RNN & LSTM

Presented by

K Sumanth Reddy

Deep Learning Engineer-Blincam

Visiting Faculty-Sathyabhama Institute Of Technology

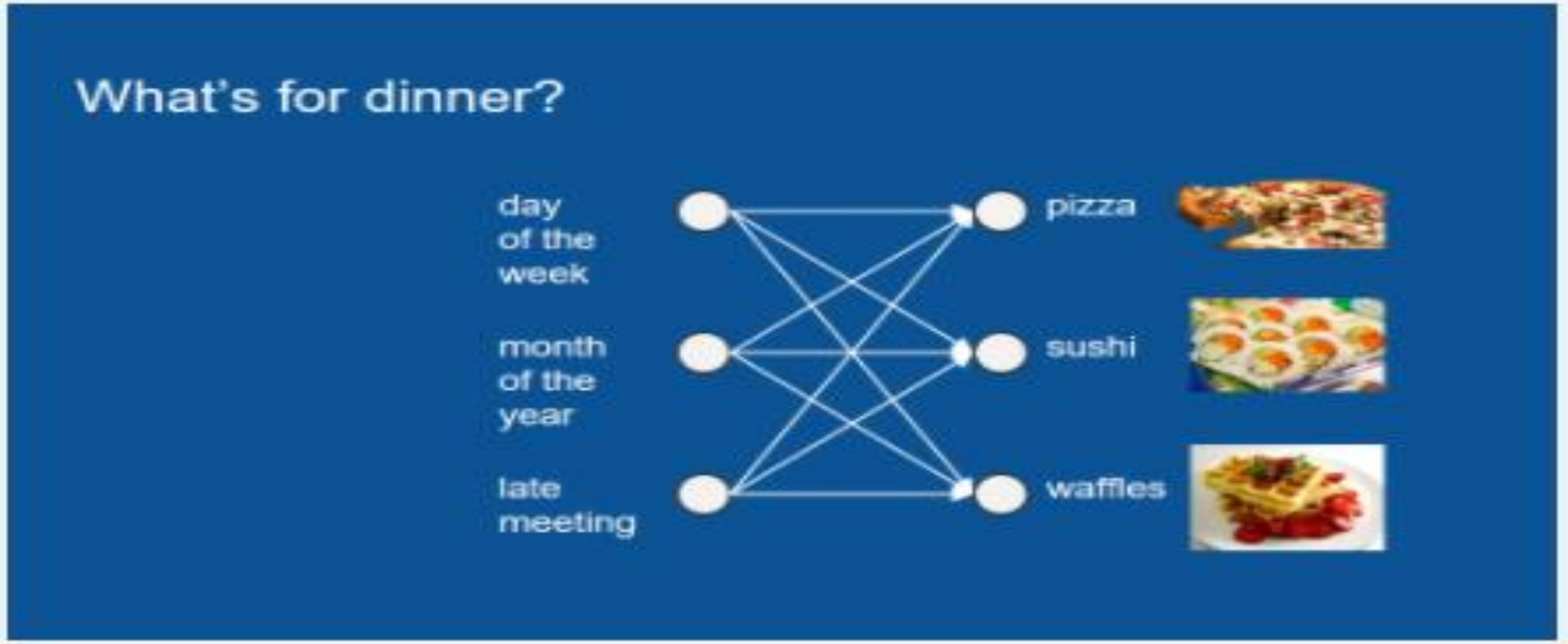
Recurrent Neural Networks

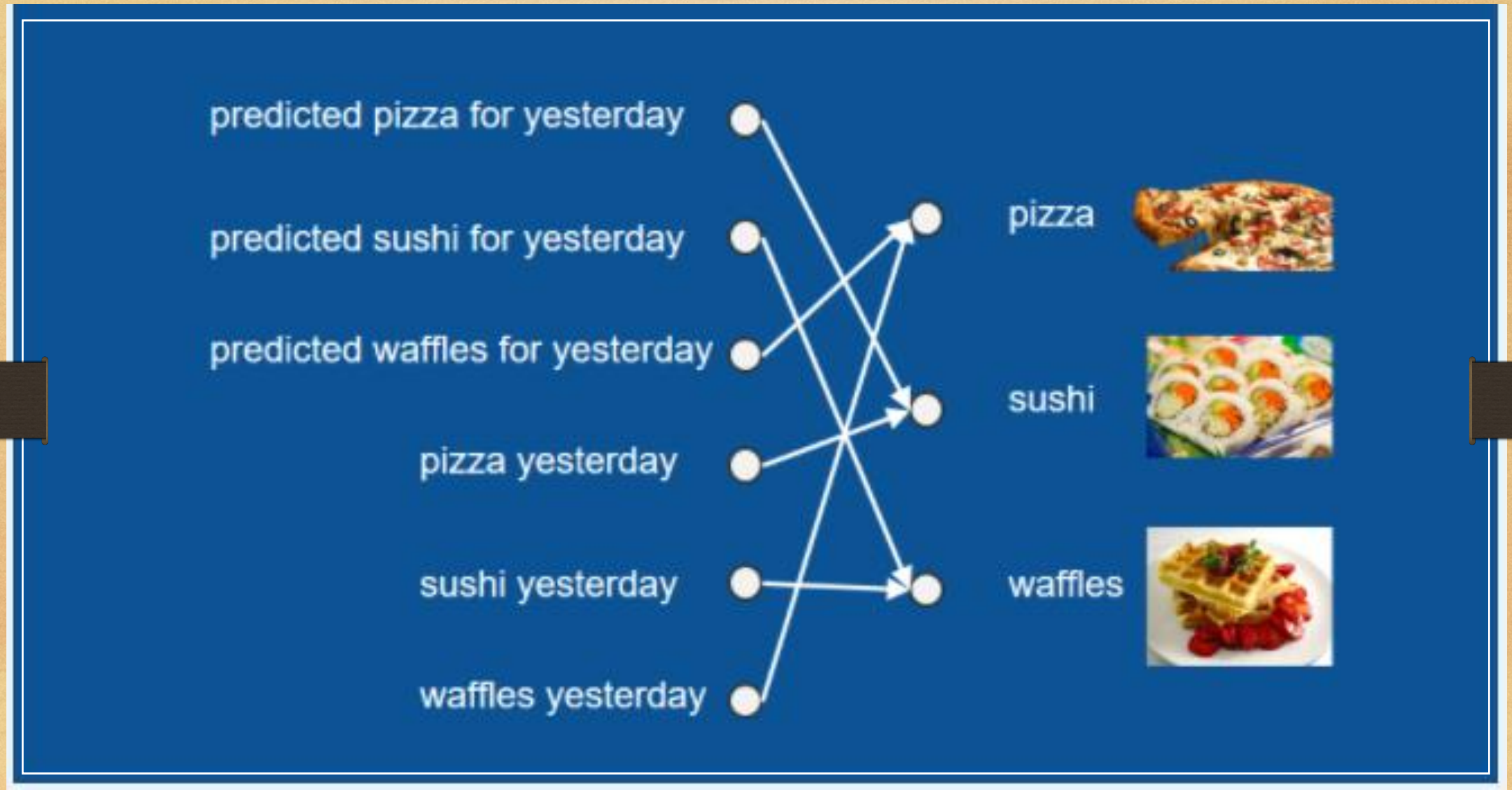
- Humans don't start their thinking from scratch every second. We understand things based on past experience.
- Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to understand what a person spoke, you can only get the meaning based on his previous words.
- Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

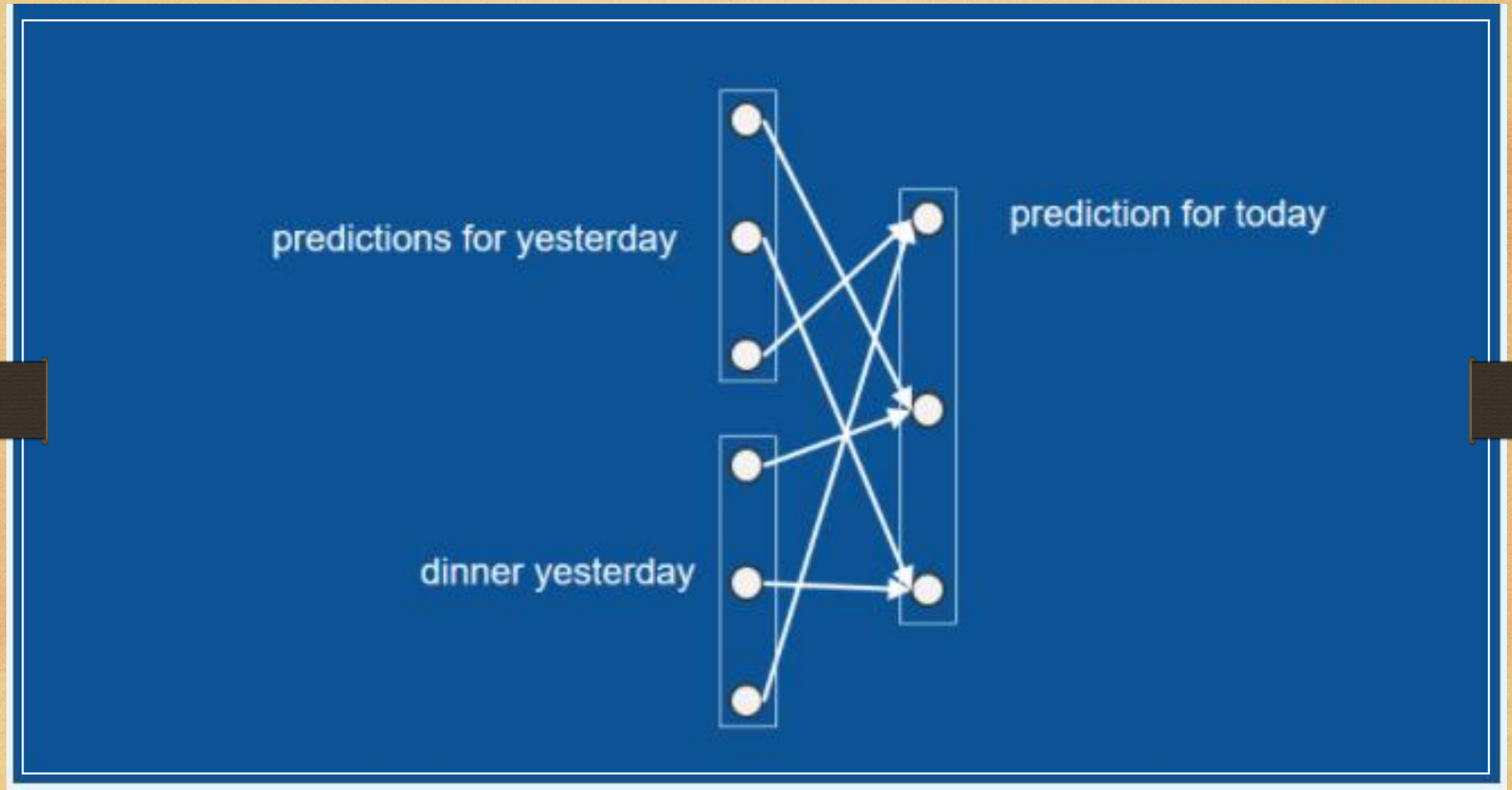
Recurrent Neural Networks

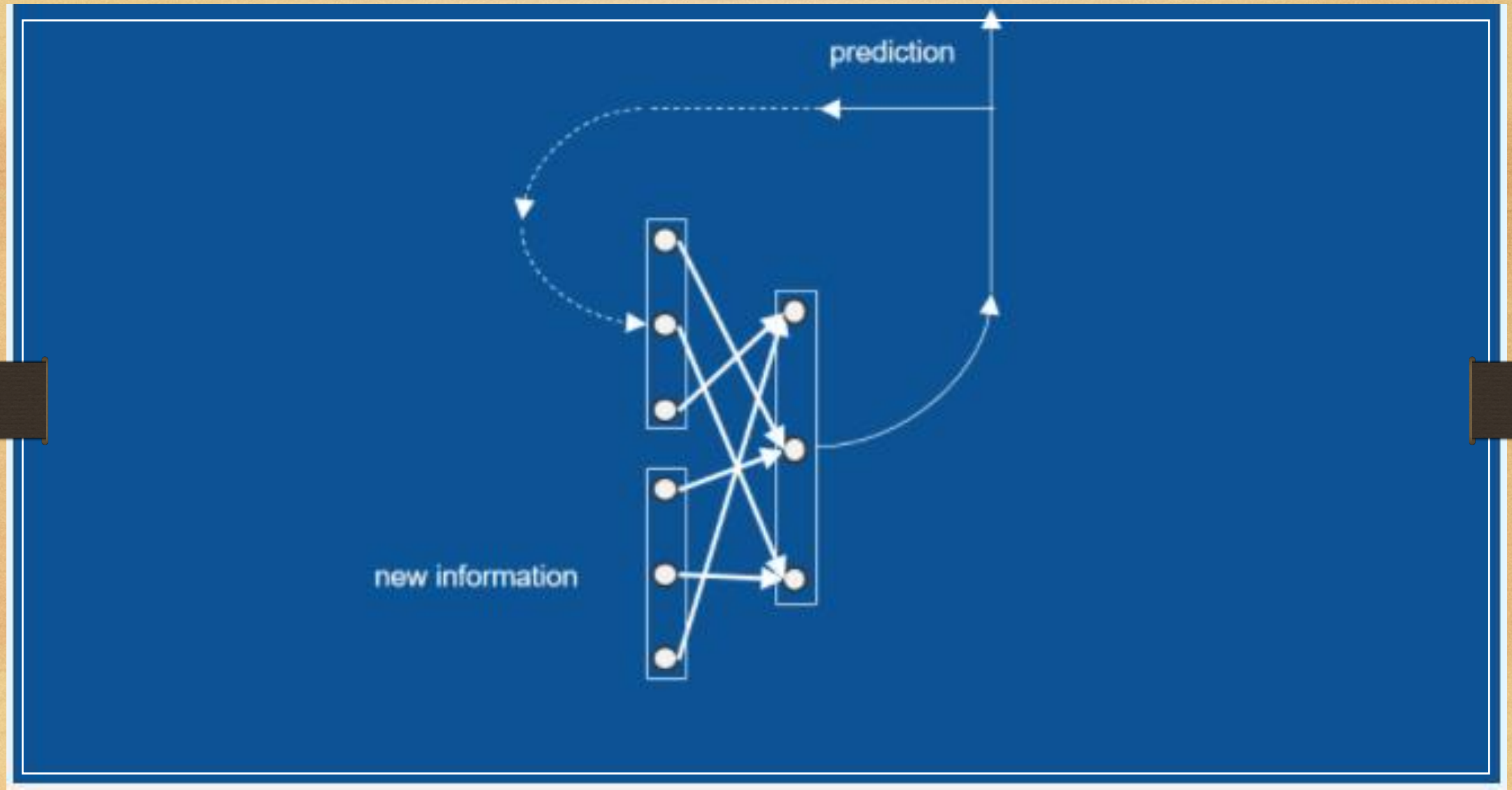
- Recurrent nets are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, or numerical times series data from sensors, stock markets and government agencies.
- They are arguably the most powerful and useful type of neural network, applicable even to images, which can be decomposed into a series of patches and treated as a sequence.
- Since recurrent networks possess a certain type of memory, and memory is also part of the human brain, we'll make repeated analogies to memory in the brain

Recurrent Neural Networks- Use Case Example




predicted pizza for yesterday
predicted sushi for yesterday
predicted waffles for yesterday
pizza yesterday
sushi yesterday
waffles yesterday
pizza
sushi
waffles

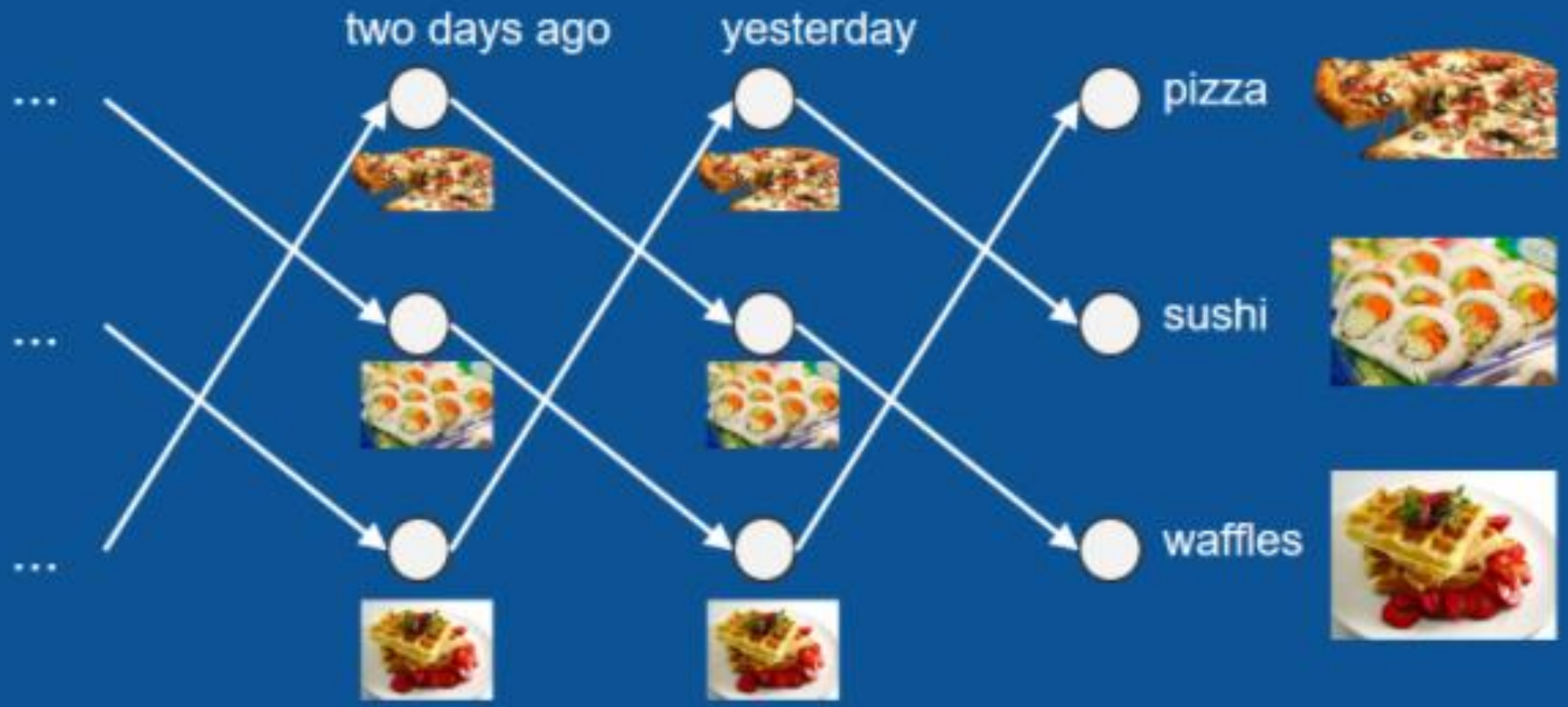

predictions for yesterday
dinner yesterday
prediction for today


prediction
new information

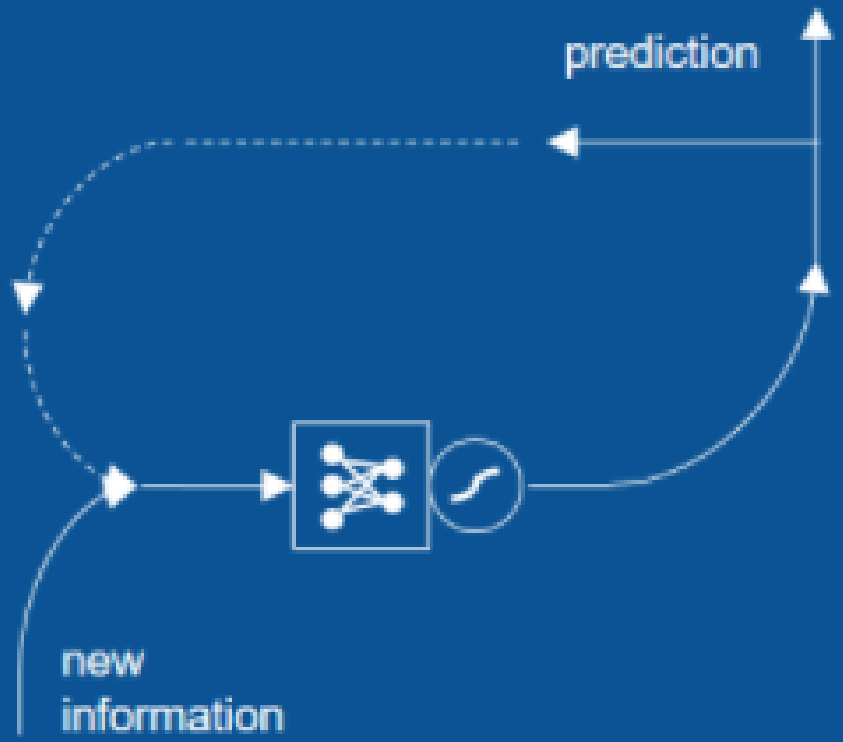
- Now we will see how if we were lacking some information, let's say we were out of town for two weeks, we can still make a good guess about what's going to be for dinner tonight, we just ignore the new information part and we can unwrap or unwind this model in time until we do have some information to base it on and then just play it forward and when it's unwrapped, it looks like this (next picture), and we can go back as far as we need to and see what was for dinner and then just trace it forward and play out our menu over the last two weeks until we find out what's for dinner tonight.

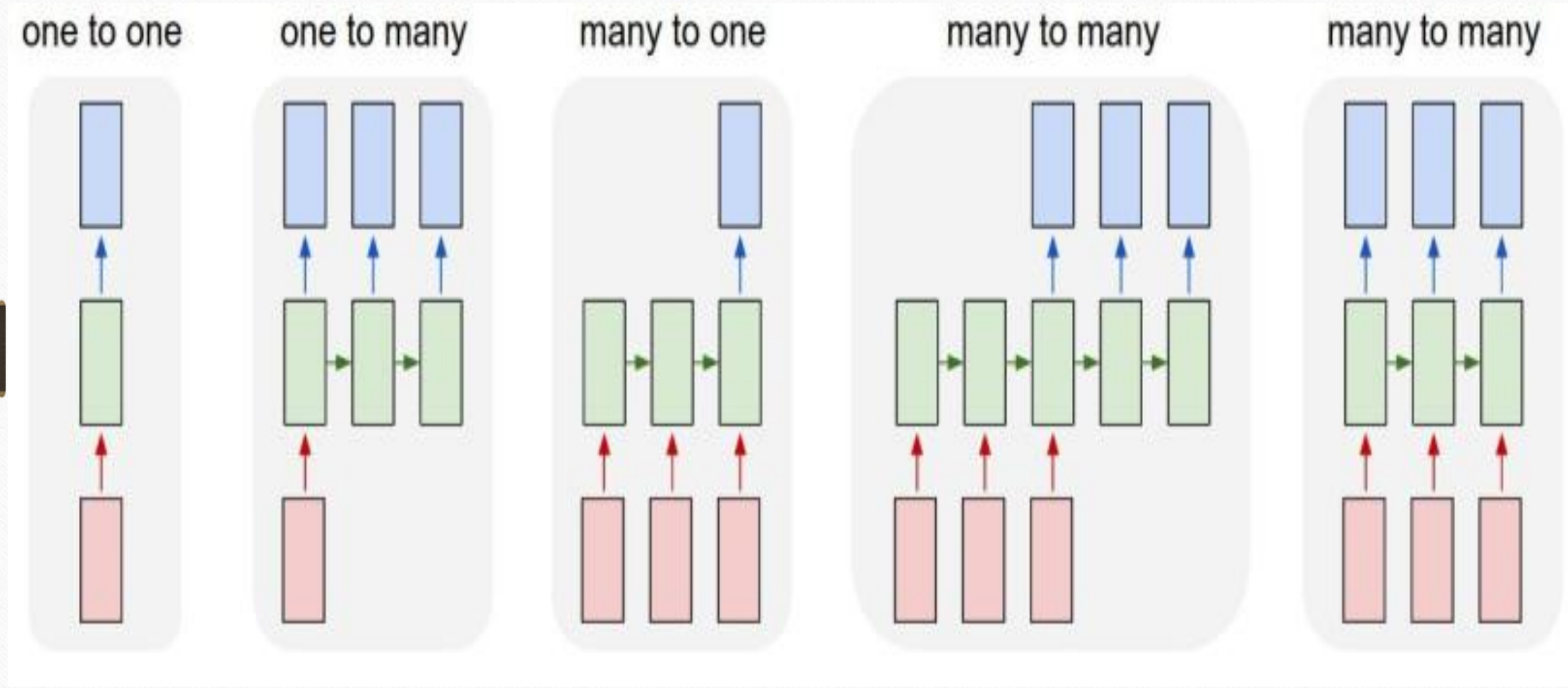

prediction

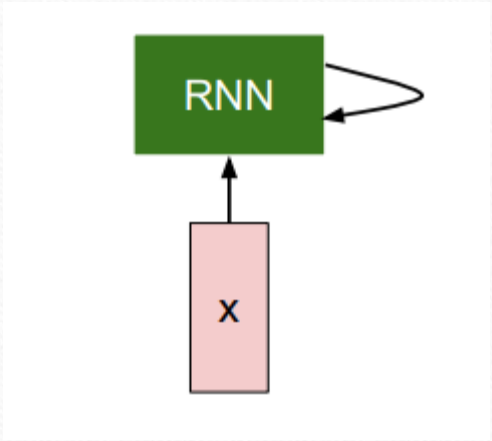
Unrolled predictions

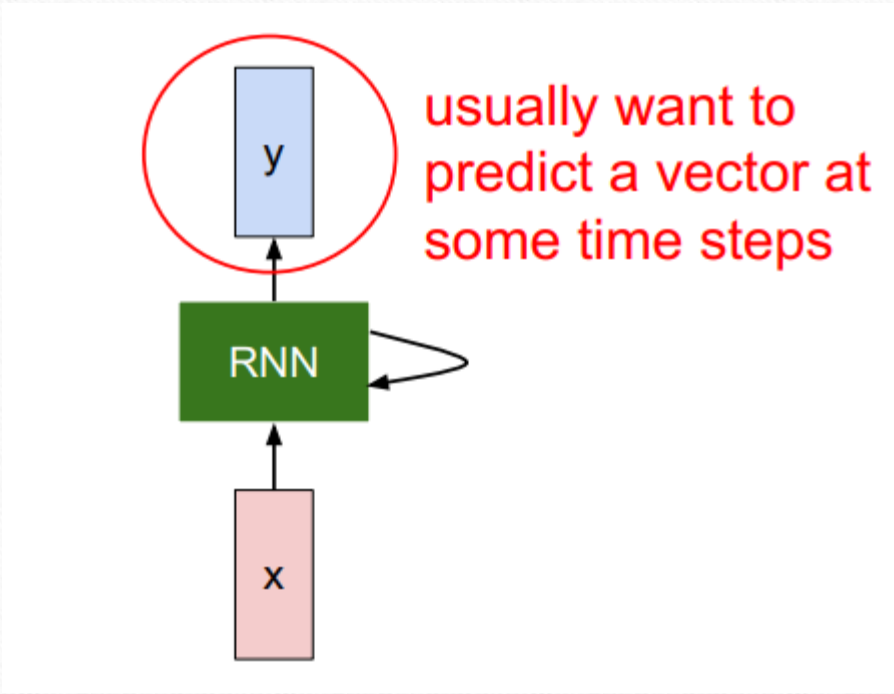


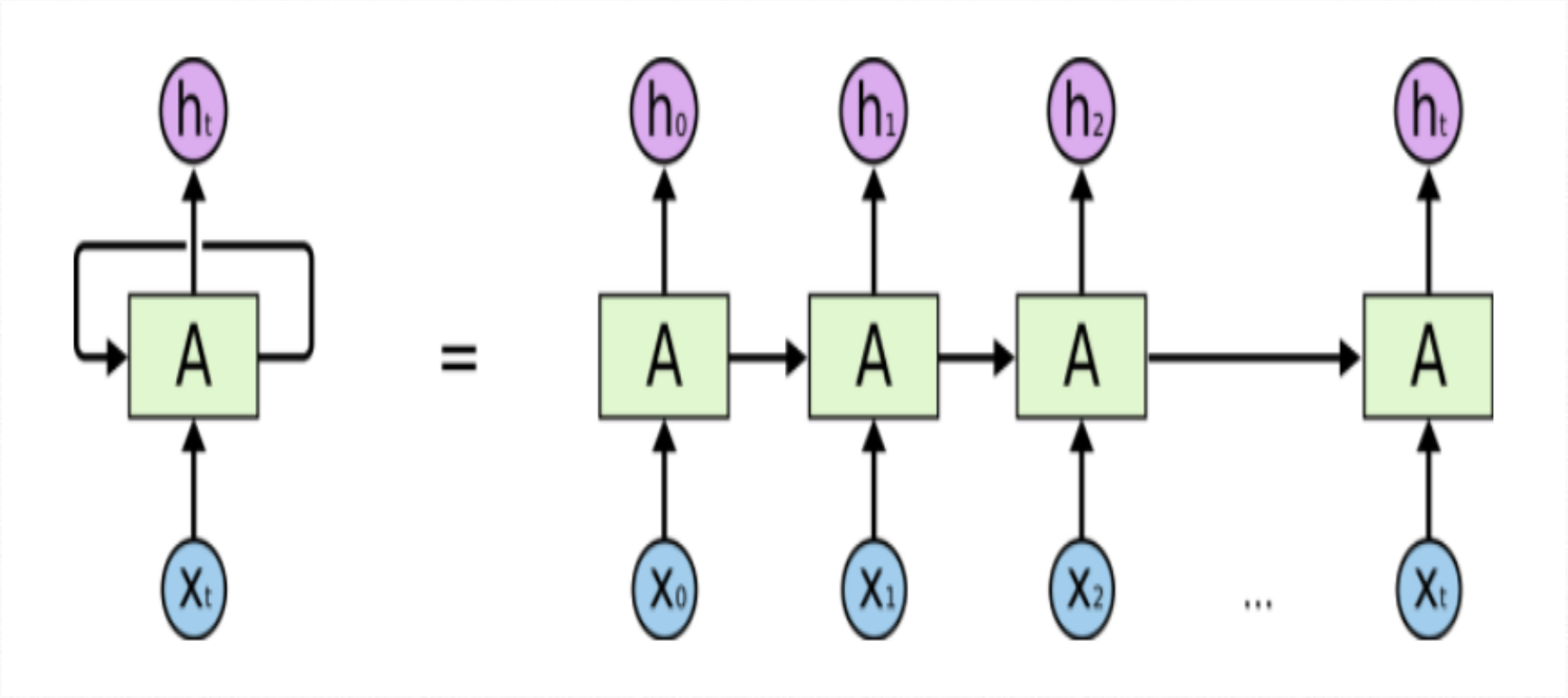
recurrent neural network




one to one
one to many
many to one
many to many
many to many


RNN
x


y
usually want to predict a vector at some time steps
RNN
x


h_t
A
X_t
=
h_0
h_1
h_2
h_t
A
A
A
A
X_0
X_1
X_2
...
X_t

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

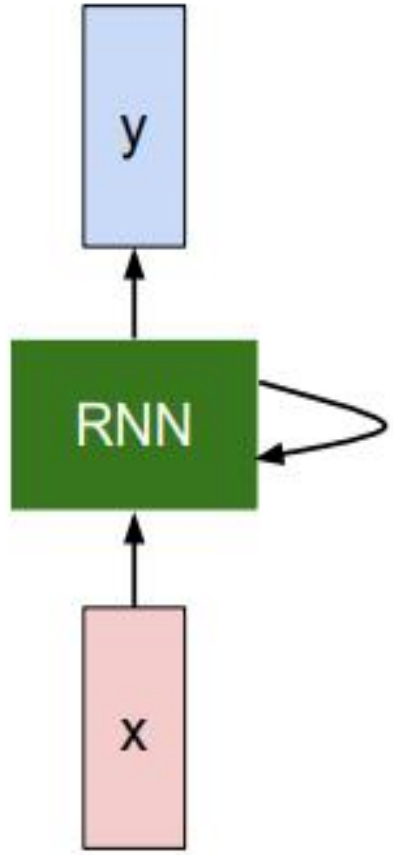
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

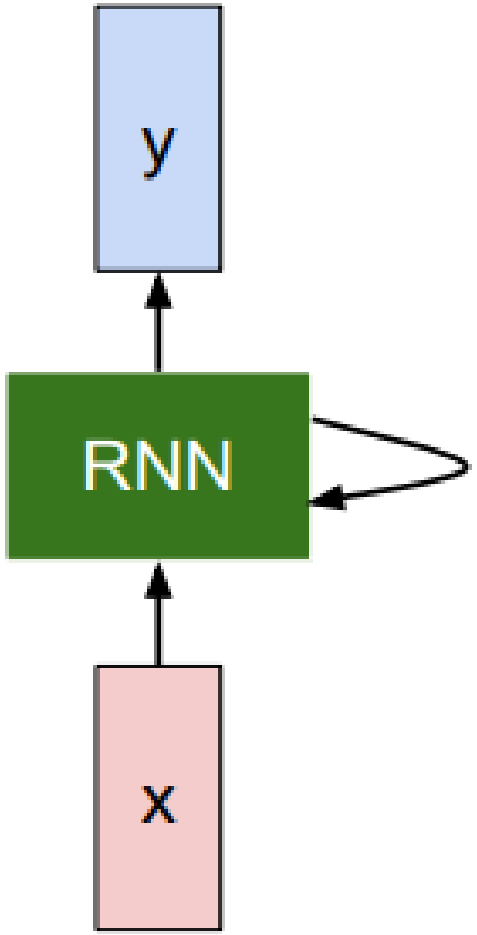
old state

input vector at some time step



Notice: the same function and the same set of parameters are used at every time step

The state consists of a single *"hidden"* vector **h**:

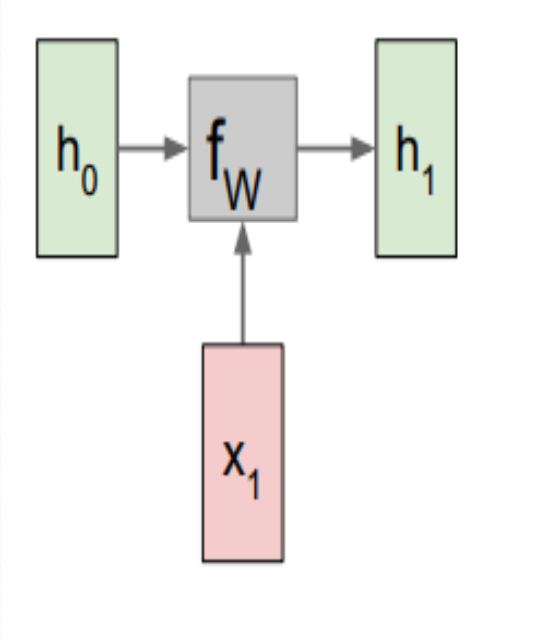


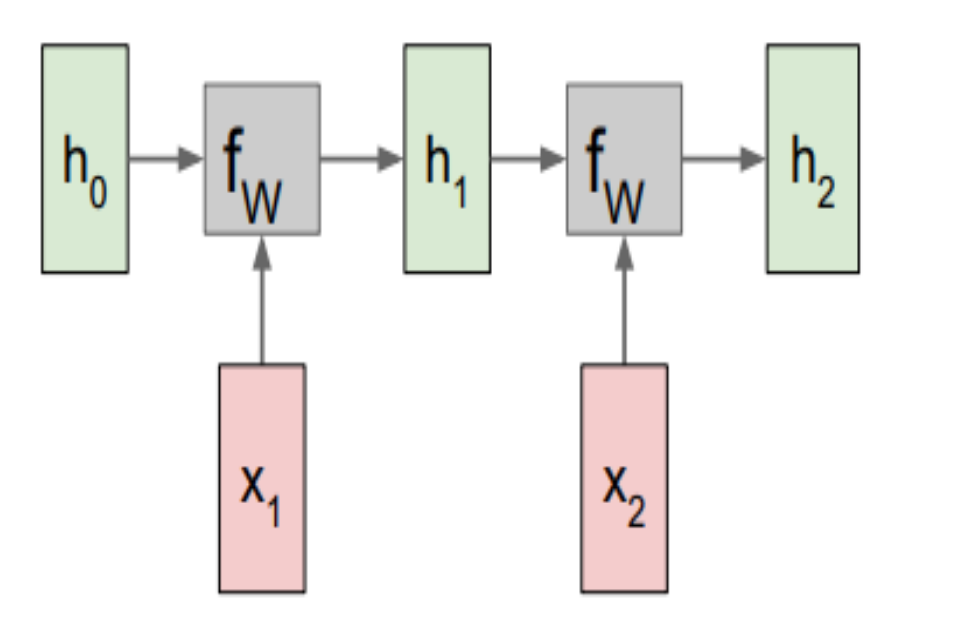
$$h_t = f_W(h_{t-1}, x_t)$$

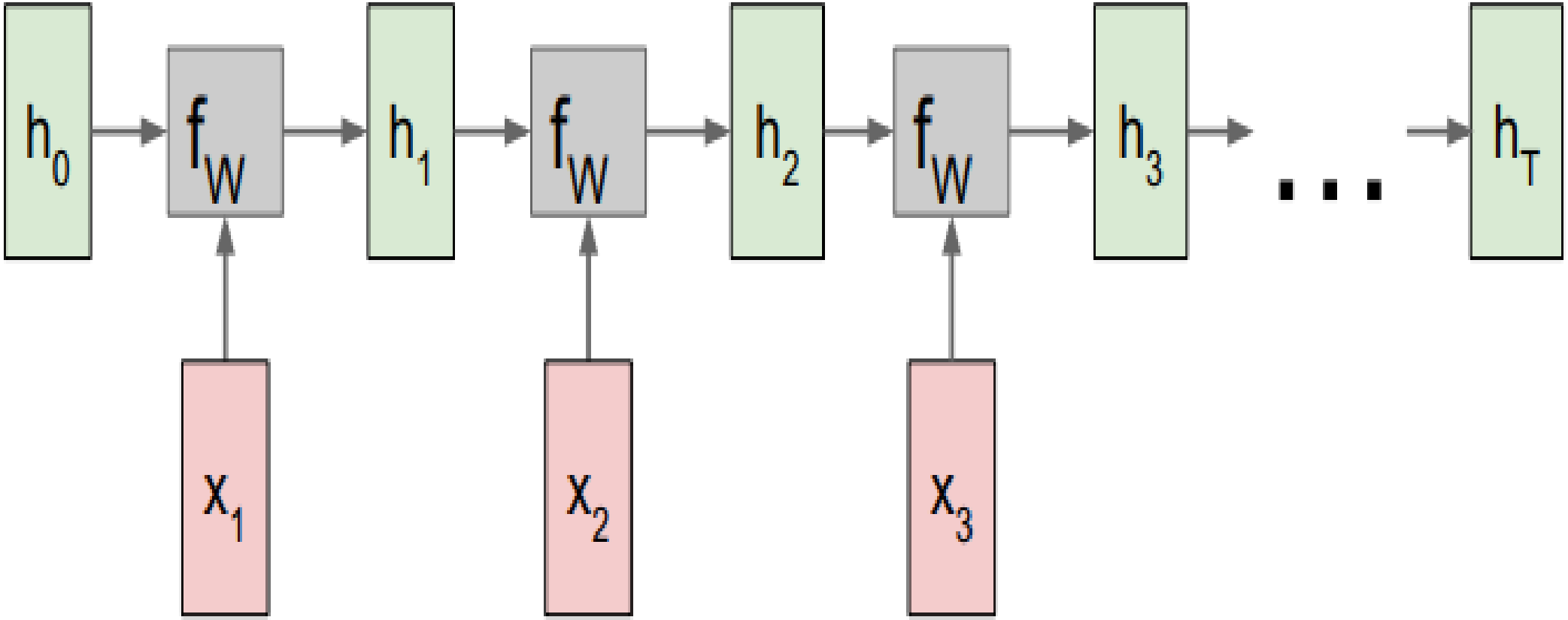
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t$$

$$y_t = W_{hy}h_t$$

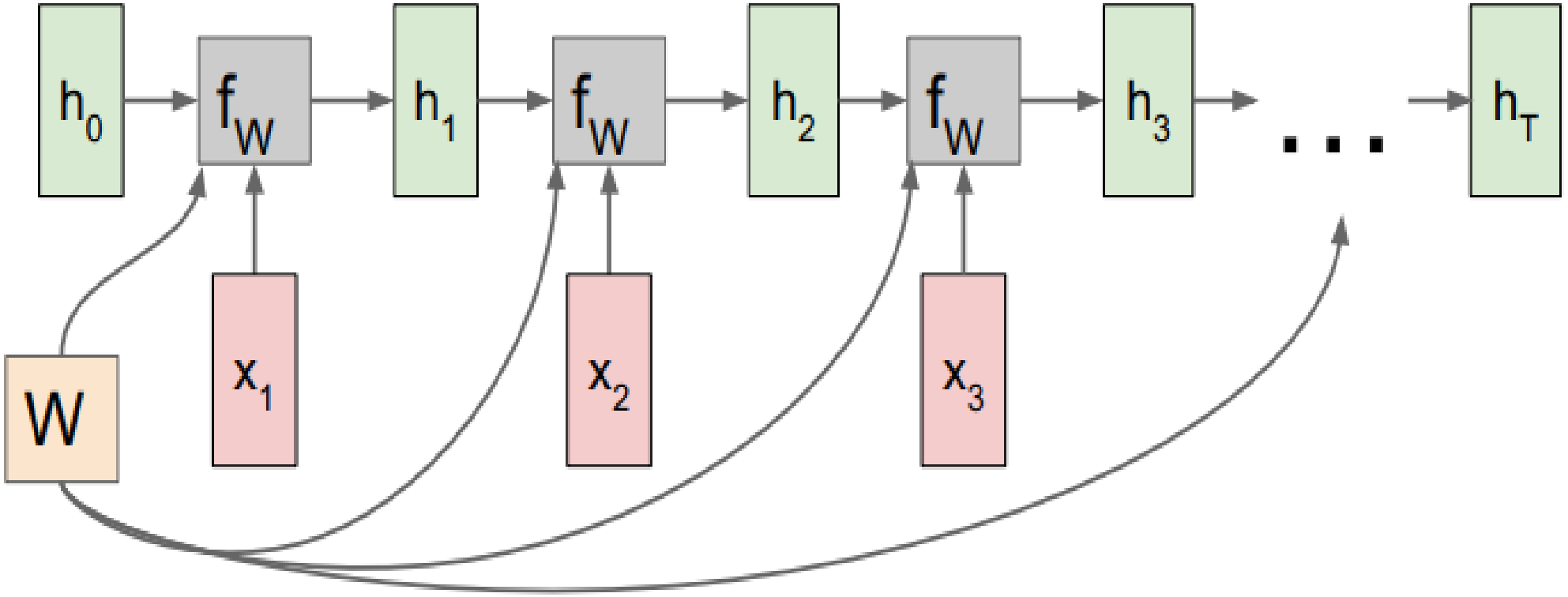
RNN Computation Graph





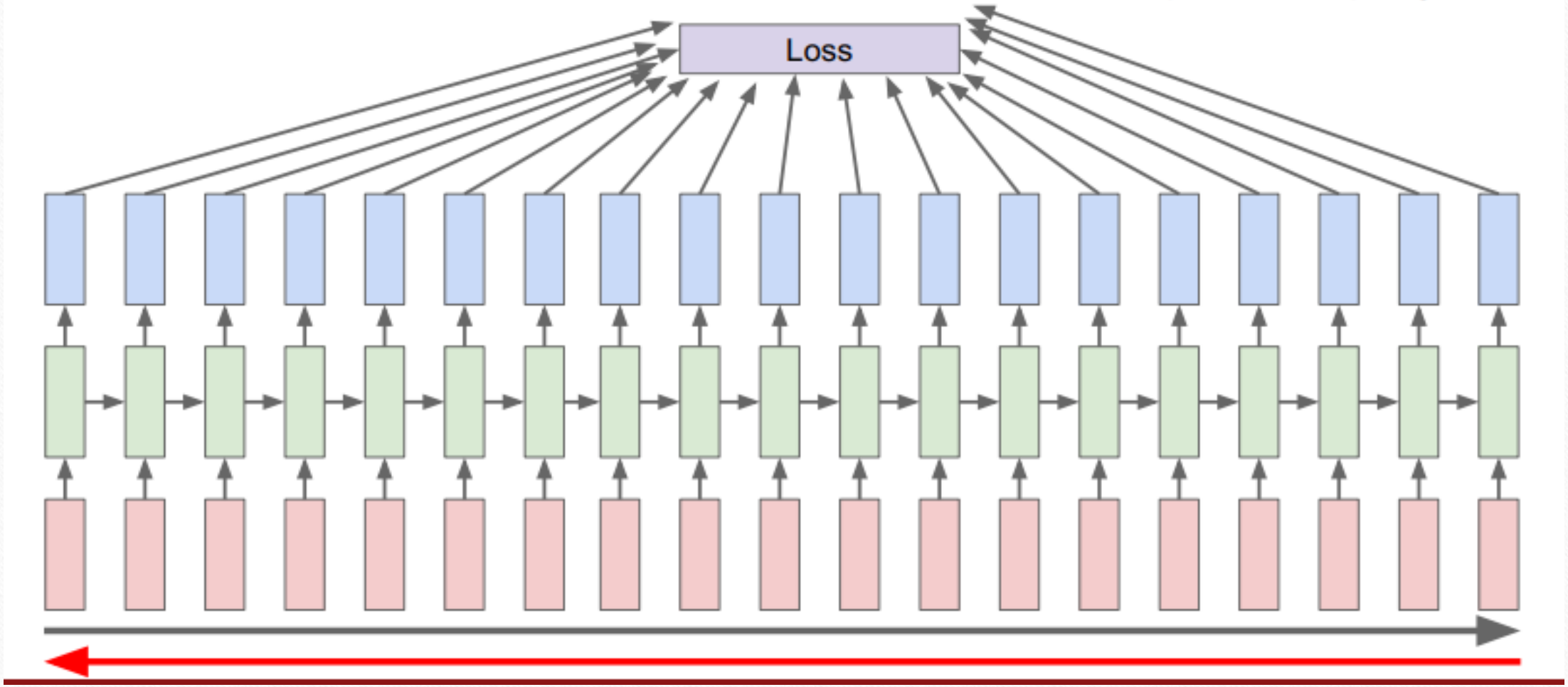

h_0
f_W
h_1
f_W
h_2
f_W
h_3
. . .
h_T
x_1
x_2
x_3

Re-use the same weight matrix at every time-step

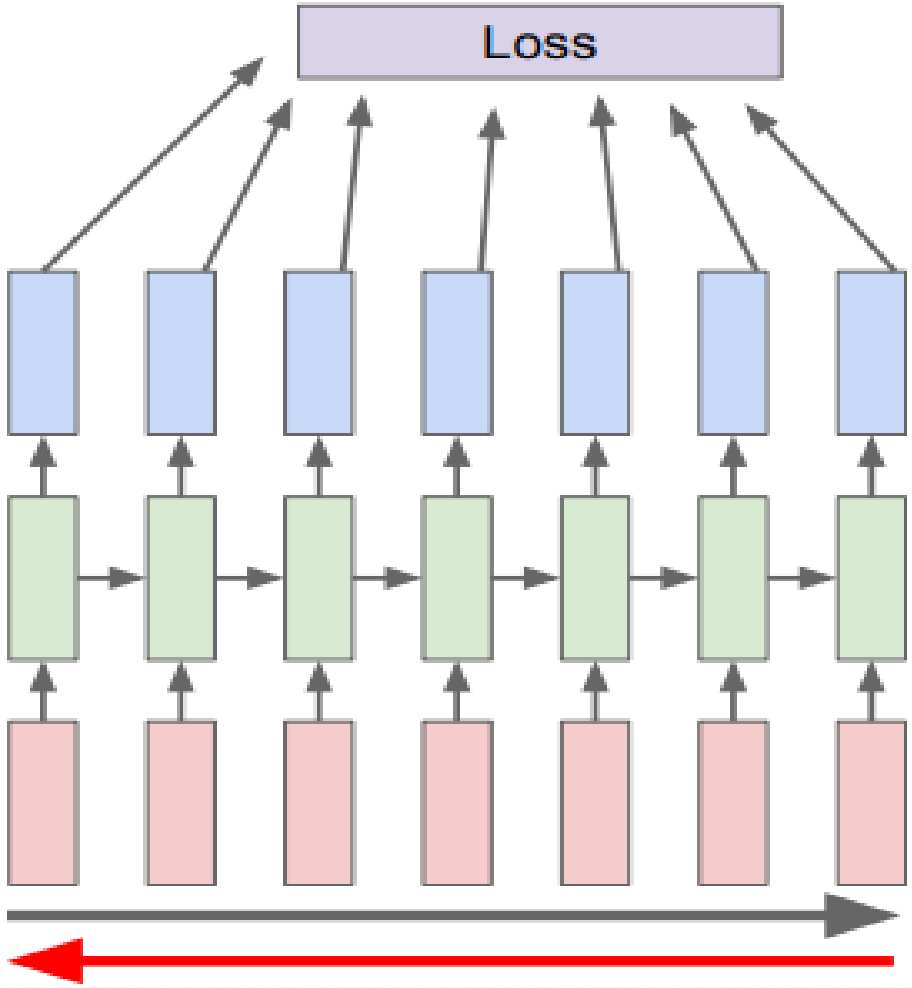


Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

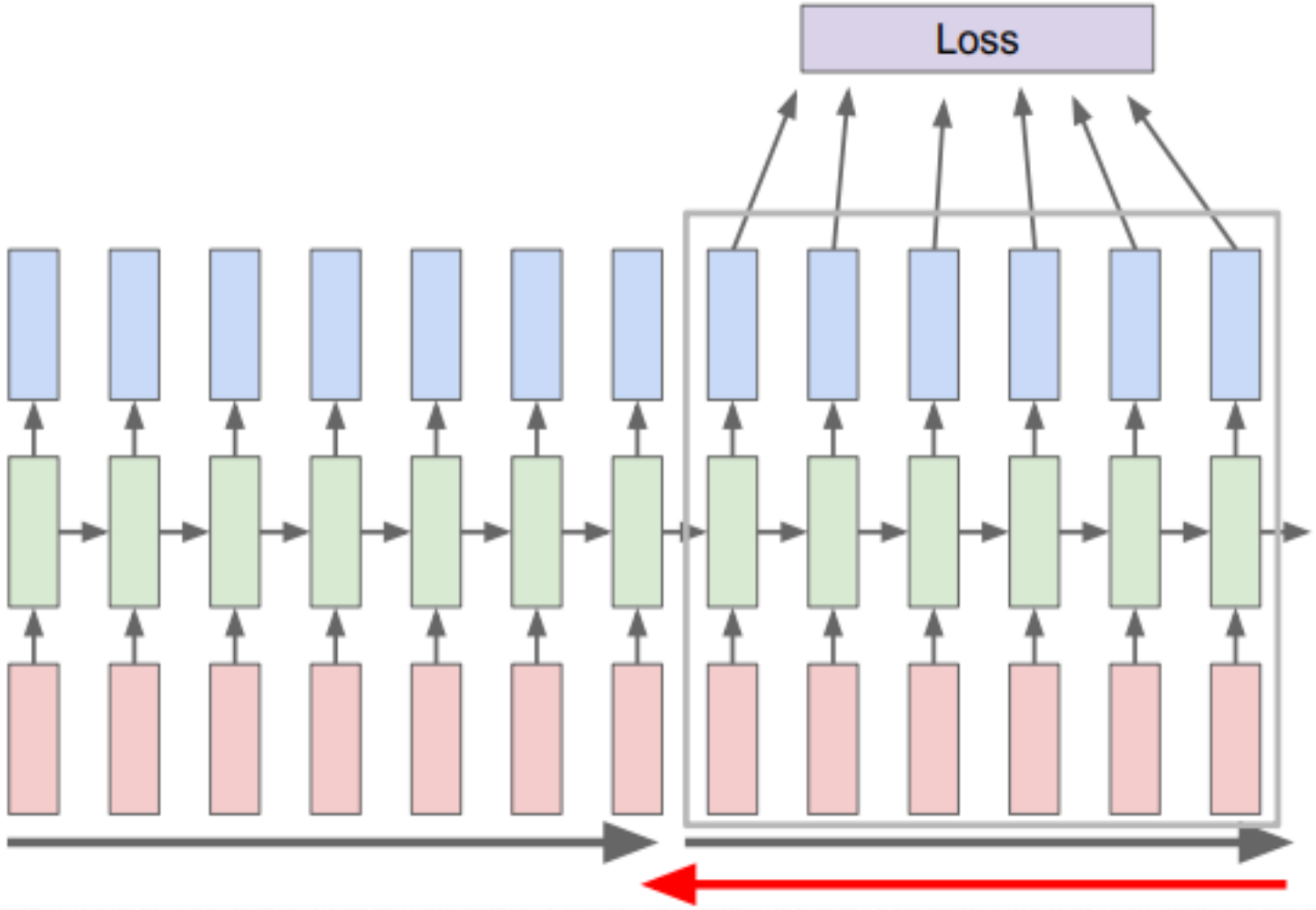


Truncated Backpropagation through time



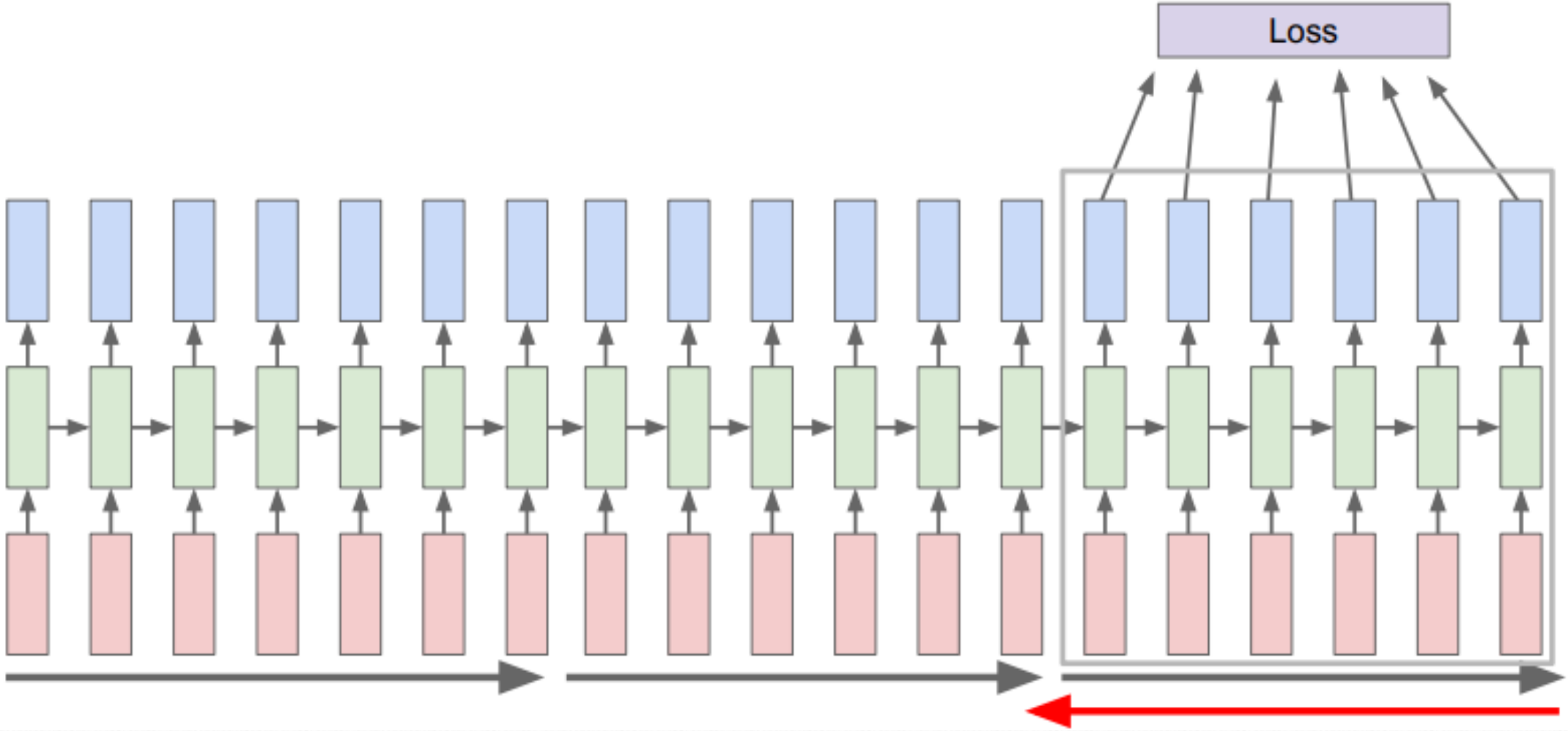
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time



Drawbacks Of RNN

Sentence: the clouds are in the *sky*

- If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

Sentence: "I grew up in France… I speak fluent *French*."

- Consider trying to predict the last word in the text "I grew up in France… I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

Drawbacks Of RNN

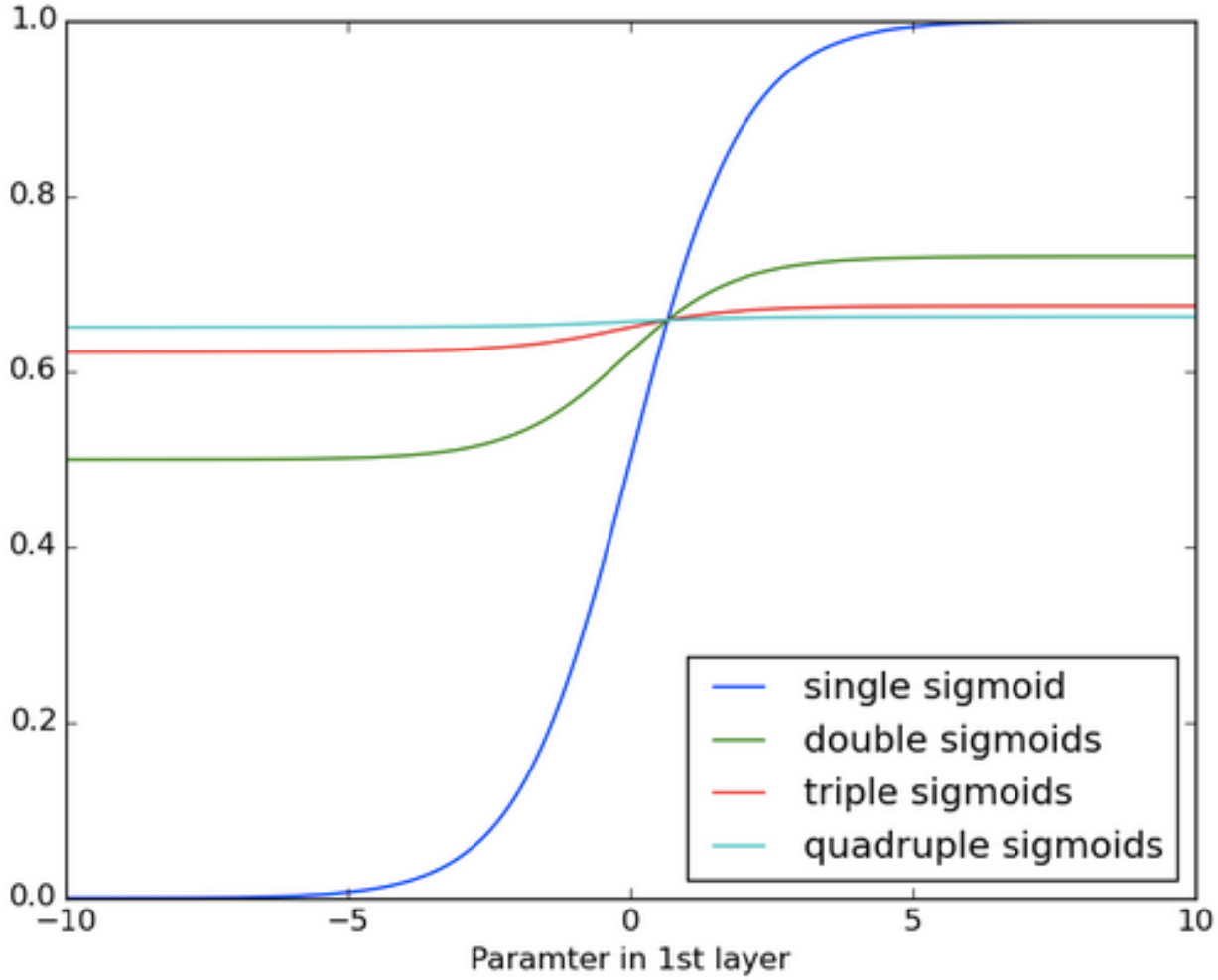
- In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.
- **In One Line:** Because each of our predictions only looks back one time step it has very short term memory, then it doesn't use the information from further back and it's subject to mistakes.

Vanishing (and Exploding) Gradients

- Like most neural networks, recurrent nets are old. By the early 1990s, the *vanishing gradient problem* emerged as a major obstacle to recurrent net performance.
- Recurrent nets seeking to establish connections between a final output and events many time steps before were hobbled, because it is very difficult to know how much importance to accord to remote inputs. (Like great-great-*-grandparents, they multiply quickly in number and their legacy is often obscure.) This is partially because the information flowing through neural nets passes through many stages of multiplication.
- Everyone who has studied compound interest knows that any quantity multiplied frequently by an amount slightly greater than one can become immeasurably large (indeed, that simple mathematical truth underpins network effects and inevitable social inequalities). But its inverse, multiplying by a quantity less than one, is also true. Gamblers go bankrupt fast when they win just 97 cents on every dollar they put in the slots.
- Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing or exploding.
- Exploding gradients treat every weight as though it were the proverbial butterfly whose flapping wings cause a distant hurricane. Those weights' gradients become saturated on the high end; i.e. they are presumed to be too powerful. But exploding gradients can be solved relatively easily, because they can be truncated or squashed. Vanishing gradients can become too small for computers to work with or for networks to learn – a harder problem to solve.

Vanishing (and Exploding) Gradients

- Please see the effects of applying a sigmoid function over and over again. The data is flattened until, for large stretches, it has no detectable slope. This is analogous to a gradient vanishing as it passes through many layers.



Long Short-Term Memory Units (LSTMs)

Long Short-Term Memory Units (LSTMs)

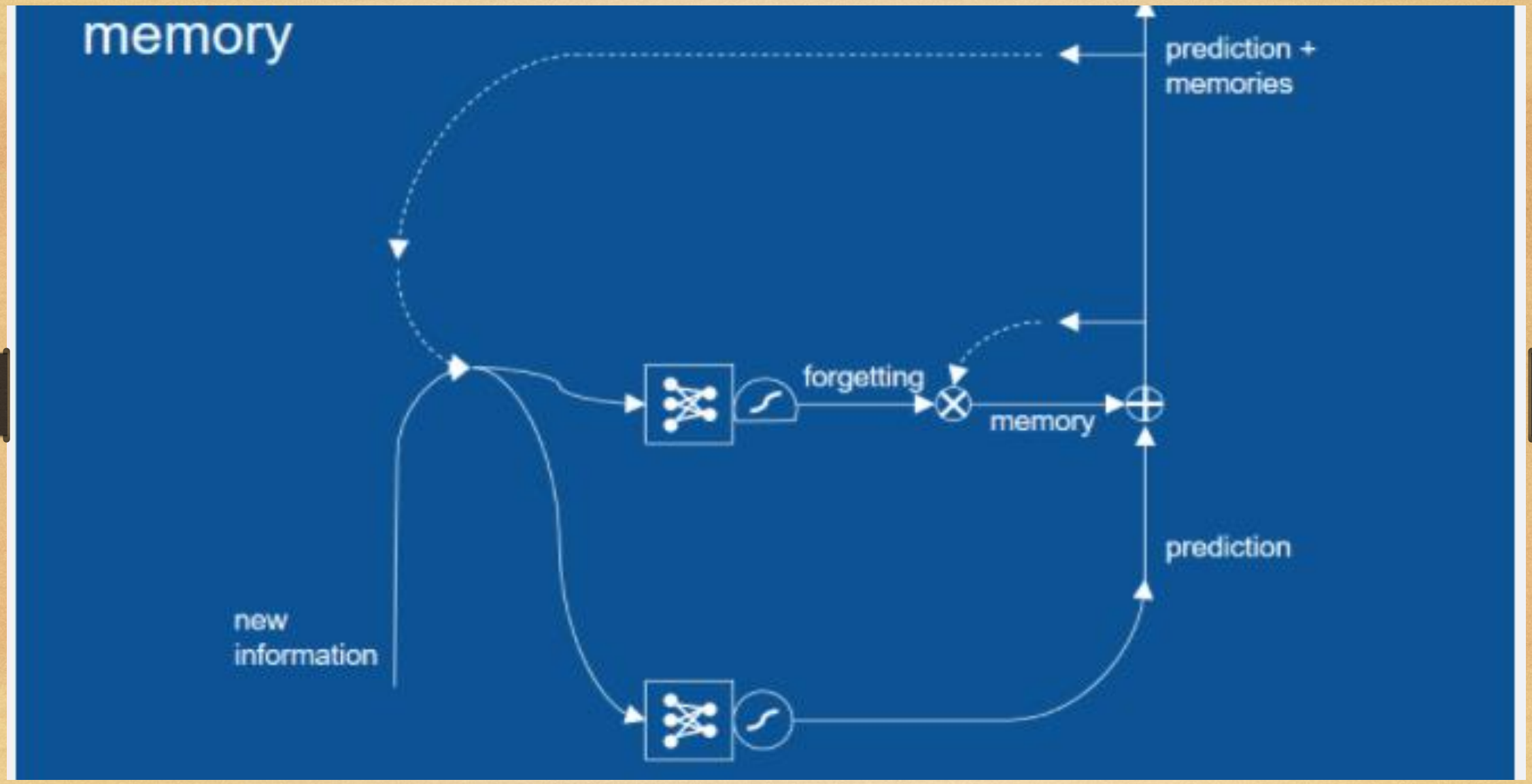
- In the mid-90s, a variation of recurrent net with so-called Long Short-Term Memory units, or LSTMs, was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem.
- LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely. This is one of the central challenges to machine learning and AI, since algorithms are frequently confronted by environments where reward signals are sparse and delayed, such as life itself. (Religious thinkers have tackled this same problem with ideas of karma or divine reward, theorizing invisible and distant consequences to our actions.)

Long Short-Term Memory Units (LSTMs)

- LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation.
- Those gates act on the signals they receive, and similar to the neural network's nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.

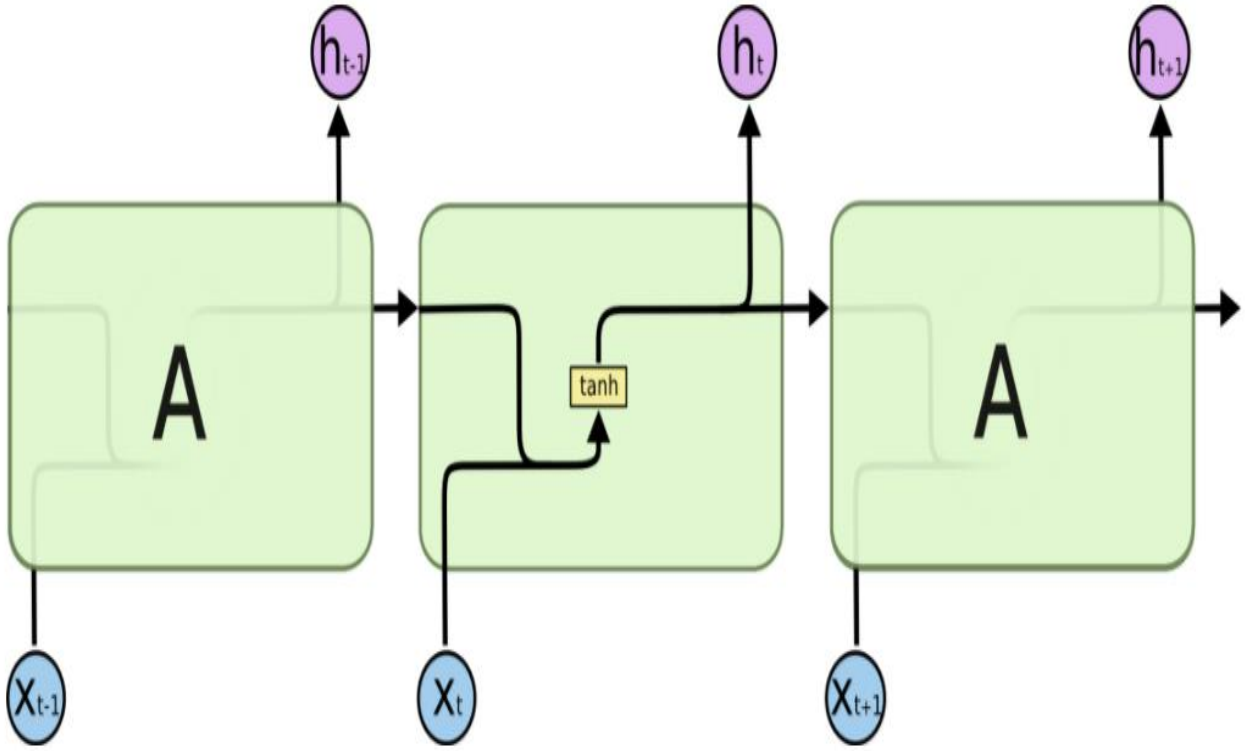
Long Short-Term Memory Units (LSTMs)

- In one sentence LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!
- The critical part here is we added memory in the middle. We want to be able to remember what happened many time steps ago


memory
prediction +
memories
forgetting
memory
prediction
new
information

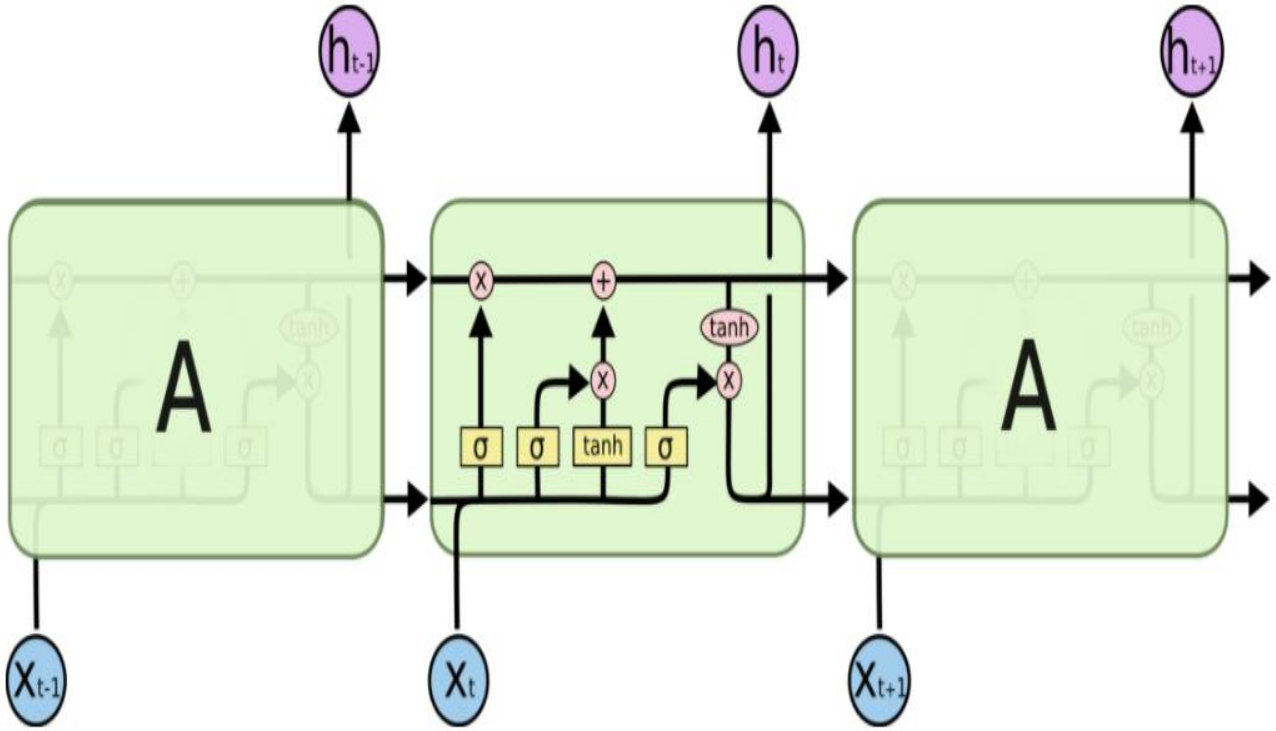
Standard RNN

- All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



Long Short-Term Memory Units (LSTMs)

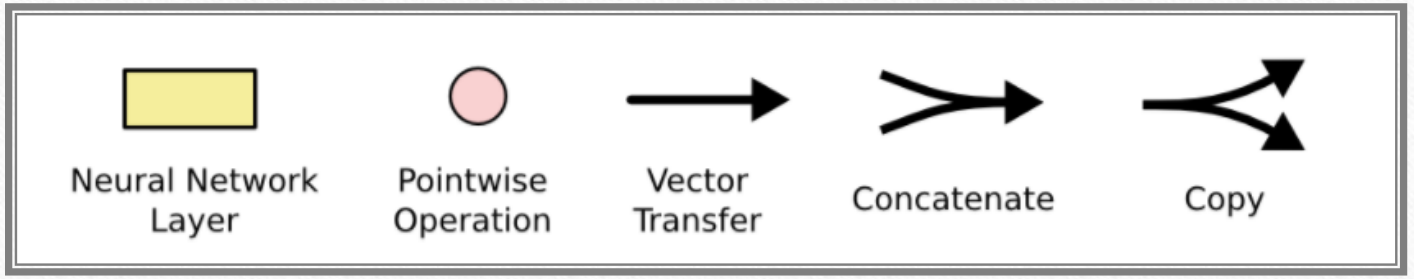
- LSTMs also have this chain like structure, but the repeating module has a different structure. **Instead of having a single neural network layer, there are four**, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

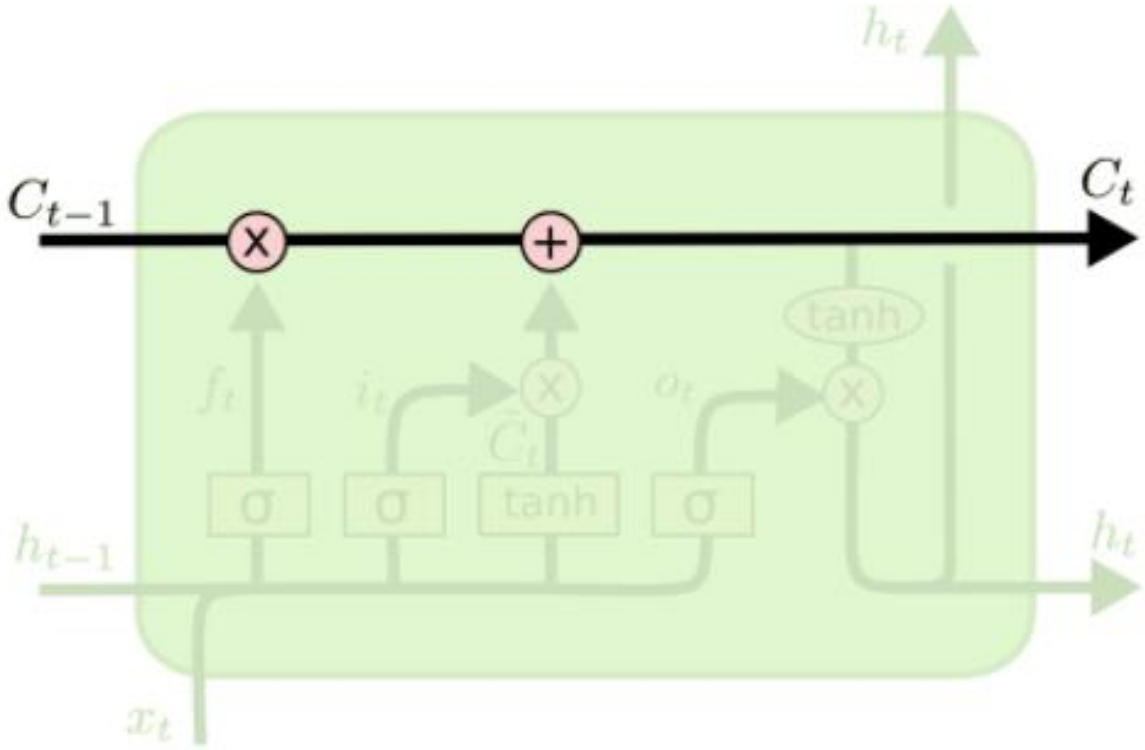
Long Short-Term Memory Units (LSTMs)

- Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.
- In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.



The Core Idea Behind LSTMs

- The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

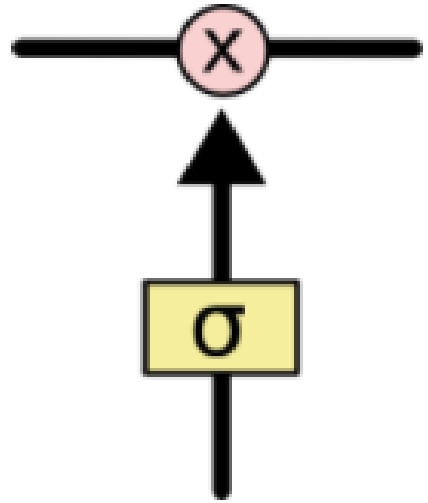


The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

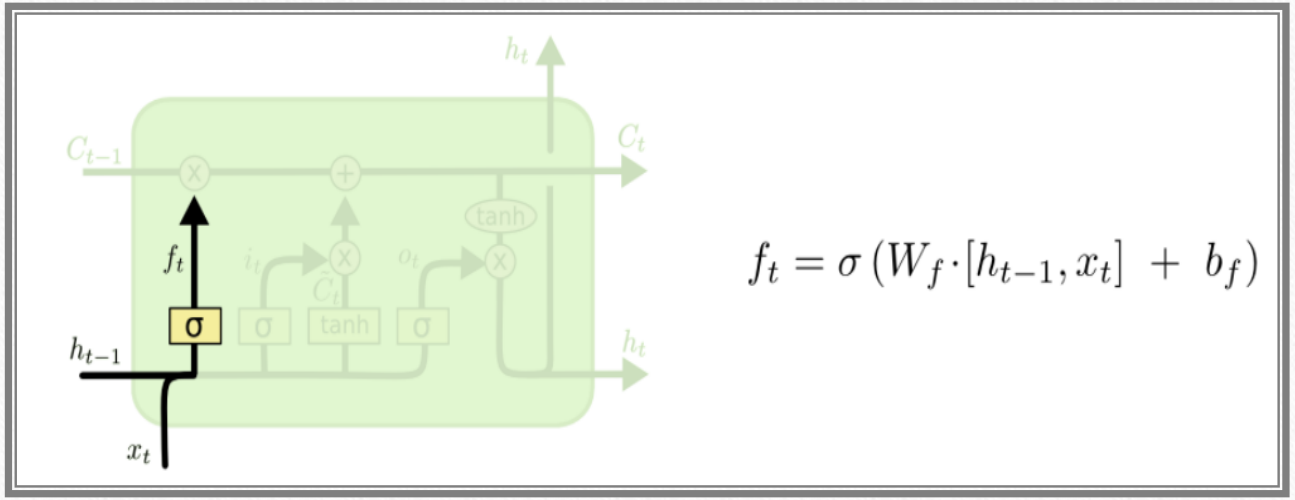
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state.



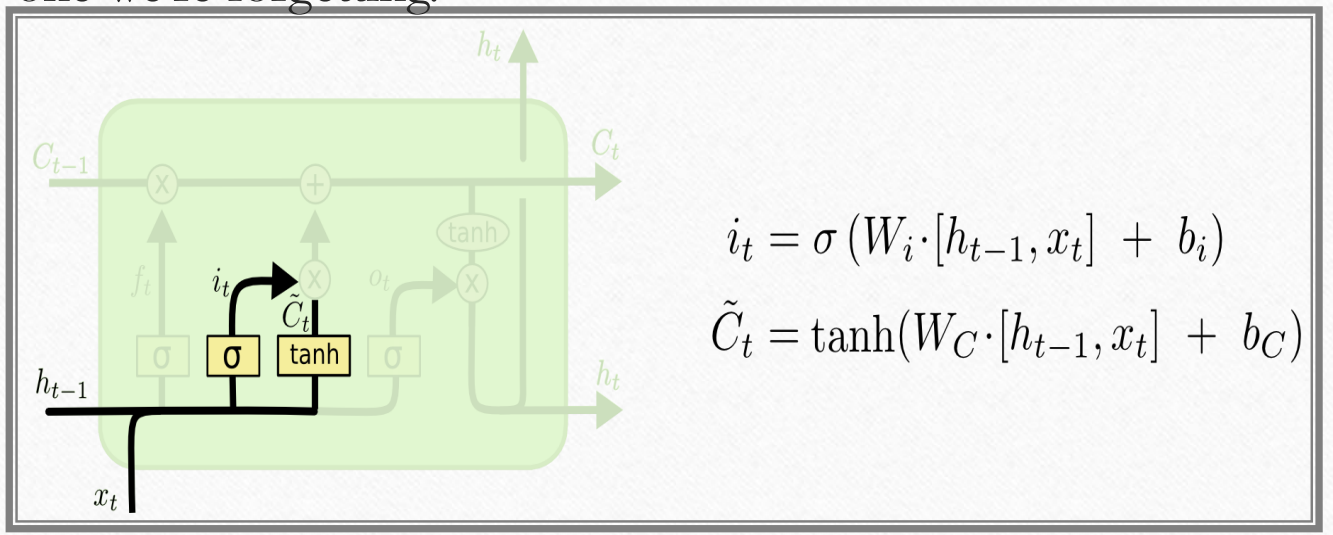
Step-by-Step LSTM Walk Through

- The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "**forget gate layer**." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. 1 represents "completely keep this" while a 0 represents "completely get rid of this."
- Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



Step-by-Step LSTM Walk Through

- The next step is to *decide what new information we're going to store in the cell state*. This has two parts. First, a *sigmoid layer* called the "input gate layer" *decides which values we'll update*. Next, a *tanh layer* creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.
- In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

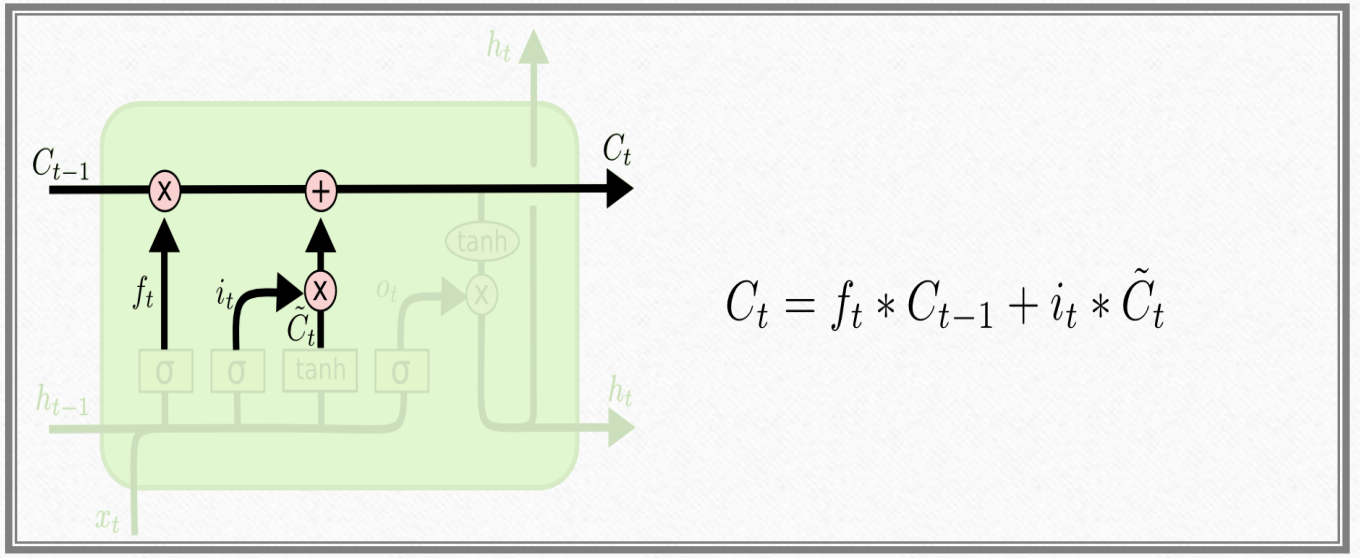


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Step-by-Step LSTM Walk Through

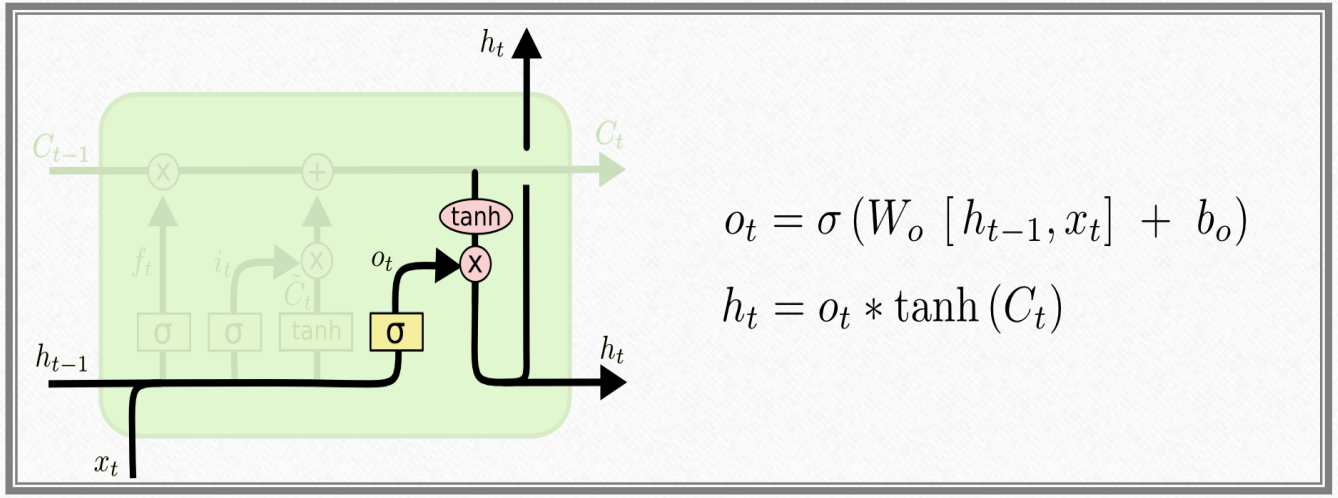
- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.
- In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

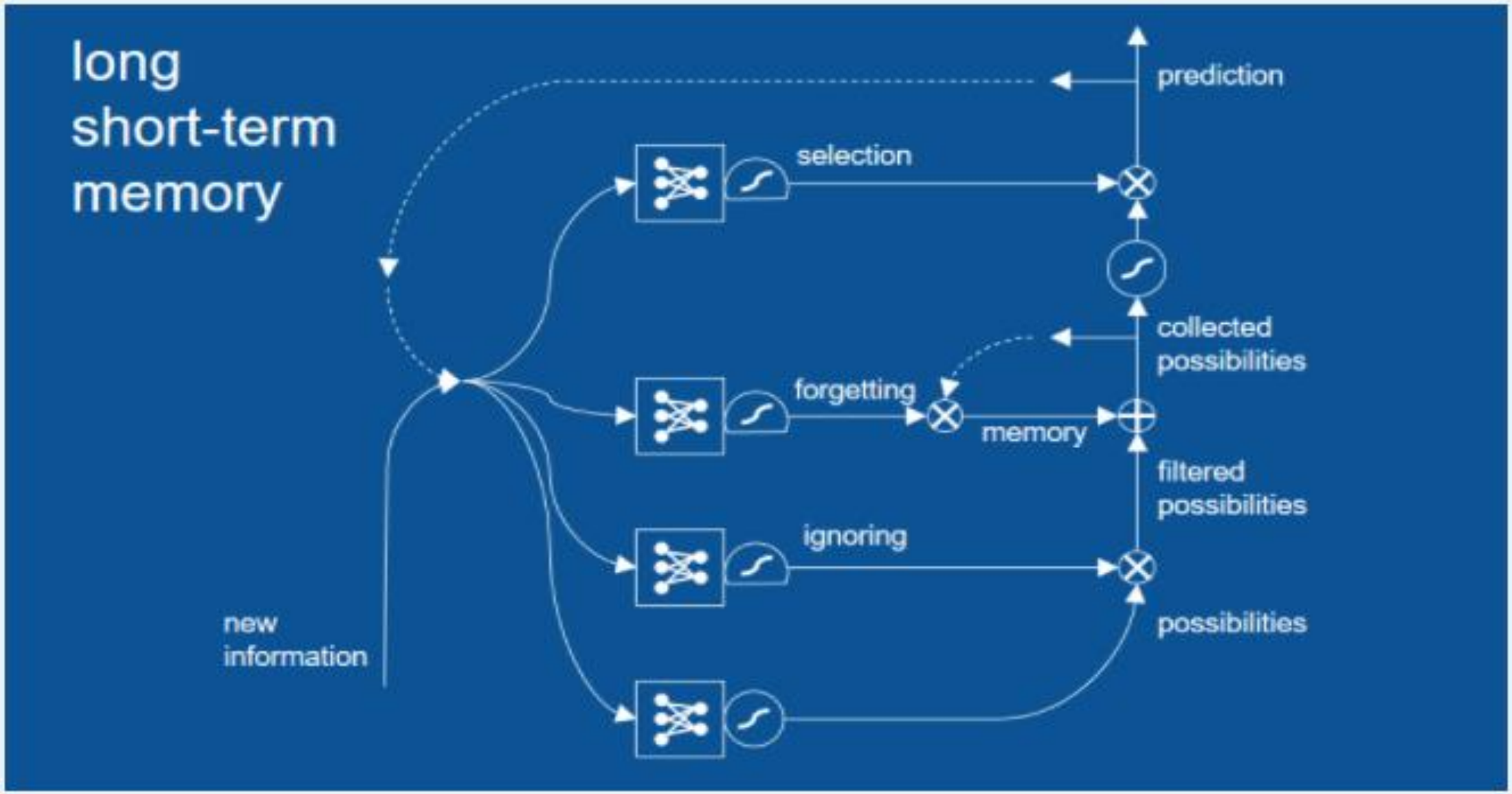
Step-by-Step LSTM Walk Through

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.
- For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$


long
short-term
memory
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities
new
information

LSTM Case Study

- now long short-term memory has a lot of pieces -a lot of bits that work together- and it's a little much to wrap your head around it all at once, so what we'll do is take a very simple example and step through it just to illustrate how a couple of these pieces work.
- So we are now in the process of writing our children's book and for the purposes of demonstration we will assume that this LSTM has been trained on our children's books examples that we want to mimic and all of the appropriate votes and weights in those neural networks have been learned. Now we'll show it in action.

Write a children's book

Doug saw Jane.

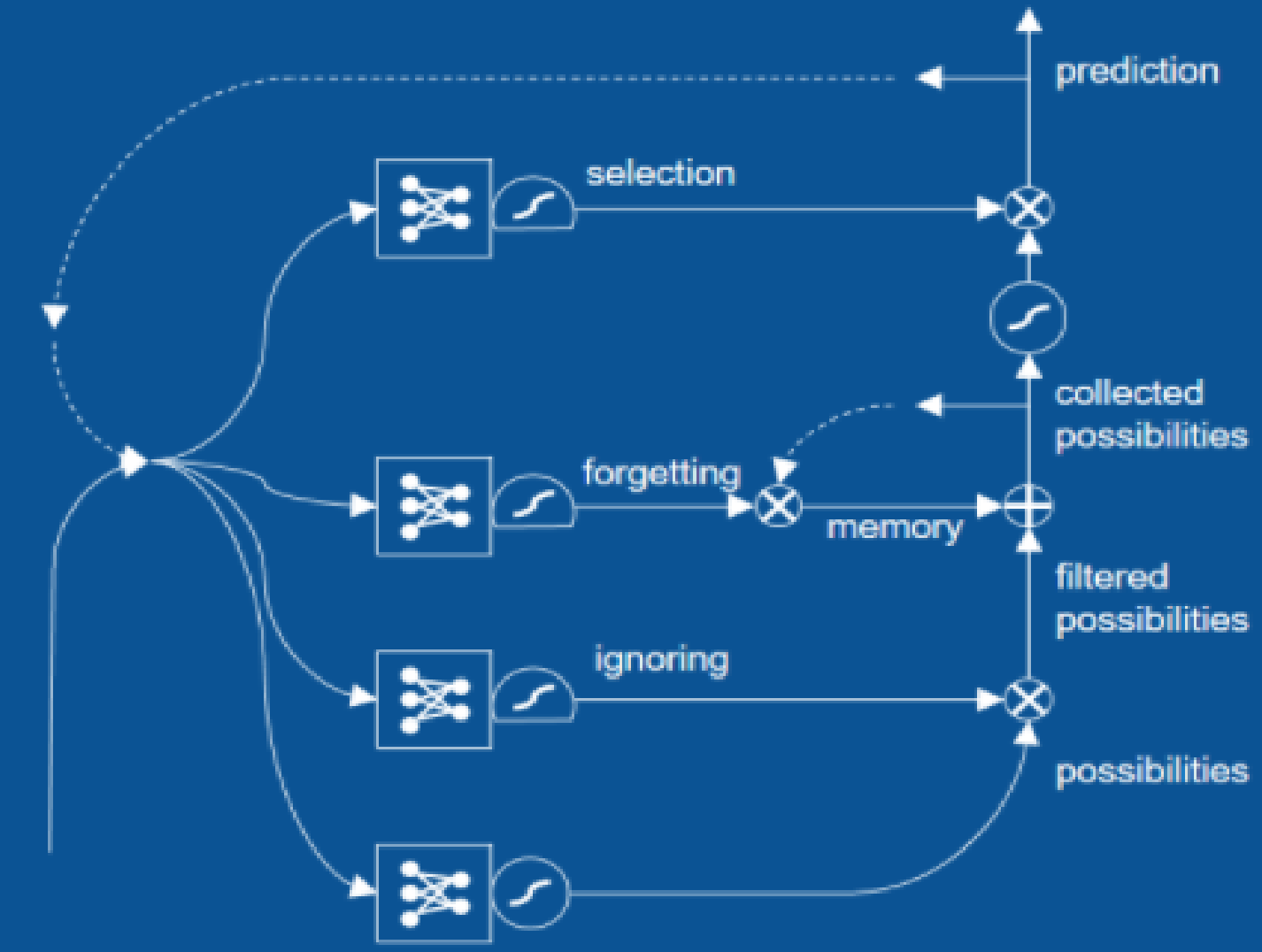
Jane saw Spot.

Spot saw Doug.

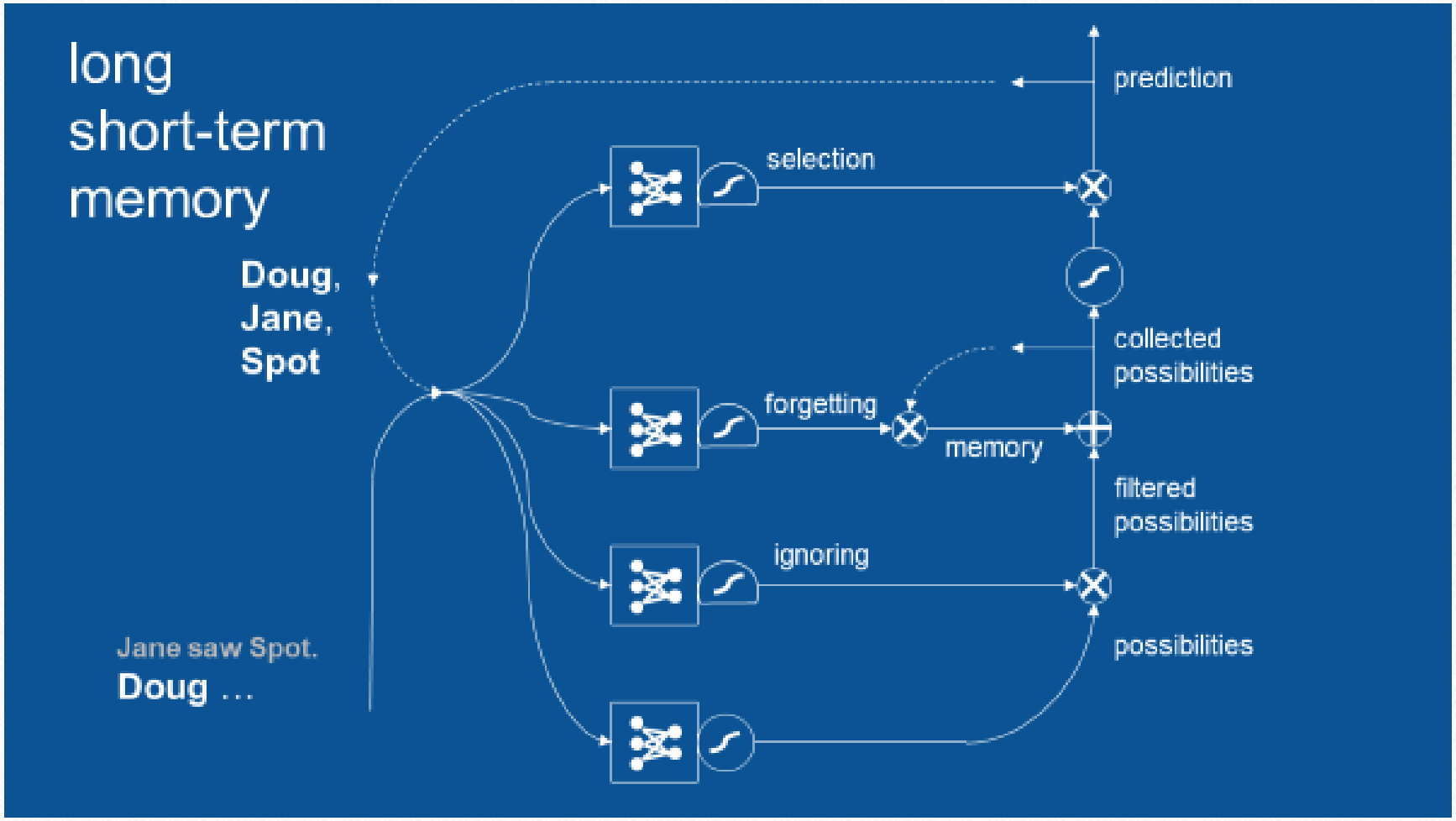
...

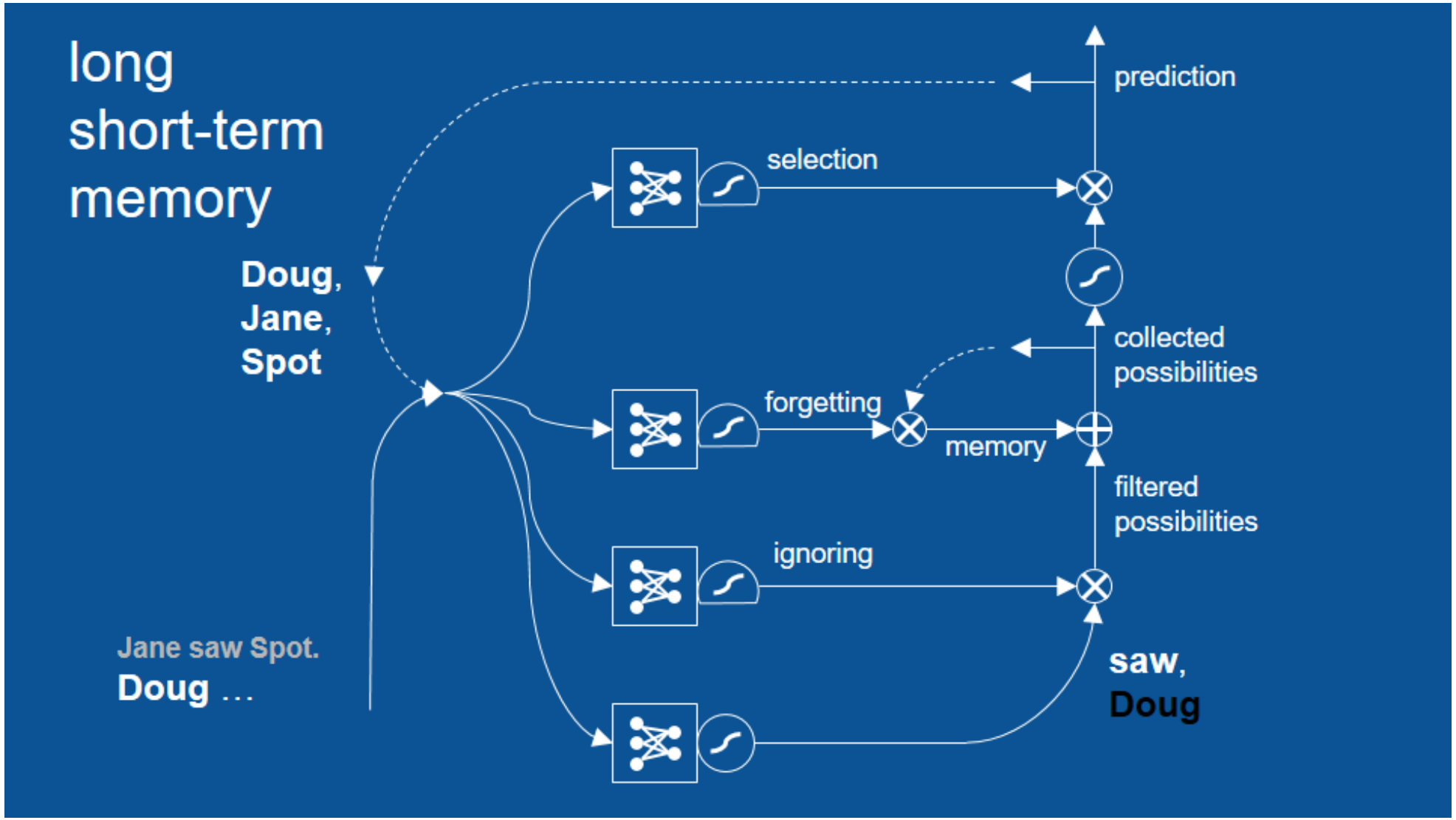
Your dictionary is small: {Doug, Jane, Spot, saw, .}

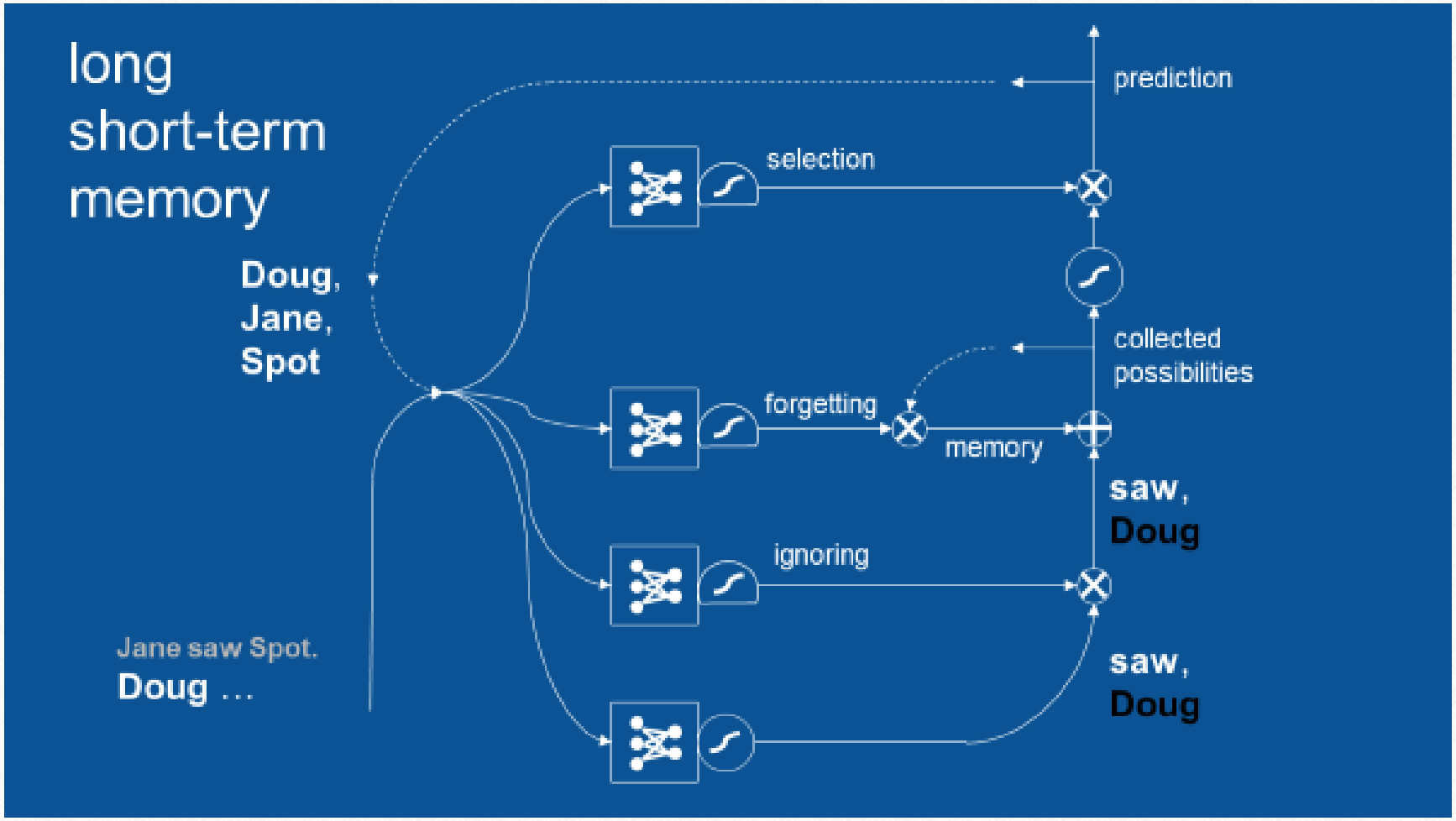
long short-term memory

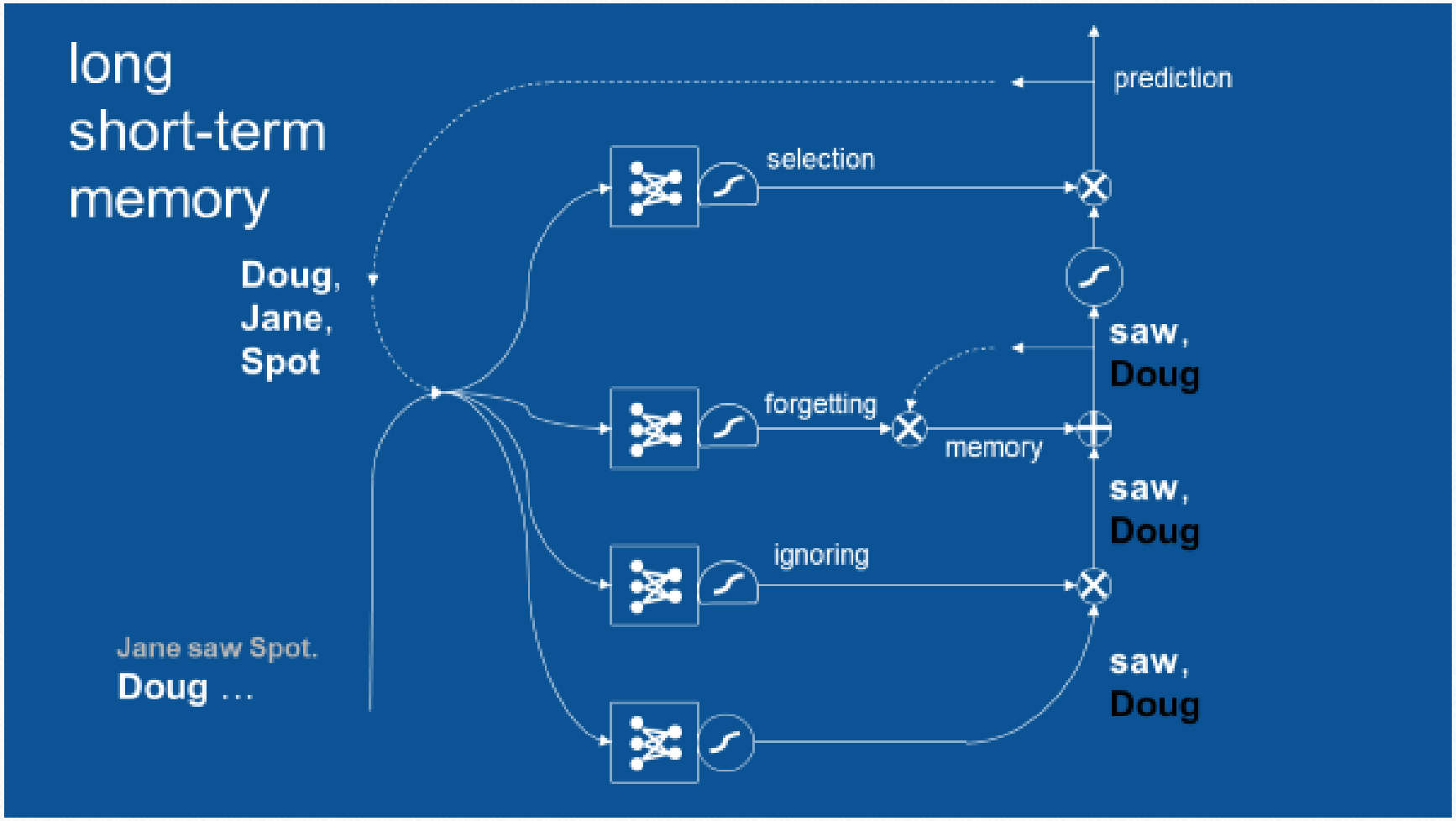


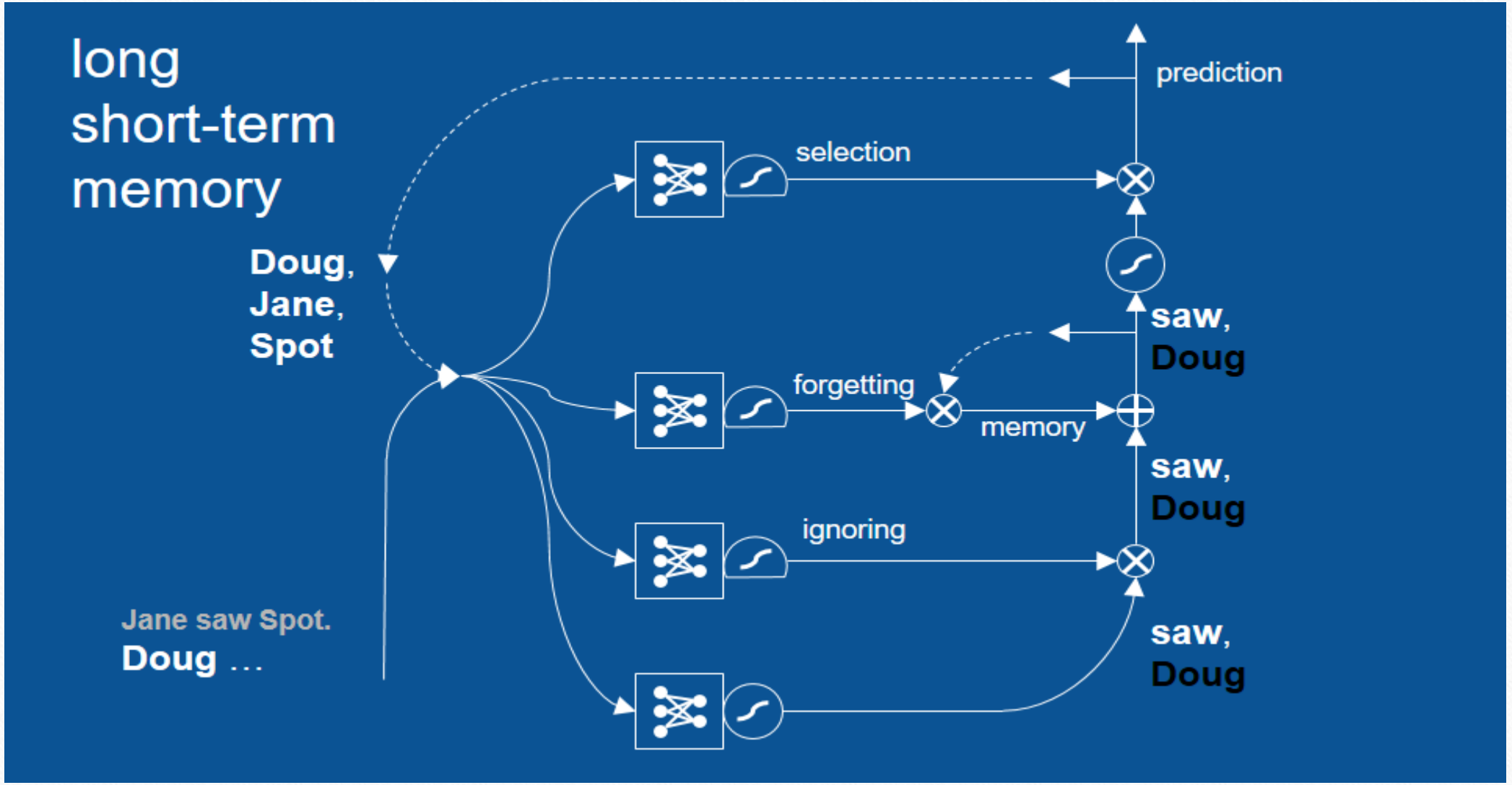
Jane saw Spot.
Doug …

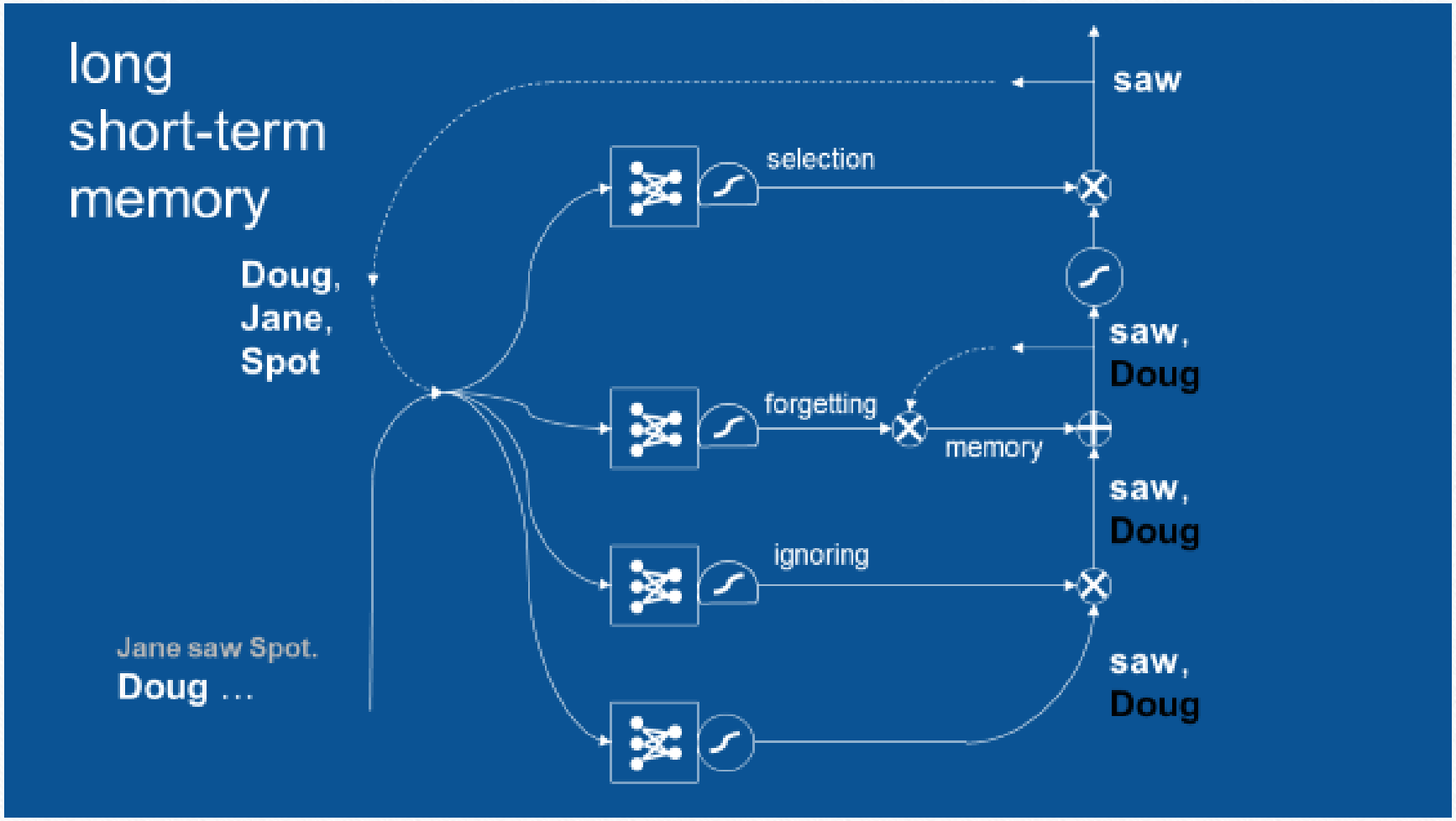

long
short-term
memory
Doug,
Jane,
Spot
Jane saw Spot.
Doug …
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities

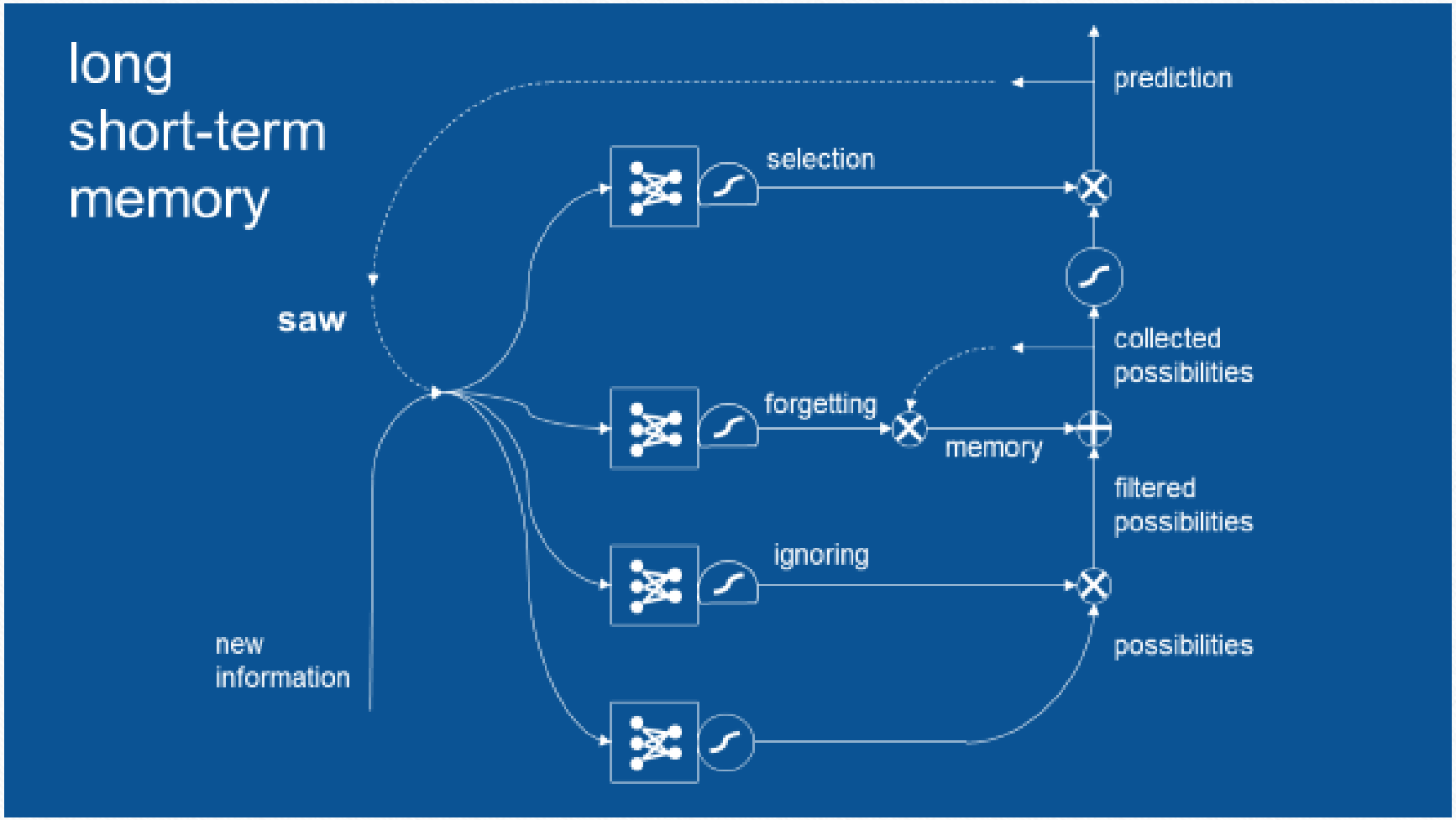

long
short-term
memory
Doug,
Jane,
Spot
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
Jane saw Spot.
Doug …
saw,
Doug

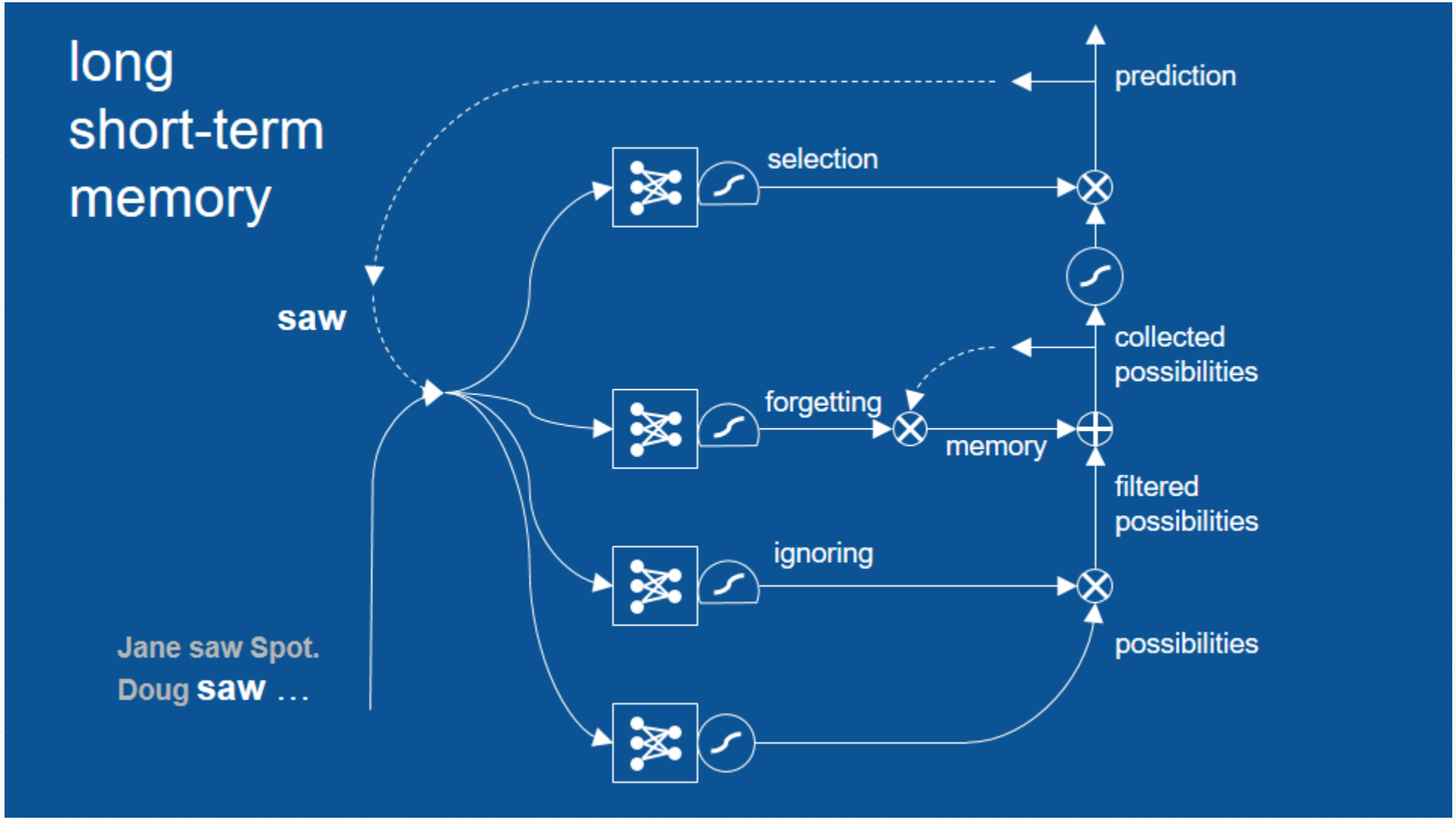

long
short-term
memory
Doug,
Jane,
Spot
prediction
selection
collected
possibilities
forgetting
memory
saw,
Doug
ignoring
saw,
Doug
Jane saw Spot.
Doug …

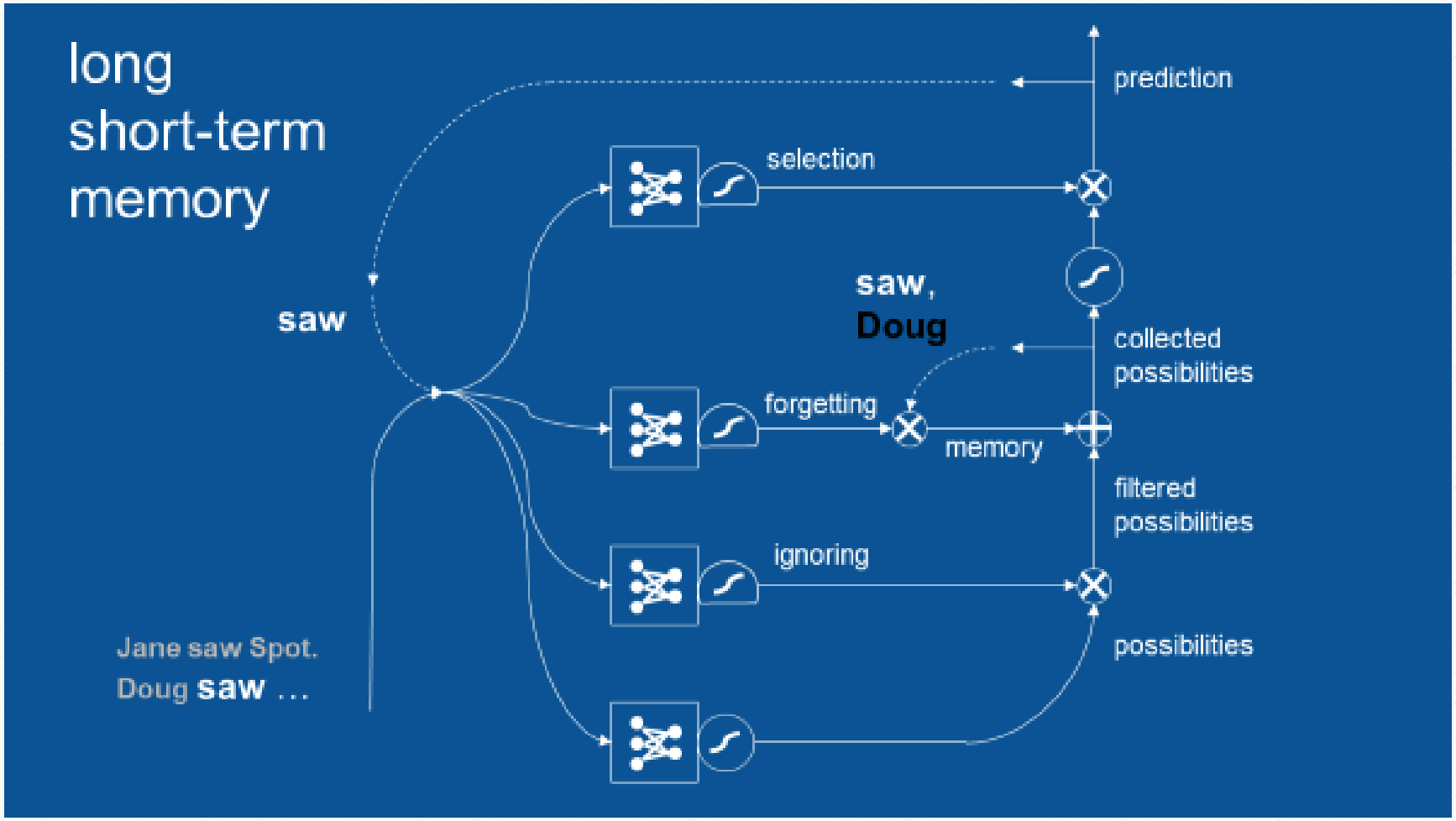

long
short-term
memory
Doug,
Jane,
Spot
Jane saw Spot.
Doug …
prediction
selection
forgetting
memory
ignoring
saw,
Doug
saw,
Doug
saw,
Doug

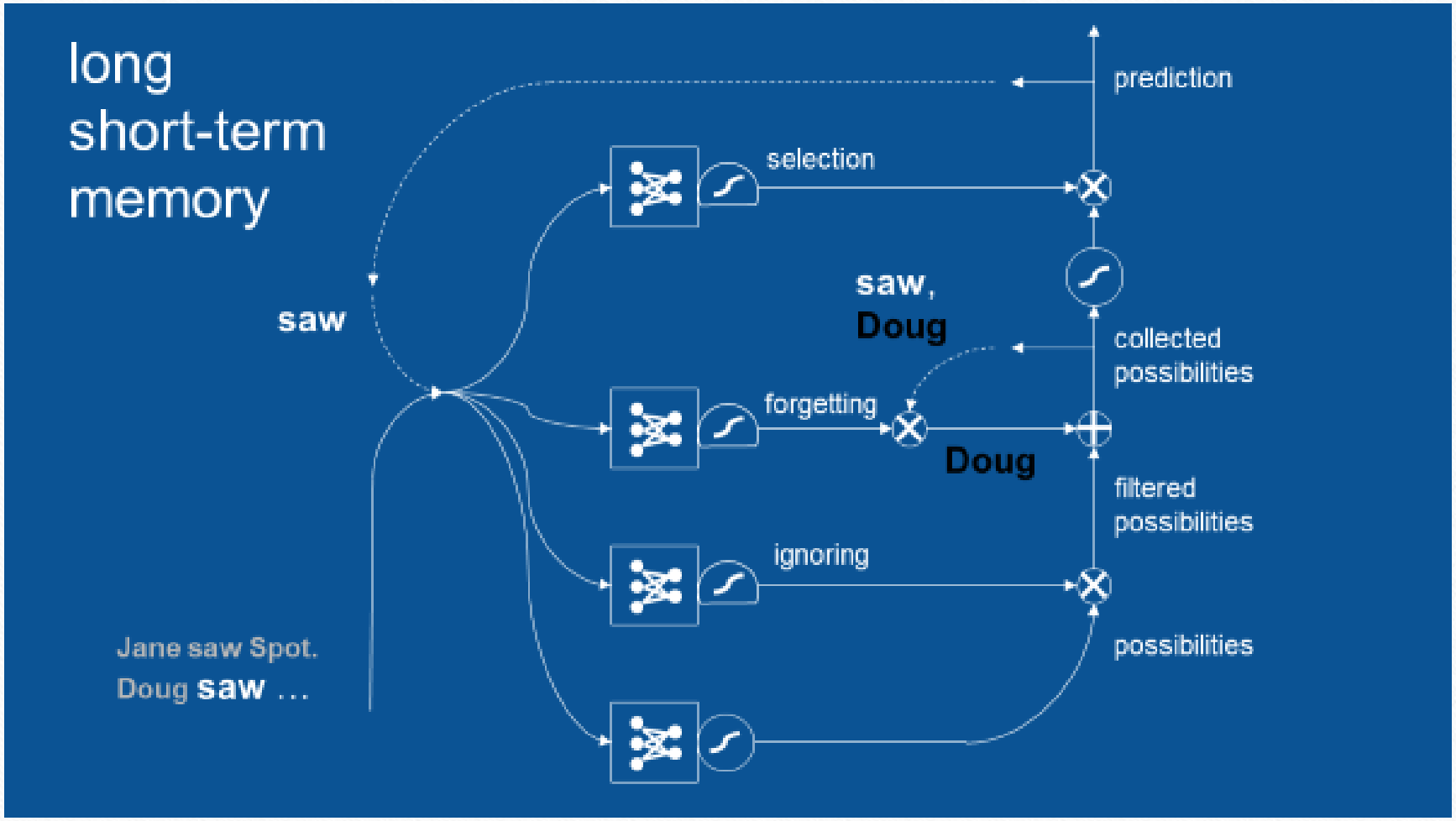

long
short-term
memory
Doug,
Jane,
Spot
Jane saw Spot.
Doug …
selection
forgetting
ignoring
memory
prediction
saw,
Doug
saw,
Doug
saw,
Doug

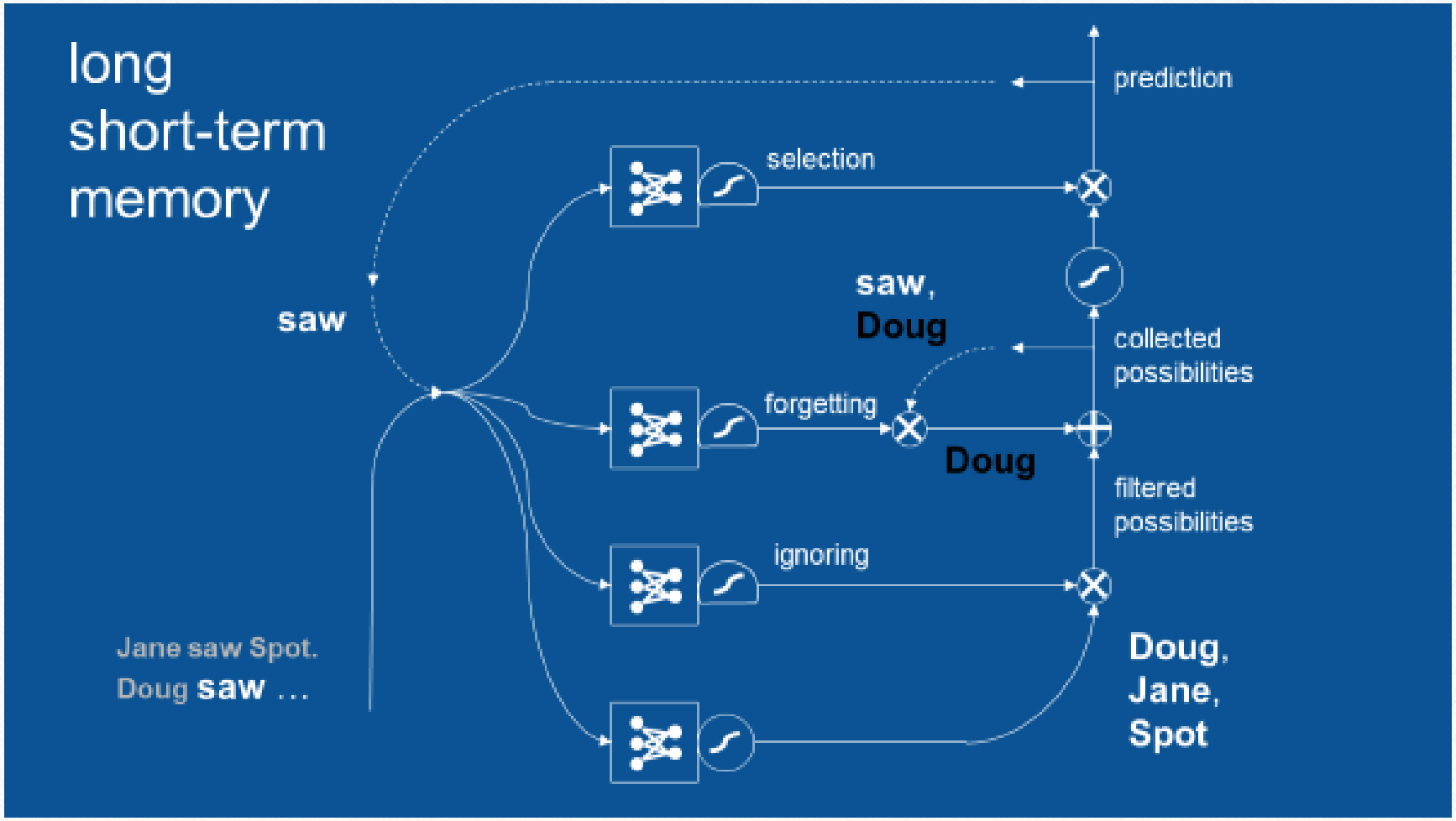

long
short-term
memory
Doug,
Jane,
Spot
selection
forgetting
memory
ignoring
saw
saw,
Doug
saw,
Doug
saw,
Doug
Jane saw Spot.
Doug …

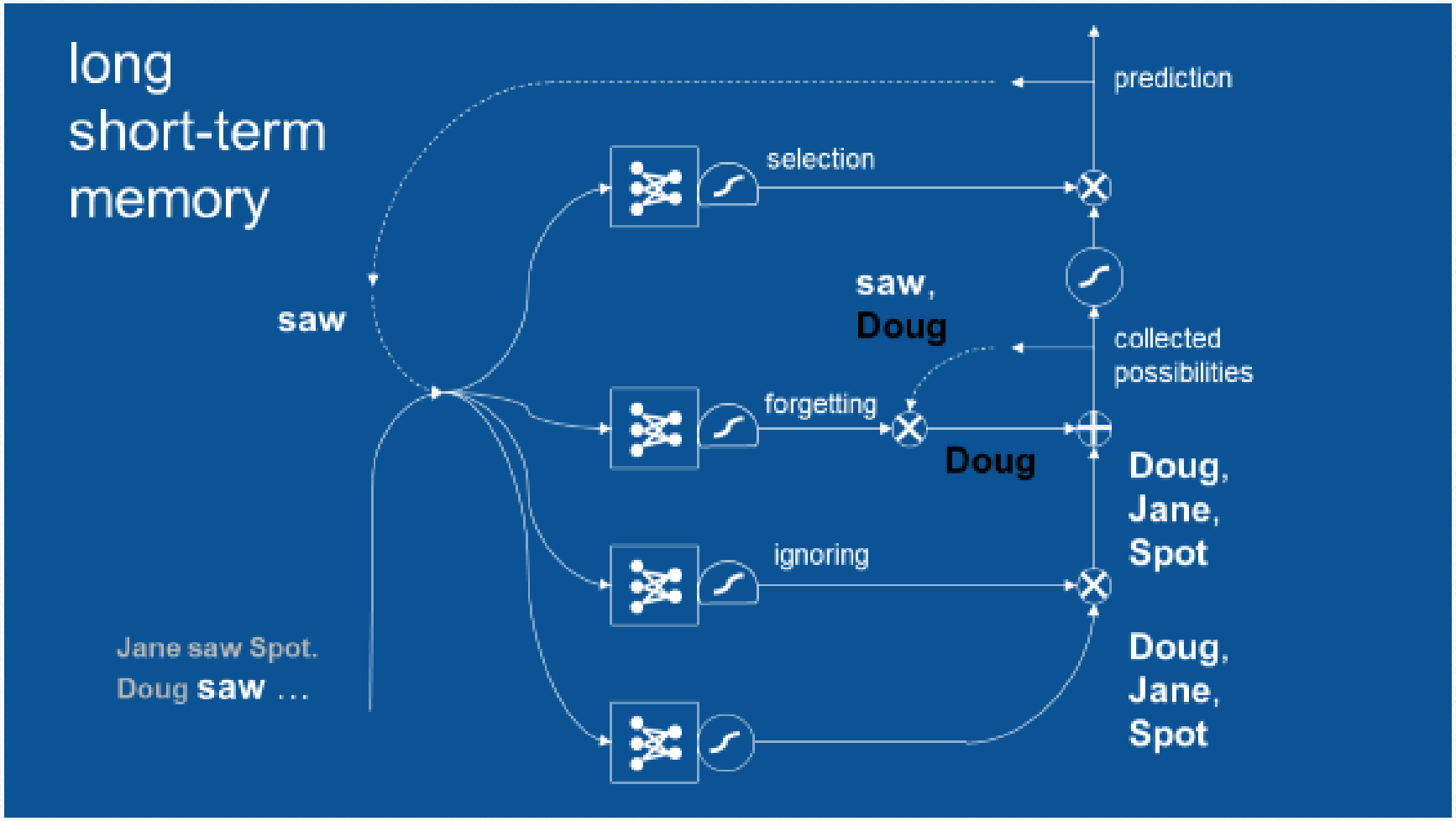

long
short-term
memory
prediction
selection
saw
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
new
information
possibilities

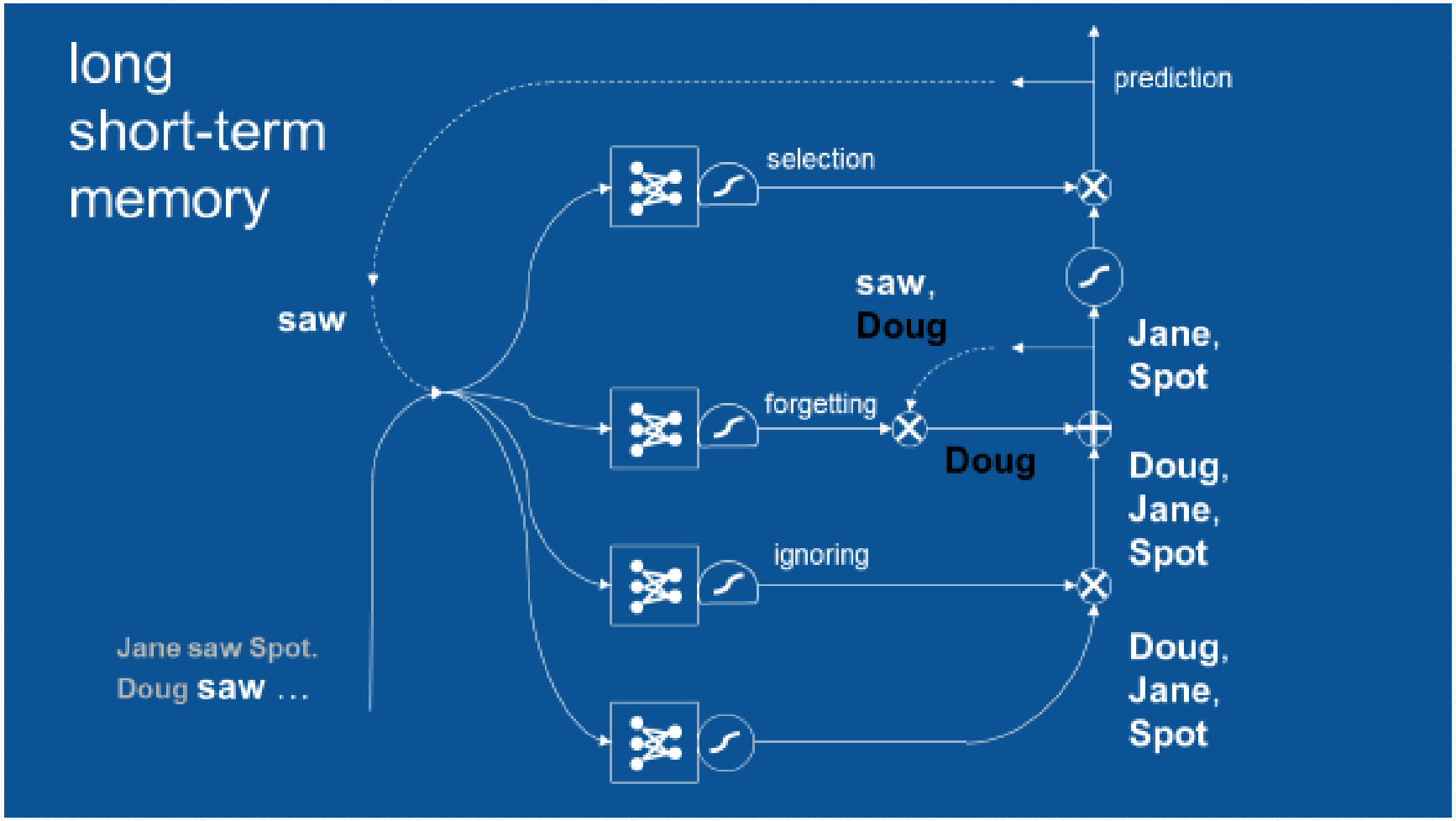

long
short-term
memory
prediction
selection
saw
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities
Jane saw Spot.
Doug saw …

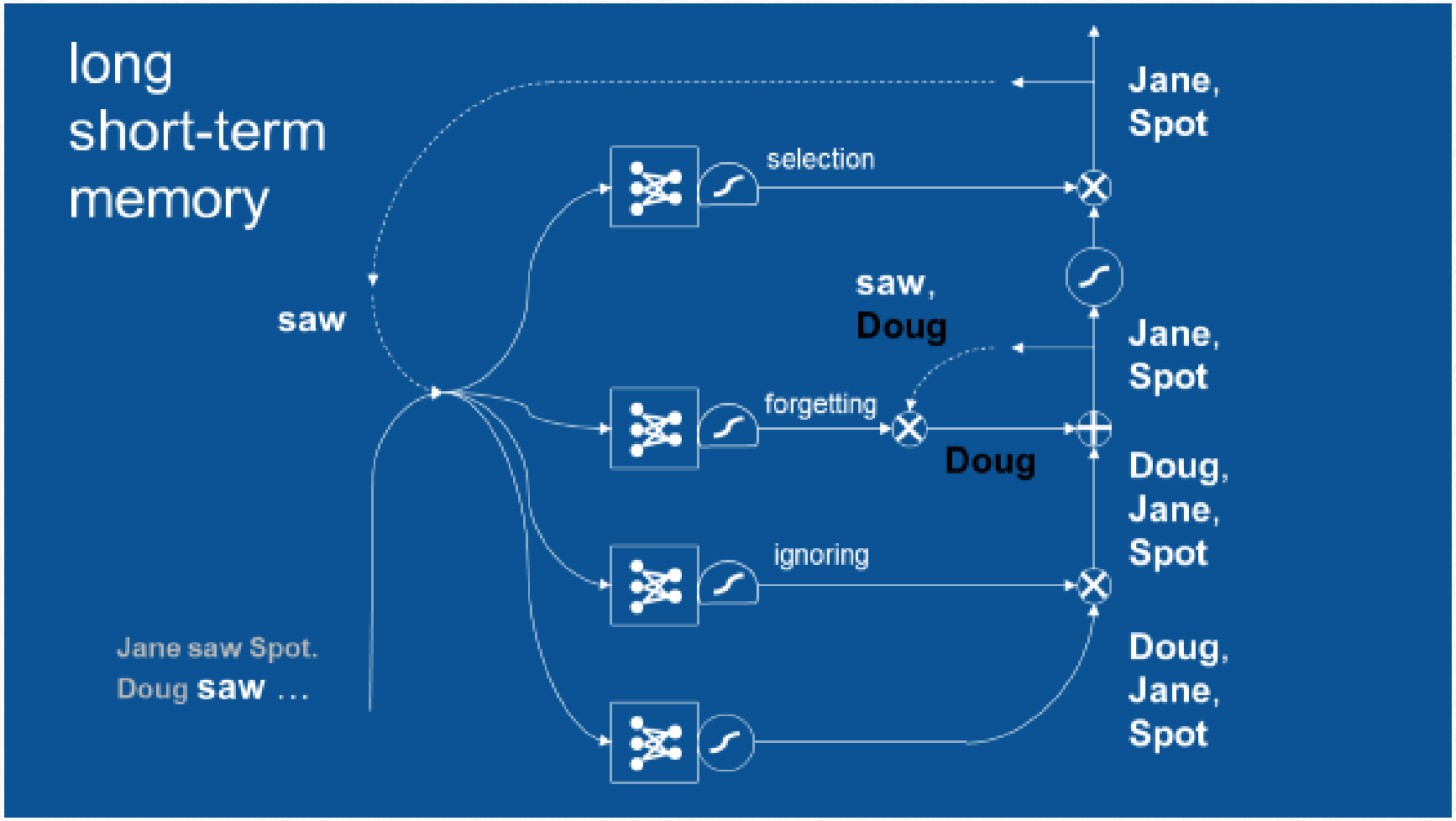

long
short-term
memory
prediction
selection
saw
saw,
Doug
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities
Jane saw Spot.
Doug saw …


long
short-term
memory
prediction
selection
saw
saw,
Doug
collected
possibilities
forgetting
Doug
filtered
possibilities
ignoring
possibilities
Jane saw Spot.
Doug saw …


long
short-term
memory
prediction
selection
saw
saw,
Doug
collected
possibilities
forgetting
Doug
filtered
possibilities
ignoring
Jane saw Spot.
Doug saw …
Doug,
Jane,
Spot


long
short-term
memory
prediction
selection
saw
saw,
Doug
collected
possibilities
forgetting
Doug
Doug,
Jane,
Spot
ignoring
Jane saw Spot.
Doug saw …
Doug,
Jane,
Spot


long
short-term
memory
prediction
selection
saw
saw,
Doug
Jane,
Spot
forgetting
Doug
Doug,
Jane,
Spot
ignoring
Doug,
Jane,
Spot
Jane saw Spot.
Doug saw …


long
short-term
memory
Jane,
Spot
selection
saw,
Doug
saw
Jane,
Spot
forgetting
Doug
Doug,
Jane,
Spot
ignoring
Jane saw Spot.
Doug saw …
Doug,
Jane,
Spot

LSTM Applications

- This is really useful in some surprisingly practical applications. If I have text in one language and I want to translate it to text to another language, LSTM's work very well. 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.

Sequential patterns

Text

Speech

Audio

Video

Physical processes

Anything embedded in time (almost everything)