How Recurrent Neural Networks and Long Short-Term Memory Work – By Example

2017-2021

Based on the notes from Brandon Brohrer

Explanation using examples

We will attempt to explain the functionality of

- RNNs
- LSTMs

By using few examples

RNN – Guess what we have for Dinner tonight?

- Every night for dinner, we have either:
  - Pizza, or
  - Sushi, or
  - Waffles
- And repeat again

Guess the dinner tonight?

Outputs: (?)

3 choices

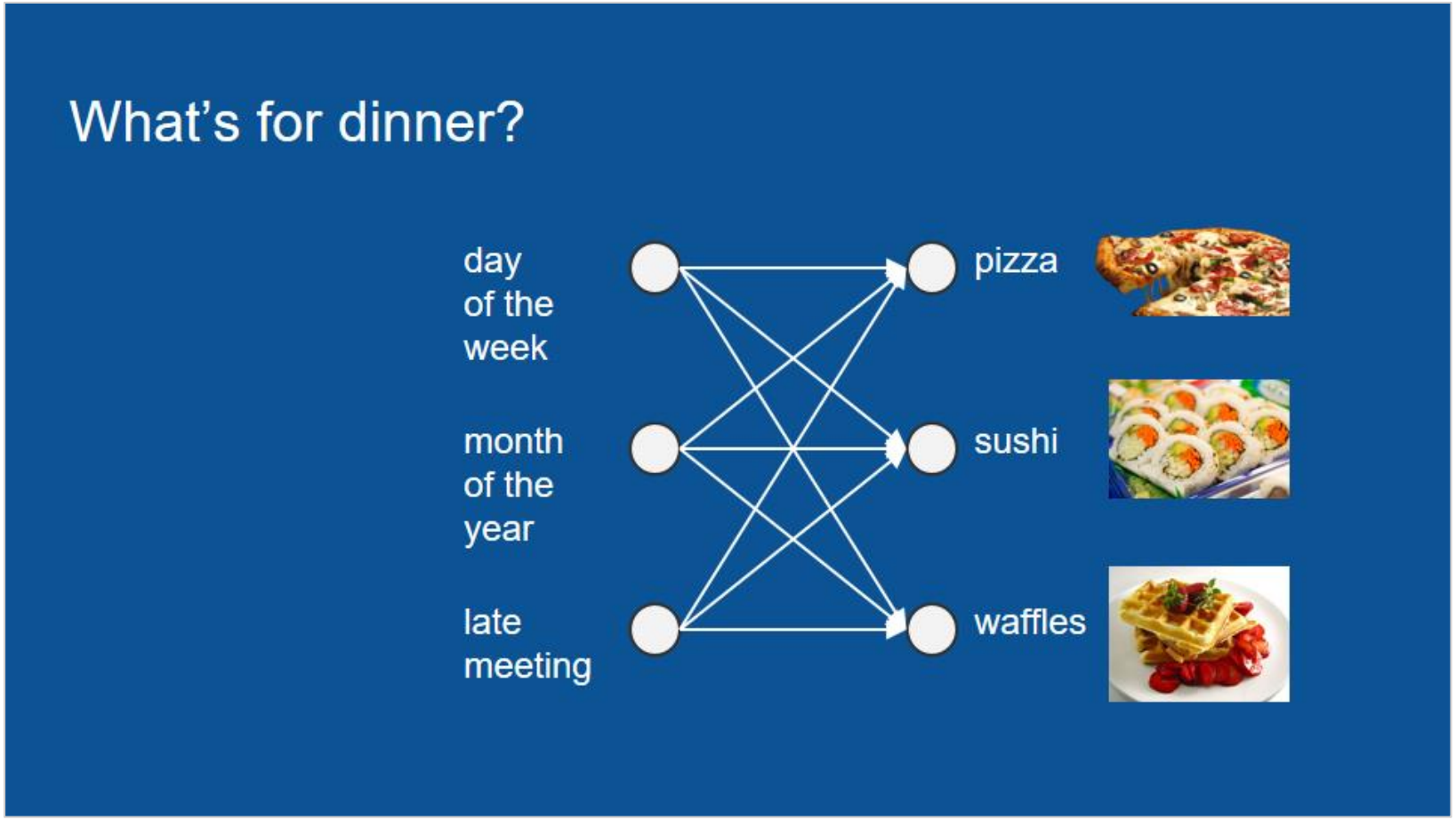
- pizza,
- sushi,
- waffles

Inputs: (?)

what can affect what we have for dinner, for example,

- day of the week,
- month,
- late meeting

Voting Process ←→ Prediction



Pizza, Sushi, Waffles, & repeat - Re-examine the data

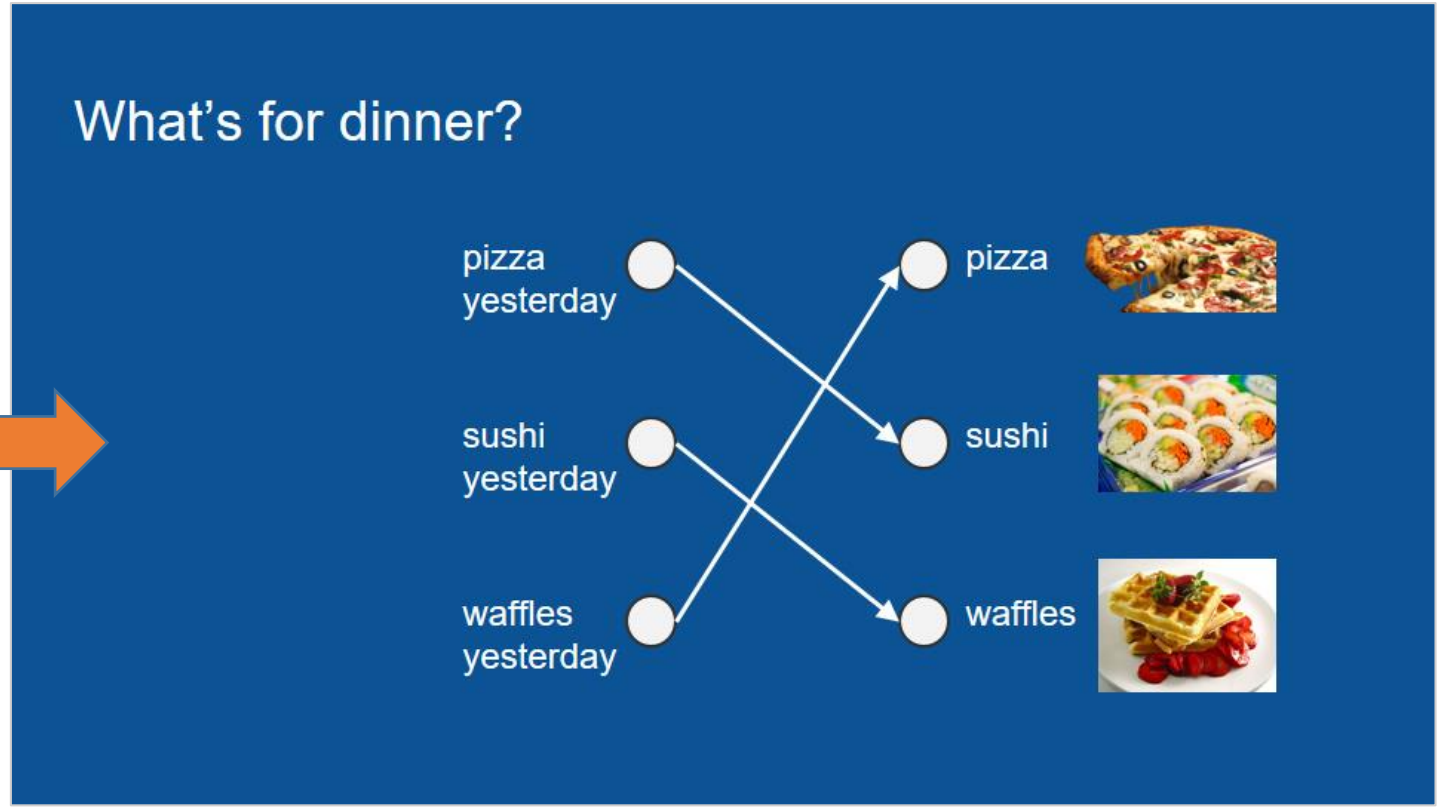
Let's simplify our assumptions

Assume that the choice of dinner does not depend on the day of the week, month, or late meetings

Let's assume that the data follows a simple pattern of

- Pizza,
- sushi,
- waffles and
- repeat

Therefore, we just need to know what we had last night



What happens if we do not know what we had last night?

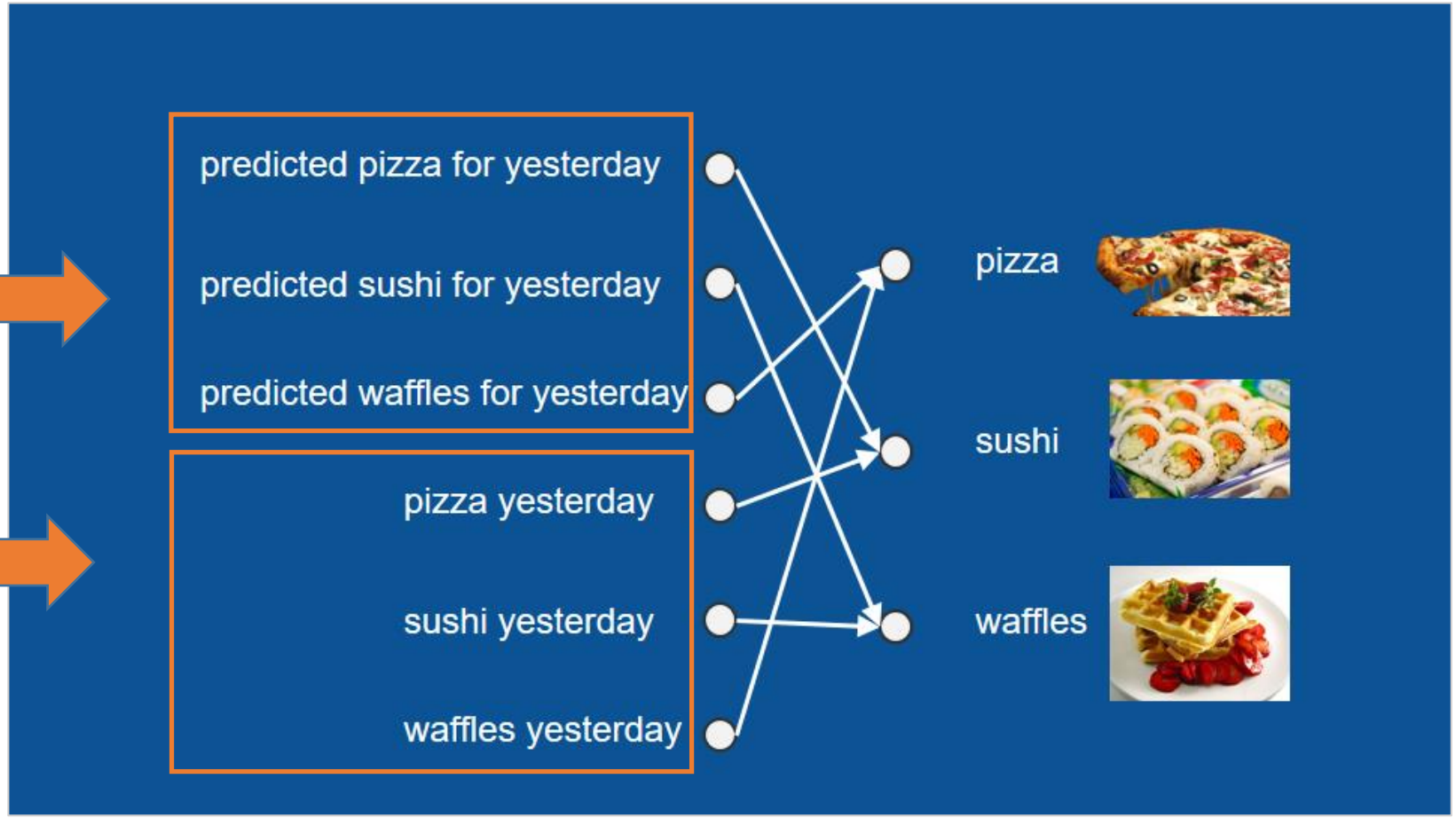
- e.g., I was not home last night,
  I cannot remember,
  …

- Then, it will be helpful to have:
  - A prediction of what we might have had yesterday night

What do we need to know to make a prediction re dinner night?

- Generally we need to know:
  - A prediction of what we might have had last night

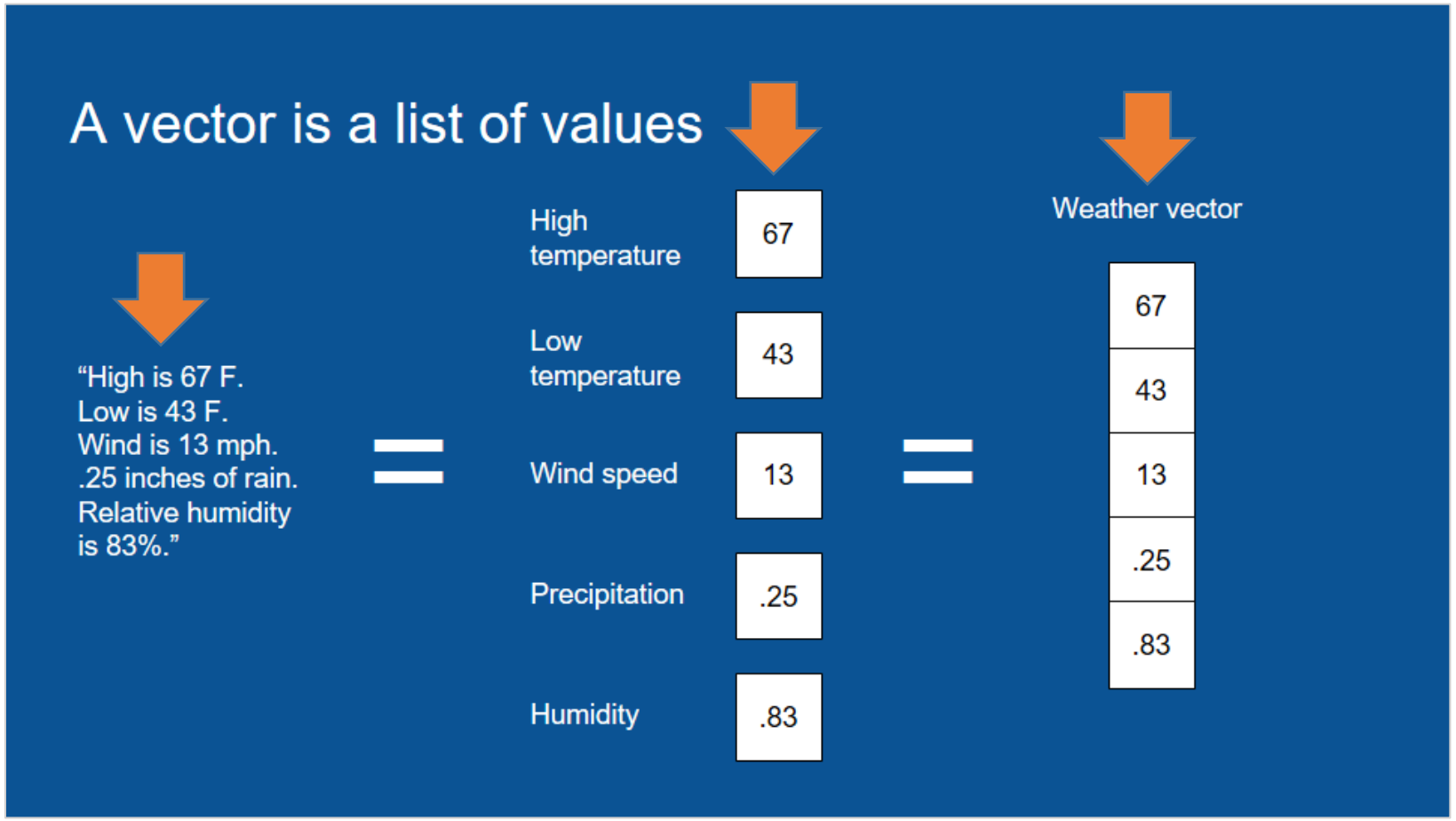
    or
  - Information about the dinner last night



Side note - Vectors

Neural networks can understand vectors better

Native language of NNs is vectors



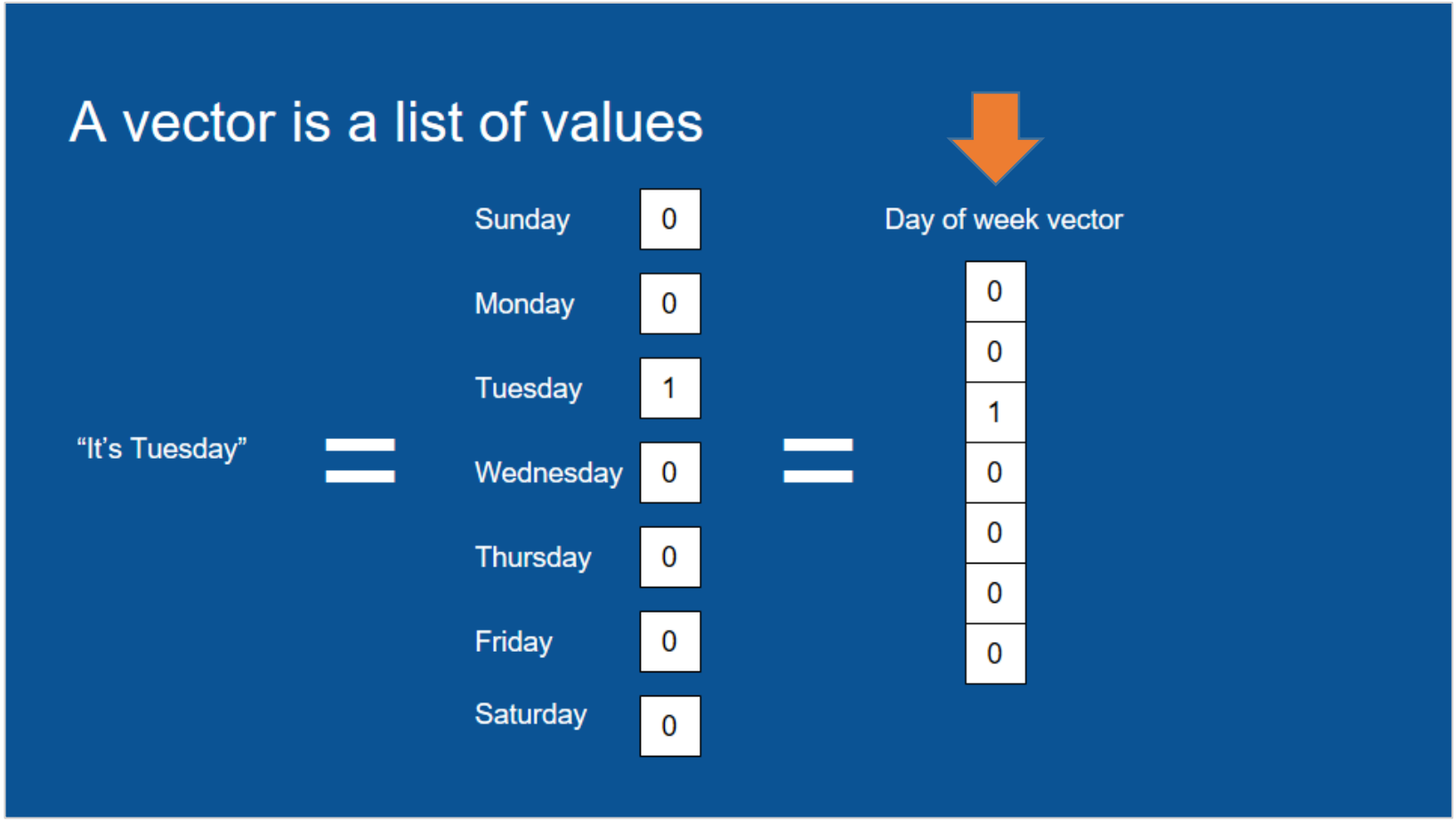
Side note - Vectors as statements

ONE HOT ENCODING

The list (vector) includes all possibilities for the days of the week

All of them are ZERO Except the one that is true that is Tuesday is ONE

“It is Tuesday”

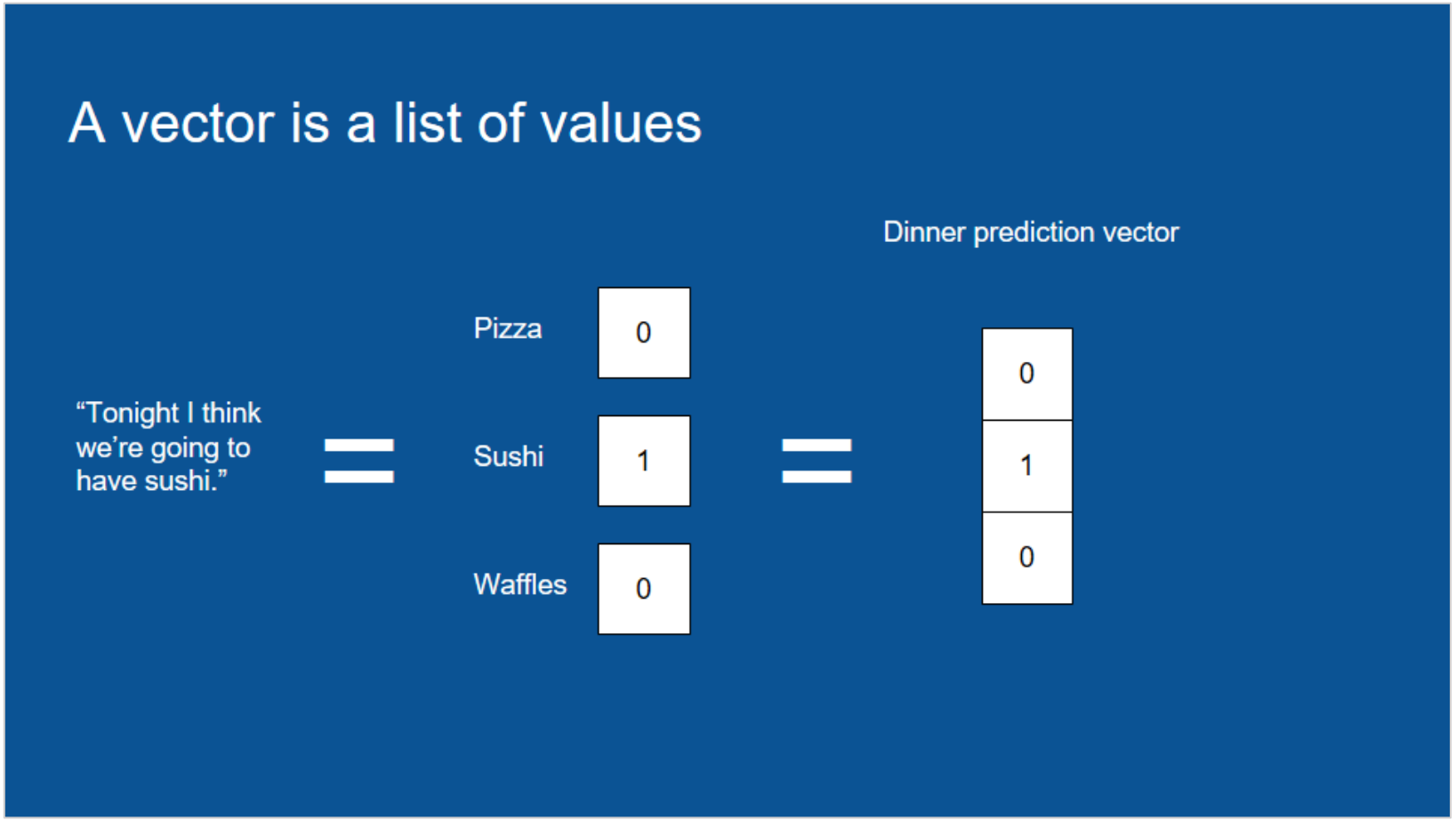


Side note - One Hot Vector for our example

A vector: a list of values
We have 3 choices for dinner
-Pizza,
-Sushi,
-Waffles

“we have Sushi”
The one hot vector representing this statement is:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



Input/Output vector

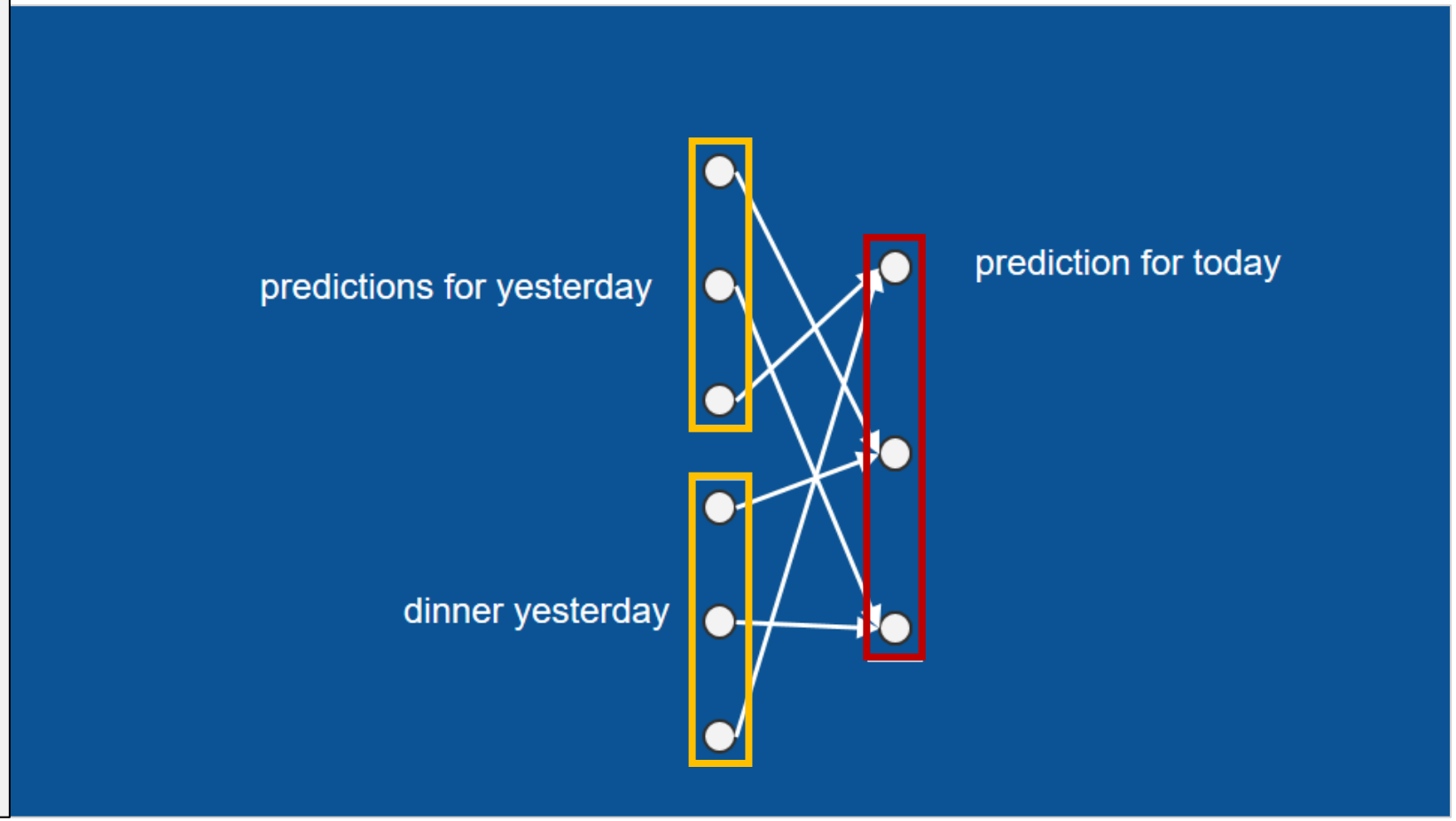
- **Input:** Two vectors

1. A vector for prediction of dinner for yesterday

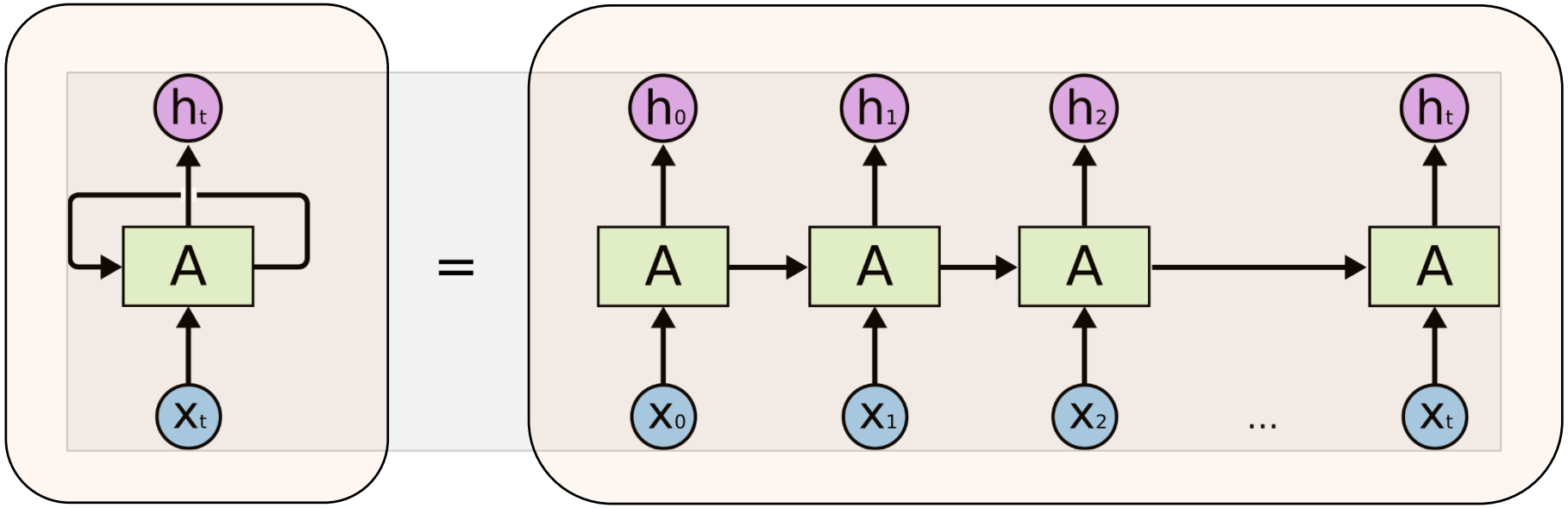
2. A vector for actual dinner yesterday

- O**utput**: One vector

1. A vector for dinner prediction for today



Recurrent Neural Networks

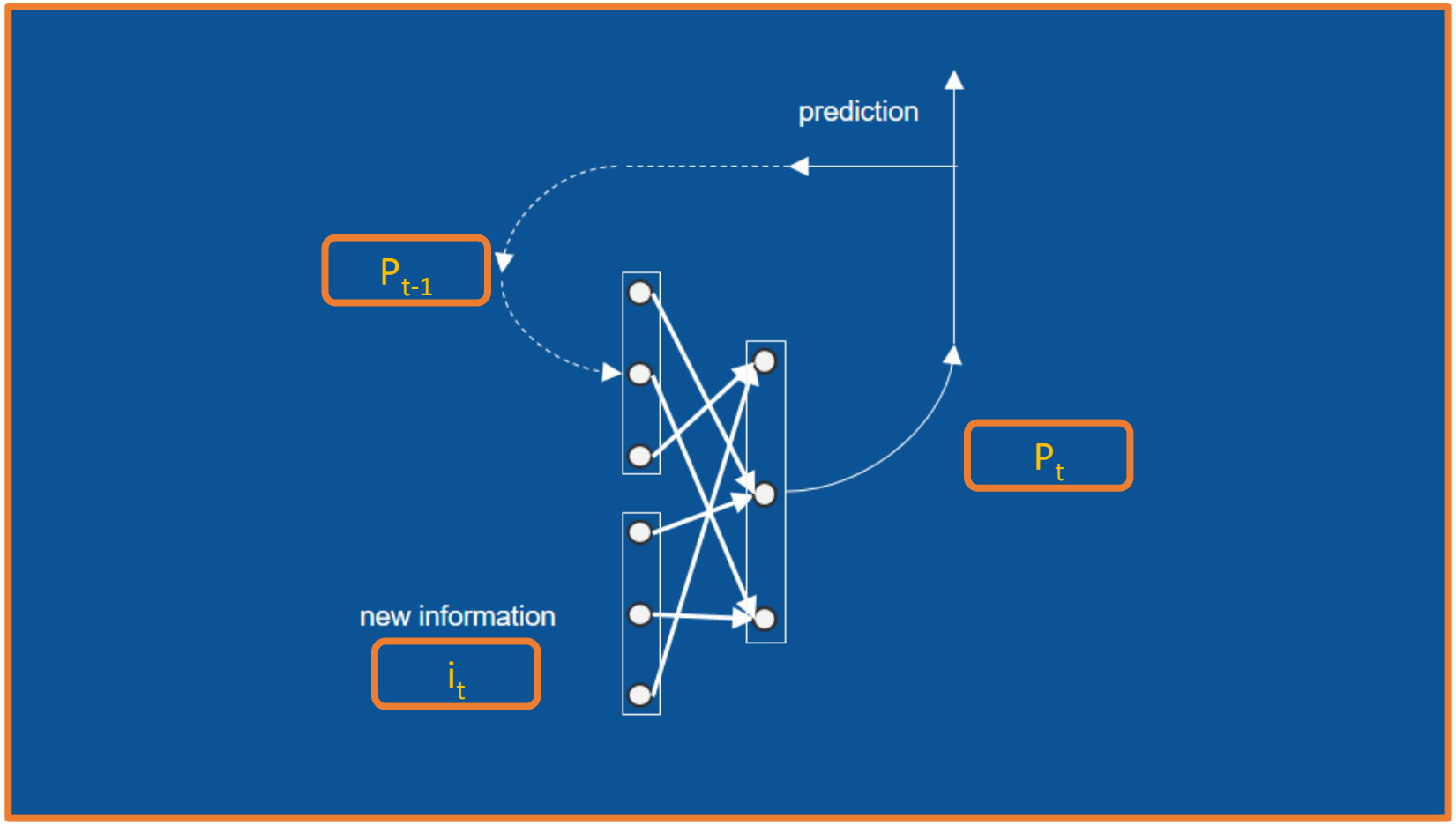


RNN - Create a feedback from output to the input

We can now connect the output to the input to create the predicted vector with a delay

Dotted line signifies the delay

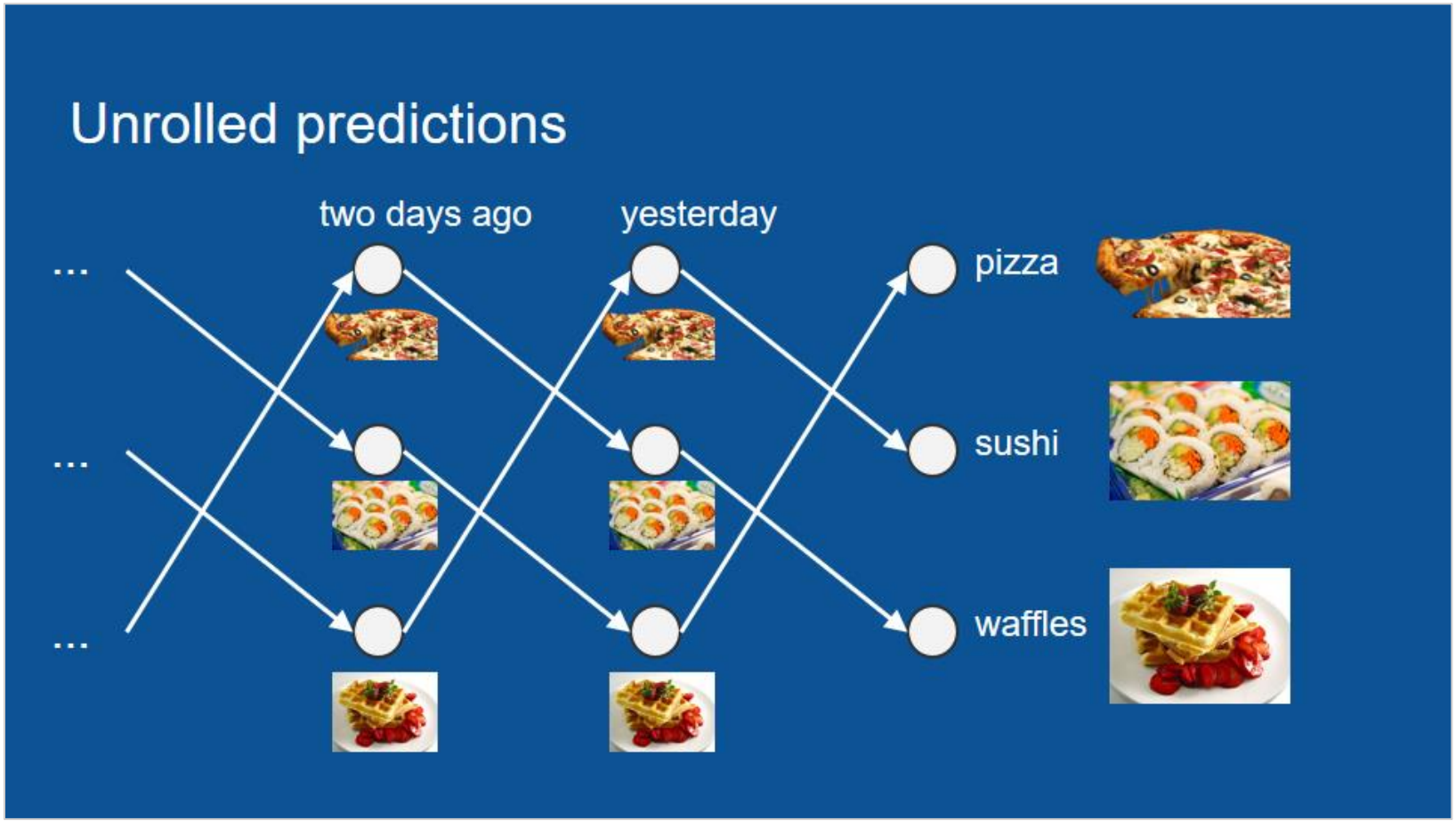
If the output vector denotes (t) then the feedback line denotes (t-1)



Dinner example - Unwrapped recurrent network

Now we can go as far back as we want

Let's say we have the dinner information for 2 weeks ago for example

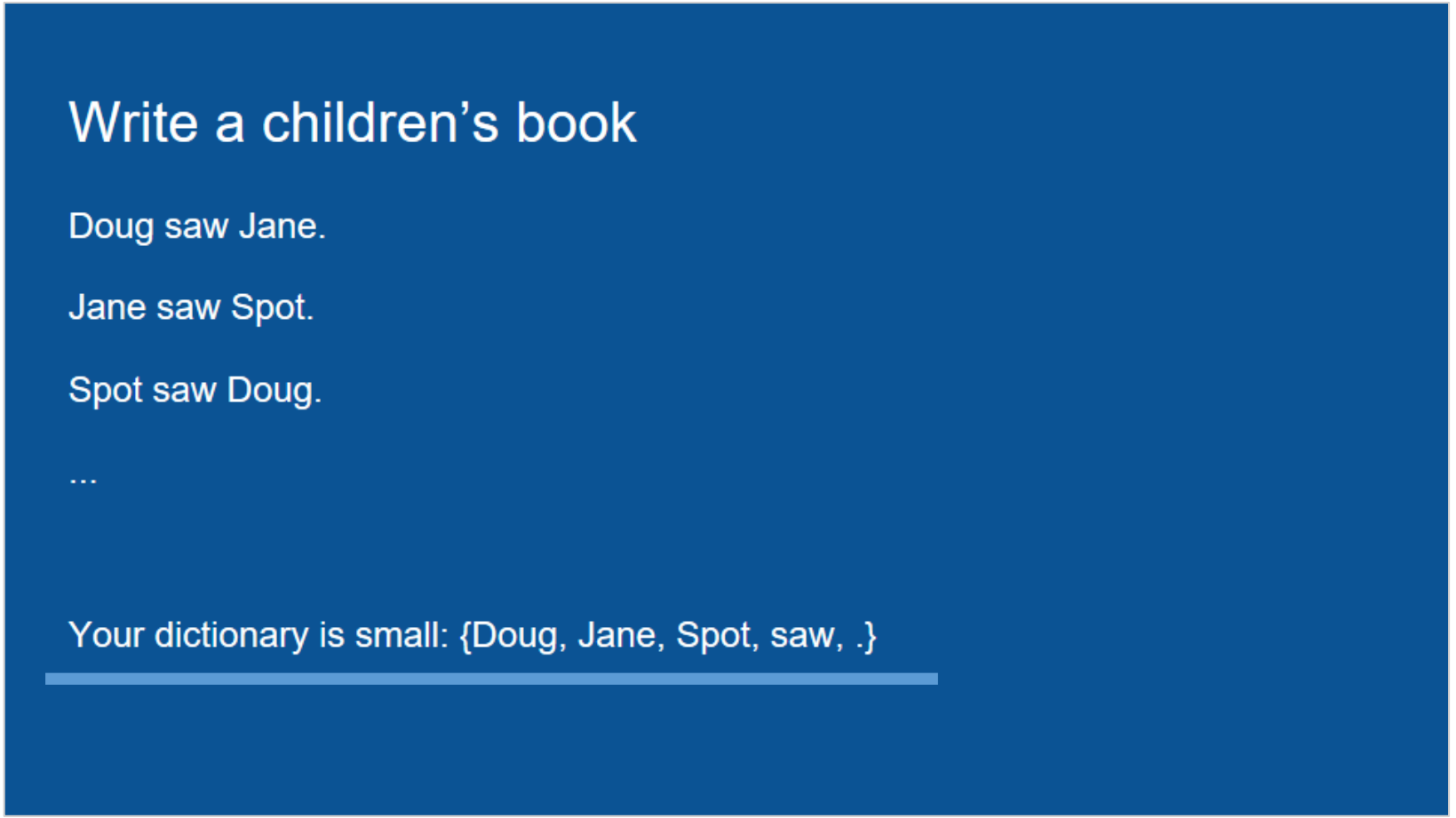


Example: A network to write a children's book

The collection and/or dictionary of the words that we have to write this book is rather small:

- **Doug**
- **Jane**
- **Spot**
- **saw**
- **.**

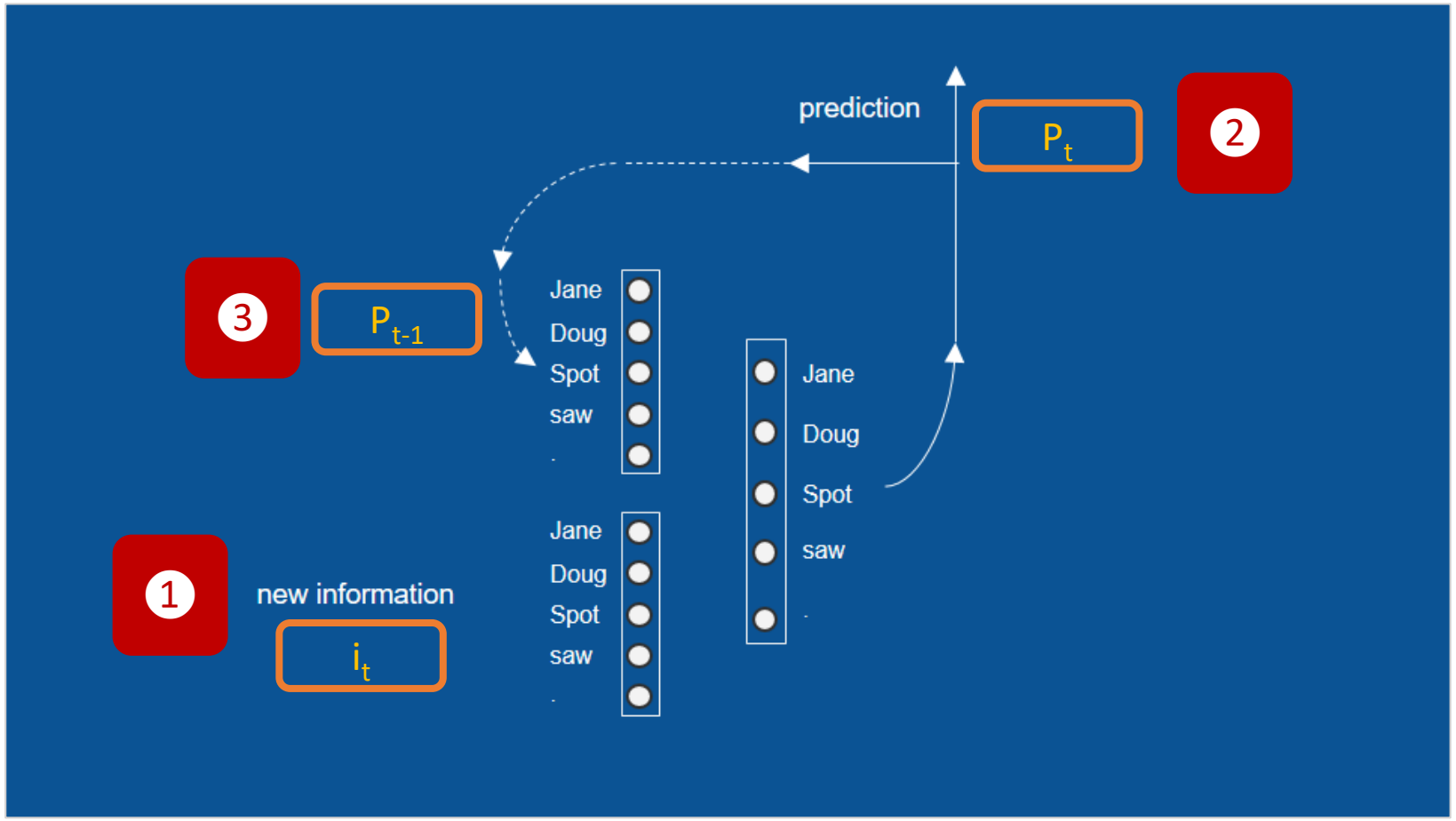
Objective: to put these words together in right order to write a book



RNN to write a book

- **3 vectors**

1. A vector of the words that we have now (i_t)
2. A vector of the prediction of the words (P_t)
3. A vector of the words that may come next (P_{t-1})



The new information (i_t) indicates what is the current word, e.g., if it is Doug then the vector is [0 1 0 0 0 0]'

Trained RNN – new information vector (i_t)

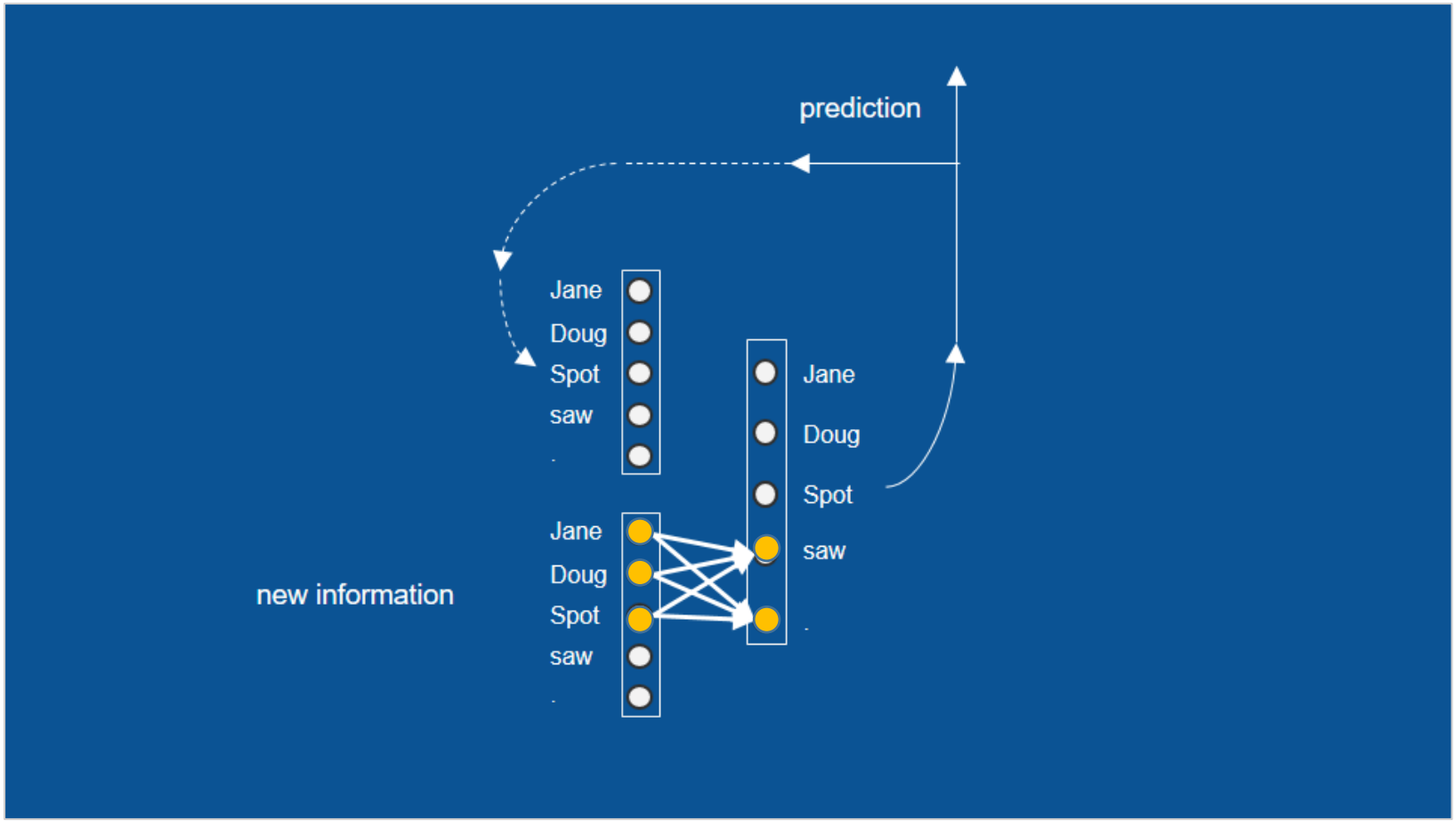
Let's try to work out this RNN

After the training is done when the <u>new information</u> is

- Jane,
- Doug or
- Spot

we expect that the trained RNN would point to

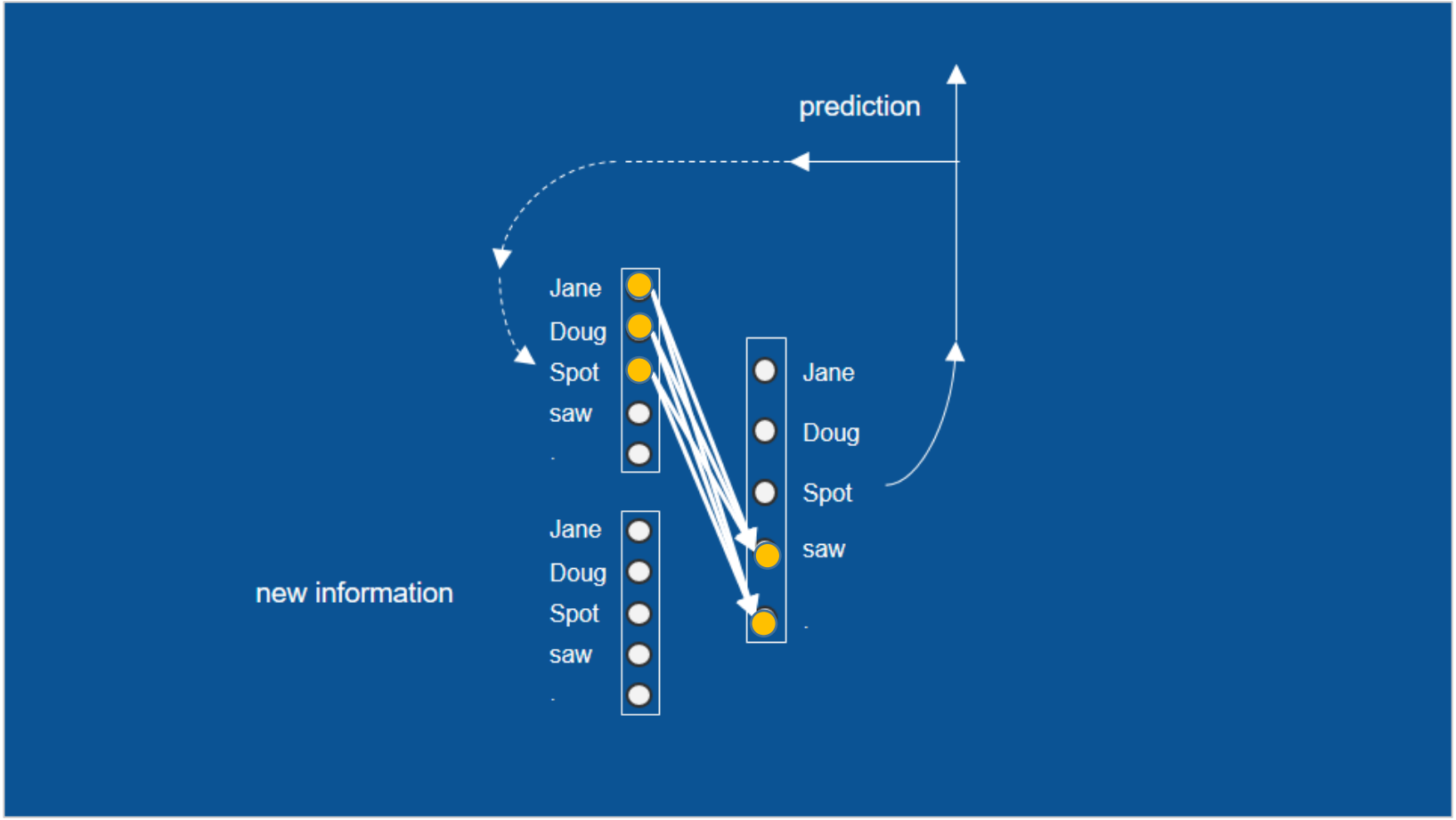
- saw or
- .



Working out our RNN – prediction vector (P_{t-1})

if the predicted word is
- Jane, or
- Doug, or
- Spot

Similarly we expect that the trained net would point to
- Saw, or
- .

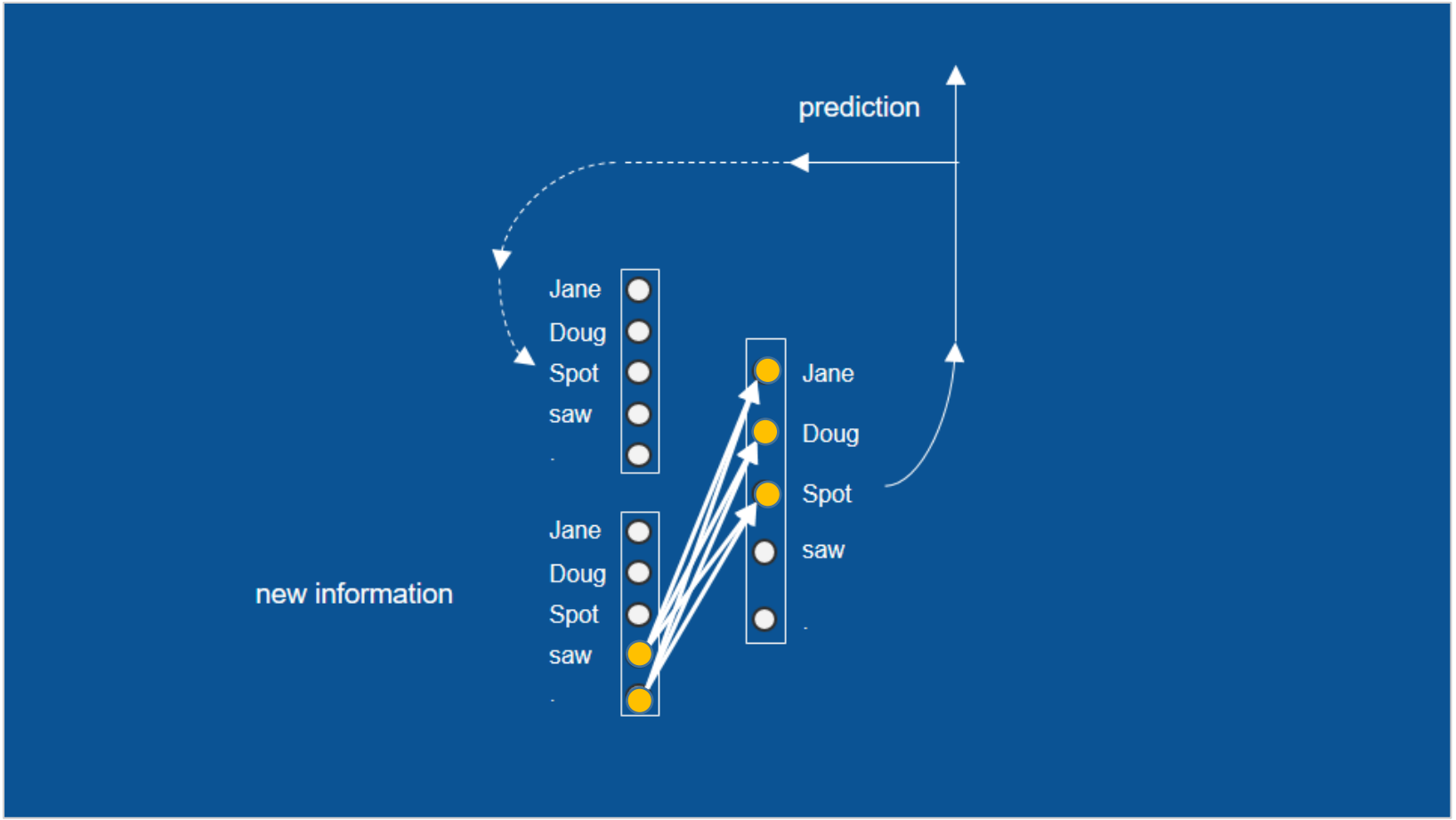


Working out our RNN

if the present word is
- saw, or
- .

The trained net would point to
- Jane or
- Doug
- Spot

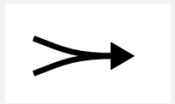
As a name should appear after saw or .



A representation for our RNN

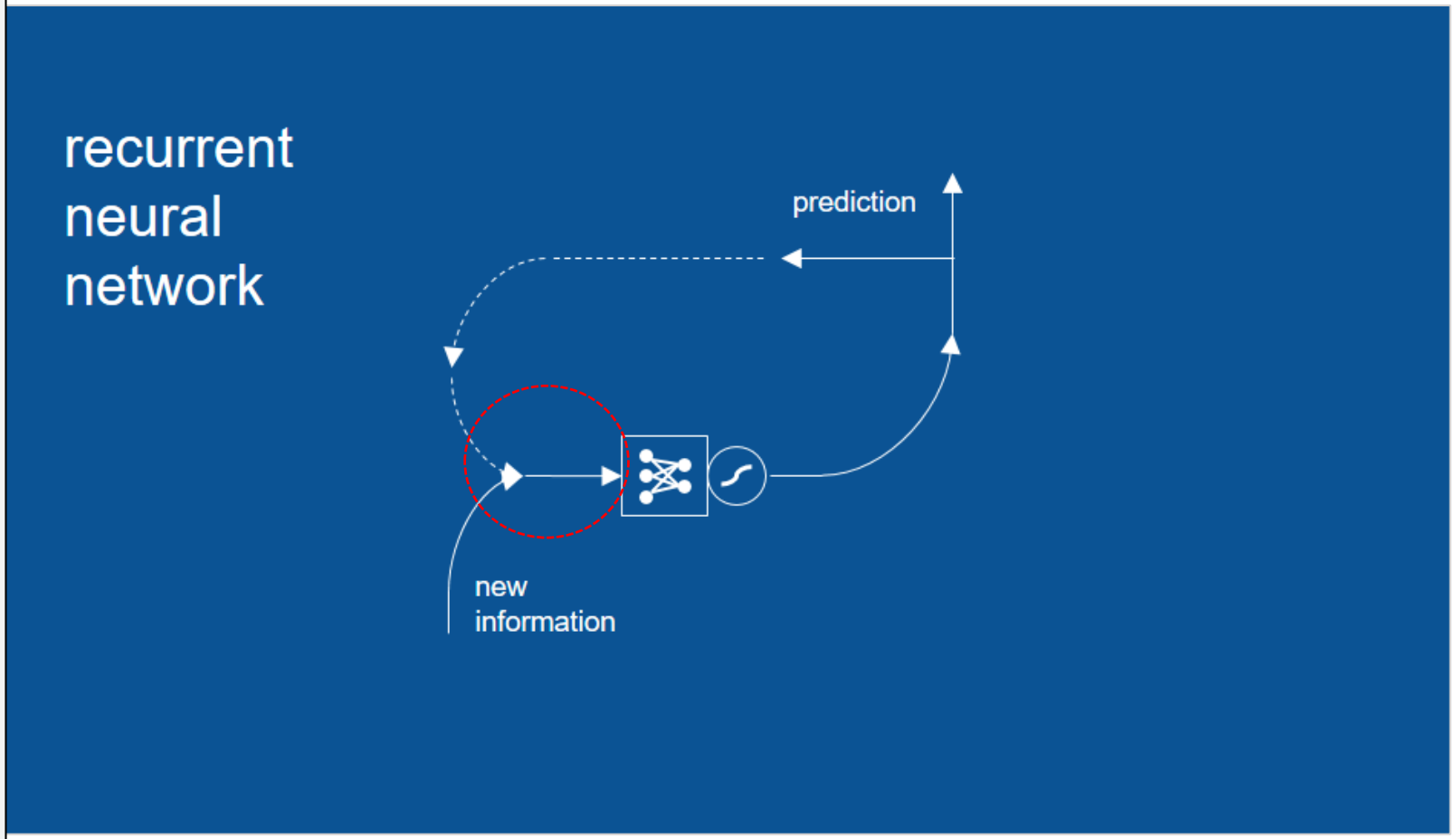
The input is a collection (concatenation) of the new information and the predicated values

This is denoted by



The activation function used here is tanh denoted by

Making the output behave well



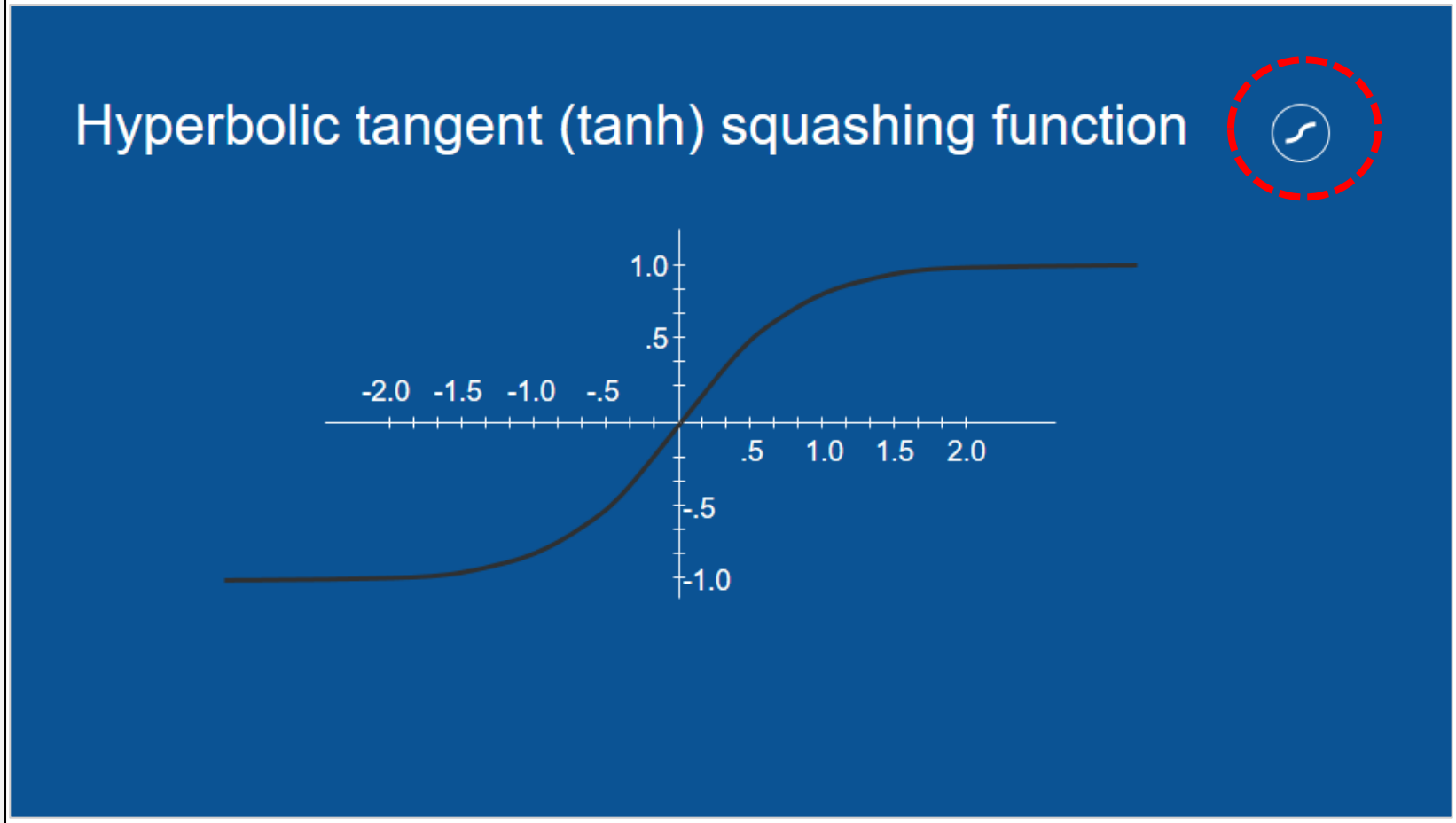
Side note – how does tanh work

Tanh is the squashing function
Regardless of the input everything will always be between -1 & +1 (very important)

For the input values between -1 & +1 the output value is very close or equal to the original input

For the values greater than +1 the output value is +1

For the values less than -1 the output value is -1

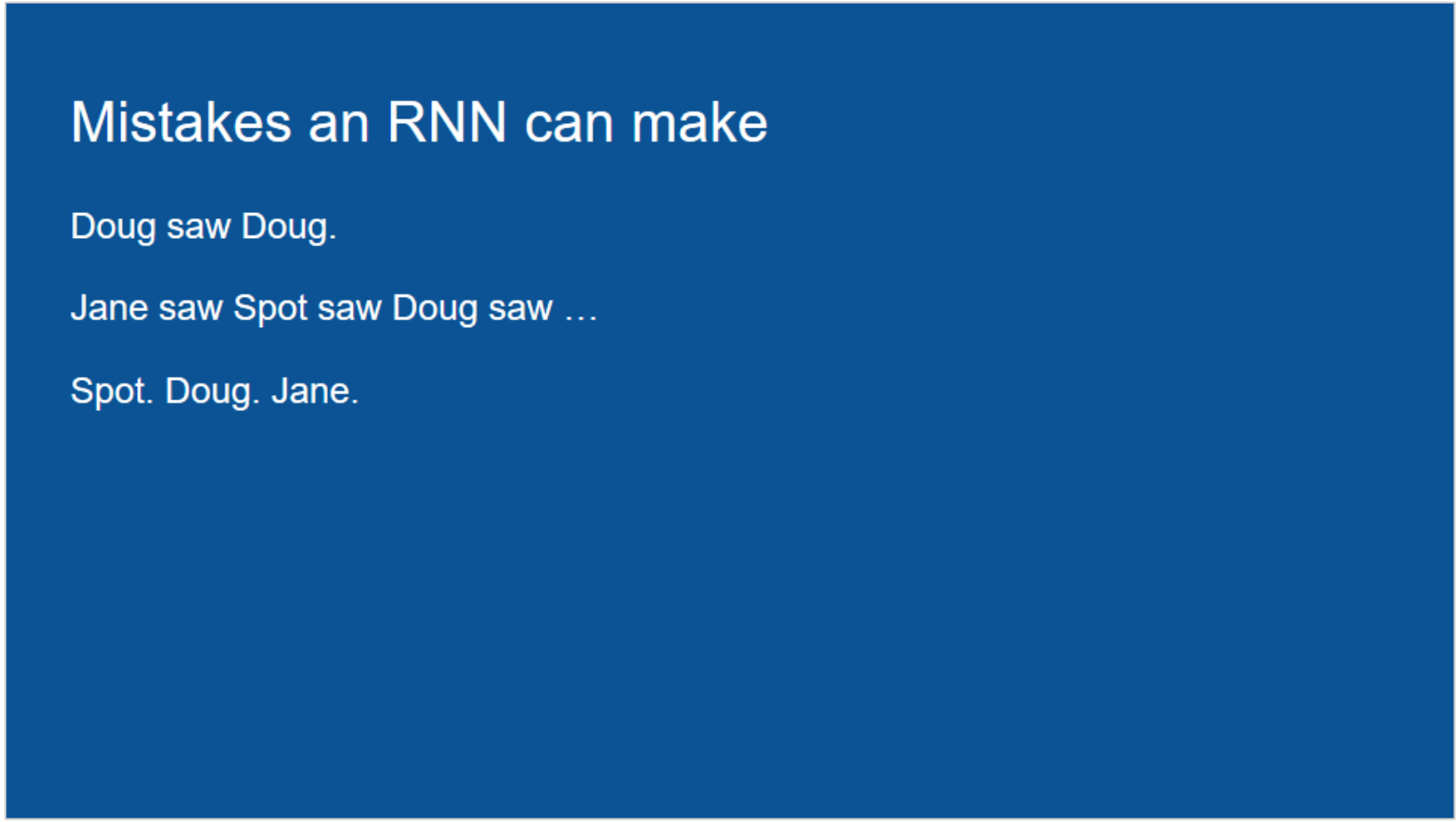


Why RNN may not work ?

Doug saw Doug.
(after saw we expect a name that name could be Doug)

Jane saw Spot saw …
(after saw we expect a name and after a name we can expect saw …)

Spot. Doug. Jane.
(after a name we can expect .)



What may not work so far?

Problem:

We have short term memory

We only look back one time step & do not use the information from further back

Mistakes an RNN can make

Doug saw Doug.

Jane saw Spot saw Doug saw …

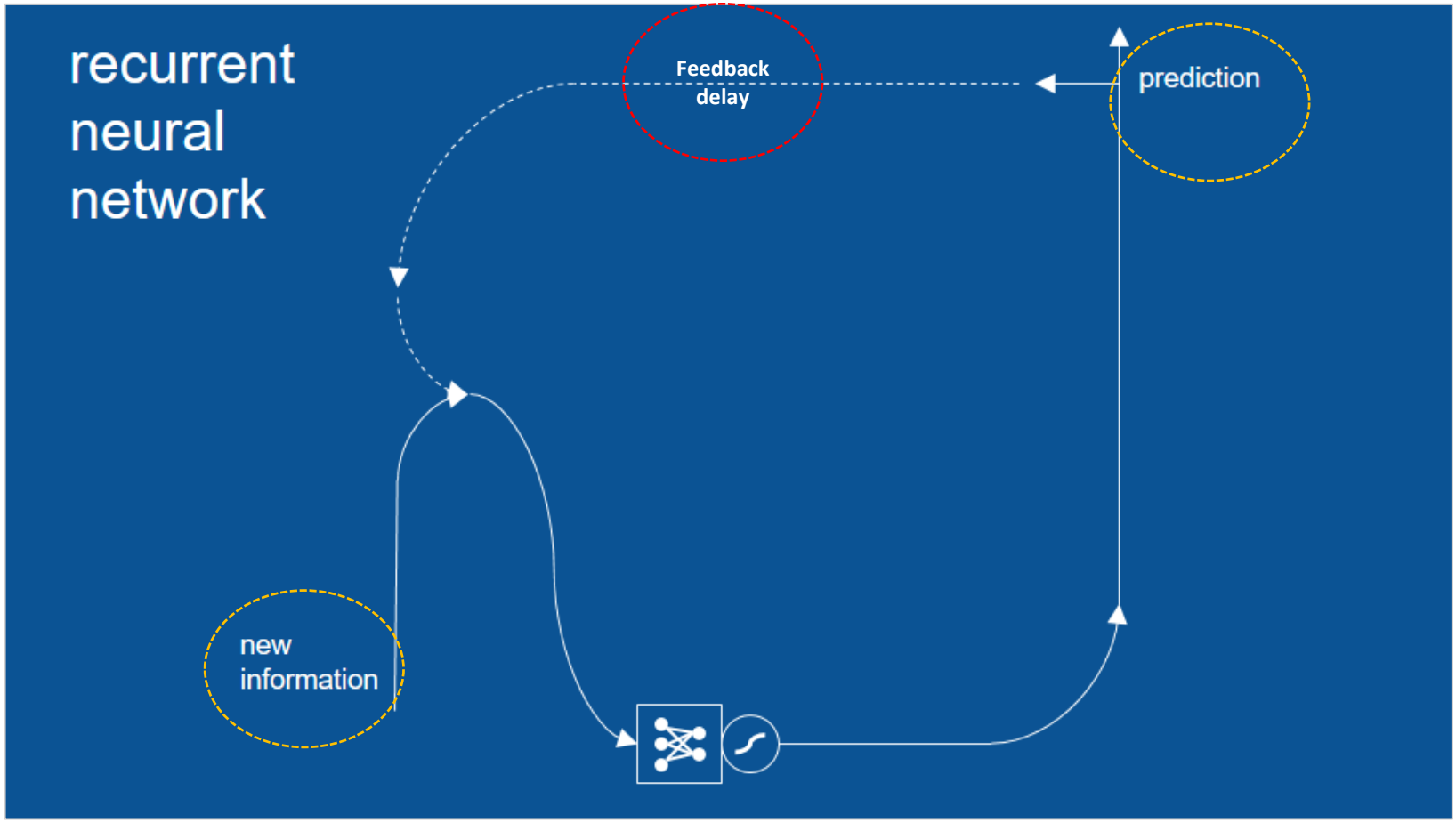
Spot. Doug. Jane.

RNN

A simple architecture of a RNN

Your input is a combination of:

- the new information &
- what you predicted in the last step (time wise)

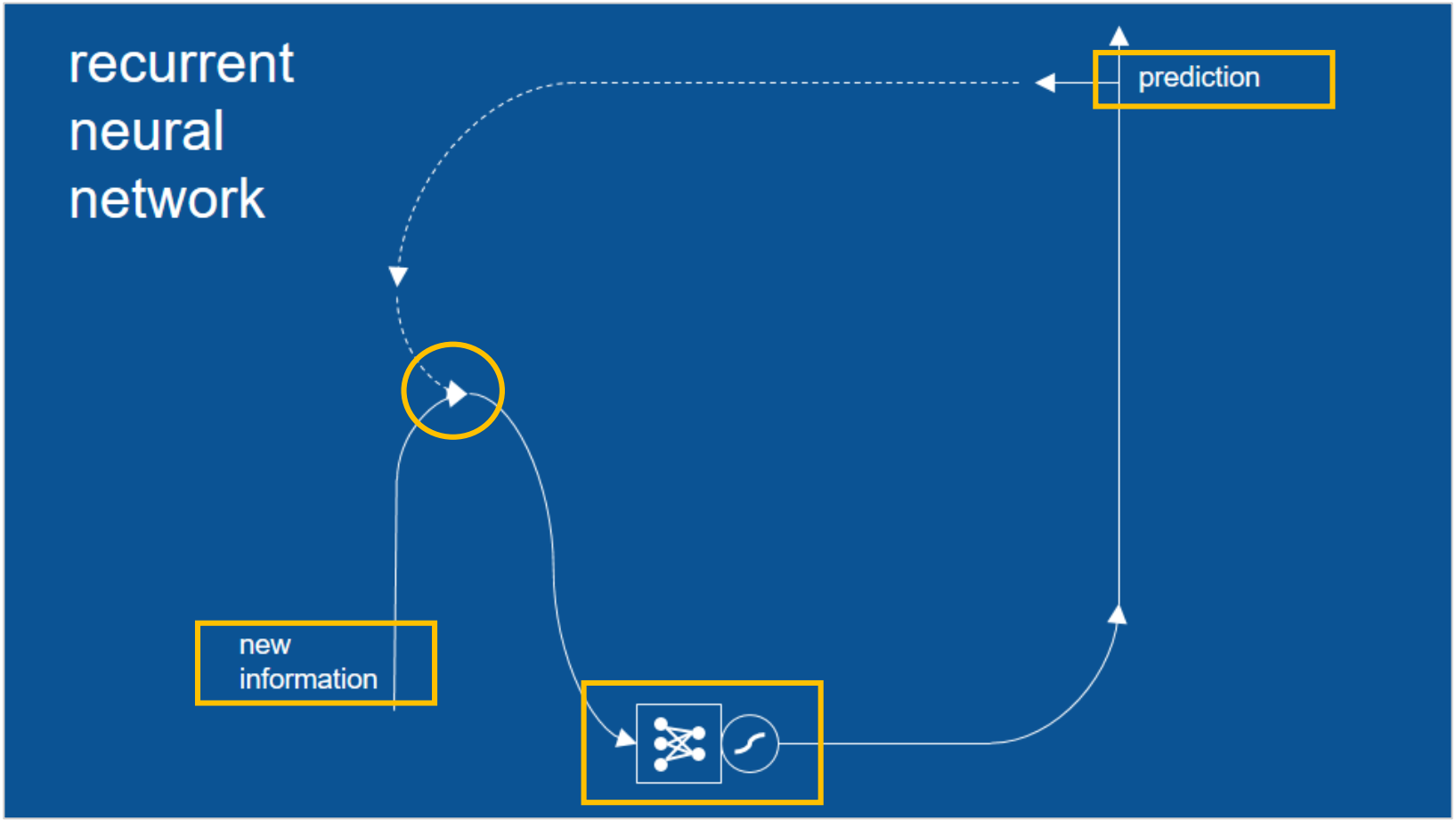


How do we fix this?

We need to modify the existing architecture

One solution is to add a memory capabilities

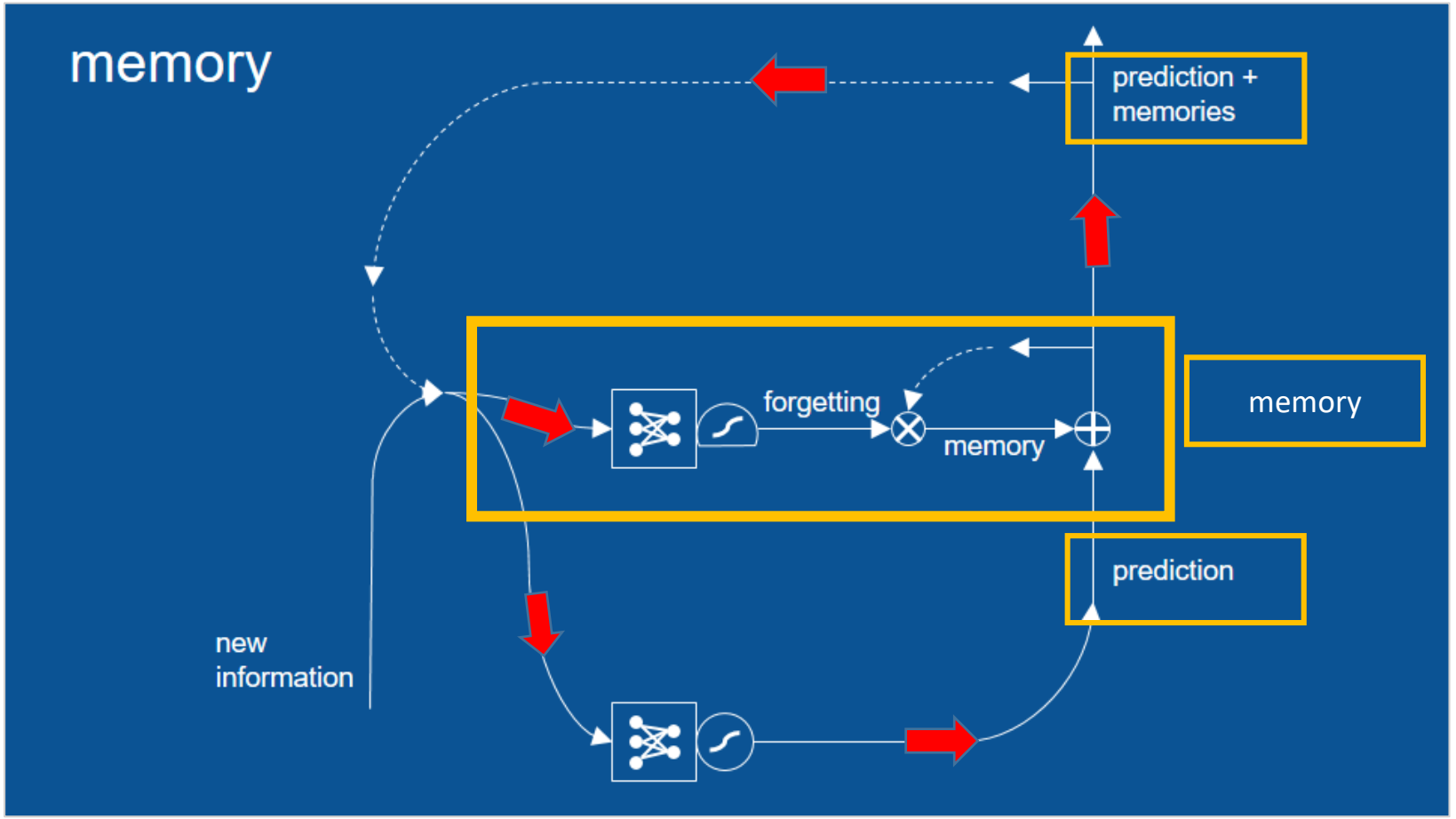
How do we add a memory component ?



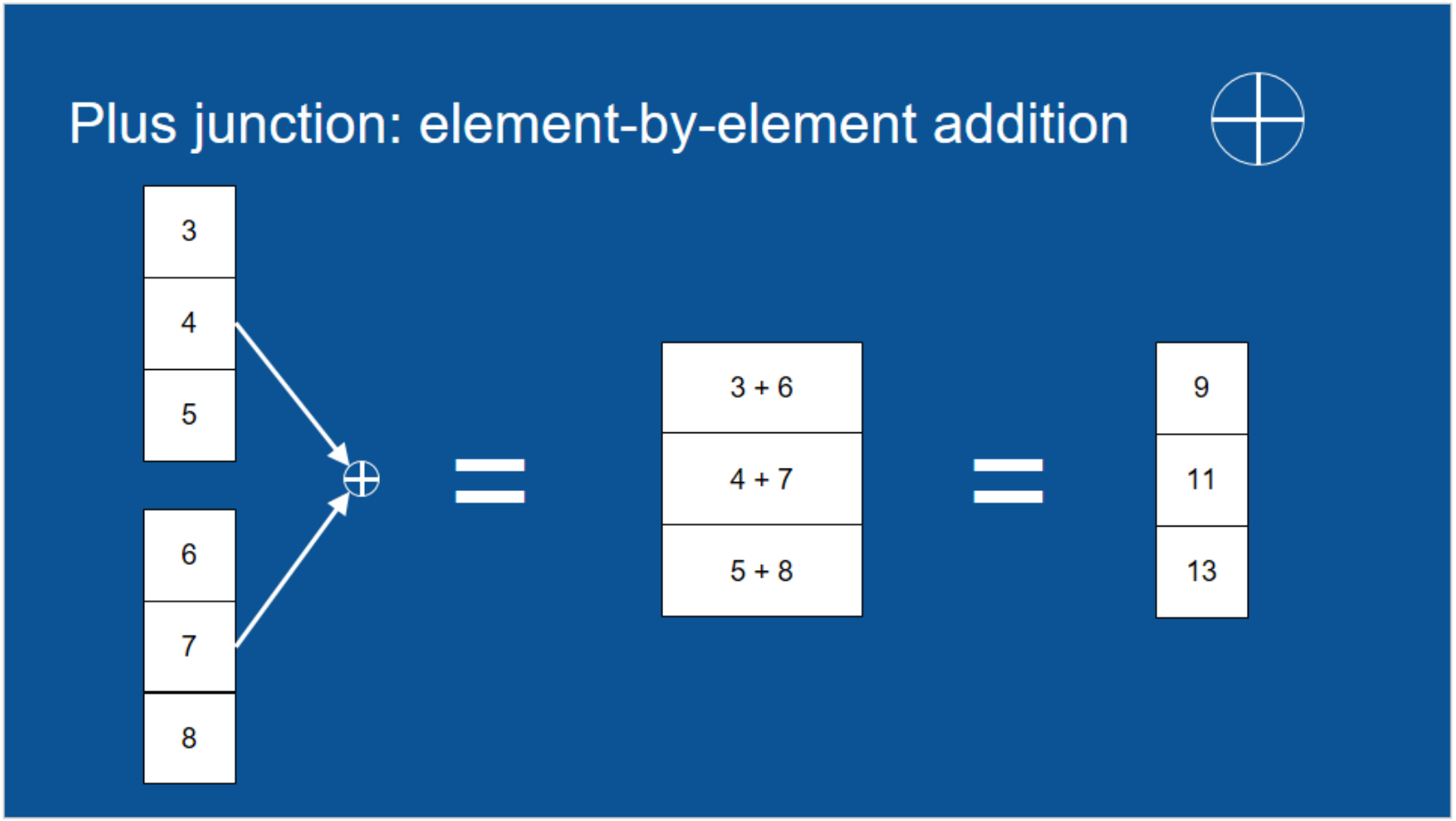
Introduction of the memory component

Adding memory component

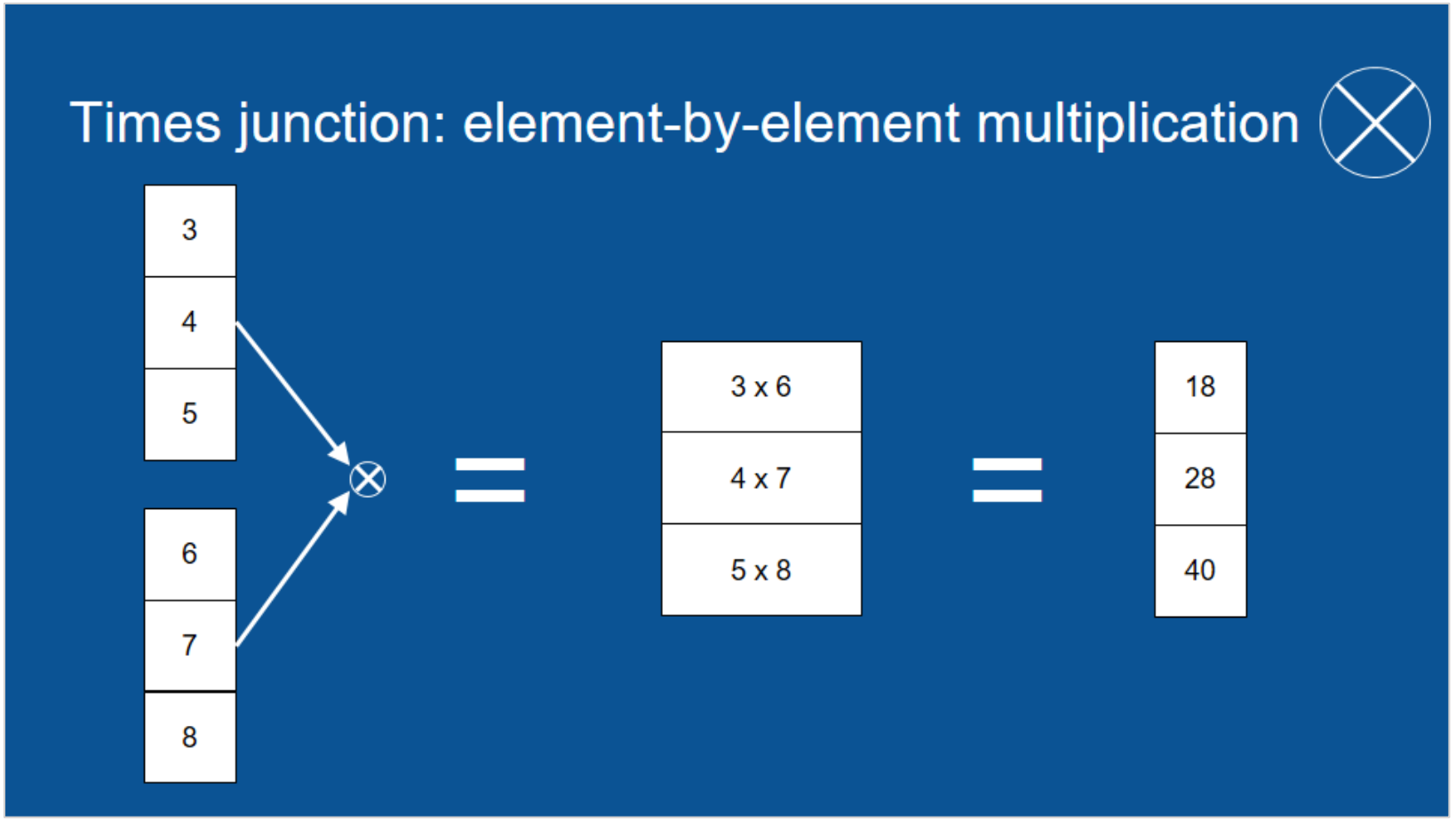
to enable the network to remember what happens many steps ago (from further back)



Side note - Element-by-Element Addition/Plus Junction



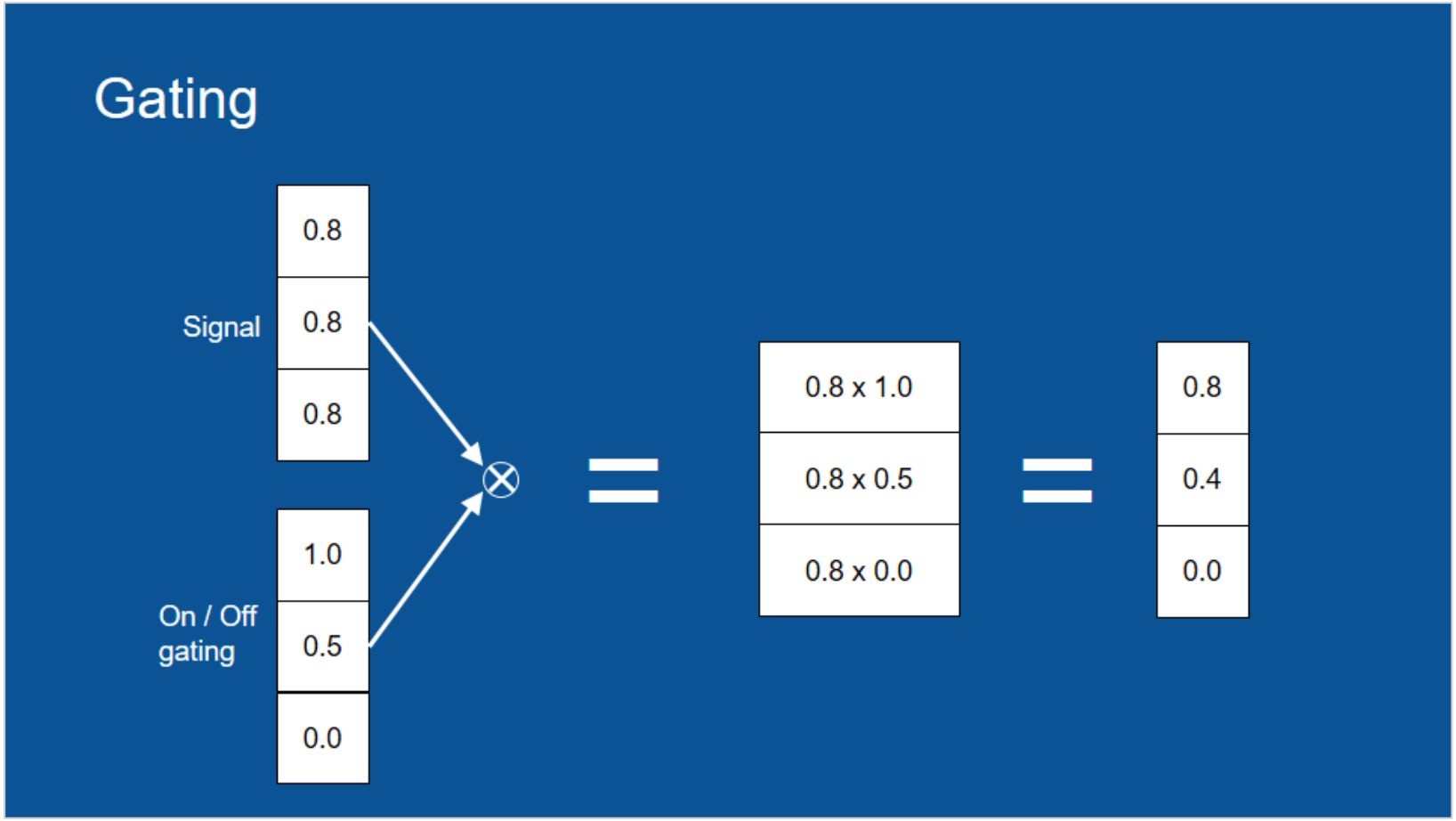
Side note - Element-by-Element Multiplication/Times Junction



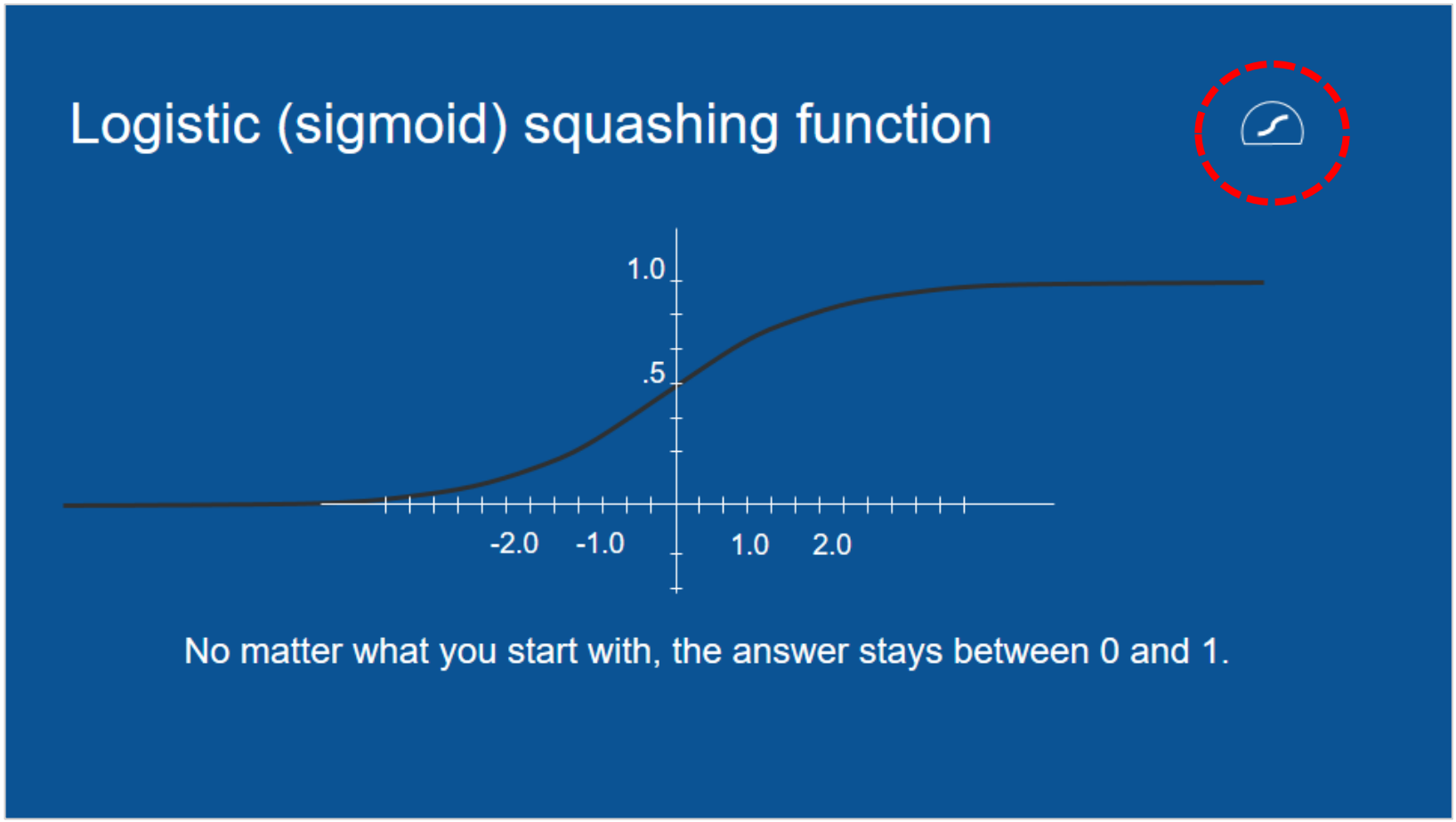
Gating

We can use time junction to control what percentage of the an input (a signal) goes through, i.e., **gating**

In this example, the 1st element of the signal goes through completely whereas the 3rd element is completely masked



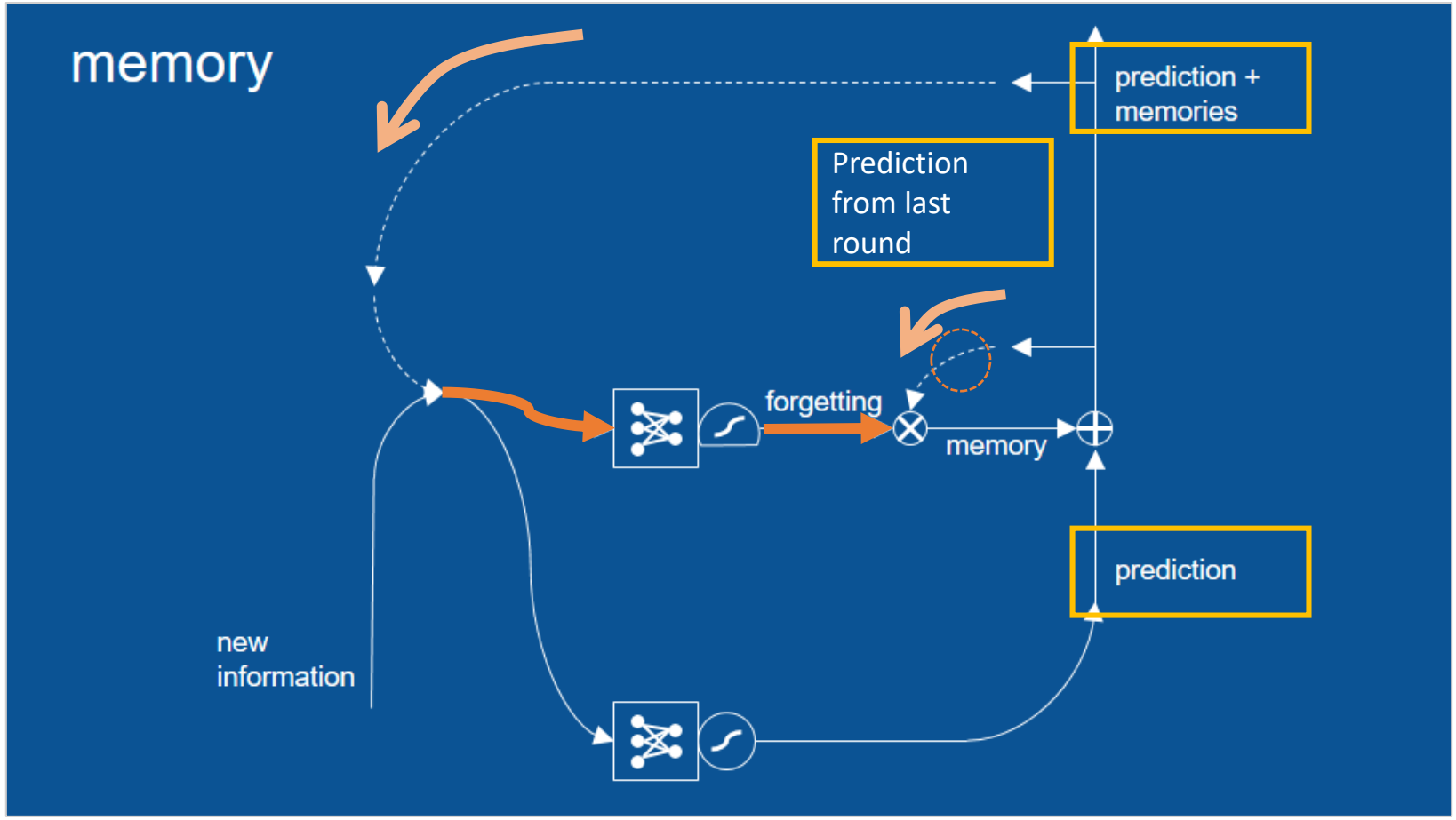
Side note - Sigmoid Function



Memory Component: forget & keep

Memory component:

- To **forget** some of the previous prediction and
- to **keep** the rest



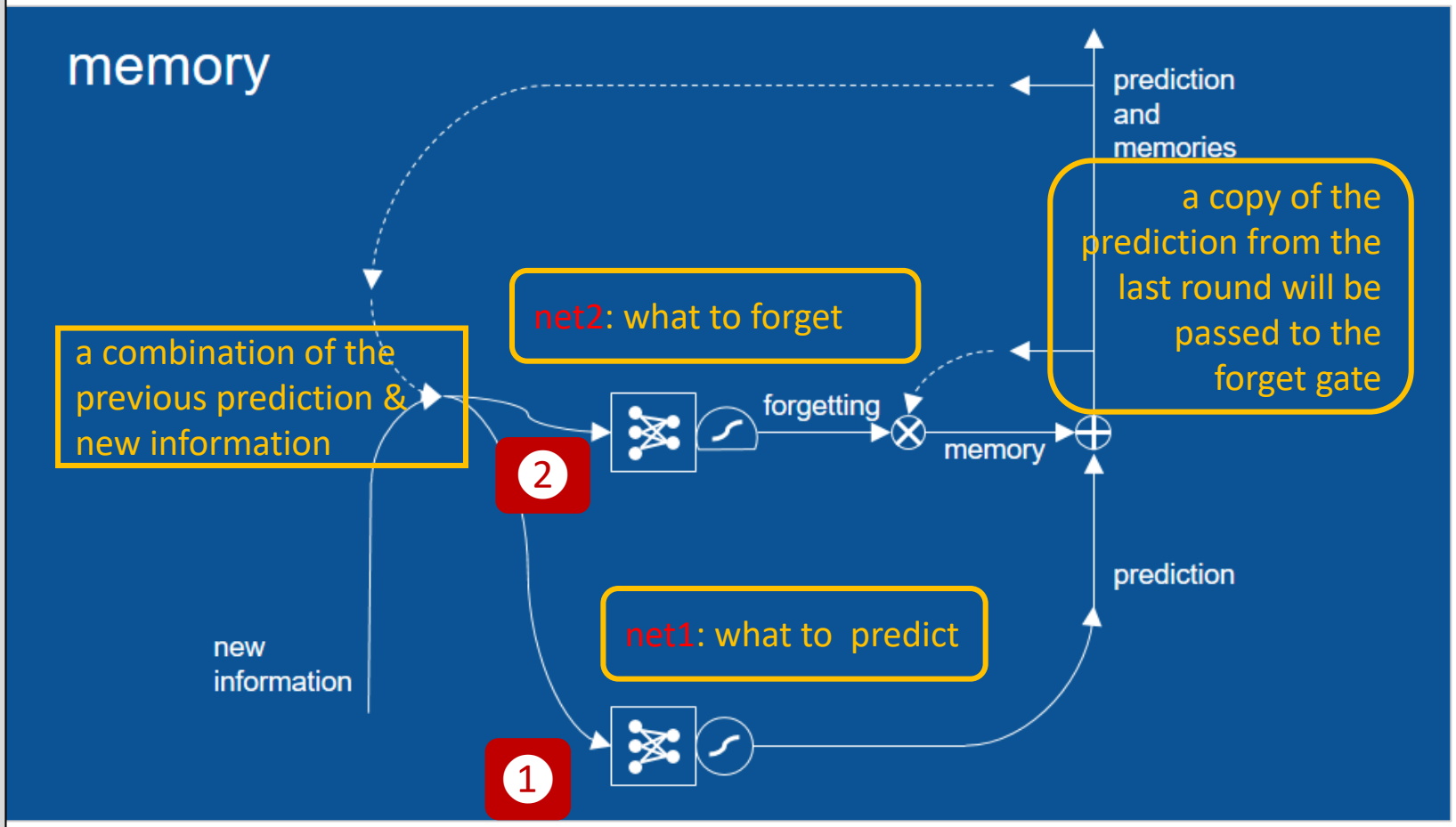
How does the forget gate work?

1. A combination of the previous prediction & new information goes thru net1 & a prediction is made accordingly

2. A copy of the prediction will be given to the forget gate

Note:
net2 is different from net1 & its task is to learn <u>what</u> to forget & <u>when</u> to forget

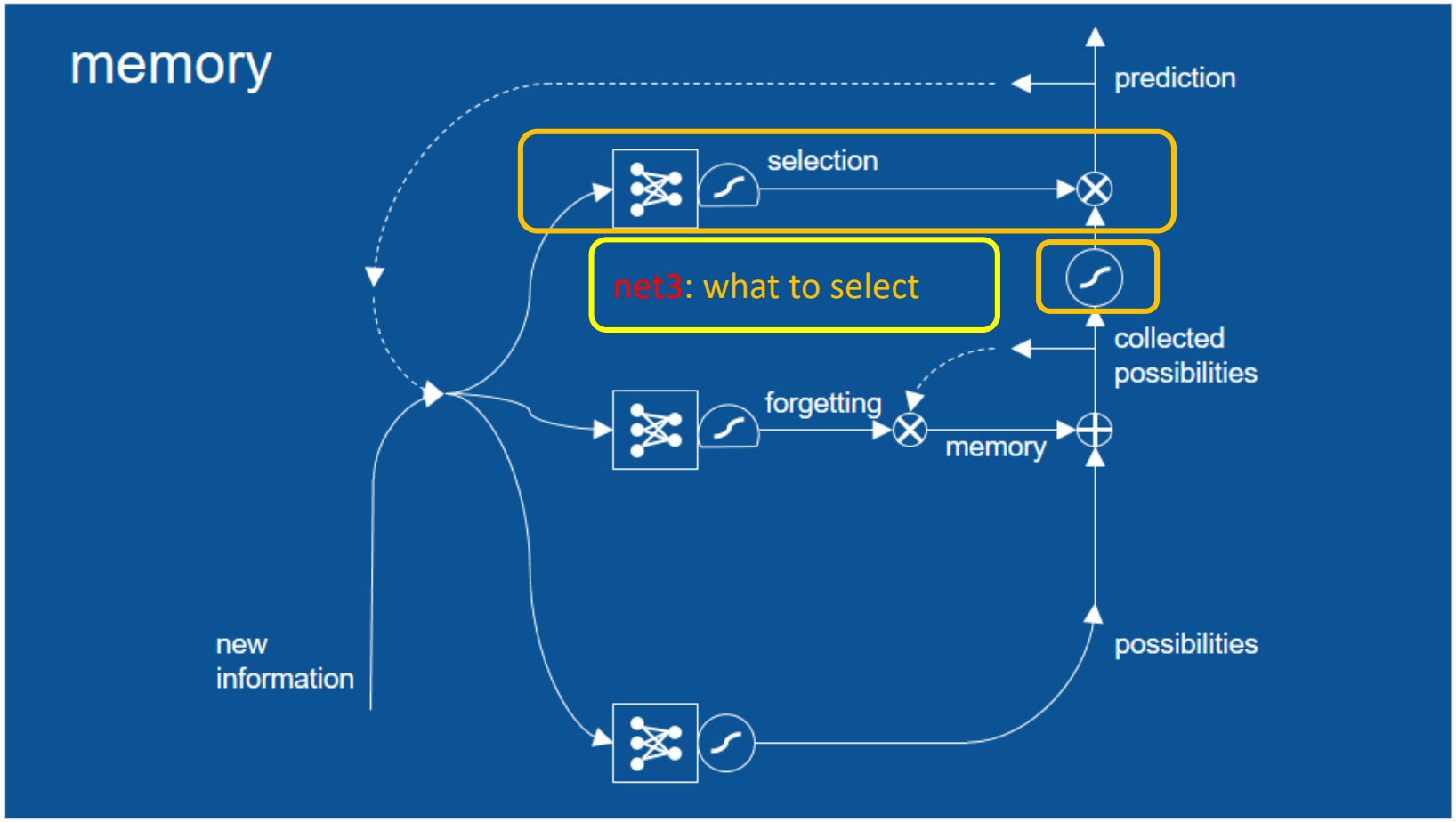
A part of this will be forgotten & the remaining will be added to the prediction



Add a selection layer – net3

We do not necessarily need to send the entire prediction to the input/output

To select with part of the prediction goes back to the input/output

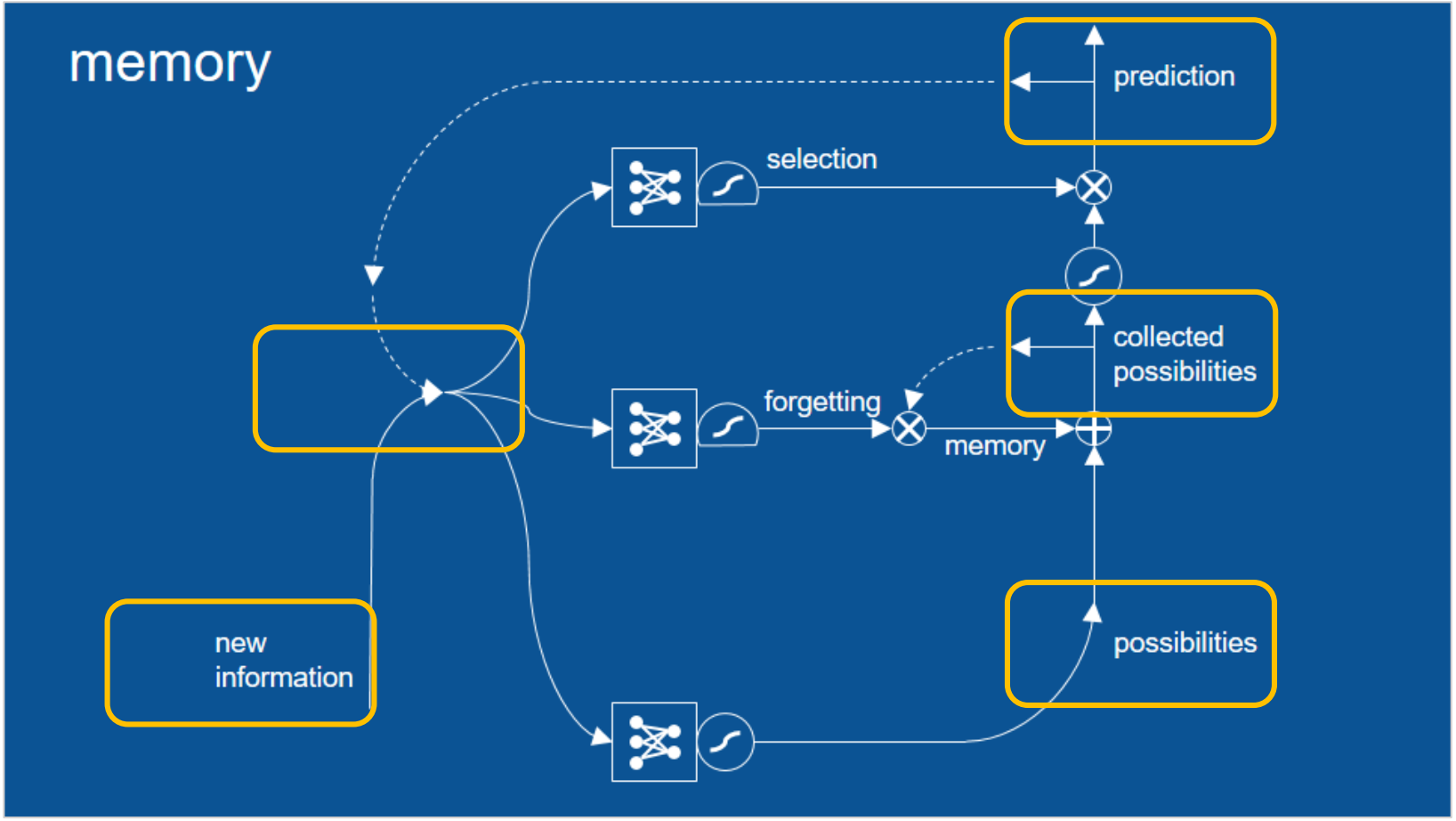


How does the selection gate work?

In the previous layer (forget/keep) we combined our memory with our prediction

1. We need to have a filter to select which part of combined memory + prediction to go out

2. We also need to add a new tanh after the elementwise add to make sure everything is still bet -1 & +1 (addition might have caused an increase beyond -1/+1)

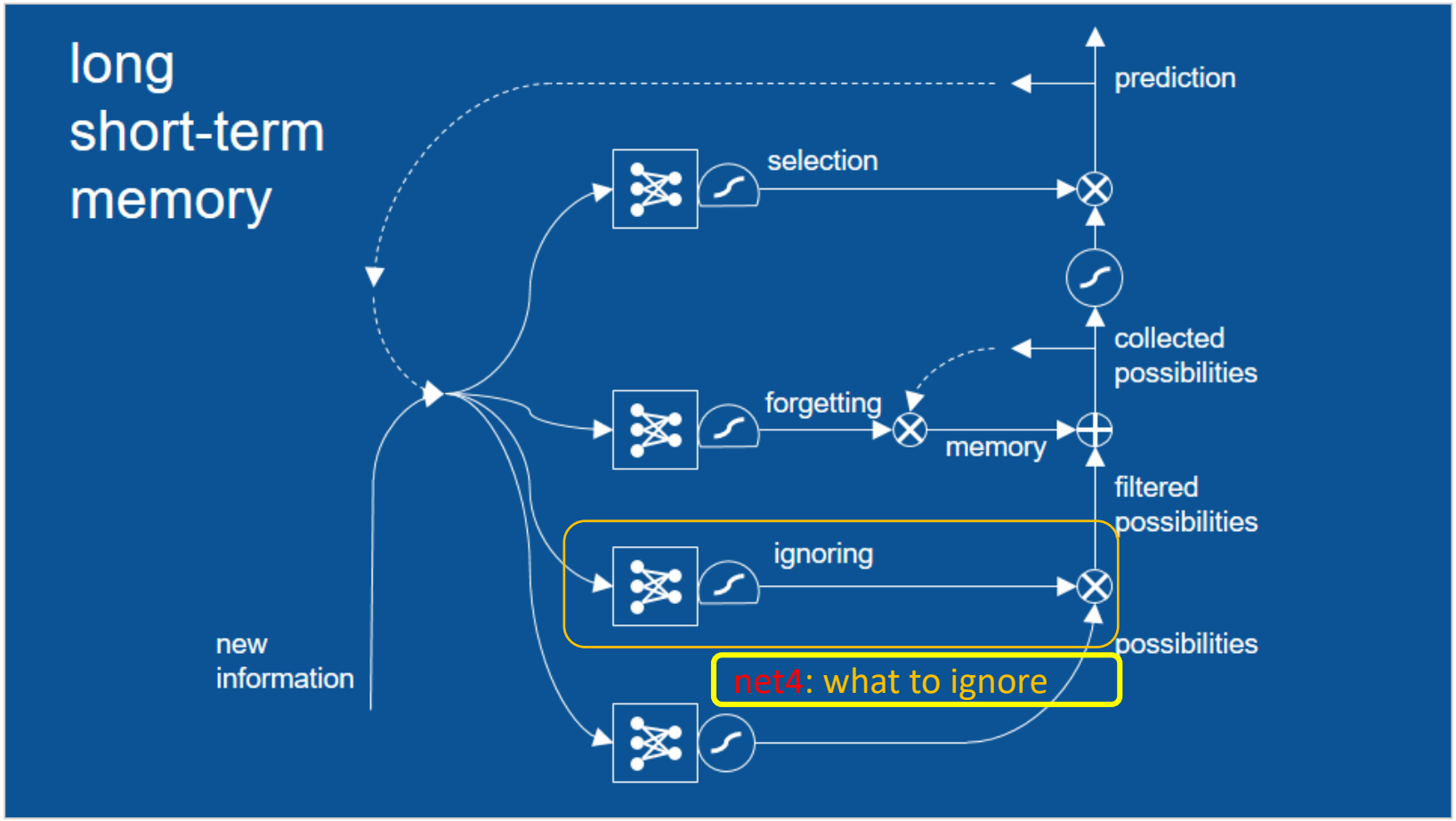


Where does learning happen so far?

- **net1**: to learn to **PREDICT**
- **net2**: to learn what to **FORGET/KEEP**
- **net3**: to learn what to **SELECT**

Add an ignore/attention layer – net4

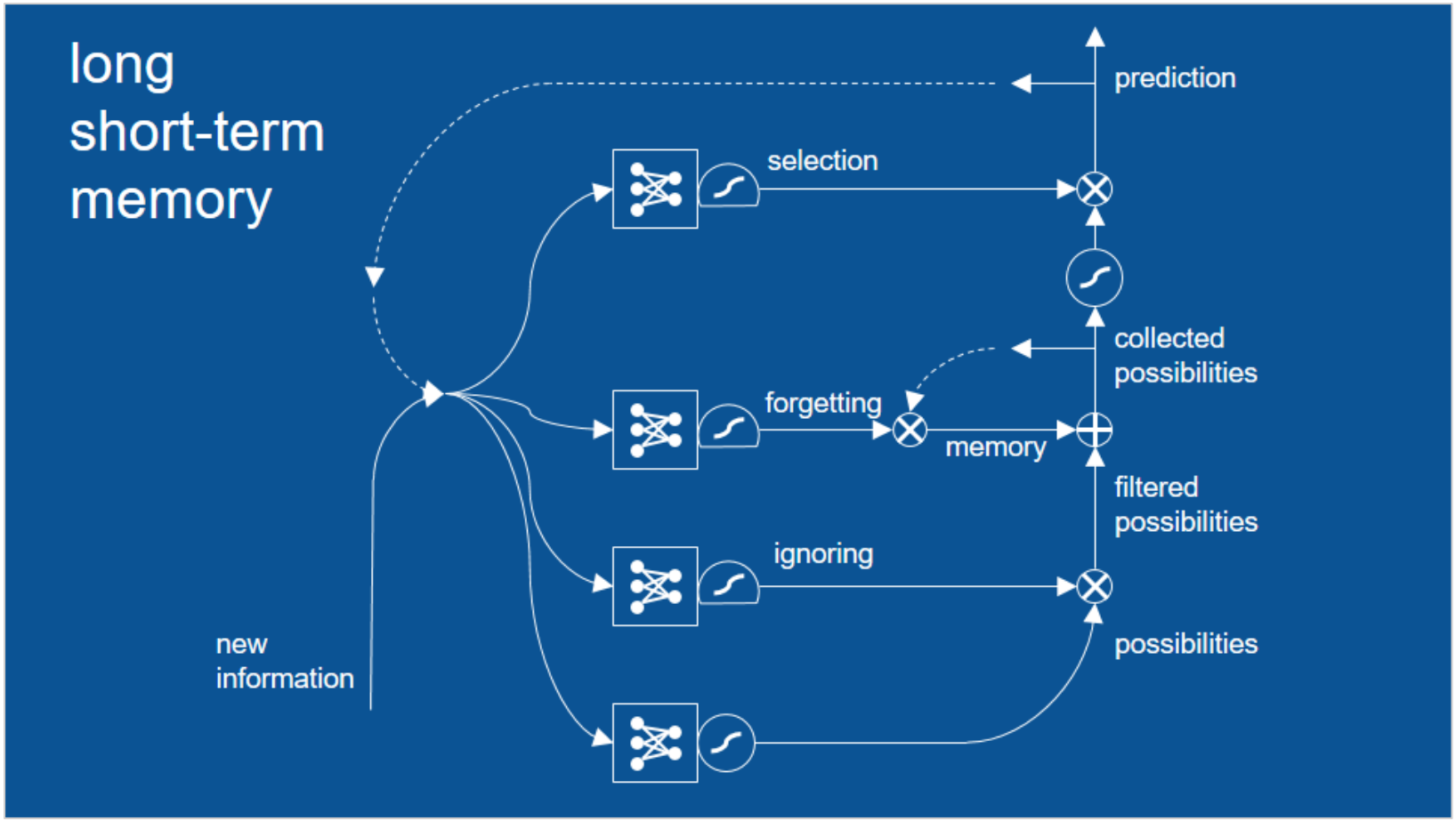
To ignore some of the possible predictions



How does the ignore layer work?

Some of the possible predictions that are not immediately relevant to be ignored

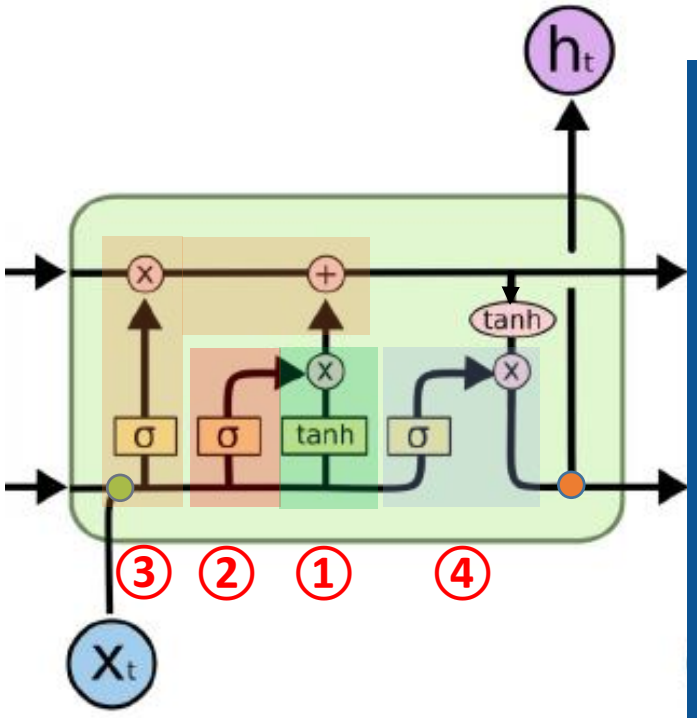
Not to unnecessarily complicate the predictions (by having too many of them) in the memory as going forward

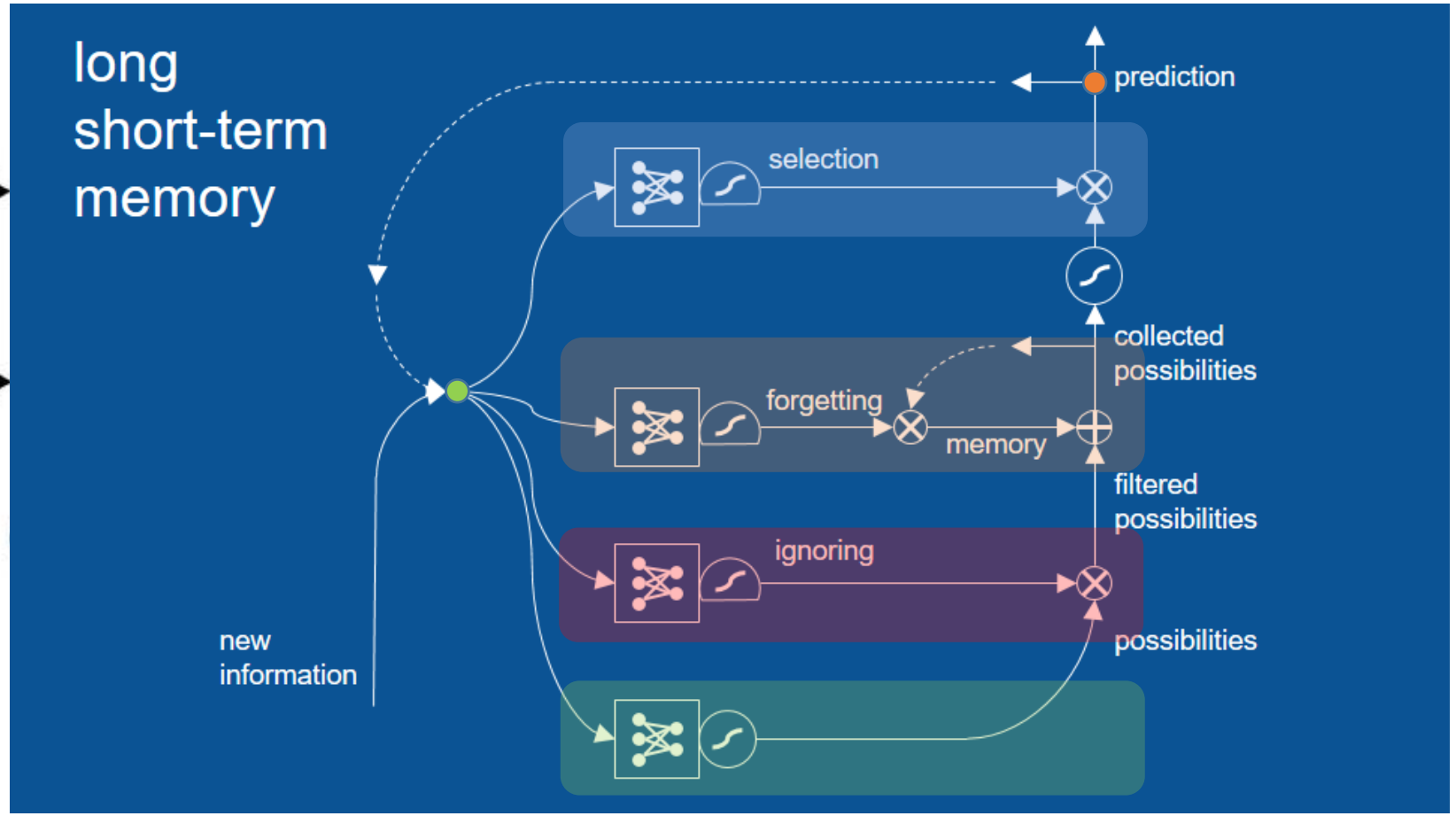


Where does learning happen?

- **net1**: to learn to predict
- **net2**: to learn what to forget/keep
- **net3**: to learn what to select
- **net4**: to learn what to ignore

LSTM Structure





Side note

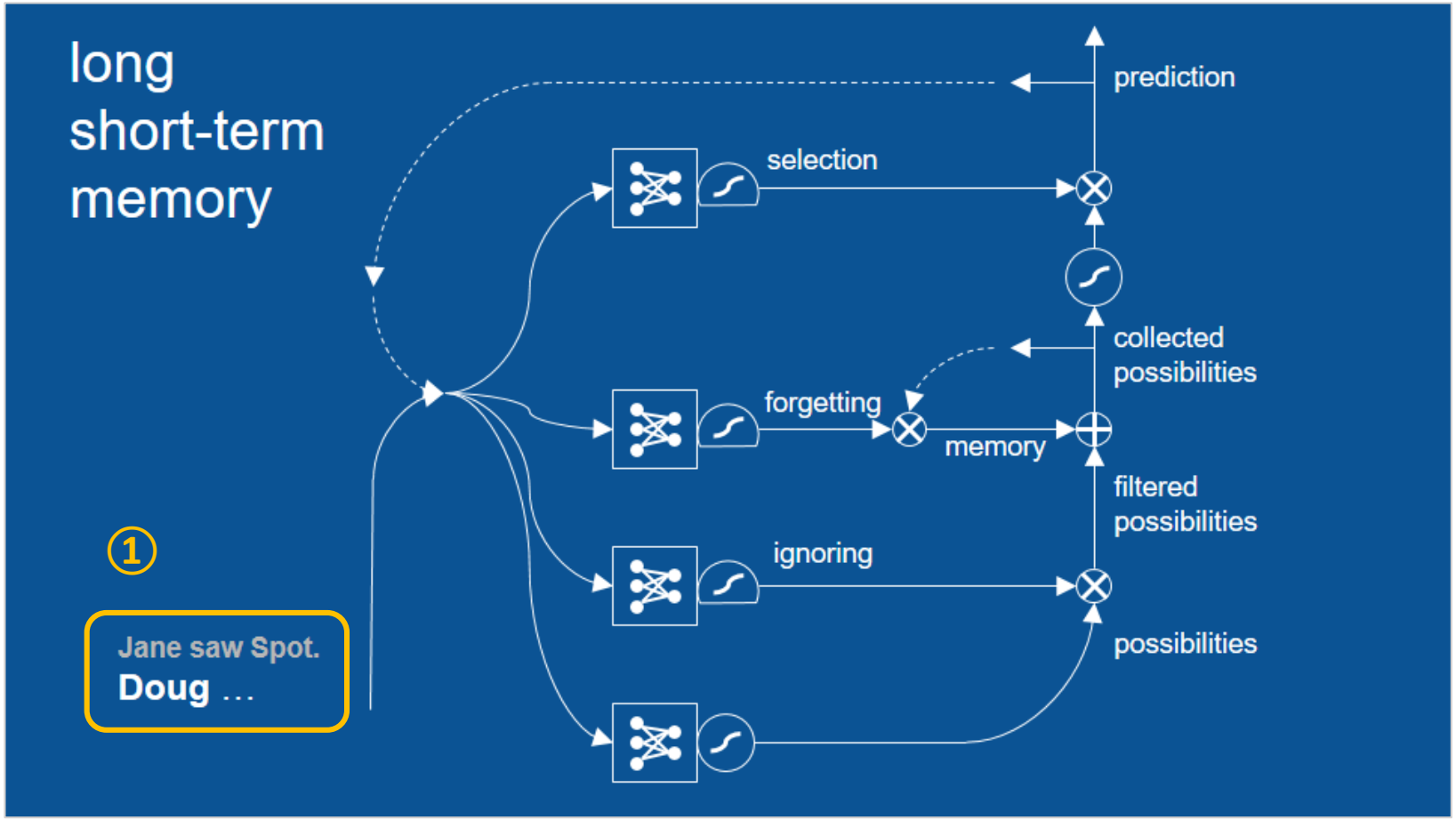
- A multiplicative input gate unit learns to protect the constant error flow within the memory cell from perturbation by irrelevant inputs

- Likewise, a multiplicative output gate unit learns to protect other units from perturbation by currently irrelevant memory contents stored in the memory cell

Running a simple example

Assume this LSTM is already trained

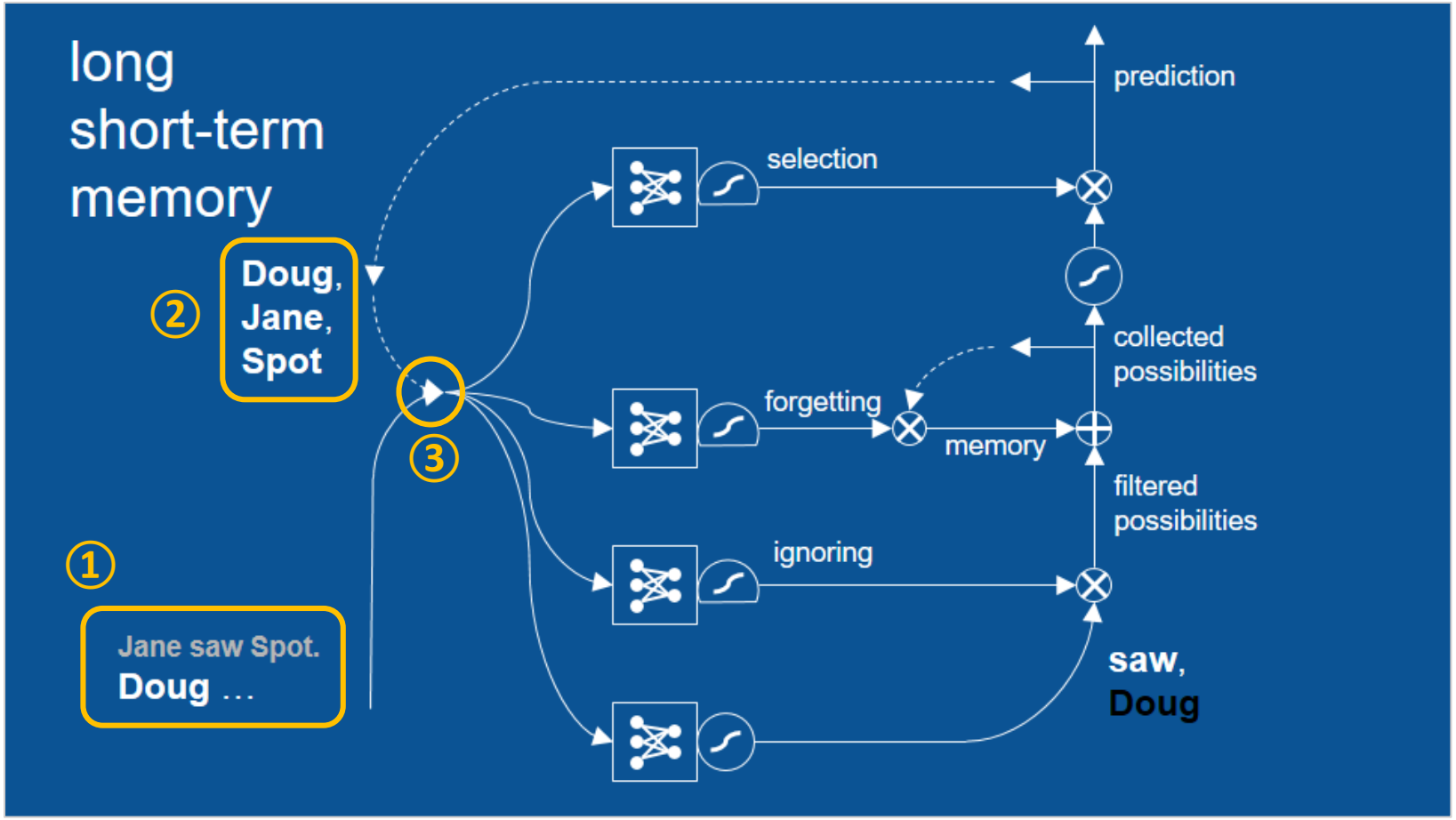
net1, net2, net3 ,net4 are known



Information going through

① So far we have …
"Jane saw Spot."
and the new word is "Doug"
② We also know from previous prediction that the next word can be "Doug, Jane, Spot"
③ We pass this info through net 1, 2, 3, 4 to

1. Predict
2. Ignore
3. Forget
4. Select

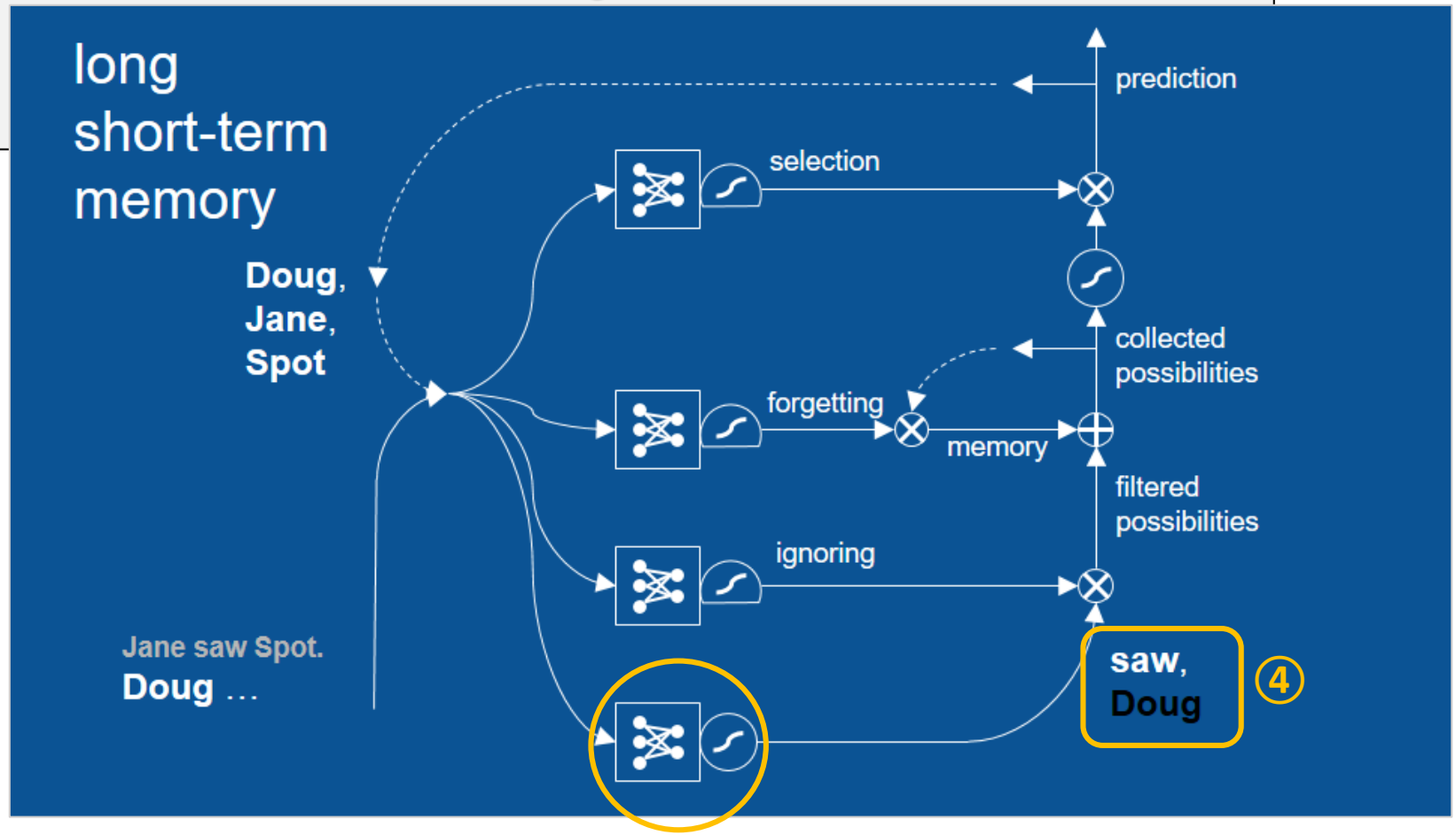


net1 - Prediction Step

④ The new word is "Doug", net1 should predict that the next word is "saw"
Also, net1 should know that since the new word is "Doug" it should not see the word "Doug" again very soon

net1 to make 2 predictions:

1. A ***positive prediction*** for "saw"
2. A ***negative prediction*** for "Doug" (do not expect to see "Doug" in the near future)



net2 - Ignore Step

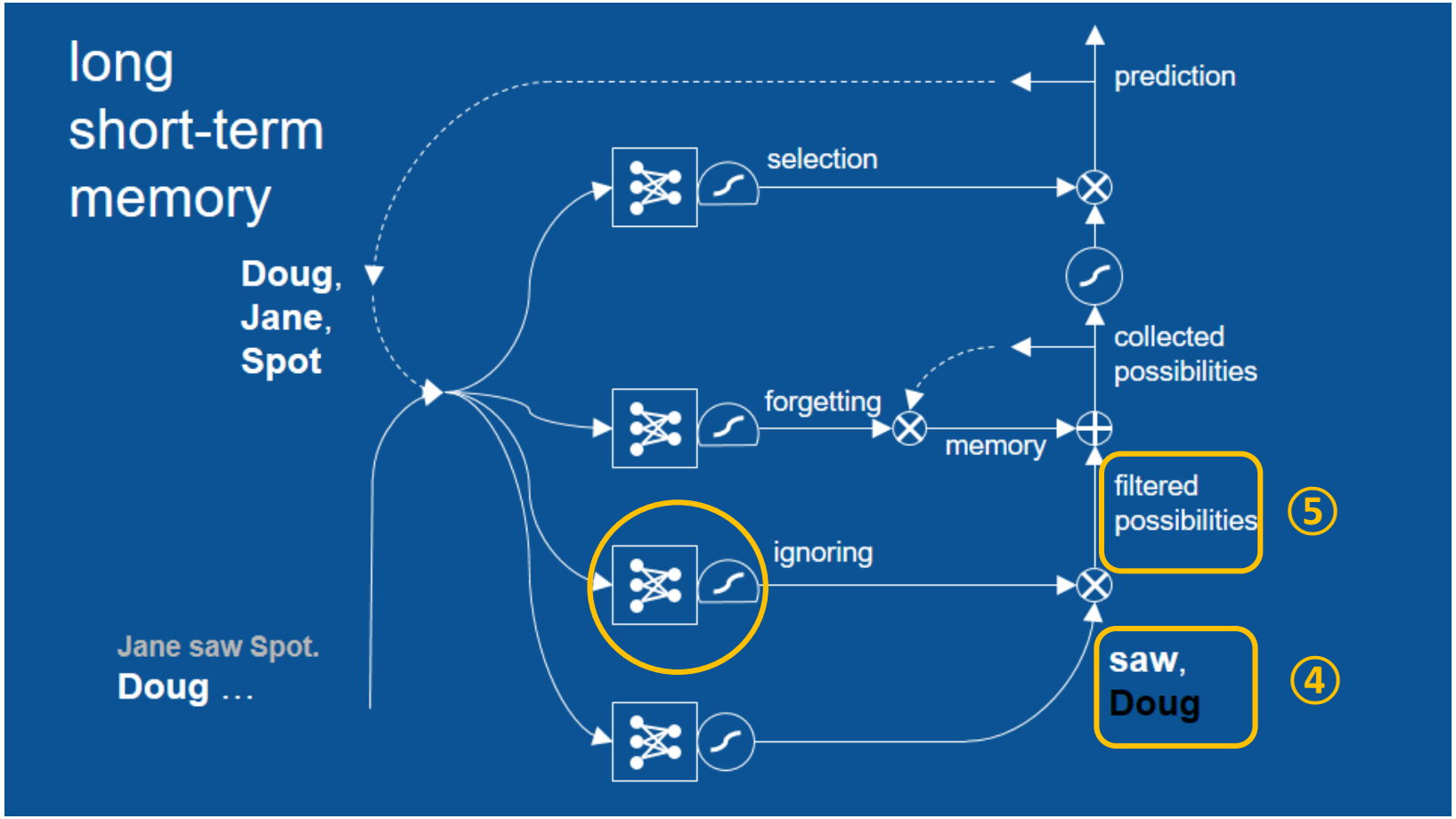
This example is simple,

we do not need to focus on ignoring anything

This prediction of

- "saw"
- "not Doug"

is passed forward



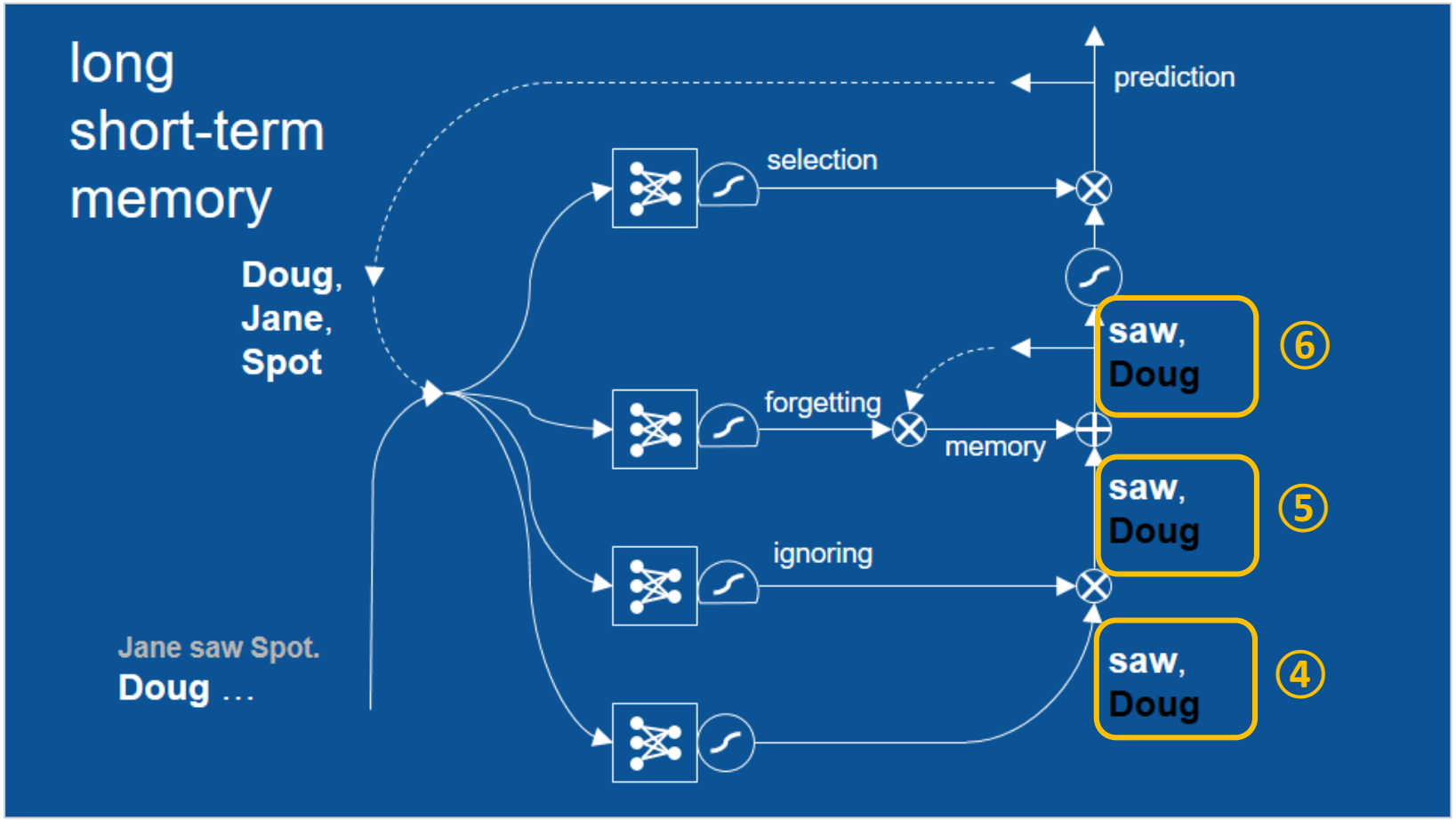
net3 - Forget Step

For the sake of simplicity, assume, there is no memory at the moment

Therefore,

- "saw"
- "not Doug"

going forward



net4 - Selection Step

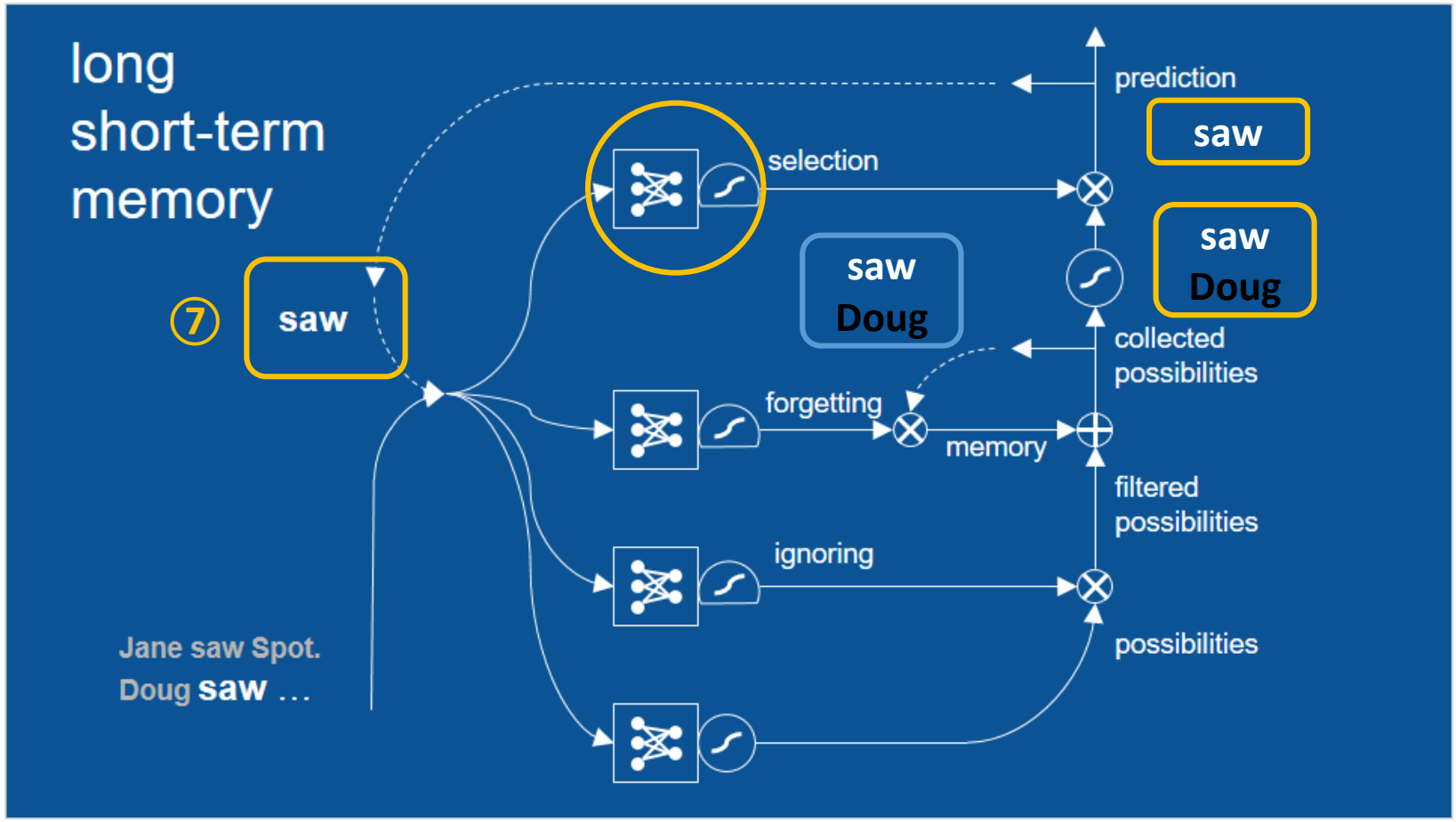
The selection mechanism (net4) has learned that when the most recent word was a name then the next is either

- "saw" or
- "."

net4 blocks any other words from coming out so

- "not Doug" gets blocked
- "saw" goes out

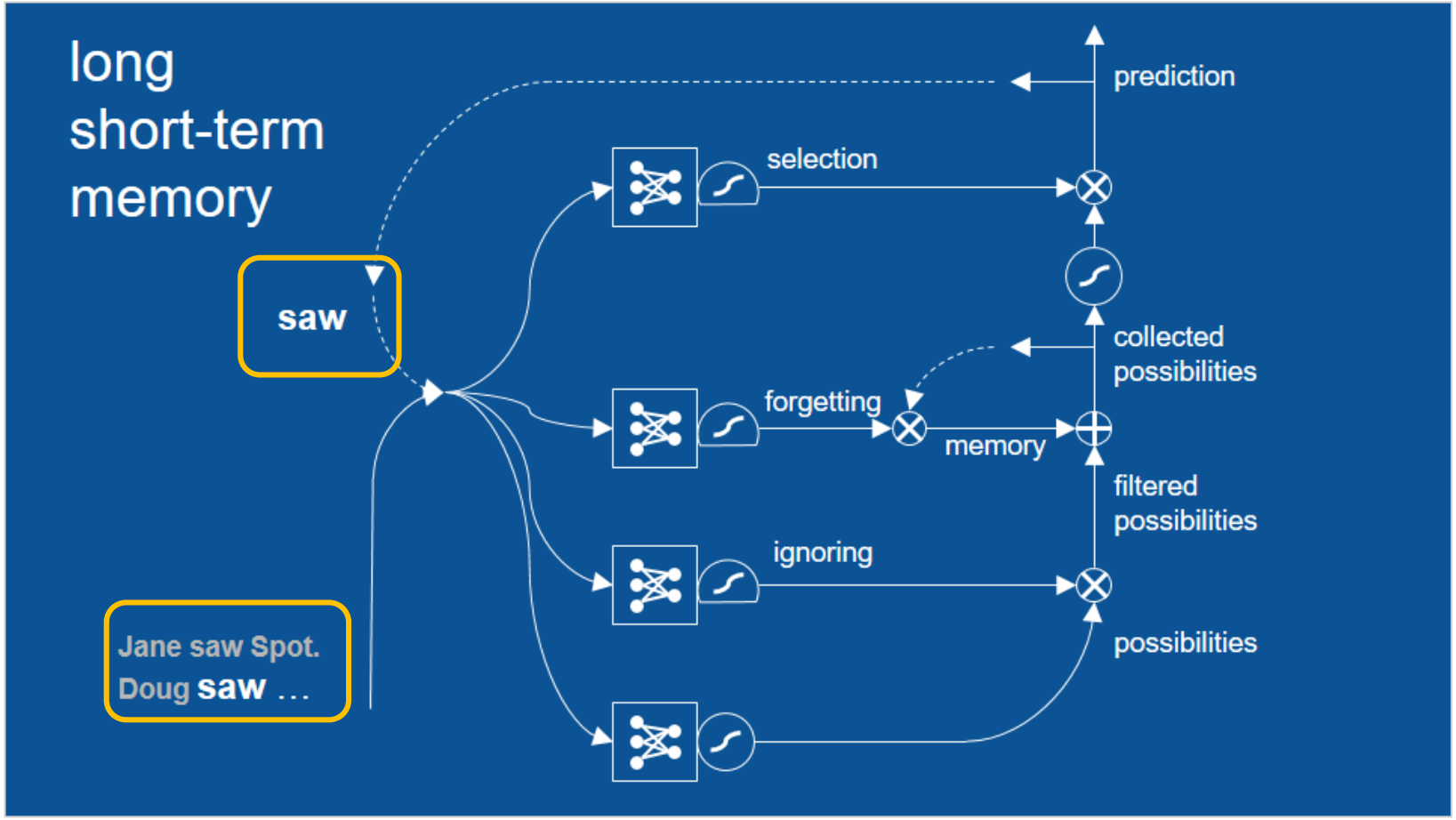
as the prediction for the next time step



Next Prediction Process

So we take a step forward in time now the word "saw" is our most recent word and our most recent prediction

They get passed forward to all of these neural networks (net 1, 2, 3, 4) and we get a new set of predictions



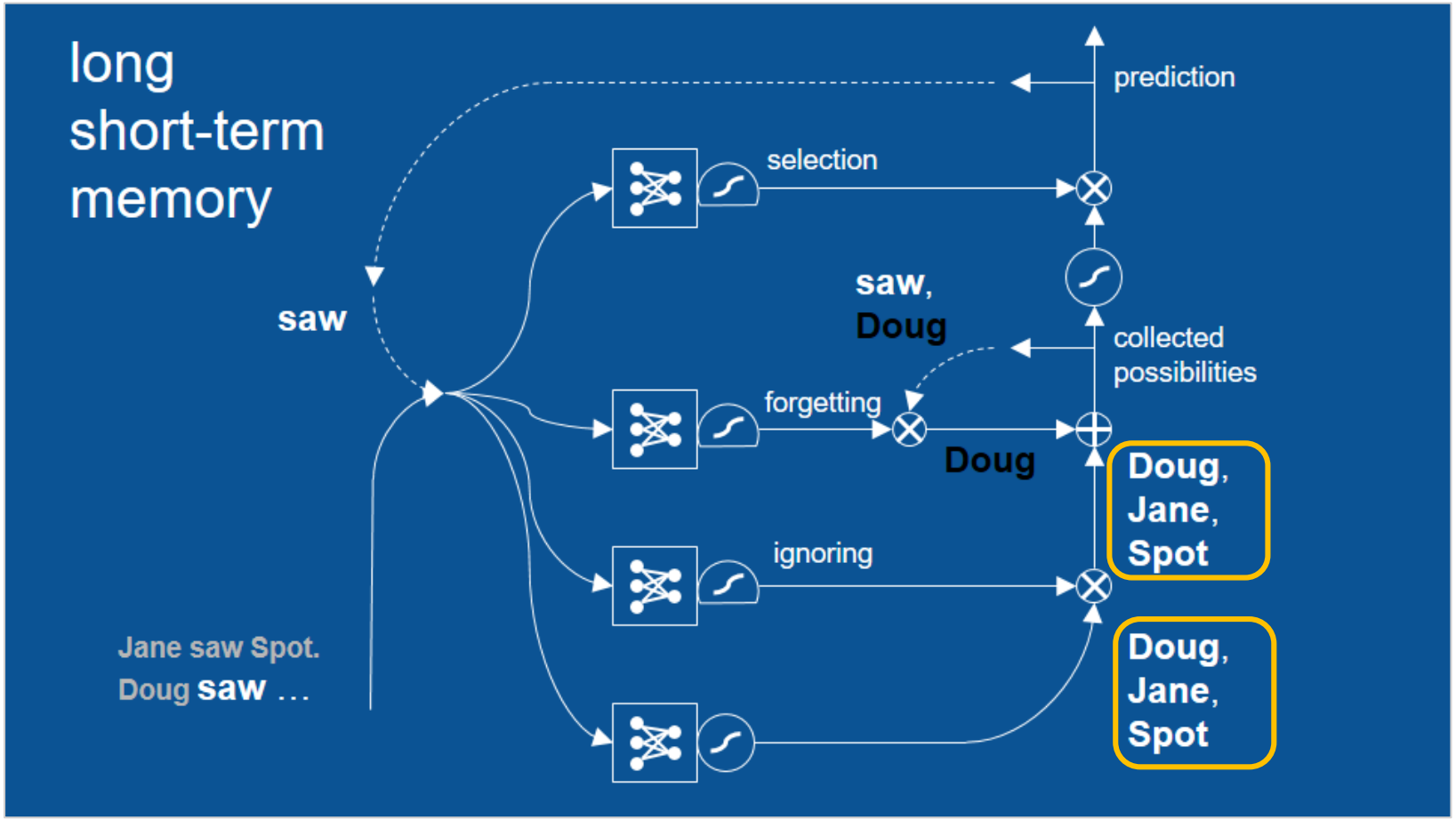
net1 - Prediction Step

Because the word "saw" just occurred we now predict that the words

- "Doug",
- "Jane", or
- "Spot"

might come next

we will pass over ignoring and attention in this example again & we will take those predictions forward



net3 - Forget Step

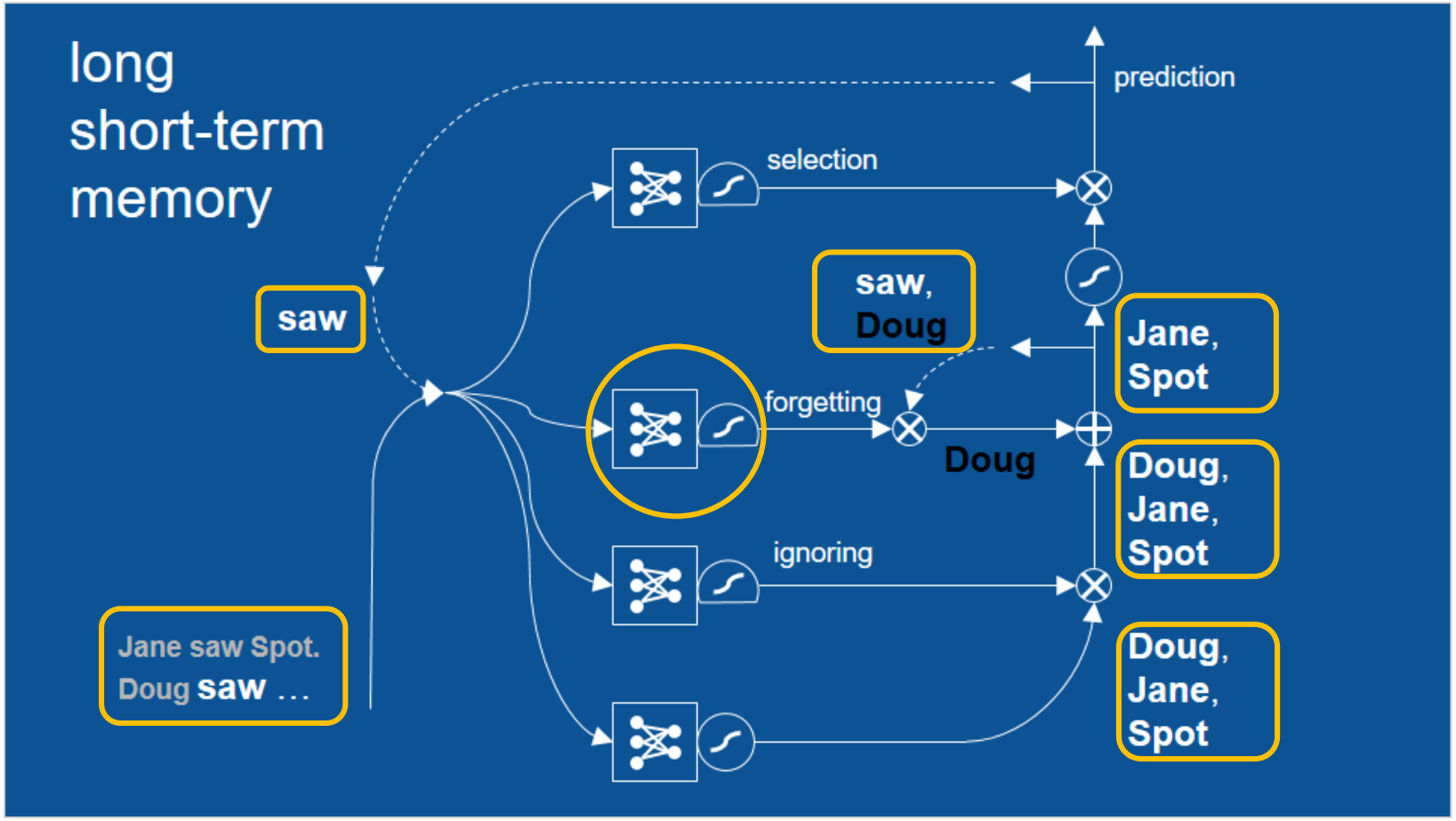
Now the other thing that we need to consider is our previous set of possibilities

Remember that we already had the words

- saw
- not Doug

that we maintain internally from previous step

They get passed to a forgetting gate



net3 - Forget Step

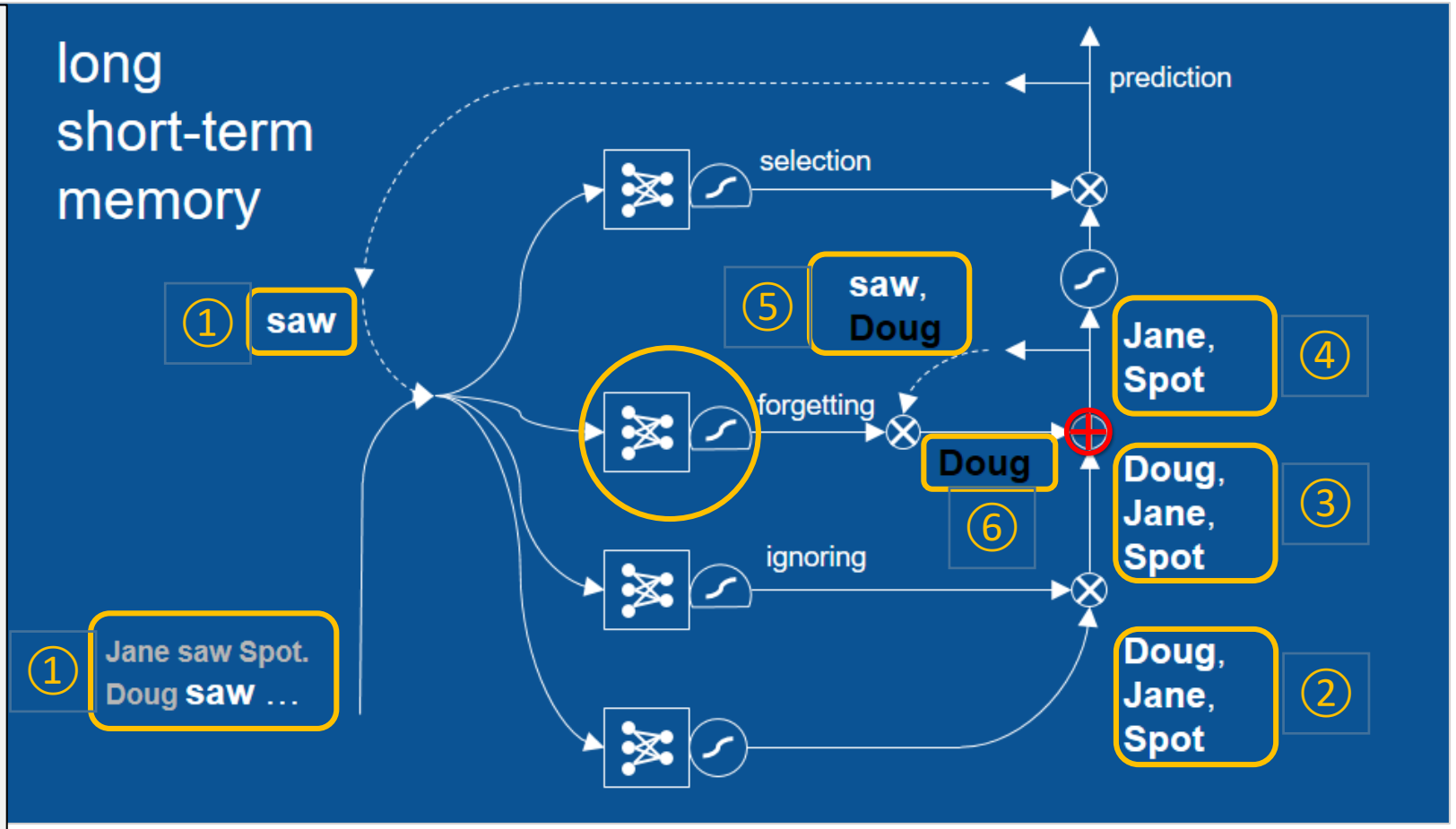
At the forgetting gate we know:
The last word that occurred was the word "saw" then the network can forget it but the network should keep any predictions about names
For net3:

- forgets "saw"
- keeps "not Doug"

& now at ⊕ we have:

- a positive vote for "Doug"
- a positive vote for "not Doug" (or a negative vote for "Doug")

they cancel each other



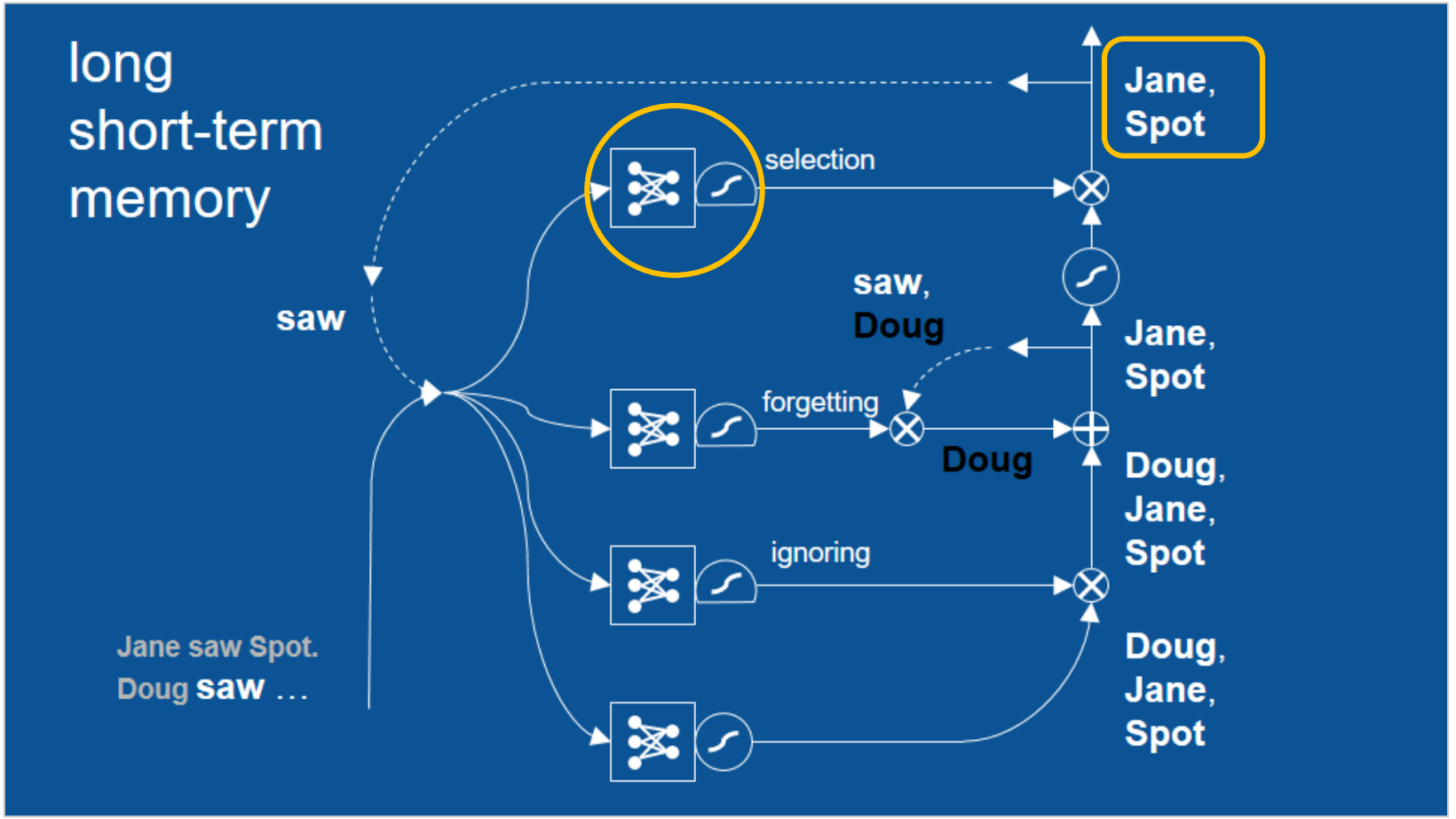
➔ After this point the network has only "Jane" & "spots" Those get passed forward

net4 - Selection Step

The selection gate knows that

- the word "saw" just occurred and
- a name should happen next

- so it passes through these predictions for names and for the next time step then we get predictions of
- "Jane"
- "spot"



Some mistakes may not happen

This network can avoid:

- Doug saw Doug.
- Jane saw Spot saw …
- Spot. Doug. Jane.

That is because LSTM can look back two, three, many time steps and use that information to make good predictions about what's going to happen next.

Note: vanilla recurrent neural networks they can actually look back several time steps as well but not very many.

LSTM Applications

- Translation of text from one language to another language

  Even though translation is not a word to word process, it's a phrase to phrase or even in some cases a sentence to sentence process, LSTMS are able to represent those grammar structures that are specific to each language and what it looks like is that they find the higher-level idea and translate it from one mode of expression to another, just using the bits and pieces that we just walked through.

LSTM Applications

- Translation of speech to text

  Speech is just some signals that vary in time. It takes them and uses that then to predict what text -what word- is being spoken and it can use the history -the recent history of words- to make a better guess for what's going to come next.

LSTM Applications

- LSTMS are a great fit for any information that is embedded in time – like audio, video
- An agent taking in information from a set of sensors and then based on that information, making a decision and carrying out an action.
- It's inherently sequential and actions taken now can influence what is sensed and what should be done many times steps down the line.

Some interesting applications

Sequential patterns

Text

Speech

Audio

Video

Physical processes

Anything embedded in time (almost everything)