Deep Learning and Neural Networks

Demetrio Labate

April 6, 2024

Part 6
Recurrent Neural Networks

Analysis of data with temporal dependencies

Several applications involve sequences of data with temporal dependencies:

- Time series forecasting (Healthcare, Finance, etc.)
- Natural Language Processing (NLP)
- Speech recognition

Common tasks:

- Predicting the next value in a sequence
- Converting data sequence to equivalent sequence in another space (translation)
- Classifying the entire sequence into specific class.

Analysis of data with temporal dependencies

The discipline of **dynamical systems** is concerned with developing tools used to model **dynamic datasets**, by which we mean **ordered data**, where often (but not always) ordering refers to the time variable.

Definition.
A **dynamical system with fixed order** consists of a function f and a sequence $(x_1, x_1, \dots)$ determined by the recursive equation

$$\begin{aligned} x_t &= \gamma_t, \qquad t = 1, \dots, W, \\ x_t &= f(x_{t-1}, \dots, x_{t-W}), \quad t > W \end{aligned}$$

where γ_t, for $t = 1, \dots, W$ are called the **initial conditions** of the system.

Analysis of data with temporal dependencies

Example: The moving average. We take a window and slide it along the time series from its start to finish and average the values inside.

We consider a time series $x_1, x_2, \ldots, x_N$
Note that it is a set of ordered points (where x_1 comes before x_2, x_2 comes before x_3, and so forth).
Choosing window size $W = 15$, the values of the moving average are simply set to the values of the input series itself for the first 15 values

$$h_t = x_t, \quad t = 1, \ldots 15$$

After these first initial values we create those that follow by averaging the preceeding 15 elements of the input series

$$h_t = \frac{1}{15} \sum_{i=1}^{15} x_{t-i}$$

Analysis of data with temporal dependencies

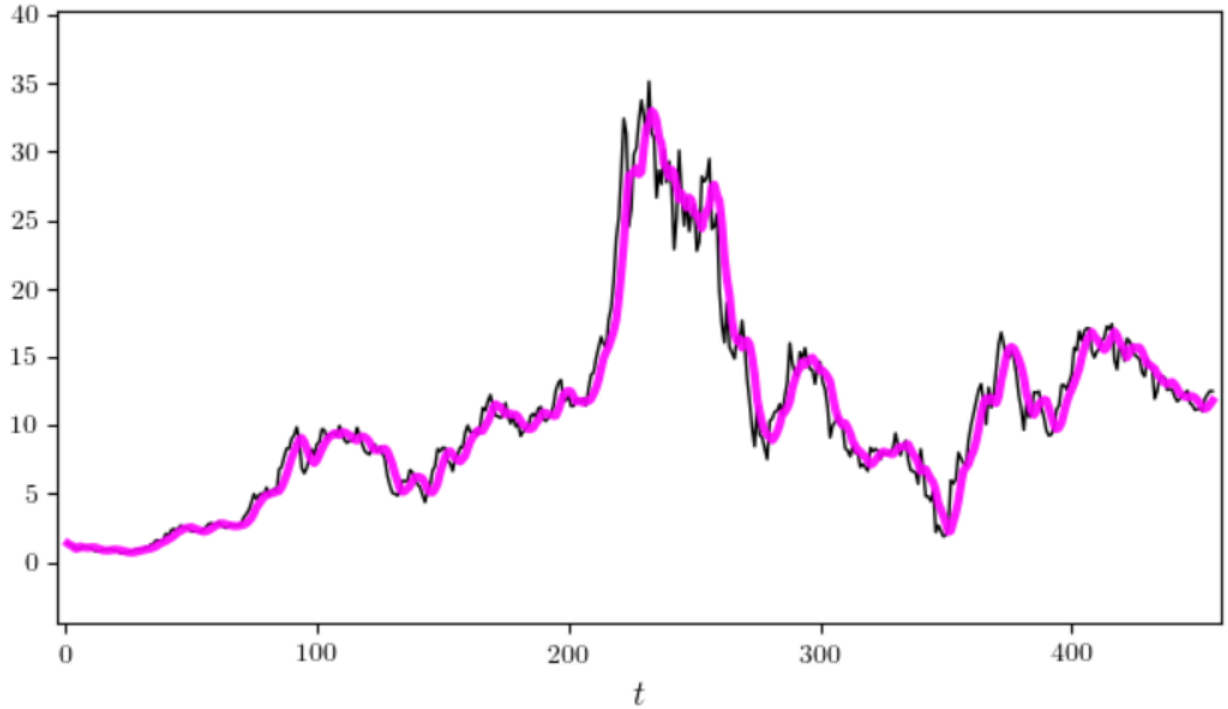
The moving average process is an example of what are called dynamic systems with fixed order.

The 'dynamic systems' part of this jargon-phrase smeans that the system - here defined by the set of moving average values $h_1, h_2, \ldots, h_N$ - is defined in terms of recent values of the input sequence $x_1, x_2, \ldots, x_N$.

The 'fixed order' part refers to just how many preceeding elements of input each value h_t is based on - here this value was 15 - and this value is fixed for each value of h_t created.

Analysis of data with temporal dependencies

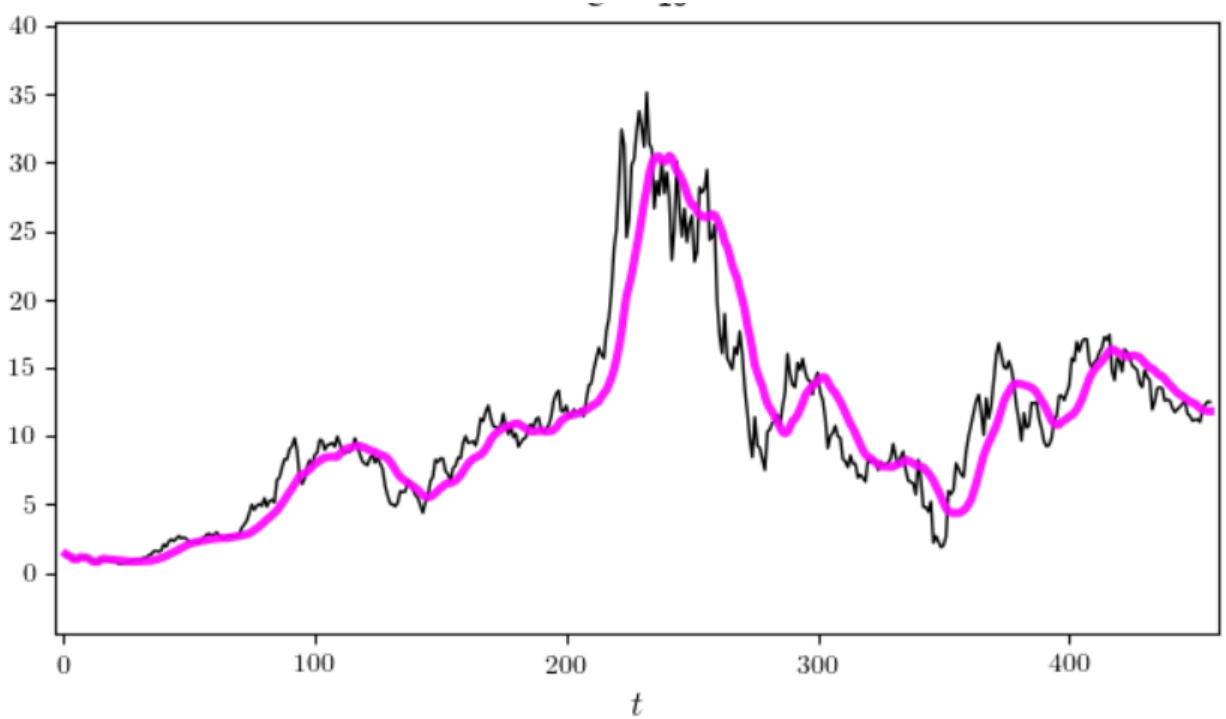
Below we show an input sequence (in black) and a set of moving average sequences with $W = 5, 15, 30$



As the order increases, the moving average gets smoother, but mirrors the structure of the original input sequence less and less. The delay of the moving average - its values trail those of the original series - increases with the order of the system and is an artifact of using a large history of equally weighted examples of the series as a predictor of its next values.

Analysis of data with temporal dependencies

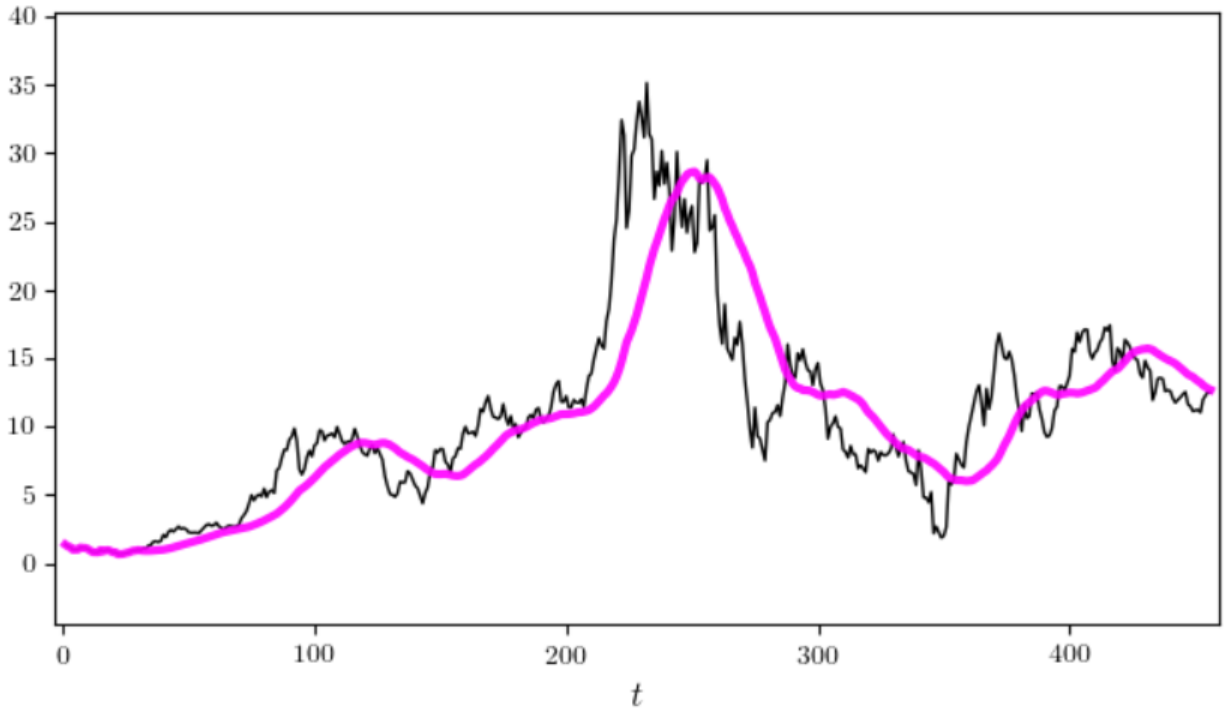
Below we show an input sequence (in black) and a set of moving average sequences with $W = 5, 15, 30$



As the order increases, the moving average gets smoother, but mirrors the structure of the original input sequence less and less. The delay of the moving average - its values trail those of the original series - increases with the order of the system and is an artifact of using a large history of equally weighted examples of the series as a predictor of its next values.

Analysis of data with temporal dependencies

Below we show an input sequence (in black) and a set of moving average sequences with $W = 5, 15, 30$



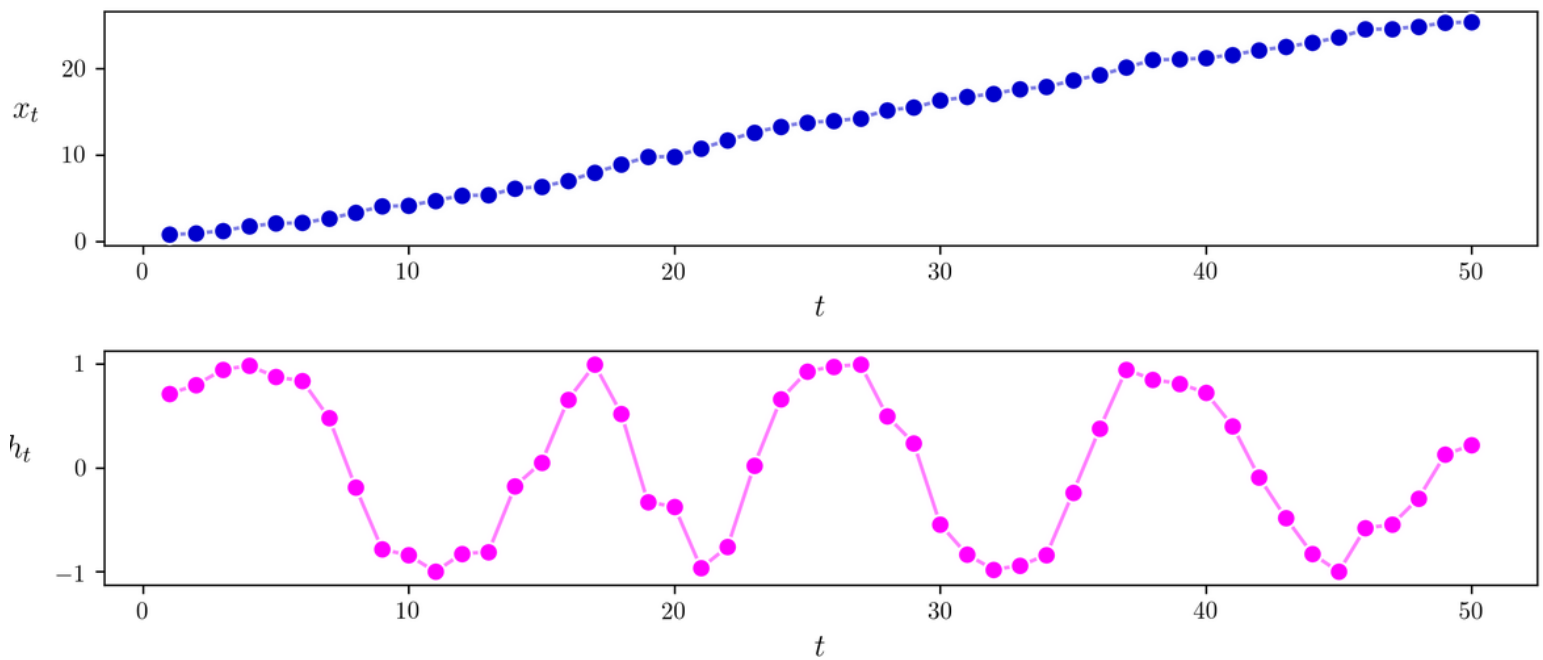
As the order increases, the moving average gets smoother, but mirrors the structure of the original input sequence less and less. The delay of the moving average - its values trail those of the original series - increases with the order of the system and is an artifact of using a large history of equally weighted examples of the series as a predictor of its next values.

Analysis of data with temporal dependencies

Using fixed order dynamic systems with input/output data we can choose order $W = 0$ where each element of the output sequence is dependent on only the current input point as $h_t = f(x_t)$.

These kind of systems are called **memoryless** since the dynamic system (here, in particular, the output sequence) is constructed without any knowledge of the past input values.

For example, the sequence $h_t = sin(x_t)$ below is memoryless.



Recurrence relations

A **recurrence relation** defines an input sequence in terms of itself as

$$x_t = f(x_{t-1}, \ldots, x_{t-W}), \quad t = W+1, \ldots, T$$

In other words, we do not begin with an input sequence, instead we generate one by recursing on a set of formulae of the form above.

To solve the equation, we need to set initial conditions, consisting here of the first W entries of the generated input sequence

$$x_t = \gamma_t, \quad t = 1, \ldots, W$$

Recurrence relations

Example: Exponential growth modeling
The recurrence relation of order $W = 1$

$$\begin{aligned} x_1 &= \gamma \\ x_t &= w_0 + w_1 x_{t-1}, \quad t > 1, \end{aligned}$$

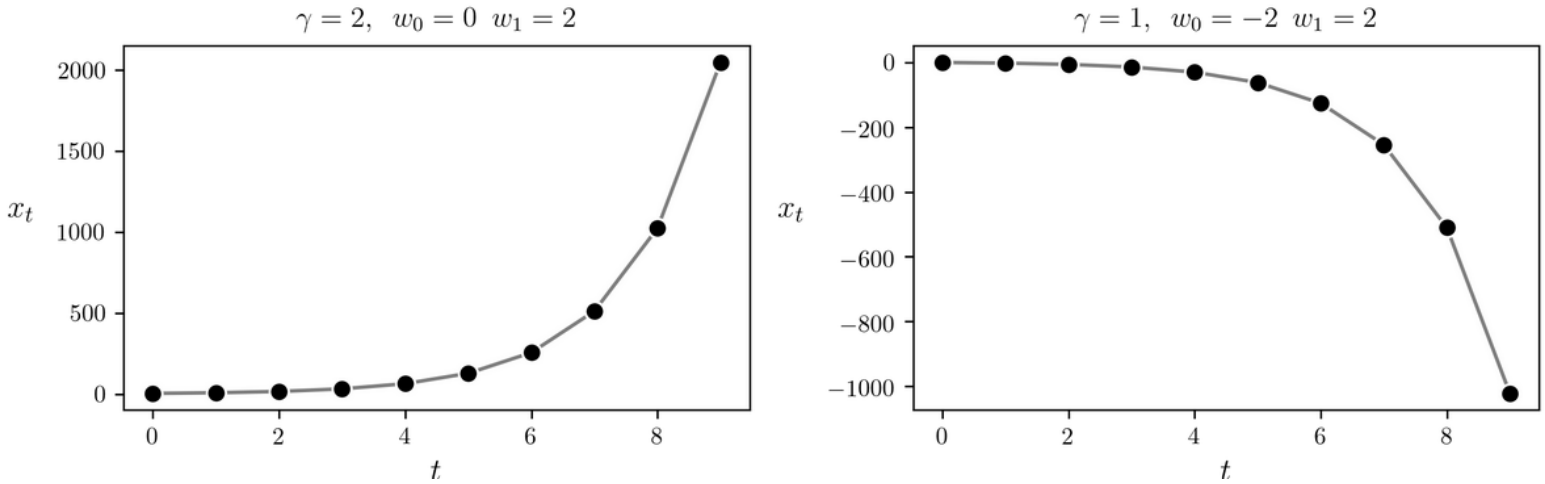
generates a sequence that exhibits exponential growth.

Here γ, w_0, and w_1 are constants which we can set to any values we please, and the function f is obviously linear.

The setting of this initial condition can significantly alter the trajectory of a sequence generated by such a system, as can the parameters of the update formula.

Recurrence relations

Below we show two example sequences of length $T = 10$



In the left panel, we set the initial condition $x_1 = 2$ and $w_0 = 0, w_1 = 2$. Note while each point in the sequence increases linearly from step to step, the data overall is increasing exponentially upwards.
In the right panel, we use an initial condition of $x_1 = 1$ with $w_0 = -2, w_1 = 2$. This data, while decreasing linearly at each step, globally is decreasing exponentially.

Recurrence relations

You can see algebraically how the update step above leads to exponential behavior by 'rolling back' the formula to its initial condition at a given value of t.

For example, say $w_0 = 0$ in the general exponential growth model above i.e., that our system takes the form

$$\begin{aligned} x_1 &= \gamma \\ x_t &= w_1 x_{t-1}, \quad t > 1, \end{aligned}$$

Then if we roll back the update step, replacing x_{t-1} with its update, we can write the update step above as

$$x_t = w_1 x_{t-1} = w_1^2 x_{t-2} = \cdots = w_1^t x_1$$

This does how how the sequence behaves exponentially depending on the coefficient value w_1.
Setting $w_0 \neq 0$, one can show a similar exponential relationship throughout the sequence by similarly rolling back to the initial condition

Recurrence relations

Example: Autoregressive models
One generalization of the exponential growth model in the previous example is the so-called **autoregressive system**.
The general order W autoregressive system takes the form

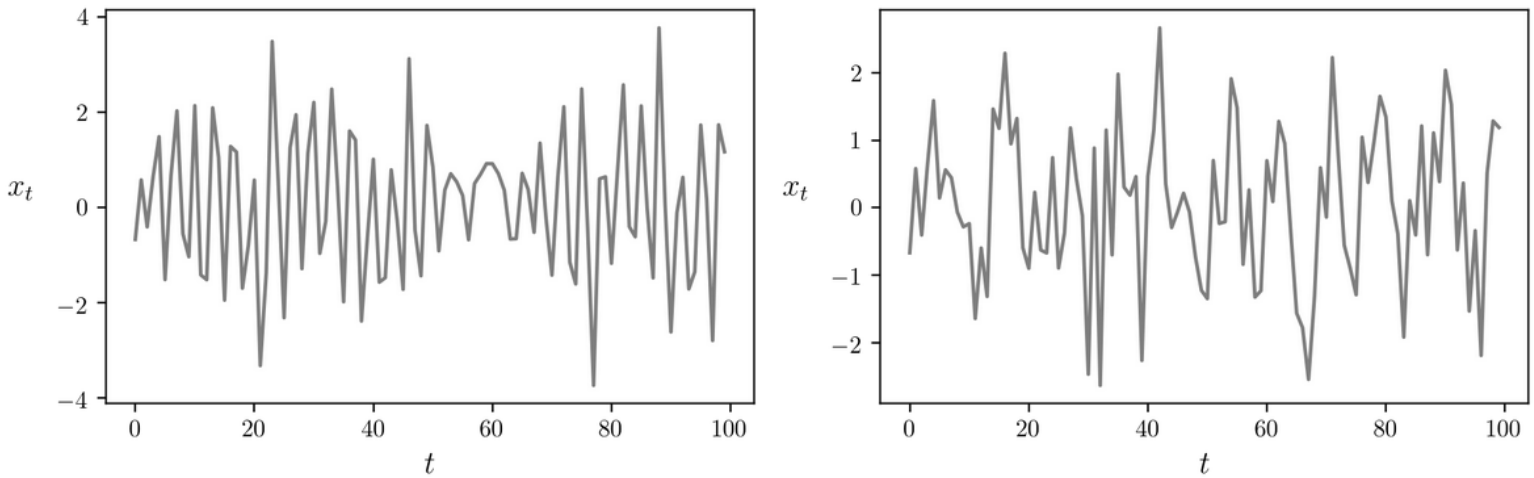
$$\begin{aligned} x_t &= \gamma_t, \quad 1 \leq t \leq W \\ x_t &= w_1 x_{t-1} + w_2 x_{t-2} + \cdots + w_k x_{t-k} + \epsilon_t, \quad t > W, \end{aligned}$$

where ϵ_t denotes the small amount of noise introduced at each step.

Sequences generated via this dynamic system tend to look like the sort of noisy financial time series commonly seen in practice after a centering procedure has been used to 'detrend' the data

Recurrence relations

Examples of autoregressive models of order $W = 4$.



Initial conditions and weights are chosen at random, with standard normal noise used at each step.

Recurrence relations

Other examples of recurrence relations are:

- **The Fibonacci Sequence.**

$$x_1 = 0, x_2 = 1$$
$$x_t = x_{t-1} + x_{t-2}, \quad t > 2.$$

It is a classic example of a dynamic system of order $W = 2$.

- **Logistic system.**

$$x_1 = \gamma$$
$$x_t = w_1 x_{t-1}(1 - x_{t-1}), \quad t > 1,$$

where the update function has the form $f(x) = wx(1 - x)$. Hence is a **non-linear** function.

Markov chains

There is another important class of dynamic systems described by **stochastic recurrence relations**; they are also called **Markov chains.**

They are defined formally as recurrence equation of the form

$$\begin{aligned} x_t &= \gamma_t, \quad 1 \leq t \leq W \\ x_t &= f(x_{t-1}, \ldots, x_{t-W}), \quad t > W, \end{aligned}$$

where the update function f is a stochastic function.

Markov chains

Motivation: Text analysis and generation.

Written text is commonly modeled as a recurrence relation.
For example, take the simple sentence

"my dog runs"

Each word in this sentence does - intuitively - seem to follow its predecessor in an orderly, functional, and predictable way.

Each word follows its immediate predecessor just like a $W = 1$ dynamic system.

Markov chains

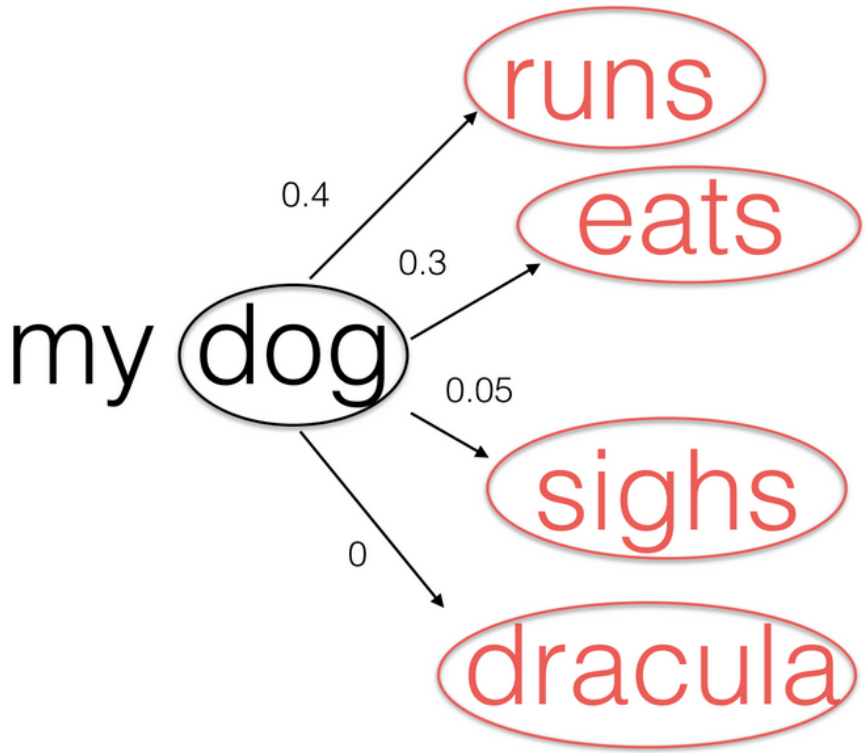
We can think of each word in a sentence following logically based not just on its immediate predecessor but on several preceding words, like an order W dynamic system.

However while text seems to have the structure of a dynamic system like the ones we have seen above, it does have one attribute that we have not seen thus far: choice. A given word does not always completely determine the word that follows it

That is, while a valid sentence of English words clearly has **ordered structure** like a dynamic system, with each word following its predecessor, there are often **many choices for subsequent words**. Moreover of the choices following a particular word we can reasonably say that some are more likely to occur than others. In other words, these choices are stochastic in nature - with each having a **different probability of occurring**.

Markov chains

Example: we show four choices of words following "dog" in the sentence above assigning a probability of occurring to each.



The probabilities shown do not add up to 1 because there could be other reasonable words that could follow "dog" other than those shown here.

Markov chains

Irregardless of how we compute the probabilities, mathematically we can codify the list possible outputs and their probability of occurrence - like the list of possible words shown above following "dog" - as a **probability mass function (pmf)** or a **histogram**.

For example,

$$Pr(\text{word after "dog"}): \begin{cases} Pr(\text{"runs"} \mid \text{"dog"}) = 0.4 \\ Pr(\text{"eats"} \mid \text{"dog"}) = 0.2 \\ Pr(\text{"sighs"} \mid \text{"dog"}) = 0.05 \\ Pr(\text{"dracula"} \mid \text{"dog"}) = 0.001 \\ \vdots \end{cases}$$

More formally, the choice of the word x_t can be written as

$$x_t = \arg\max_p pmf(x_{t-1}) := f(x_{t-1})$$

Our prediction for the next word in the sentence (in an order-1 model) is the one with the largest probability.

Markov chains

Example: Generating text word-by-word via a Markov chain.

In this example we generate a Markov chain model of text using the classic novel War of the Worlds by H.G. Wells to define our transition probabilities. we consider models with increasing order $W = 1$, $W = 2, \ldots$

Using an **order** W **model** we pick a word randomly from the text, and start generating the following text using a Markov chain model.
We will display a chunk of 30 words from the text following our initial input, followed by the result of the Markov model.
In the result, the input words $x_1 = \gamma_1, \ldots, x_W = \gamma_W$ is colored red, and the 30 words generated using it are colored blue.

Markov chains

```
-------- TRUE TEXT -------
had done for countless years as though no planet mars existed in the sky even at woking station and h
orsell and chobham that was the case in woking junction until late


-------- ORDER = 1 MODEL TEXT -------
had been the martians were the martians were the martians were the martians were the martians were th
e martians were the martians were the martians were the martians were the martians were the martians
```

The Markov model of order 1 does not generate anything meaningful.

```
-------- TRUE TEXT -------
trucks bearing huge guns and carriages crammed with soldiers these were the guns that were brought up
from woolwich and chatham to cover kingston there was an exchange of pleasantries you ll


-------- ORDER = 2 MODEL TEXT -------
trucks bearing huge guns and such morbidities never enter the scheme of their houses astonished how a
re all things made for the most part the daily telegraph for instance in the darkness
```

As we increase the order to $W = 2$, the generated sentence starts to make more sense, matching the following 3 words of the original text.

Markov chains

As we increase the order of the model the generated text will begin to match the original more and more. By the time we reach the order $W = 10$ the text generated by the model is identical to the original.

```
-------- TRUE TEXT -------
we hurried along and the artilleryman talked in whispers and looked now and again over our shoulders
once or twice we stopped to listen after time we drew near the road and as we did so we heard the cla
tter
```

```
-------- ORDER = 10 MODEL TEXT -------
we hurried along and the artilleryman talked in whispers and looked now and again over our shoulders
once or twice we stopped to listen after time we drew near the road and as we did so we heard the cla
tter
```

When we increase the order enough, there remains only a single exemplar in the text to construct each associated histogram, that is, every input sequence used to determine the transition probabilities is unique.

Fixed order and limited memory

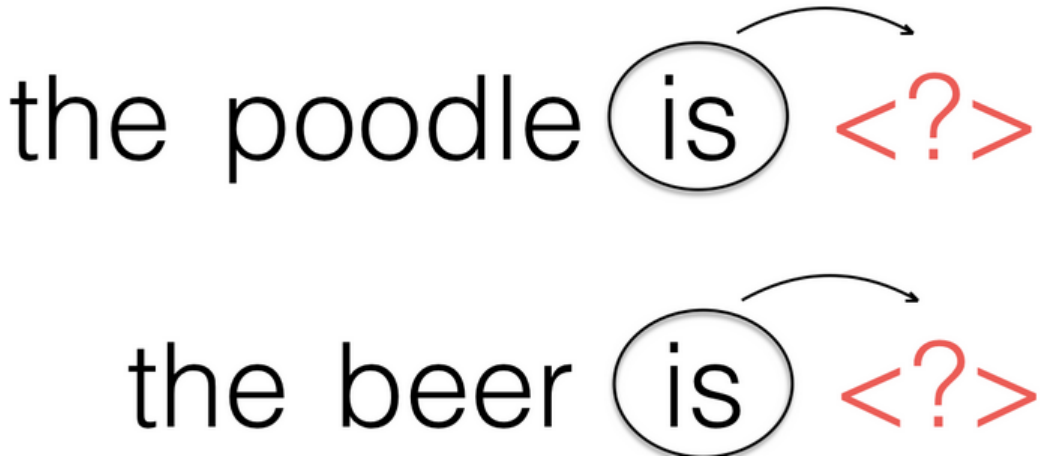
- Markov chains have **fixed order**.
- An order W system with generic update steps

$$x_t = f(x_{t-1}, \ldots, x_{t-W})$$

has the property that the update for any x_t is only on dependent x_{t-1} through x_{t-W} and no point coming before x_{t-W}. So the range of values used to build each subsequent step is **limited by the order of the system** and **cannot use any information from earlier in a sequence**.

Fixed order and limited memory

The main shortcoming of fixed order systems is exemplified in this toy example.



Let us consider an order-1 model whose transition probabilities have been determined on a large training corpus. Here we use our order-1 to predict the next word of each sentence, following the word "is".
However since the model is order-1 the same word will be predicted for each sentence. Given the different context of each, this will likely mean that at least one of the sentences will not make sense.

Analysis of data with temporal dependencies

Definition. A **dynamical system with variable order** is defined formally as recurrence equation of the form

$$\begin{aligned} h_1 &= \gamma_1, \\ h_t &= f(h_{t-1}, x_t), \quad t > 1. \end{aligned}$$

The variable h_t is often referred to as the **state variable**. It provides a sort of summary of the corresponding input sequence at each step of the system

Note that, in this formulation, the order of the dynamical system increases from iteration-to-iteration.

Dynamical system with variable order

Example: the exponential average

The exponential average is another smoothing technique applied to time series data as a preprocessing step to remove high-frequency oscillations.

Instead of taking a sliding window and averaging the input series inside of it, we compute the average of the entire input sequence in an online fashion, adding the contribution of each input one element at-a-time.

To do this we form an average of the first two points x_1 and x_2; we next take this result and make a weighted combination of it and the third point x_3 giving an average of the first three points. We continue in this fashion until the final element of the sequence is reached.

Dynamical system with variable order

We could write down this running average as

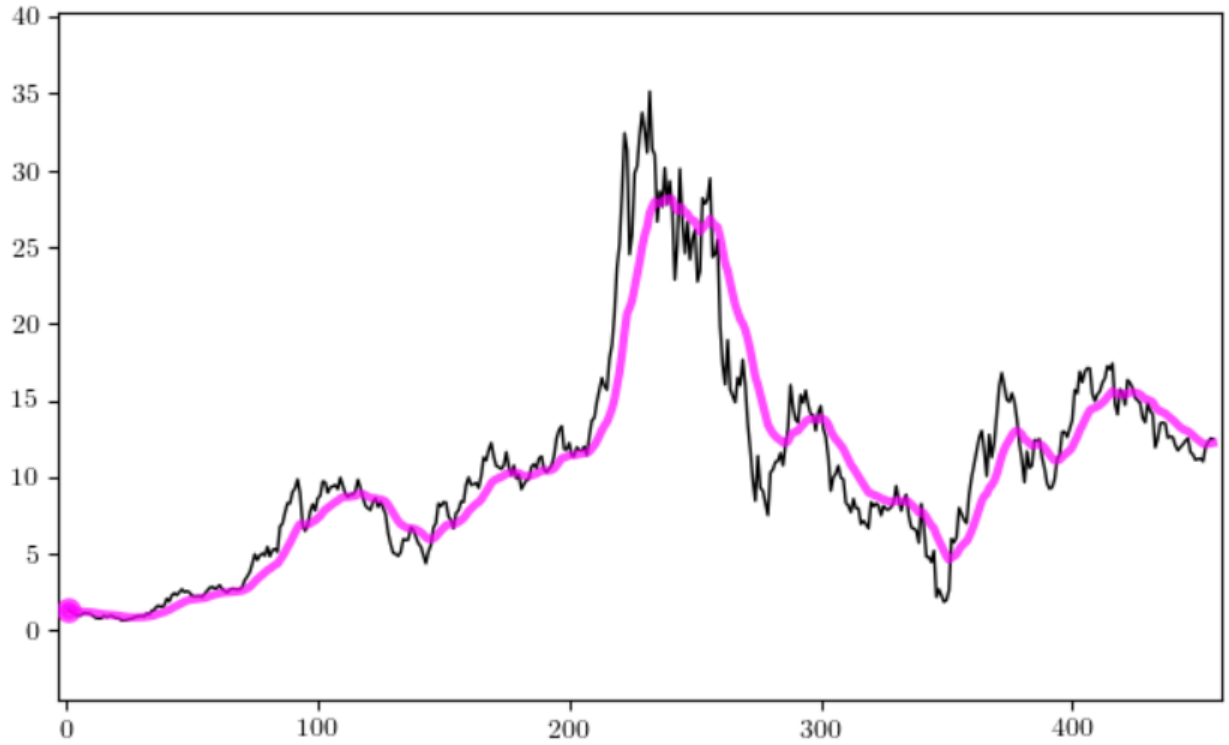
$$\begin{aligned} h_1 &= x_1 \\ h_2 &= \frac{x_1+x_2}{2} \\ h_3 &= \frac{x_1+x_2+x_3}{3} \\ \vdots & \quad \vdots \end{aligned}$$

Notice how at each step here the average computation h_t summarizes the input points via a simple summary statistic, i.e., their sample mean.

Also note that this dynamic system does not have a fixed order: at each step the order of the dynamic system increases by 1.

Dynamical system with variable order

Illustration of a a time series with the resulting exponential average shown as a pink curve.



Dynamical system with variable order

We can write this running average more efficiently as

$$h_2 = \frac{x_1+x_2}{2} = \frac{h_1+x_2}{2} = \frac{h_1}{2} + \frac{x_2}{2}$$

$$h_3 = \frac{x_1+x_2+x_3}{3} = \frac{2\frac{x_1+x_2}{2}+x_3}{3} = \frac{2h_2+x_3}{3} = \frac{2h_2}{3} + \frac{x_3}{3}$$

In general we have that

$$h_t = \frac{(t-1)h_{t-1}}{t} + \frac{x_t}{t}$$

From a computational perspective, this representation is far more **memory efficient** since at the t step we only need to store and deal with two values as opposed to t values.

Dynamical system with variable order

Using the same initial value $h_1 = x_1$ and taking a value $\alpha \in [0, 1]$ we can define a running **exponential average** via the formula

$$\begin{aligned} h_1 &= x_1 \\ h_t &= \alpha h_{t-1} + (1-\alpha)x_t, \quad t > 1. \end{aligned}$$

Why is this recurrent relation called an exponential average?

$$\begin{aligned} h_t &= \alpha h_{t-1} + (1-\alpha)x_t \\ &= \alpha(\alpha h_{t-2} + (1-\alpha)x_{t-1}) + (1-\alpha)x_t \\ &= \alpha^2 h_{t-2} + \alpha(1-\alpha)x_{t-1} + (1-\alpha)x_t \\ &= \dots \\ &= \alpha^t x_1 + \alpha^{t-1}(1-\alpha)x_2 + \cdots + (1-\alpha)x_t \end{aligned}$$

This shows that h_t summarizes the inputs x_1 through x_t via its **exponential mean**.

The exponential average is a dynamic system whose order changes at each step; it increases by 1 element at each step.

Dynamical system with variable order

Other examples of dynamical system with variable order are the following:

Example: The running sum.

$$\begin{aligned} h_1 &= x_1 \\ h_t &= h_{t-1} + x_t, \quad t > 1. \end{aligned}$$

Example: The running product.

$$\begin{aligned} h_1 &= x_1 \\ h_t &= h_{t-1} x_t, \quad t > 1. \end{aligned}$$

Example: Historical maximum.

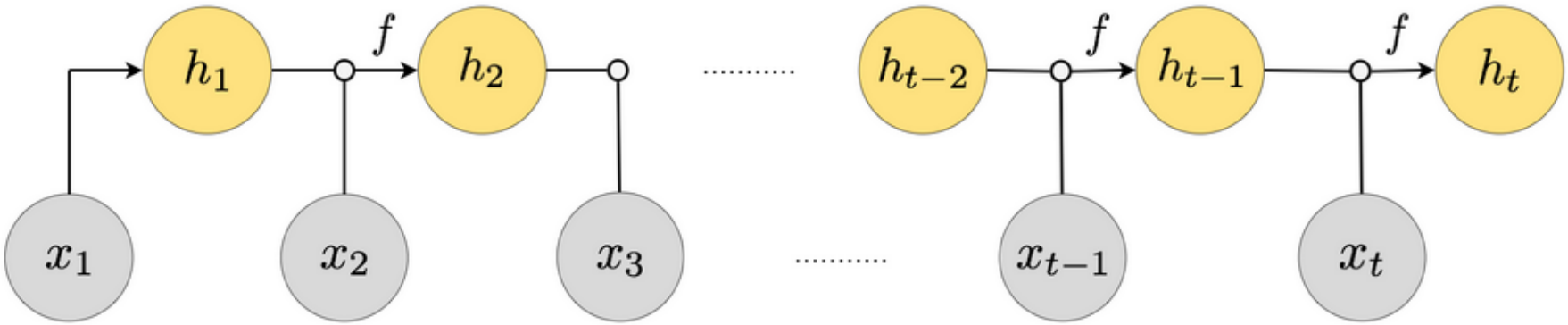
$$\begin{aligned} h_1 &= x_1 \\ h_t &= \max(h_{t-1}, x_t), \quad t > 1. \end{aligned}$$

Dynamical system with variable order

In the **graphical model representations of the fixed order dynamic system**, when we 'roll back' the recursion we see - in the end - that every point in the system depends entirely on the system's initial condition(s).

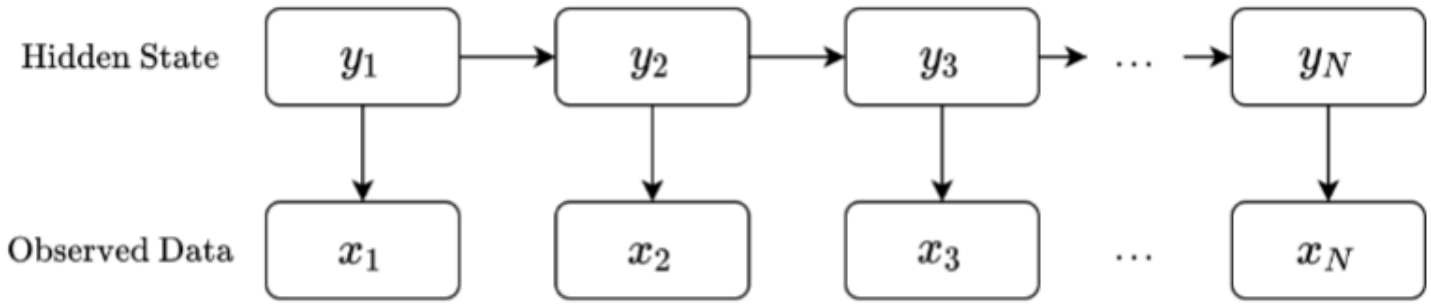
Dynamical system with variable order

In the **graphical model representations of the variable order dynamic system**, when we 'roll back' the recursion we see that every preceding input plays a role in the value of its next step, and is embedded in the system.



Dynamical system with variable order

Hidden Markov Models. They use a hidden state to represent higher level information about sequence:



Example: we model a sequence of temperature measurements and attempt to encode information about season into hidden state.

Limitations:

- Updates between hidden states generally have to be linear
- Markov assumption: no long term dependencies possible

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) offer several advantages with respect to classical methods:

- Non-linear hidden state updates allow for high representational power.
- They can represent long term dependencies in hidden state (theoretically).
- Shared weights can be used on sequences of arbitrary length

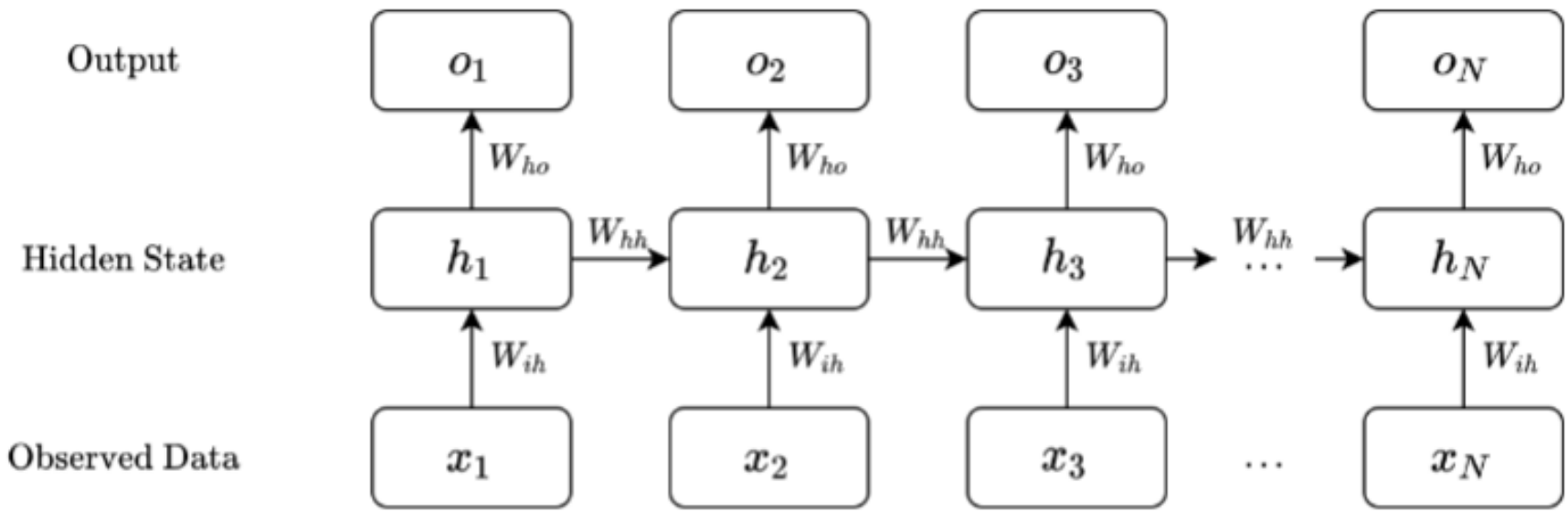
Historica note.

• The first recurrent neural network was propsed by Shunichi Amari in 1972.

• The popularity and success of these types of networks started with the **Long short-term memory** (LSTM) networks invented by Hochreiter and Schmidhuber in 1997.

Recurrent Neural Networks (RNNs)

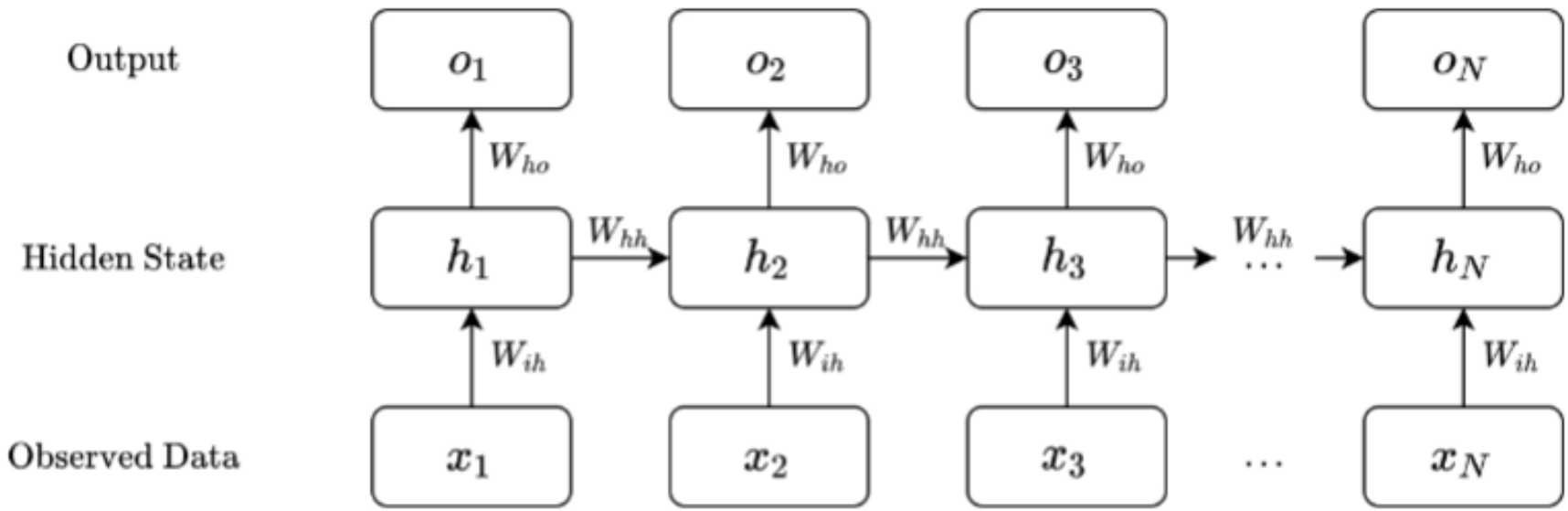
In contrast to the feedforward neural network, it is a **bi-directional artificial neural network**, meaning that it allows the output from some nodes to affect subsequent input to the same nodes.
They have the ability to use an **internal state** or **memory** to process arbitrary sequences of input.



Scheme of a Simple Recurent Network (SRN)

Recurrent Neural Networks (RNNs)

Scheme of a **Simple Recurrent Network** (SRN), also called Jordan Network [Jordan, 1997] (similar to the Elman Network, 1990)



$$h_t = \sigma\left(W_{ih}\, x_t + W_{hh}\, h_{t-1} + b_{ih} + b_{hh}\right)$$
$$o_t = softmax(W_{ho}\, h_t + b_{ho})$$

In the original implementation, $\sigma = \tanh$.

Recurrent Neural Networks (RNNs)

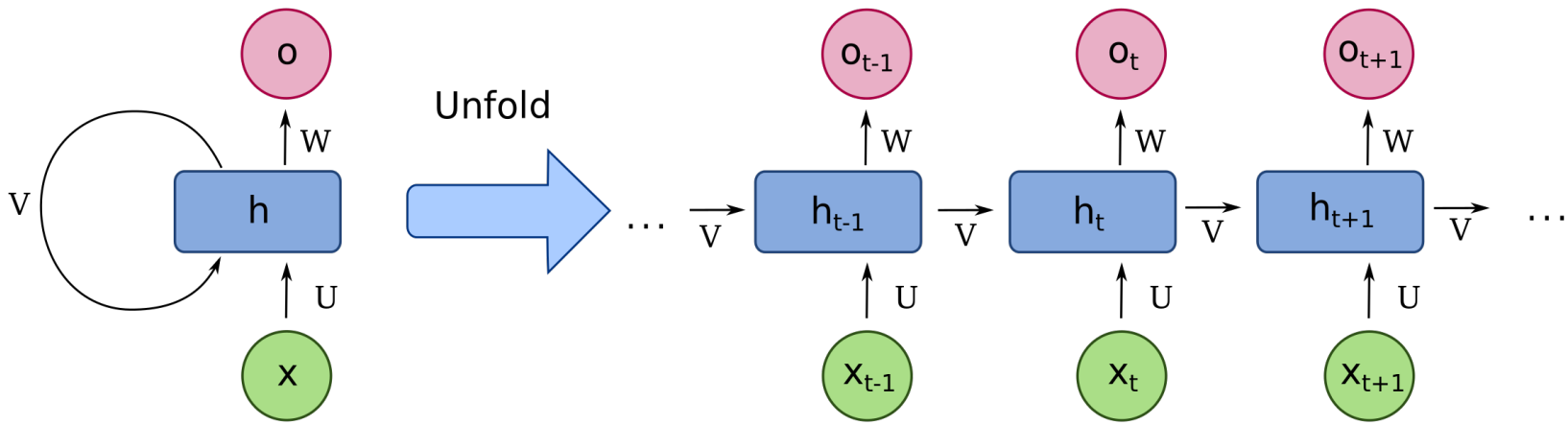
$$
\begin{aligned}
h_t &= \sigma\left(W_{ih}\, x_t + W_{hh}\, h_{t-1} + b_{ih} + b_{hh}\right) \\
o_t &= softmax(W_{ho}\, h_t + b_{ho})
\end{aligned}
$$

RNN learns weights and biases through training using back propagation.

These weights decide the importance of hidden state of previous timestamp and the importance of the current input. Essentially, they decide how much value from the hidden state and the current input should be used to generate the current input. The activation function σ adds non-linearity to RNN, thus simplifying the calculation of gradients for performing back propagation

Recurrent Neural Networks (RNNs)

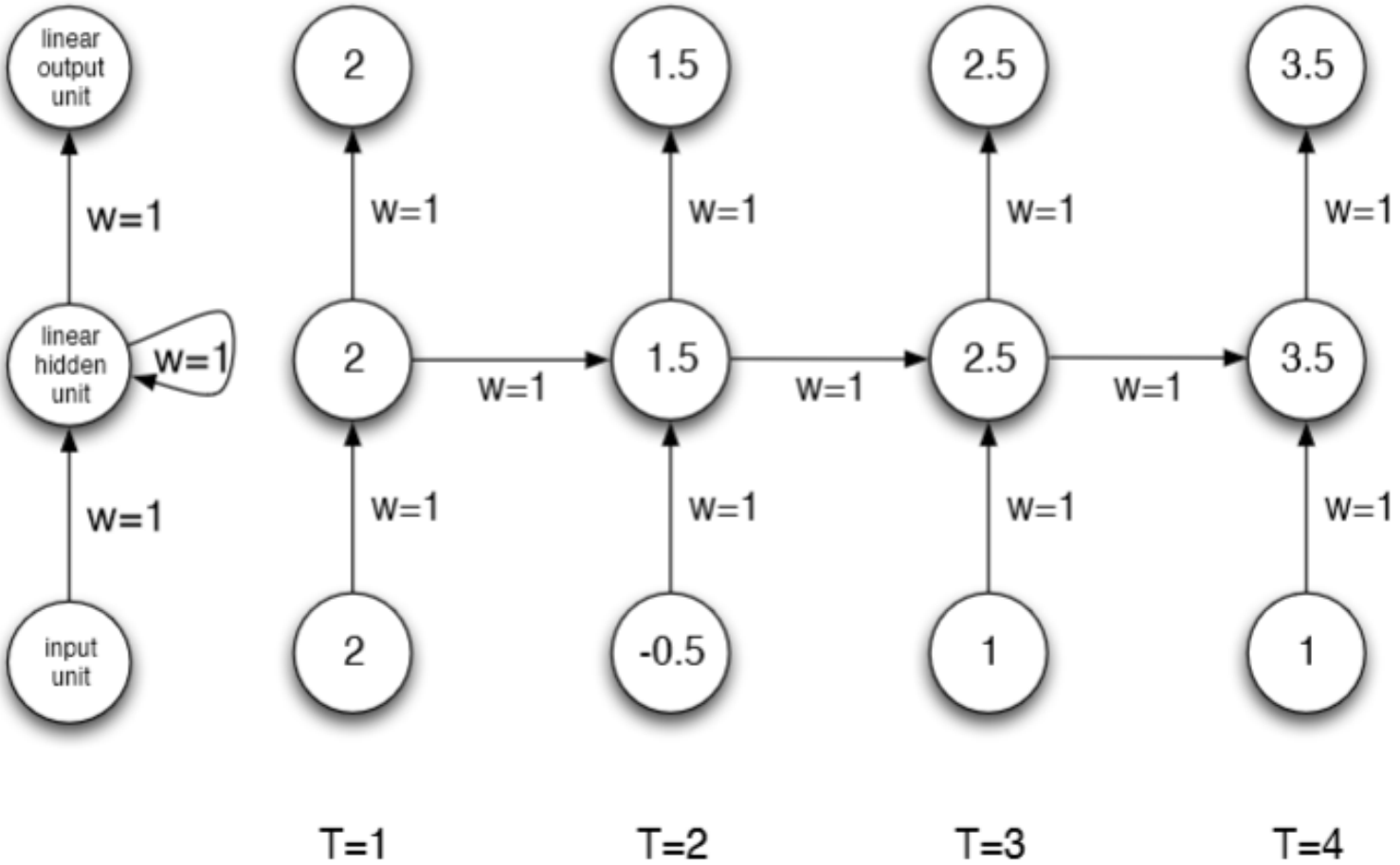
Remark. The SNR illustration may be misleading because what appears to be layers are, in fact, **different steps in time of the same fully recurrent neural network**. The drawing unfolds the time steps to create the appearance of a multi-layer architecture.



The left-most item in the illustration above shows the recurrent connections as the arc labeled 'v'. It is "unfolded" in time to produce the appearance of layers.

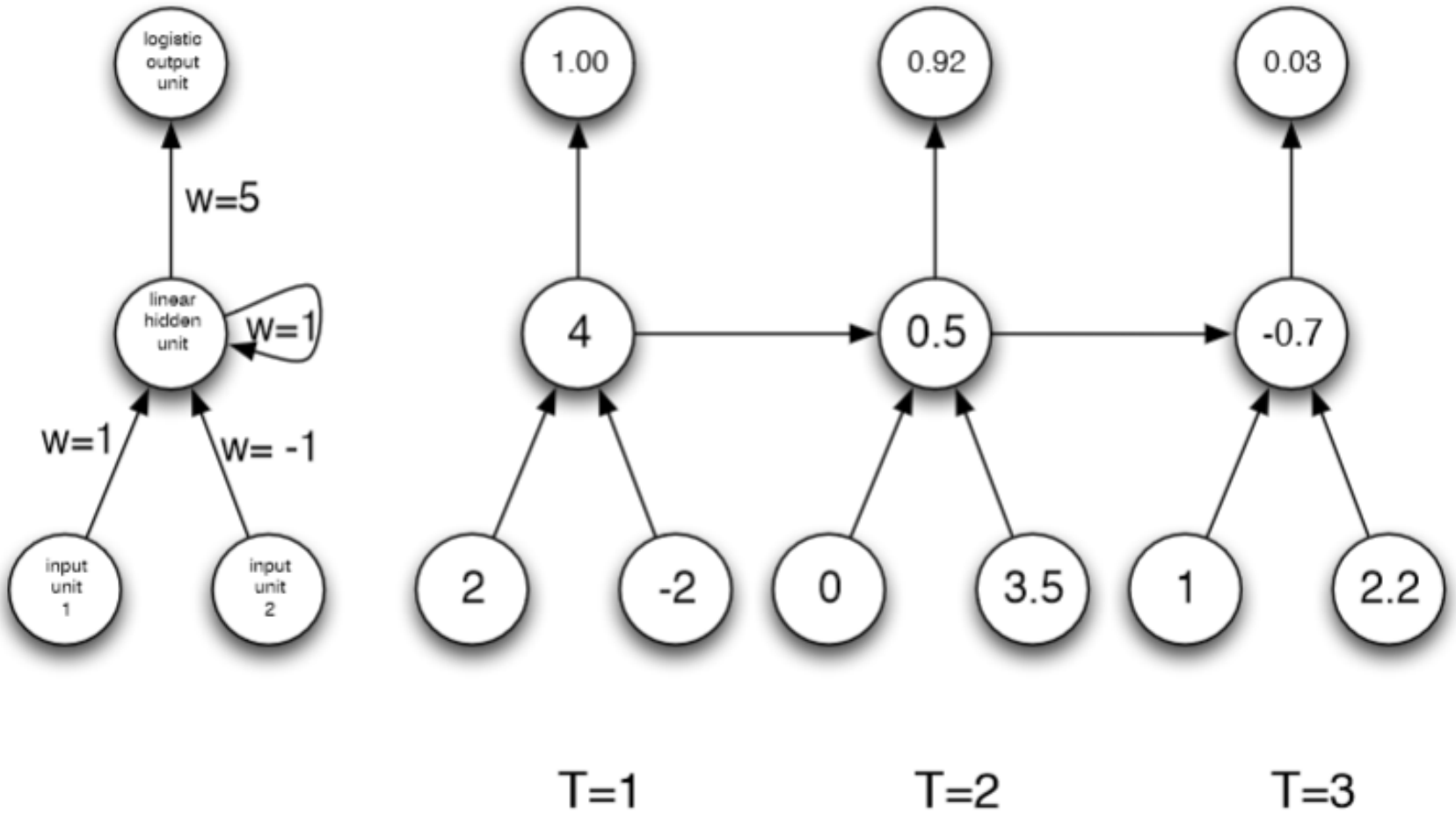
Recurrent Neural Networks (RNNs)

Example. This SRN sums its inputs.



Recurrent Neural Networks (RNNs)

Example. This SRN determines if the aggregated values of the first or second input are larger.



Recurrent Neural Networks (RNNs)

Example: Parity Check

Assume we have a sequence of binary inputs. We want to determine the parity incrementally i.e., to keep track whether the number of 1's is even or odd.

Parity bits:	0	1	1	0	1	1	...			
Input:	0	1	0	1	1	0	1	0	1	1

Note that each parity bit is the **XOR of the input and the previous parity bit**

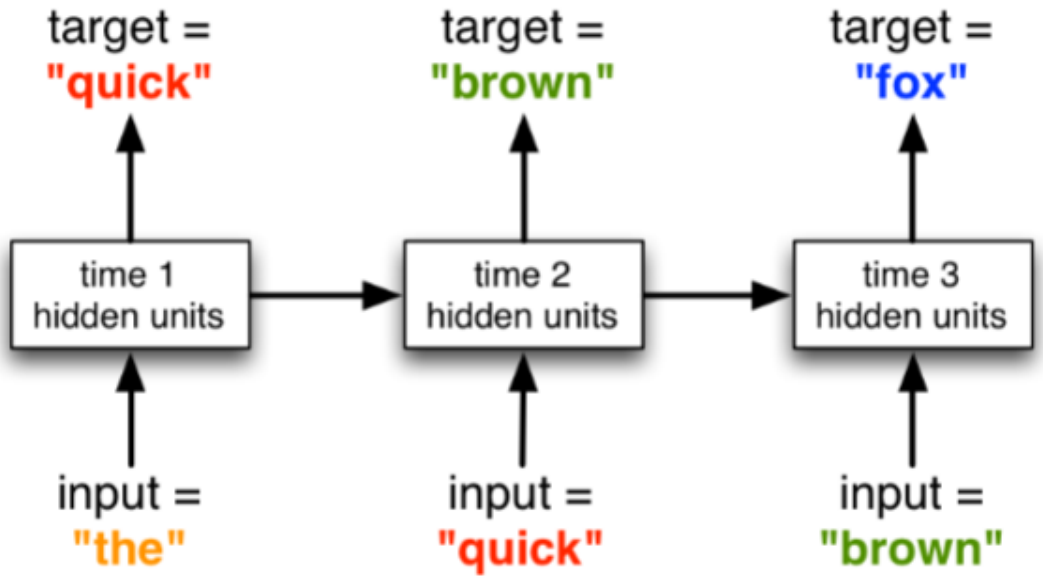
$$h_t = XOR(h_{t-1}, x_t)$$

Parity is a classic example of a problem that is hard to solve with a shallow feed-forward net, but easy to solve with an RNN.

Recurrent Neural Networks (RNNs)

Example: Language Modeling

RNNs are used for language models.



Each word is represented as an indicator vector and the model predicts a distribution that we can train it with cross-entropy loss. This model can learn long-distance dependencies.

Recurrent Neural Networks (RNNs)

RNN Limitations.

Simple Recurrent Networks suffers from a major drawback, known as the **vanishing gradient problem**, which prevents it from high accuracy.

As the context length increases, layers in the unrolled RNN also increase. Consequently, the network becomes deeper and the gradients flowing back in the back propagation step becomes smaller.

As a result, the learning rate becomes really slow and makes it **infeasible to expect long-term dependencies of the language**.

In other words, RNNs experience difficulty in memorizing previous words very far away in the sequence and can only make predictions based on the most recent words.

Recurrent Neural Networks (RNNs)

More sophisticated types of RNNs have been developed to deal with this shortcoming of the standard RNN model.

- **Bidirectional RNNs** are composed of 2 RNNs stacking on top of each other. The output is then composed based on the hidden state of both RNNs. The idea is that the output may not only depend on previous elements in the sequence but also on future elements.
- **Long Short-Term Memory Networks (LSTM)**. They inherit the architecture from standard RNNs, with the exception of the hidden state. The memory in LSTMs (called cells) take as input the previous state and the current input. Internally, these cells decide what to keep in and what to eliminate from the memory. Next, they combine the previous state, the current memory, and the input. This process efficiently solves the vanishing gradient problem.

Recurrent Neural Networks (RNNs)

- **Gated Recurrent Unit Networks** extends LSTM with a gating network generating signals that act to control how the present input and previous memory work to update the current activation, and thereby the current network state. Gates are themselves weighted and are selectively updated according to an algorithm.
- **Neural Turing Machines** extend the capabilities of standard RNNs by coupling them to external memory resources, which they can interact with through attention processes. The analogy is that of Alan Turing's enrichment of finite-state machines by an infinite memory tape.

Long short-term memory (LSTM)

Long short-term memory (LSTM) [Hochreiter, Schmidhuber 1995] is a deep learning system that was introduced to avoid the vanishing gradient problem of RNN.

Motivationi. In theory, (classic) RNNs can keep track of arbitrary long-term dependencies in the input sequences. However, when training a classic RNN using back-propagation, the long-term gradients which are back-propagated can vanish or explode because of the computations involved in the process.

The intuition behind the LSTM architecture is to create an additional module in a neural network that learns when to remember and when to forget pertinent information. That is, the network effectively learns which information might be needed later on in a sequence and when that information is no longer needed. This creates a short-term memory that can last thousands of timesteps, hence the name "long short-term memory".

Long short-term memory (LSTM)

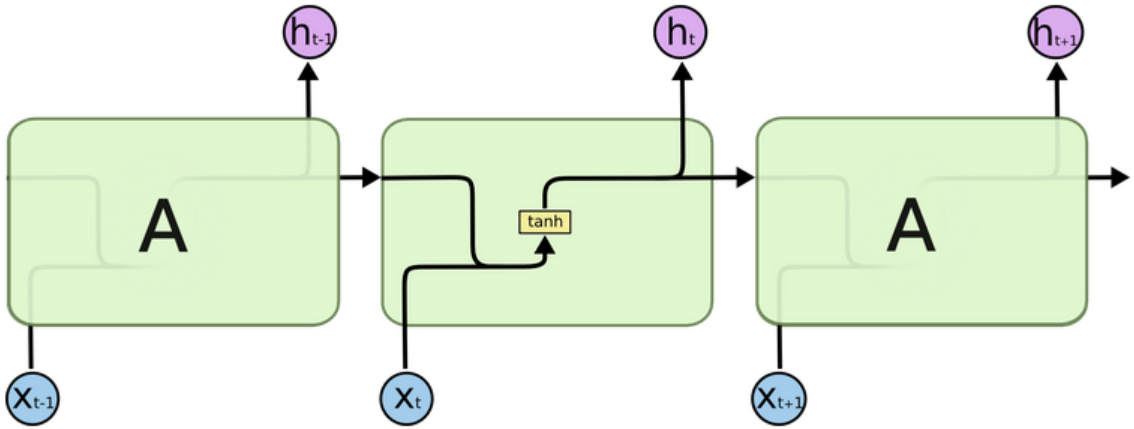
A common LSTM unit is composed of a cell and 3 gates regulating the flow of information into and out of the cell.

1. **The cell** remembers values over arbitrary time intervals.
2. **Forget gates** decide what information to discard from a previous state by assigning a previous state, compared to a current input, a value between 0 and 1. A (rounded) value of 1 means to keep the information, and a value of 0 means to discard it. They prevent backpropagated errors from vanishing or exploding.
3. **Input gates** decide which pieces of new information to store in the current state, using the same system as forget gates.
4. **Output gates** control which pieces of information in the current state to output by assigning a value from 0 to 1 to the information, considering the previous and current states.

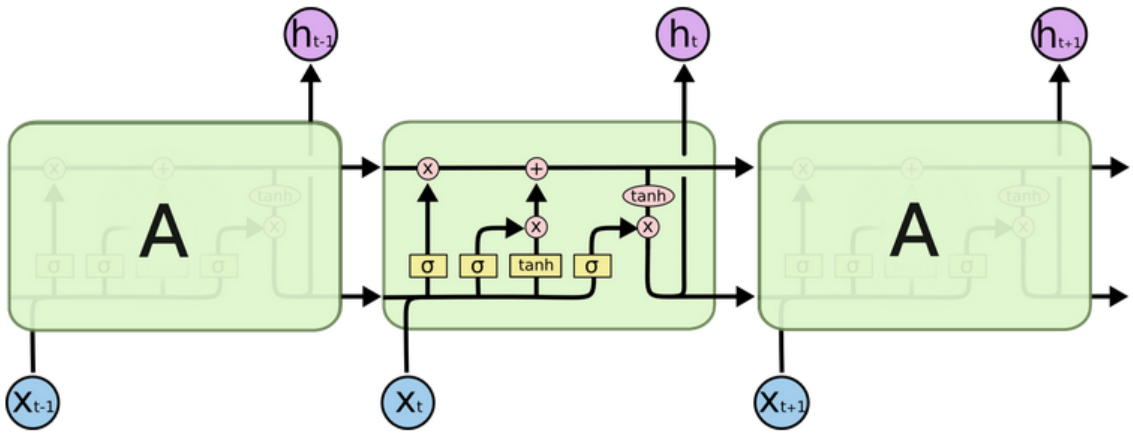
Selectively outputting relevant information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions, both in current and future time-steps.

LSTM Architecture

RNNs have the form of a chain of repeating modules of neural network

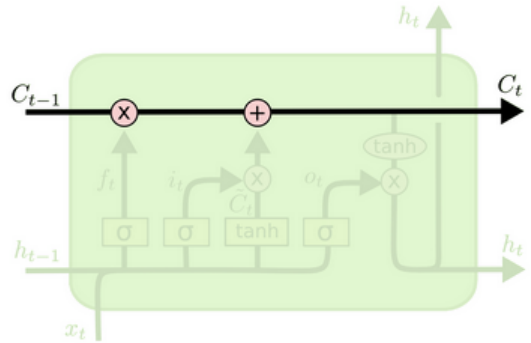


LSTMs also have this chain like structure, but the repeating module has a different structure



LSTM Architecture

The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.

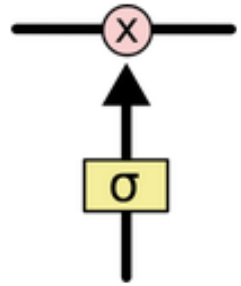


The cell state acts like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions.

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

LSTM Architecture

Gates 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.

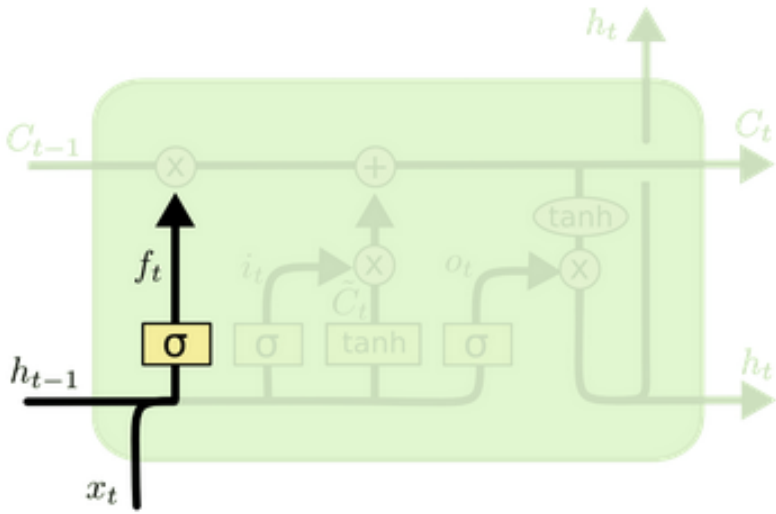
A value of zero means "let nothing through," while a value of one means "let everything through!"

As we indicated above, LSTM has three of these gates, to regulate the flow of information into and out of the cell.

LSTM Architecture

The first step in the LSTM is to decide what information we are 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}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

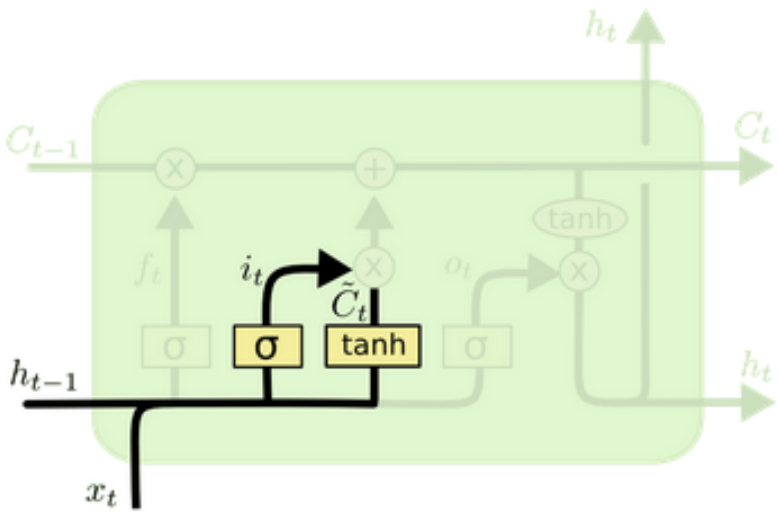


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM Architecture

The next step is to decide what new information we are going to store in the cell state. This has two parts.

1. First, a sigmoid layer called the **input gate layer** decides which values to update.
2. Next, a tanh layer creates a vector of new candidate values, $\widetilde{C}_t$, that could be added to the state.



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

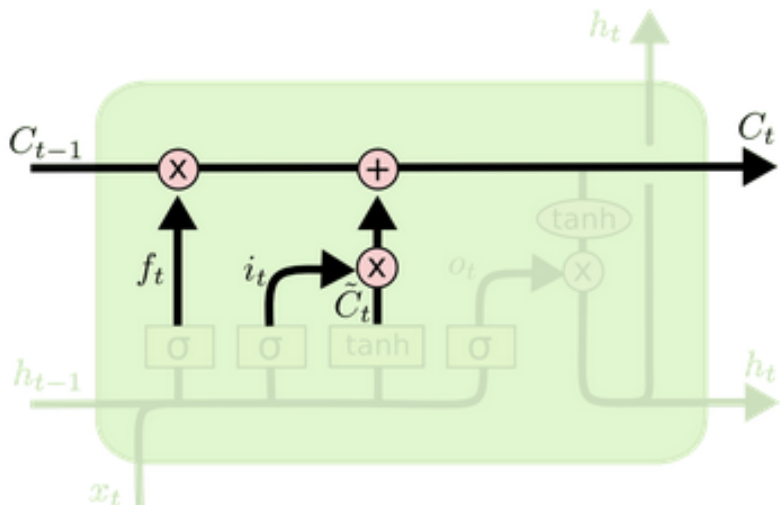
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

In the next step, we will combine these two vectors to create an update to the state.

LSTM Architecture

We now update the old cell state, C_{t-1}, into the new cell state C_t based on the information from the previous steps.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \widetilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

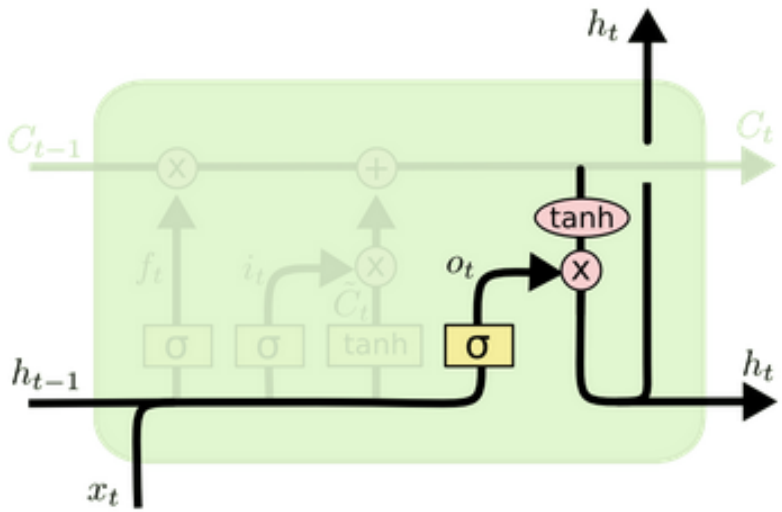


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM Architecture

Finally, we need to decide what we are going to output. This output will be based on our cell state, but will be a filtered version.

1. First, we run a sigmoid layer which decides what parts of the cell state we are going to output.
2. Next we put the cell state through tanh (to set 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.

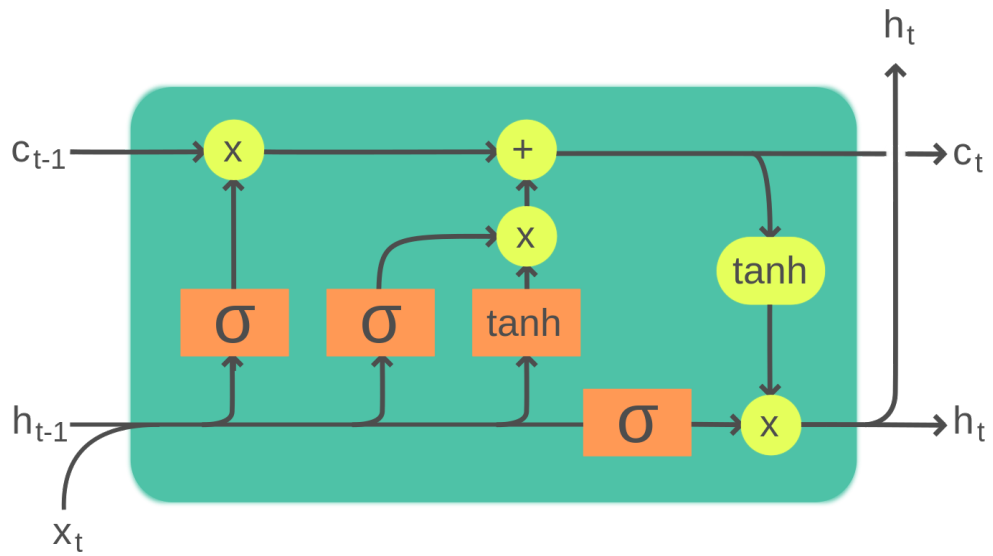


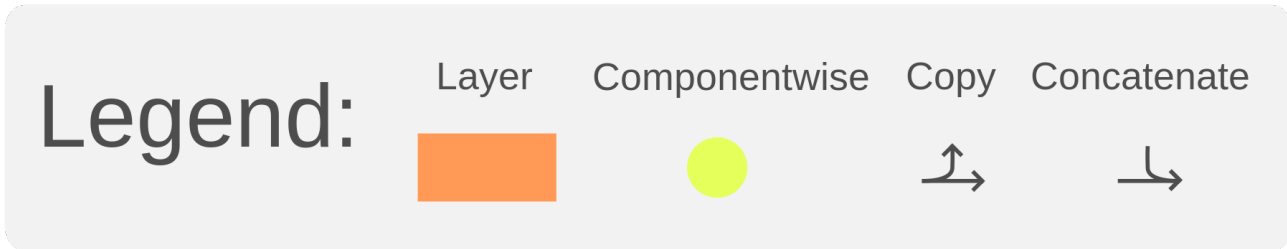
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

LSTM Architecture

All together, here is the LSTM cell with a forget gate.

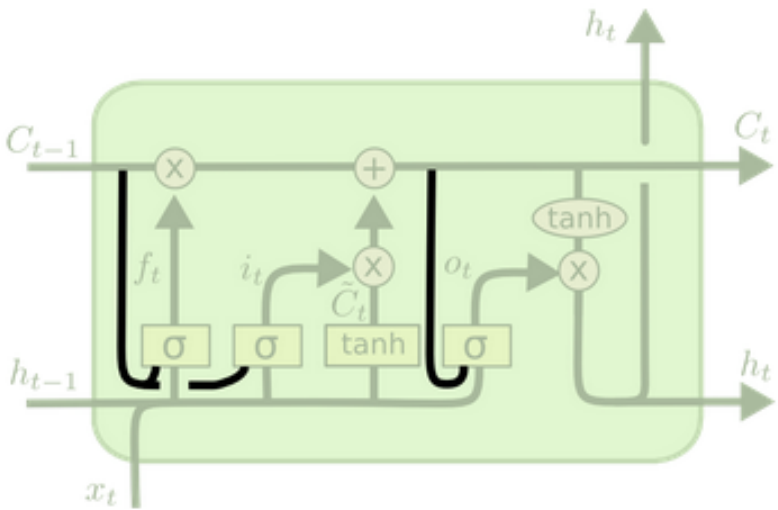




LSTM Architecture

What was described above is the basic LSTM cell but not all LSTMs are the same. In fact, there are many variants.

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding **peephole connections**.
This means that we let the gate layers look at the cell state.

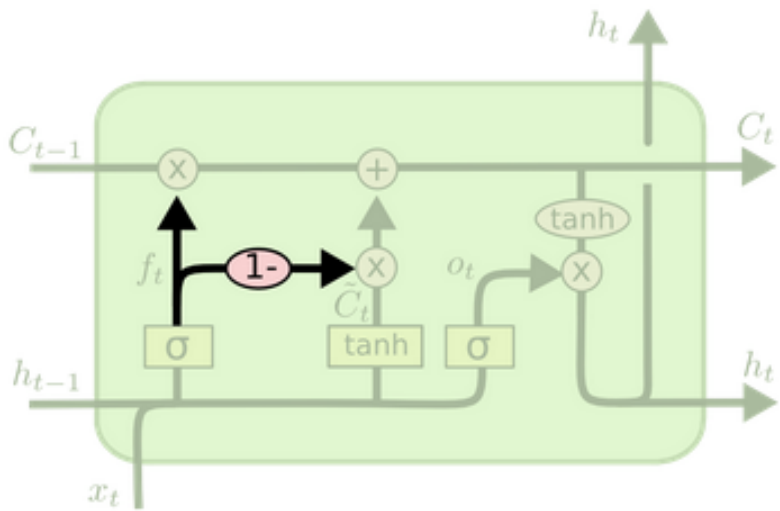


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

LSTM Architecture

Another variation is to use **coupled forget and input gates**.

Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we are going to input something in its place. We only input new values to the state when we forget something older.

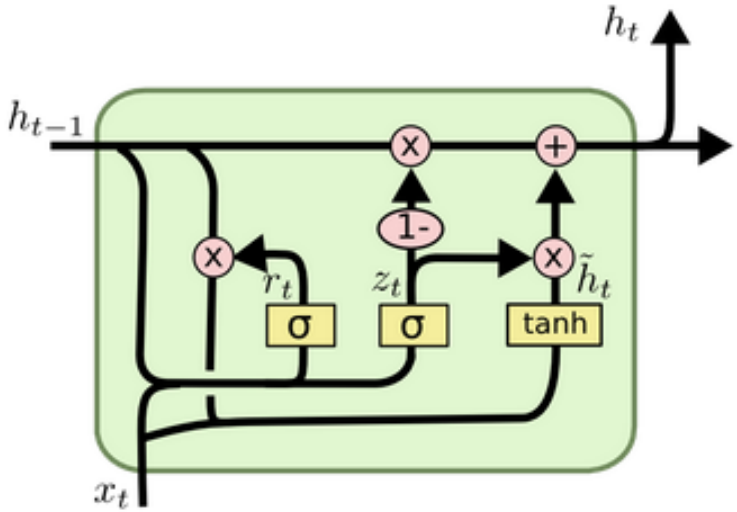


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

LSTM Architecture

Yet another variation on the LSTM is the **Gated Recurrent Unit**, or GRU, introduced by Cho, et al. (2014).

It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Long short-term memory (LSTM)

LSTMs were a big step in what can be accomplished with RNNs.

- 2009: A method based on LSTM won the ICDAR connected handwriting recognition competition [Graves et al, 2009].
- 2015: Google started using an LSTM for speech recognition on Google Voice.
- 2016: Google released the Google Neural Machine Translation system for Google Translate which used LSTMs.
- 2016: Amazon released Polly, which generates the voices behind Alexa, using a bidirectional LSTM for the text-to-speech technology.
- 2018: OpenAI used LSTM trained by policy gradients to beat humans in the complex video game of Dota 2.
- 2019: DeepMind used LSTM trained by policy gradients to excel at the complex video game of Starcraft II.

Recurrent Neural Networks (RNNs)

Examples of Generating Text with RNN Language Model

- **Machine Generated Political Speeches.** Here the author used RNN to generate hypothetical political speeches given by former president Barrack Obama. Taking in over 4.3 MB / 730,895 words of text written by Obama's speech writers as input, the model generates multiple versions with a wide range of topics including jobs, war on terrorism, democracy, China.
- **Machine Generated Harry Potter.** Here the author trained an LSTM Recurrent Neural Network on the first 4 Harry Potter books. Then he asked it to produce a chapter based on what it learned.
- **Machine Generated Seinfeld Scripts.** A cohort of comedy writers fed individual libraries of text (scripts of Seinfeld Season 3) into predictive keyboards for the main characters in the show. The result is a 3-page script with uncanny tone, rhetorical questions, stand-up jargons that match the rhythms and diction of the show.

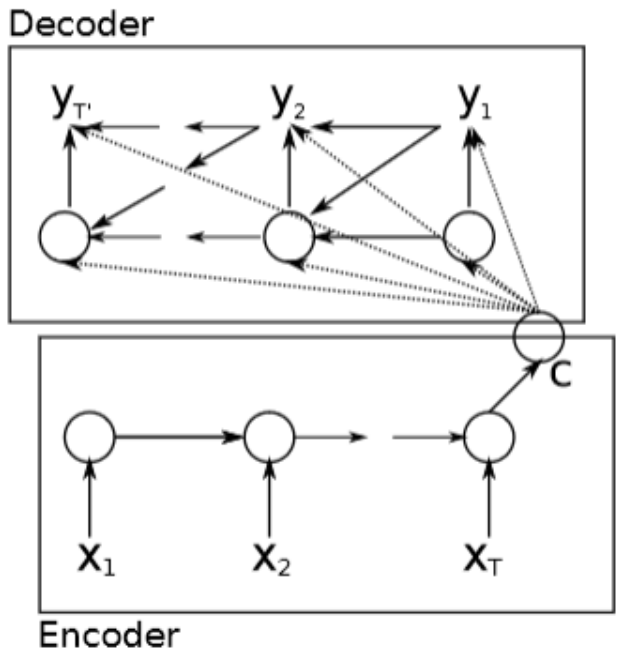
RNNs: Neural Machine Translation

The **Neural Machine Translation** (also **Neural MT** or **NMT**) is an approach for modeling language translation by predicting the likelihood of a sequence of words, typically modeling entire sentences in a single integrated model.

- End-to-end neural machine translation had their breakthrough with Kalchbrenner & Blunsom (2013) who introduced **Recurrent Continuous Translation Models** designed to map a sentence from the source language to a probability distribution over the sentences in the target language.
  - The generation of the translation employs a target Recurrent Language Model, whereas the conditioning on the source sentence is modelled with a Convolutional Sentence Model.
  - The convolutional sentence model transforms the source word representations into a representation for the source sentence.

RNNs: Neural Machine Translation

- Cho et al. (2014) and Sutskever et al. (2014) introduced a method based on **RNN Encoder–Decoder pair**.
  - An encoder network reads and encodes a source sentence into a fixed-length vector.
  - A decoder then outputs a translation from the encoded vector.
  - The whole encoder–decoder system is jointly trained to maximize the probability of a correct translation given a source sentence.



RNNs: Neural Machine Translation

- The original RNN Encoder–Decoder pair performed poorly on longer sentences. This problem was addressed when Bahdanau et al. (2014) introduced an **attention mechanism** to their encoder-decoder architecture.
  - The new architecture consists of a bidirectional RNN as an encoder and a decoder that emulates searching through a source sentence during decoding a translation.
  - At each decoding step, the state of the decoder is used to calculate a source representation that focuses on different parts of the source and uses that representation in the calculation of the probabilities for the next token.
  - The decoder (soft-)search for a set of input words, or their annotations computed by an encoder, when generating each target word. This frees the model from having to encode a whole source sentence into a fixed-length vector and lets the model focus only on information relevant to the generation of the next target word.

RNNs: Neural Machine Translation

- Based on the idea of encoder-decoder RNN-based architectures, Baidu launched the "first large-scale NMT system" in 2015, followed by Google in 2016 with the **Google's Neural Machine Translation System**.

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

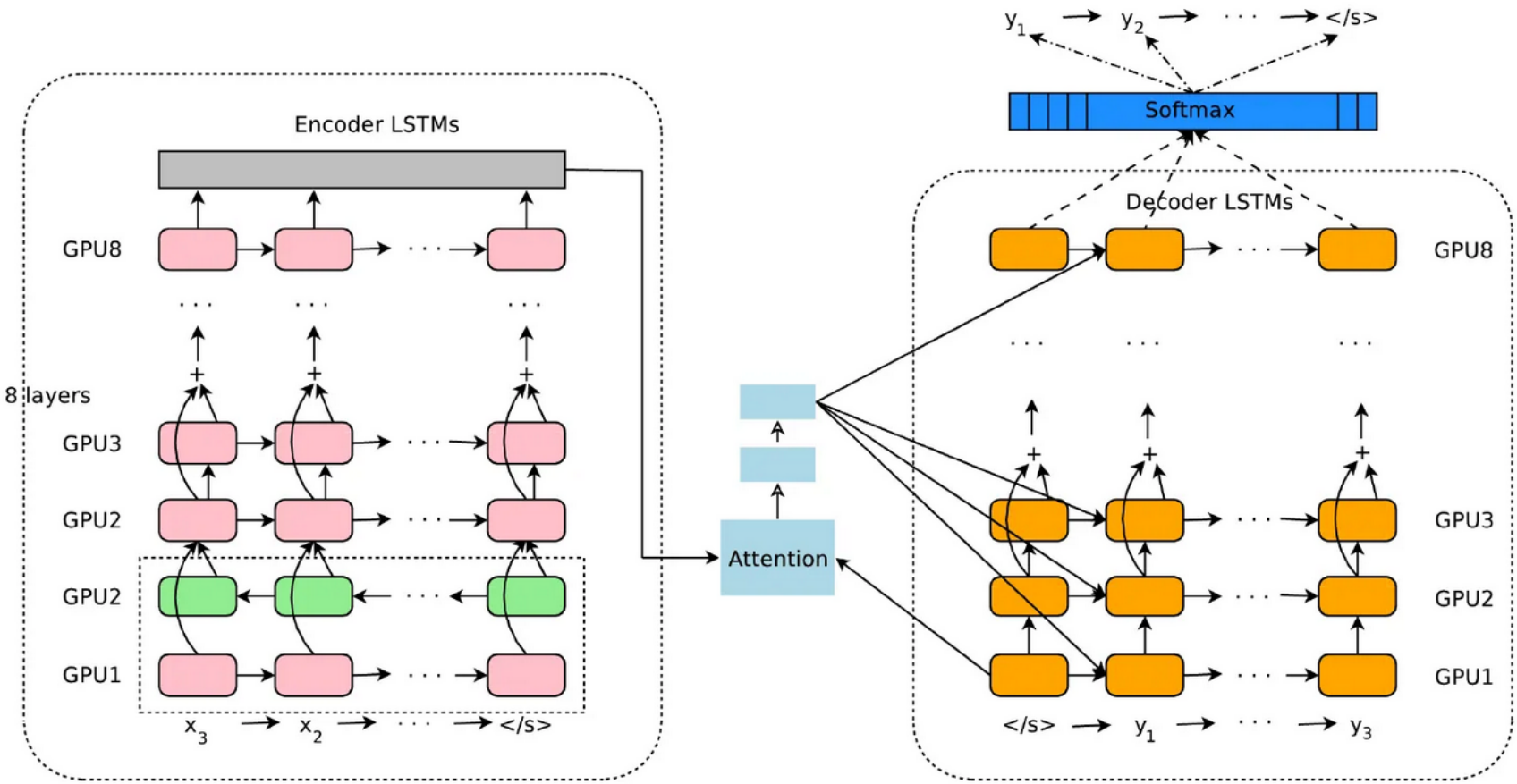
Mean of side-by-side scores on production data

PBST = phrase-based machine translation

- Starting 2016, neural models became the prevailing choice in the translation conference on Statistical Machine Translation.
- DeepL Translator, launched in 2017, uses a proprietary algorithm with CNNs.

Google's Neural Machine Translation (GNMT) System

The GNMT network architecture includes a encoder, an attention module and a decoder.



Recurrent Neural Networks (RNNs)

Besides Machine Translation, RNNs have shown great success in are other major Natural Language Processing tasks:

- **Sentiment Analysis**. An example is to classify Twitter tweets into positive and negative sentiments. The input would be a tweet of different lengths, and the output would be a fixed type and size.
- **Image Captioning**. Together with CNNs, RNNs have been used in models that can generate descriptions for unlabeled images. Given an image in need of textual descriptions, the output would be a series or sequence of words. While the input might be of a fixed size, the output can be of varying lengths.
- **Speech Recognition**. An example is that given an input sequence of electronic signals we can predict a sequence of phonetic segments together with their probabilities. Think applications such as SoundHound (voice-enabled digital assistant, music recognition) and Shazam (music identification from a short sample).