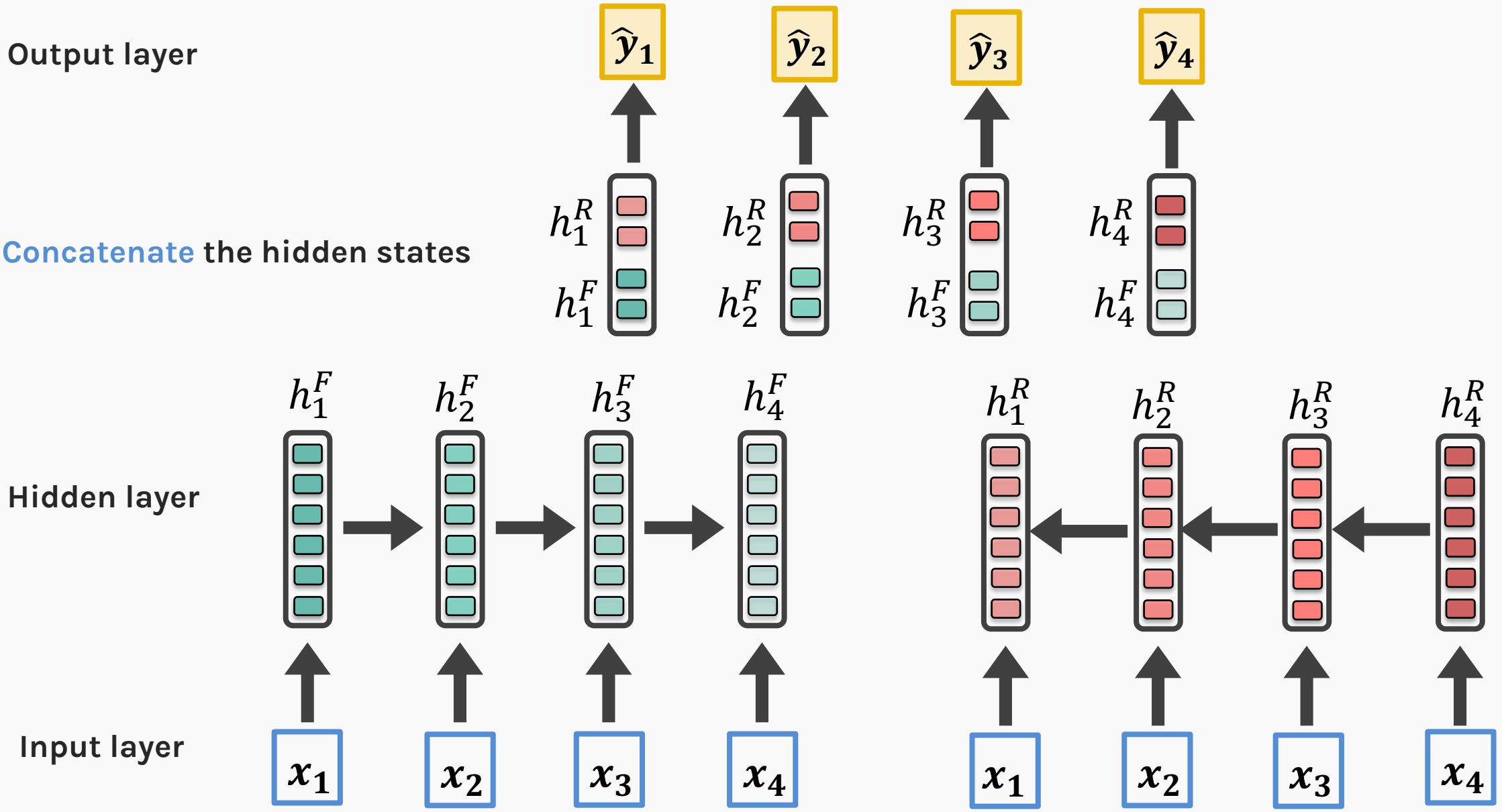
Final hidden state

The output equation is similar to that of simple RNNs:

$$\hat{y}_t\ = \sigma(h_t^T W + \ \beta_2\ )$$

Often it is concatenation and not addition

Bidirectional RNNs



Deep RNN

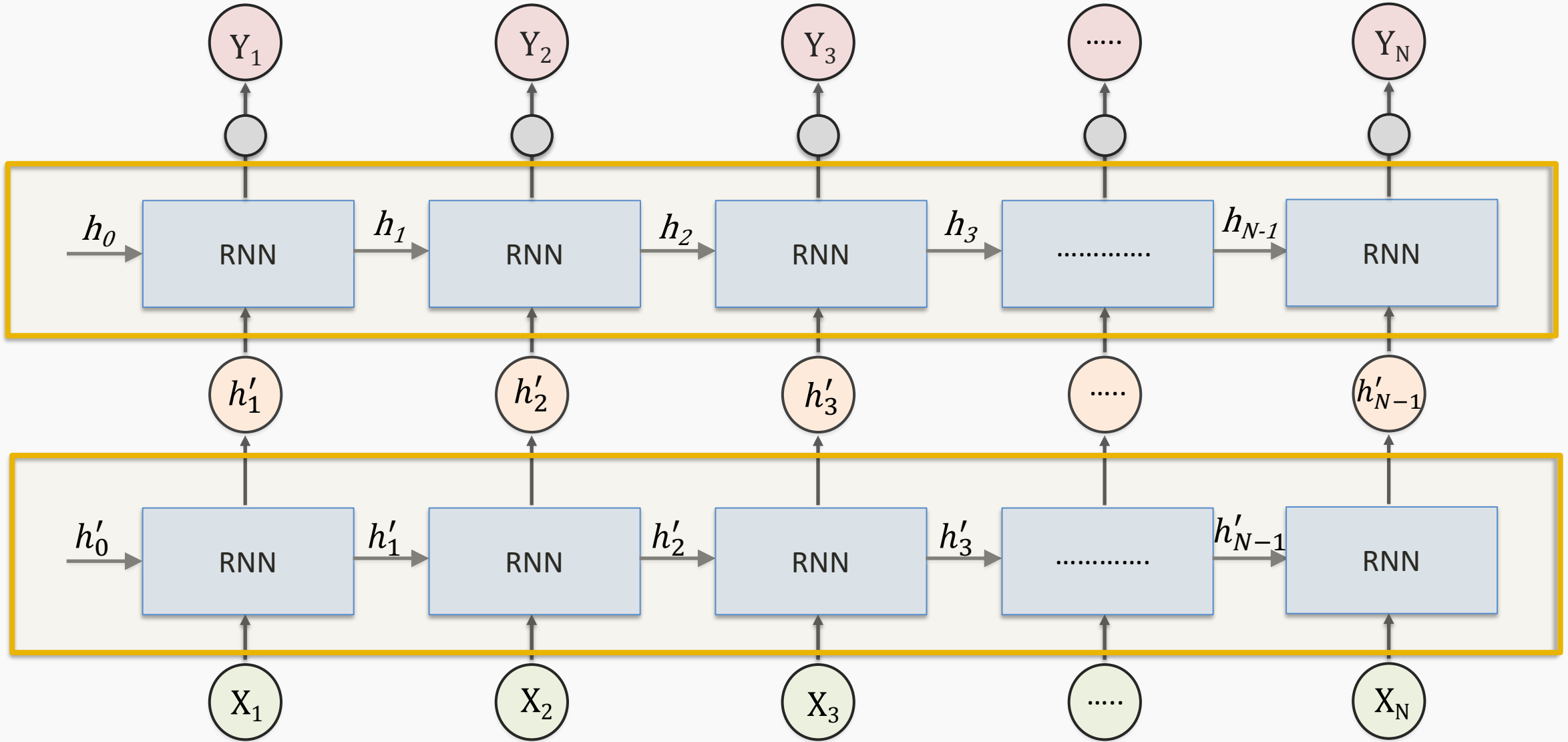
RNNs units can be arranged in layers, so that the output of each unit is the input to the other units. This is called **a deep RNN**, where the adjective “deep” refers to these multiple layers.

- Each layer feeds the RNN on the next layer
- First time step of the input is fed to the first layer RNN, which processes that data and produces an output (and a new state for itself).
- That output of the first layer is fed to the next RNN, which does the same thing, and so on.
- Then the second time step arrives at the first RNN, and the process repeats.

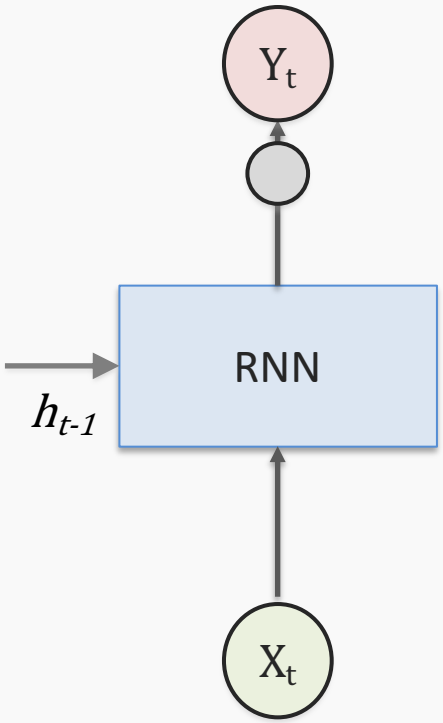
Deep RNN

Hidden layers provide an abstraction (holds “meaning”).

Stacking hidden layers provides increased abstractions.

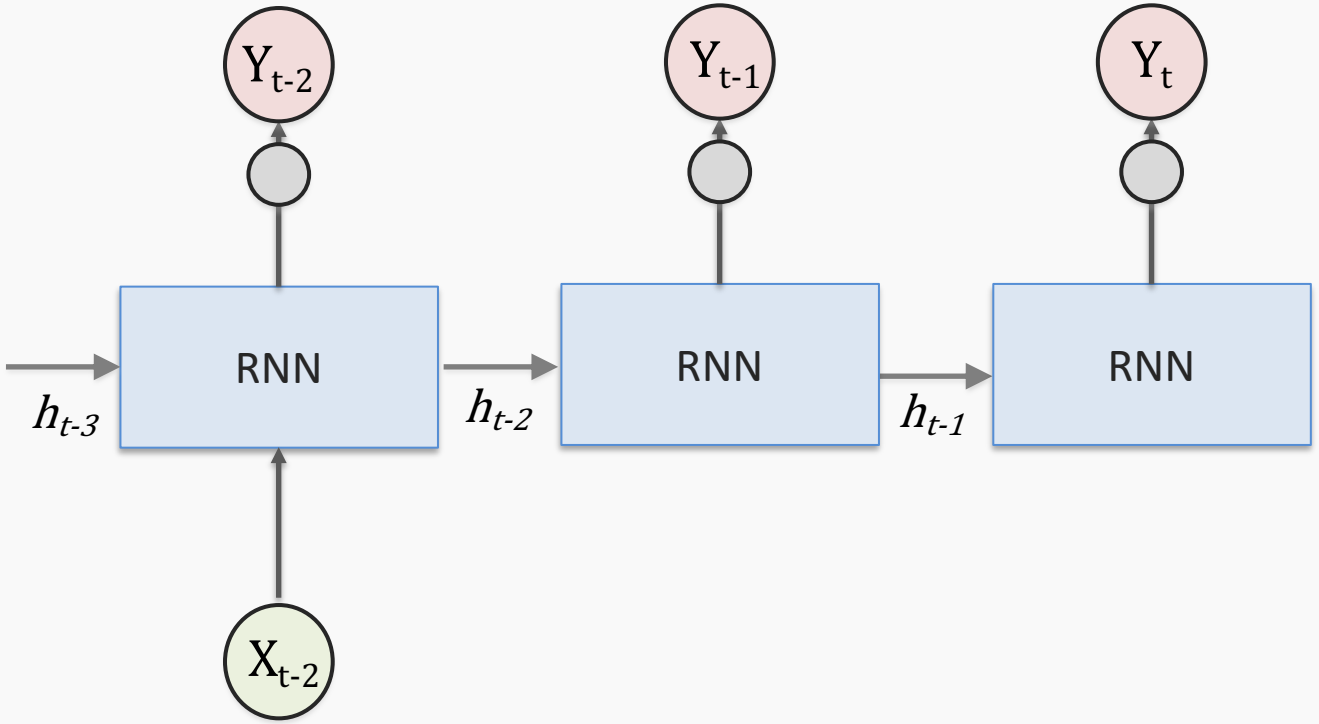


Flavors of RNN : **One to One**



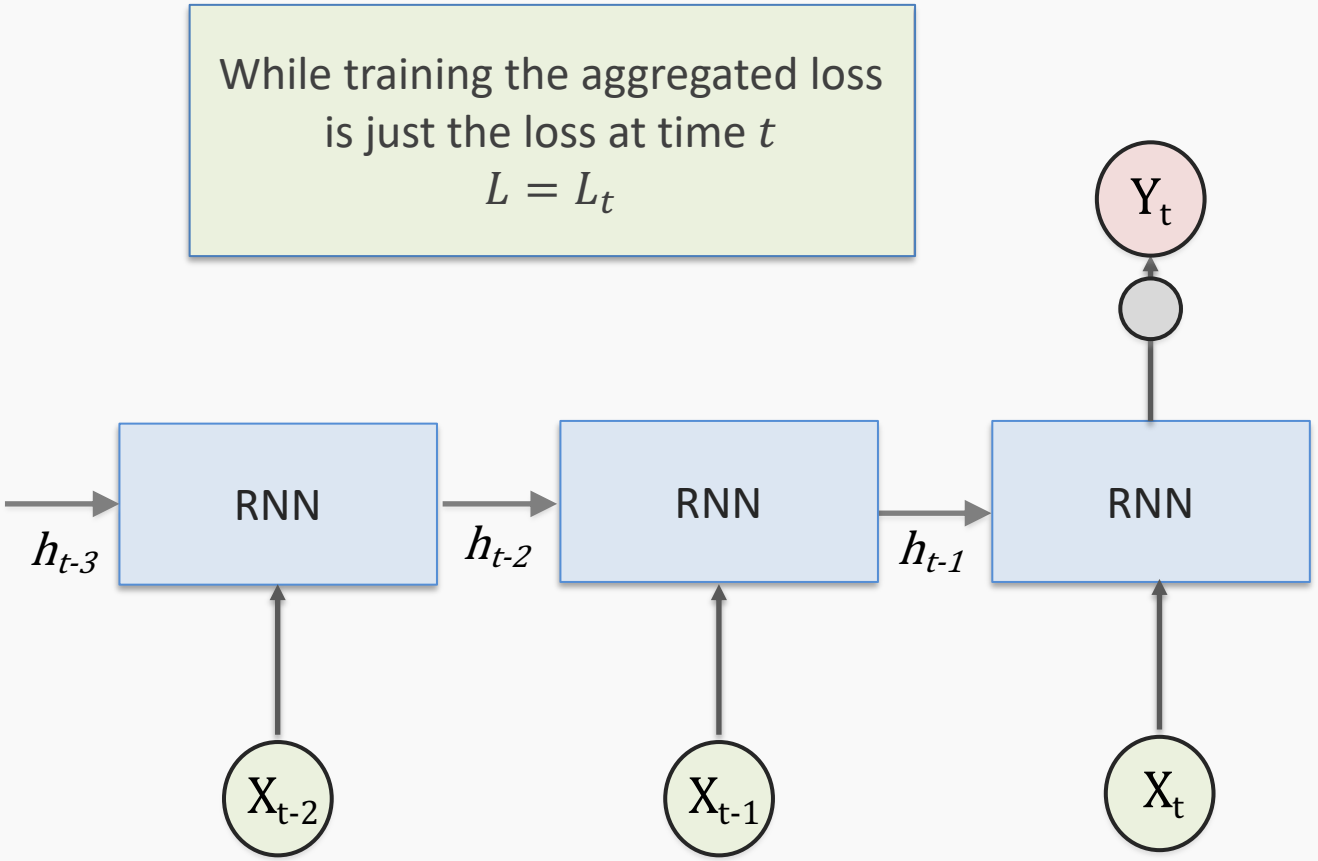
- The **One to One** structure is useless.
- It takes a single input and it produces a single output.
- Not useful because the RNN cell is making little use of its unique ability to remember things about its input sequence.

Flavors of RNN : **One to Many**



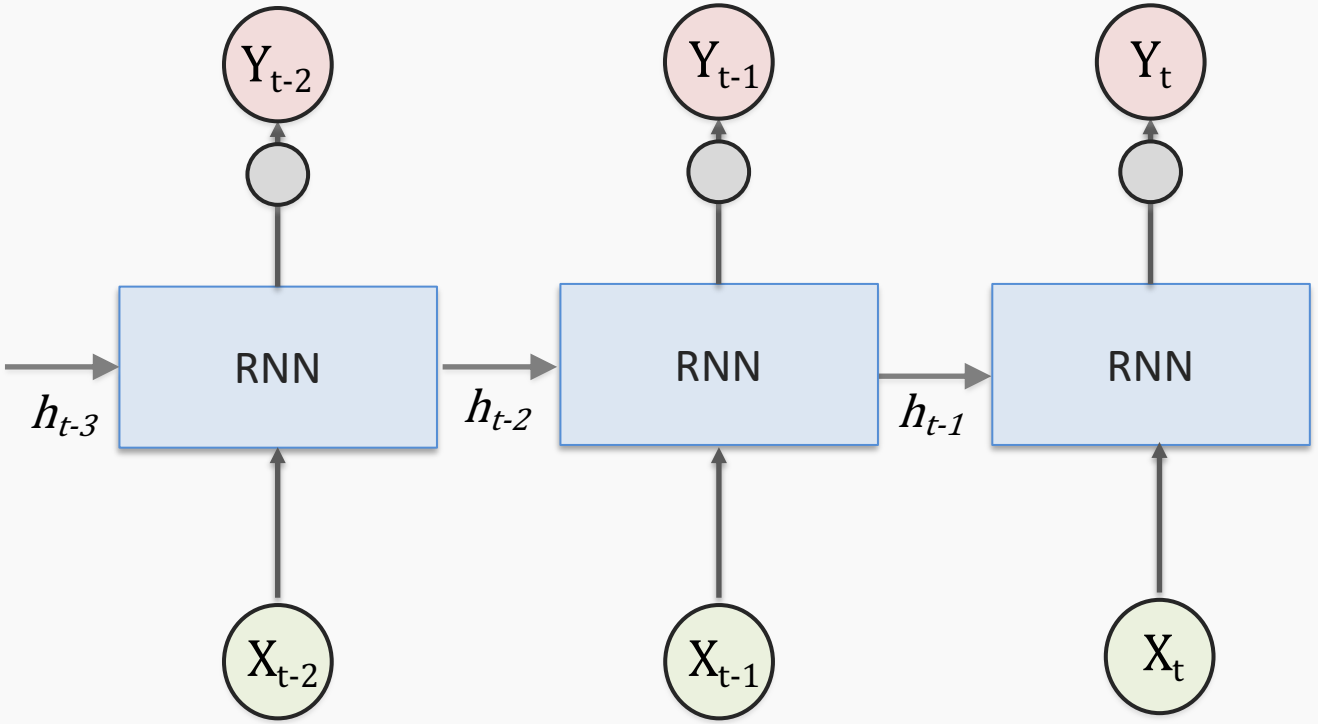
- The **One to Many** takes in a single piece of data and produces a sequence.
- For example, we give it the starting note for a song, and the network produces the rest of the melody for us.

Flavors of RNN : **Many to One**



- The **Many to One** structure reads in a sequence and gives us back a single value.
- Example: **Sentiment analysis**, where the network is given a piece of text and then reports on some quality inherent in the writing. A common example is to look at a movie review and determine if it was positive or negative.
- This structure is also used for any kind of **classification**.

Flavors of RNN : **Many to Many**

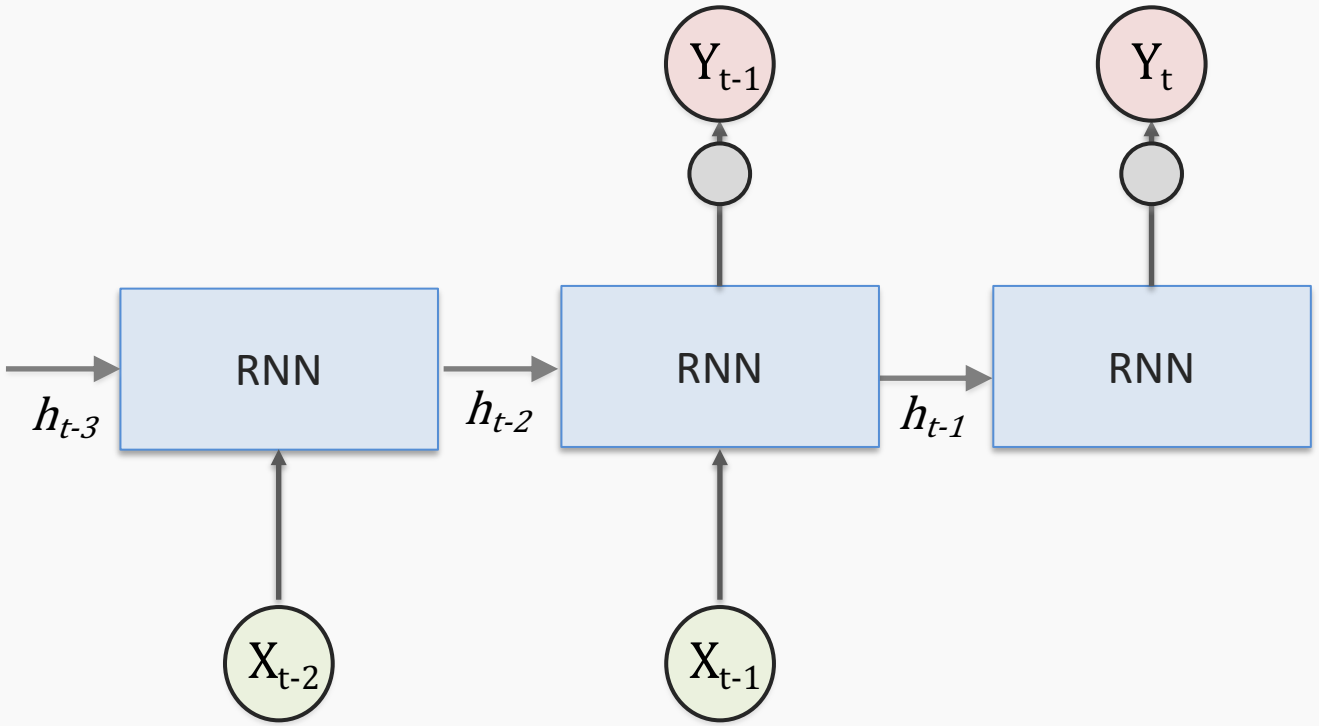


- The **Many to Many** structures are in some ways the most interesting.

- Examples:
  - Predict if it will rain given some inputs.
  - Language Model: In this case we predict the next word:

$$Y_t = X_{t+1}$$

Flavors of RNN : **Many to Many**



- This form of **Many to Many** can be used for machine translation.

- For example, the English sentence:
  “The white dog jumped over the cat”

  In Spanish would be:
  “**El perro blanco saltó sobre el gato**”
  In Spanish, the adjective “blanco” (white) follows the noun “perro” (dog), so we need to have a buffer so we can produce the words in their proper order in Spanish.