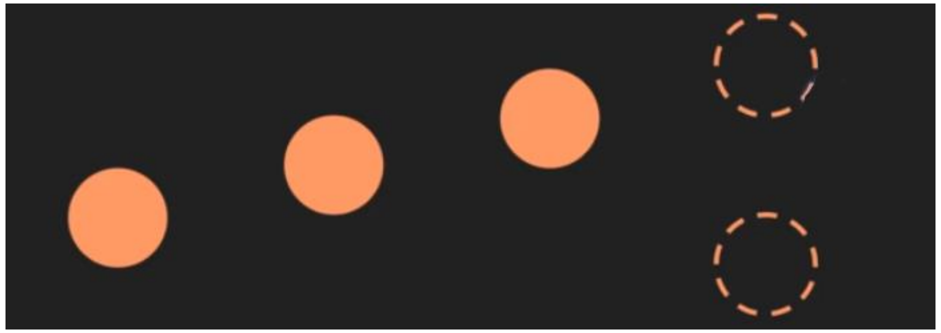
Deep Dive into RNN

Farid Afzali, Ph.D., P.Eng.

Sequence

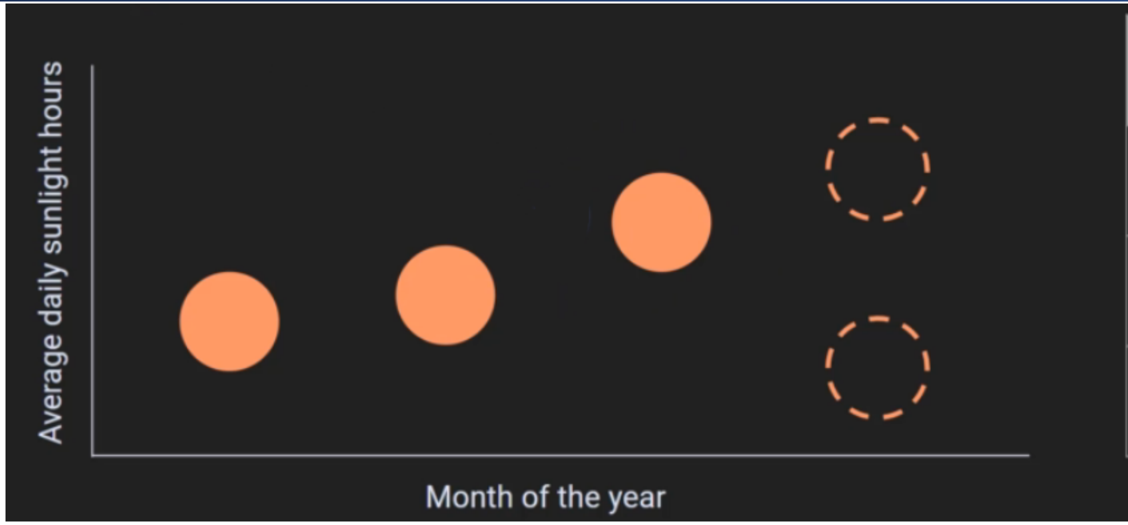
- ❑ A sequence in the context of Recurrent Neural Networks (RNNs) refers to a series of data points or inputs that are processed one at a time, in order, by the RNN model.
- ❑ Each data point in the sequence is typically associated with a time step, and the RNN processes these inputs sequentially, maintaining a hidden state that captures information from previous time steps.
- ❑ This allows RNNs to model sequential dependencies and patterns in data, making them particularly useful for tasks such as time series prediction, natural language processing, and speech recognition.

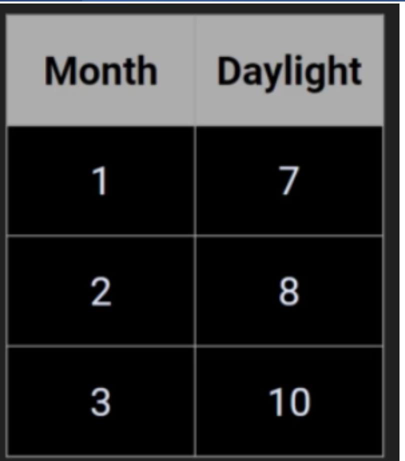


Sequence-Examples

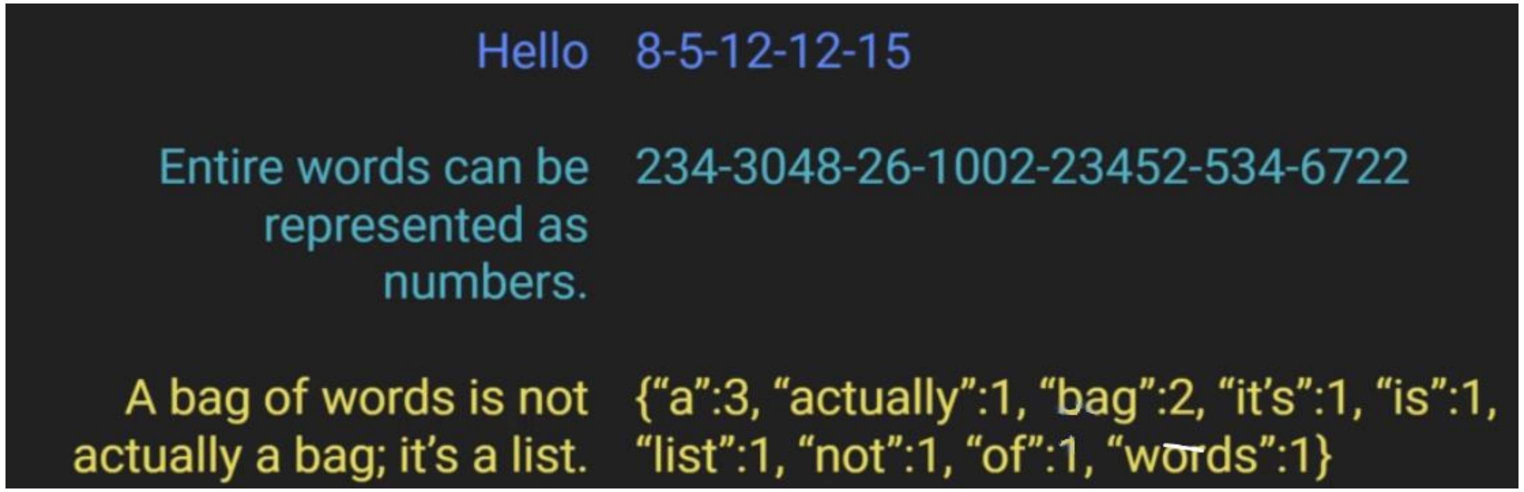
1. **Text Generation**: RNNs can be used to generate text, whether it's natural language text, code, or poetry. They can learn from existing text data and generate new text that is coherent and contextually relevant.
2. **Language Translation**: RNNs, especially variants like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), are used in machine translation systems, such as Google Translate. They can convert text from one language to another.
3. **Speech Recognition**: RNNs can be employed to convert spoken language into text. This is essential for applications like virtual assistants (e.g., Siri, Alexa) and transcription services.
4. **Time Series Forecasting**: RNNs can predict future values in a time series, making them valuable for financial forecasting, weather prediction, stock market analysis, and more.
5. **Handwriting Recognition**: RNNs can recognize and convert handwritten text or characters into digital text. This is used in digitizing handwritten forms and signature recognition.
6. **Video Analysis**: RNNs can be applied to video analysis tasks, such as action recognition in sports videos, tracking objects in surveillance videos, and generating video captions.
7. **Music Generation**: RNNs can generate music compositions based on patterns learned from existing music data. They are used in creating new melodies and harmonies.
8. **Sentiment Analysis**: RNNs can analyze text sentiment, determining whether a given piece of text is positive, negative, or neutral. This is used in social media monitoring and customer feedback analysis.

Sequence-Examples



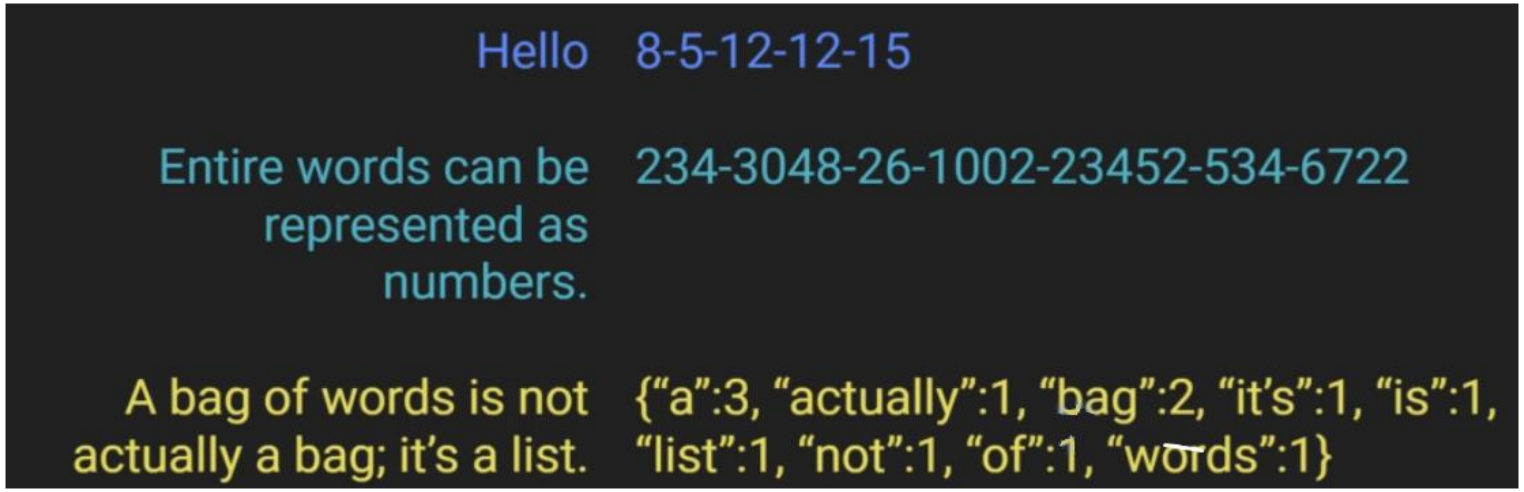


Month	Daylight
1	7
2	8
3	10

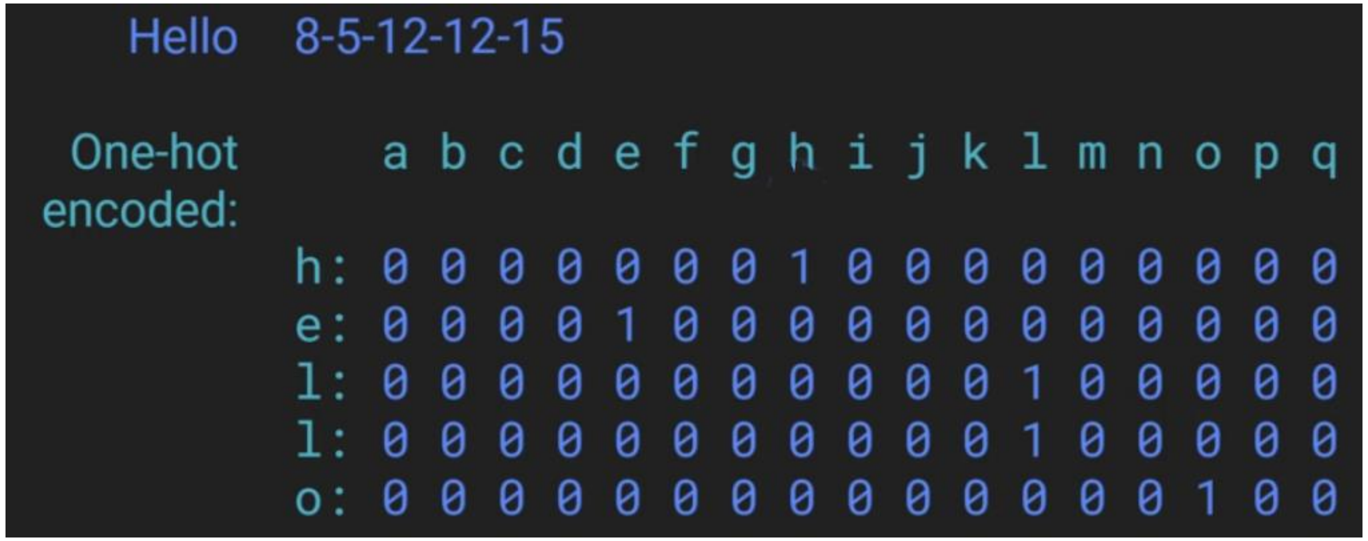


Hello	8-5-12-12-15
Entire words can be represented as numbers.	234-3048-26-1002-23452-534-6722
A bag of words is not actually a bag; it's a list.	{"a":3, "actually":1, "bag":2, "it's":1, "is":1, "list":1, "not":1, "of":1, "words":1}

Sequence-Examples



Hello	8-5-12-12-15
Entire words can be represented as numbers.	234-3048-26-1002-23452-534-6722
A bag of words is not actually a bag; it's a list.	{“a”:3, “actually”:1, “bag”:2, “it's”:1, “is”:1, “list”:1, “not”:1, “of”:1, “words”:1}



Hello 8-5-12-12-15

One-hot encoded:

```
    a b c d e f g h i j k l m n o p q
h:  0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
e:  0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
l:  0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
l:  0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
o:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

Limitations with previous models

For FFN architectures, the limitations listed are:

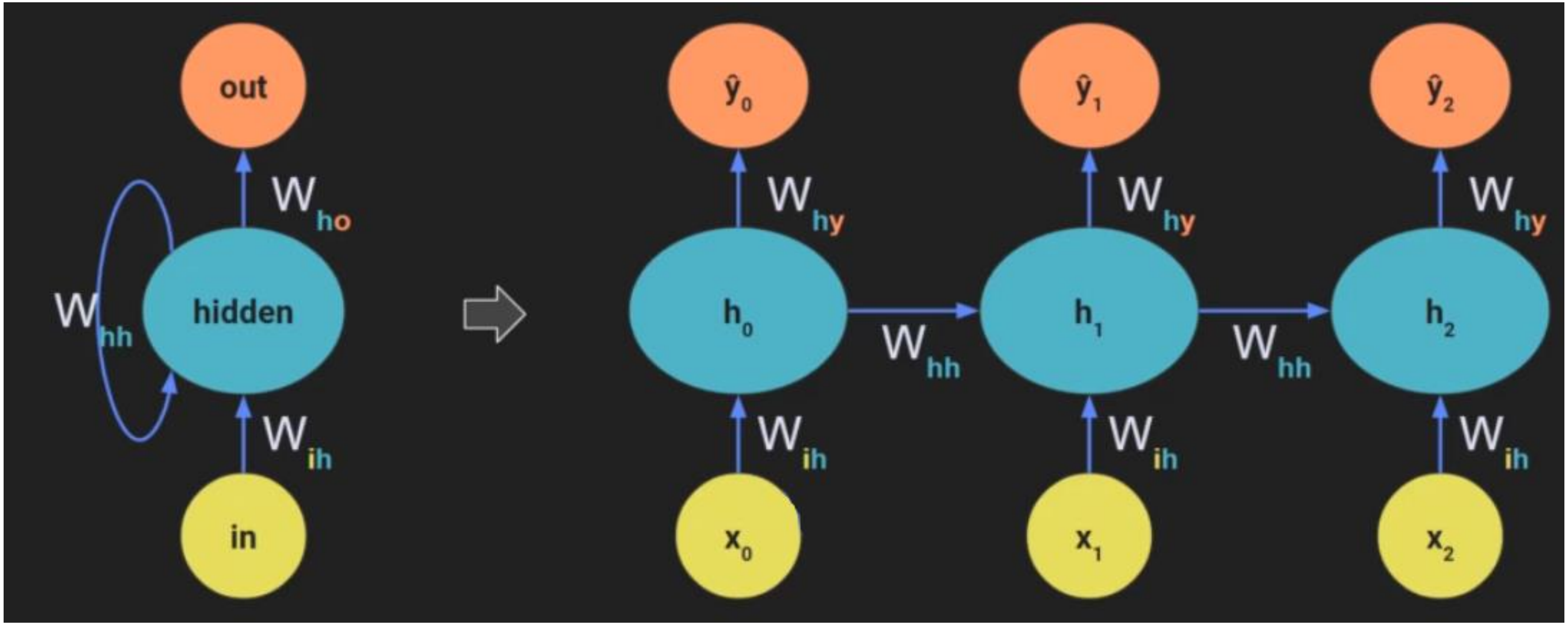
- ❑ **Ignores sequence information.** - FFNs do not take into account the order of the input, which is crucial for sequence processing.
- ❑ **Requires fixed sequence (input) length.** - FFNs cannot handle variable-length sequences without some form of padding or truncation.
- ❑ **Learned parameters not shared across the sequence.** - Each parameter in an FFN is used only once per input, unlike in architectures like RNNs where parameters are shared across different positions in the sequence.

Limitations with previous models

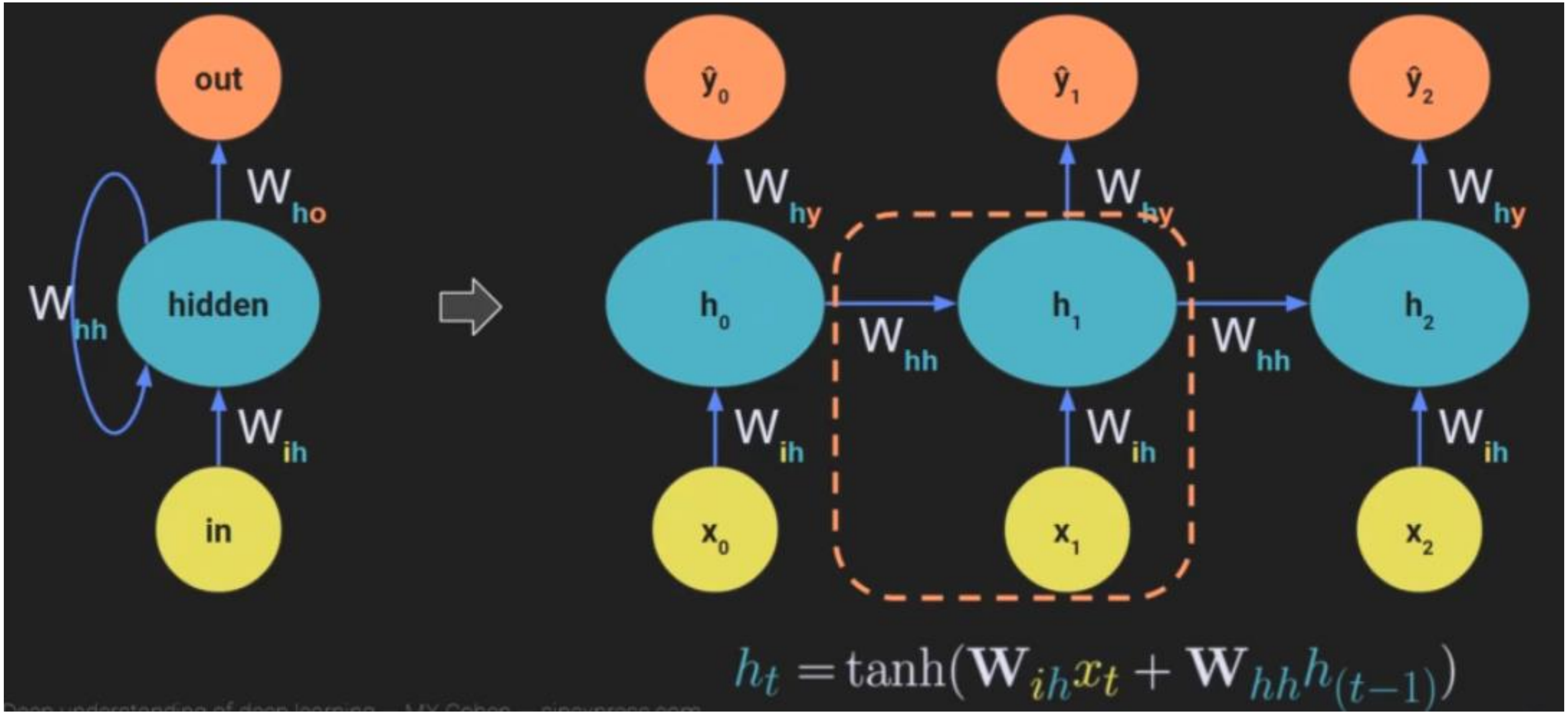
For CNN architectures, the limitations listed are:

- ❑ **Can leverage sequence information in some cases (e.g., converting a time series into an image).** - While CNNs can capture local dependencies and are better at processing sequences than FFNs, they are not inherently designed for sequence processing.
- ❑ **Requires fixed sequence length (image size).** - Similar to FFNs, CNNs also require a fixed-size input, which means sequences must be transformed or adjusted to fit this requirement.
- ❑ **Works in some cases, but usually suboptimal.** - CNNs can be used for sequence processing and have been successful in some applications, but they are often not as effective as architectures specifically designed for sequence data, like Recurrent Neural Networks (RNNs) or Transformers.

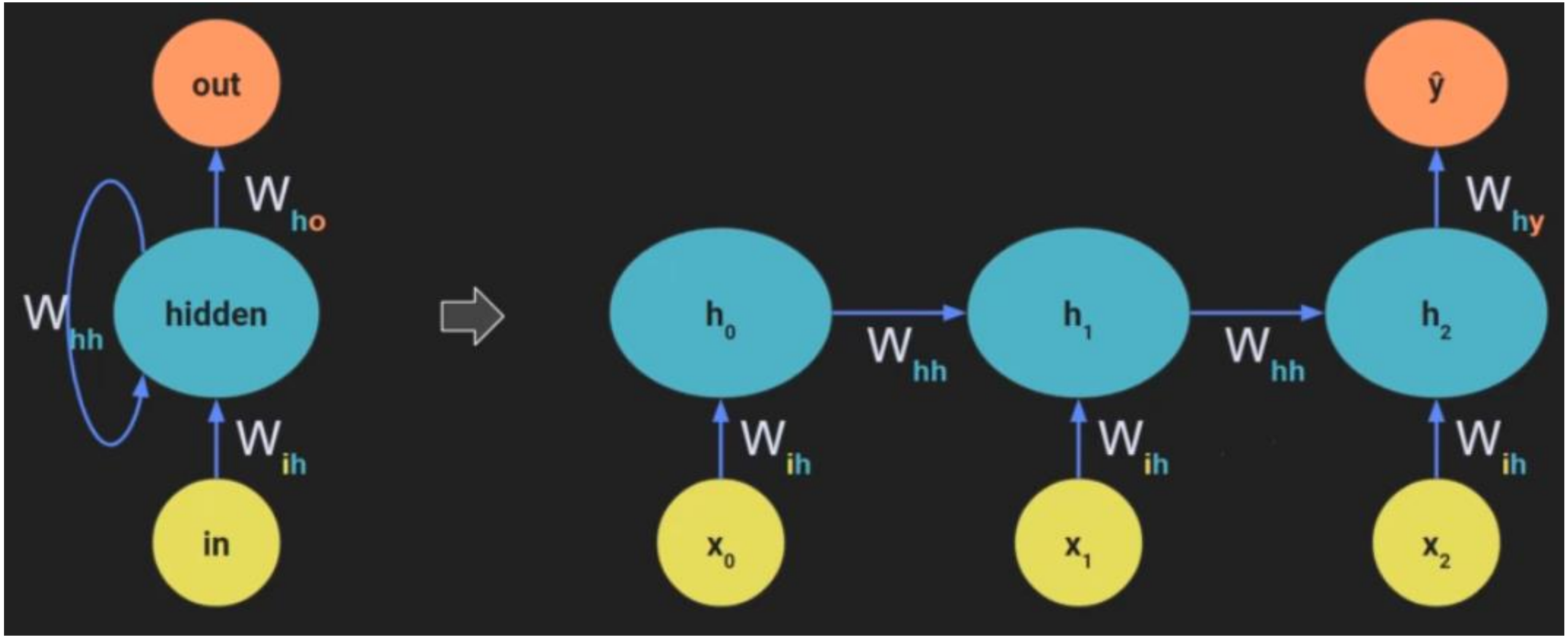
Basic RNN architecture



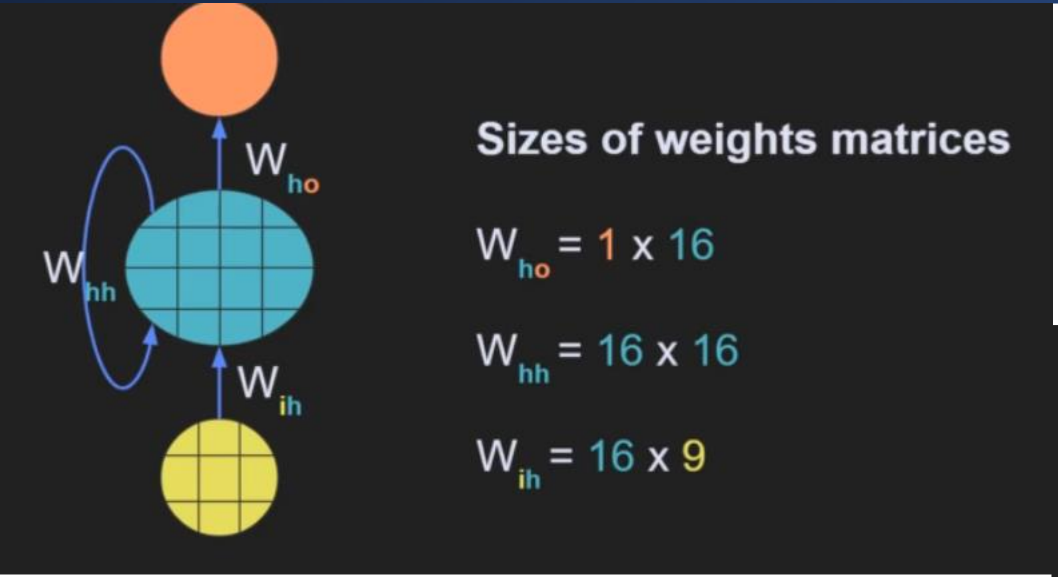
Basic RNN architecture


out
hidden
in
W_{ho}
W_{hh}
W_{ih}
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
h_0
h_1
h_2
x_0
x_1
x_2
W_{hy}
W_{hh}
W_{ih}
$h_t = \tanh(\mathbf{W}_{ih} x_t + \mathbf{W}_{hh} h_{(t-1)})$
Deep understanding of deep learning — MX Cohen — sincxpress.com

Basic RNN architecture



Basic RNN architecture

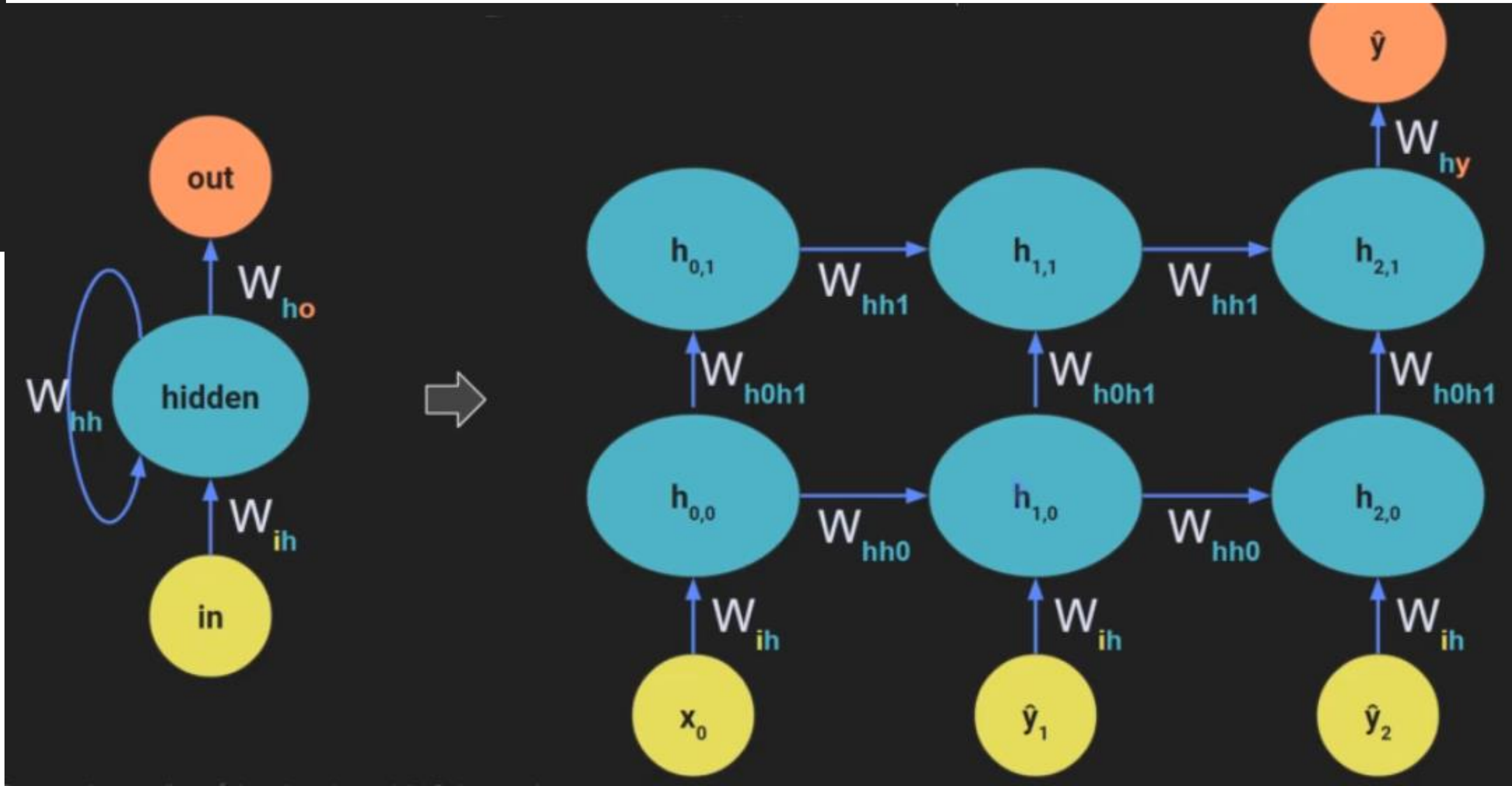


Sizes of weights matrices

$W_{ho} = 1 \times 16$

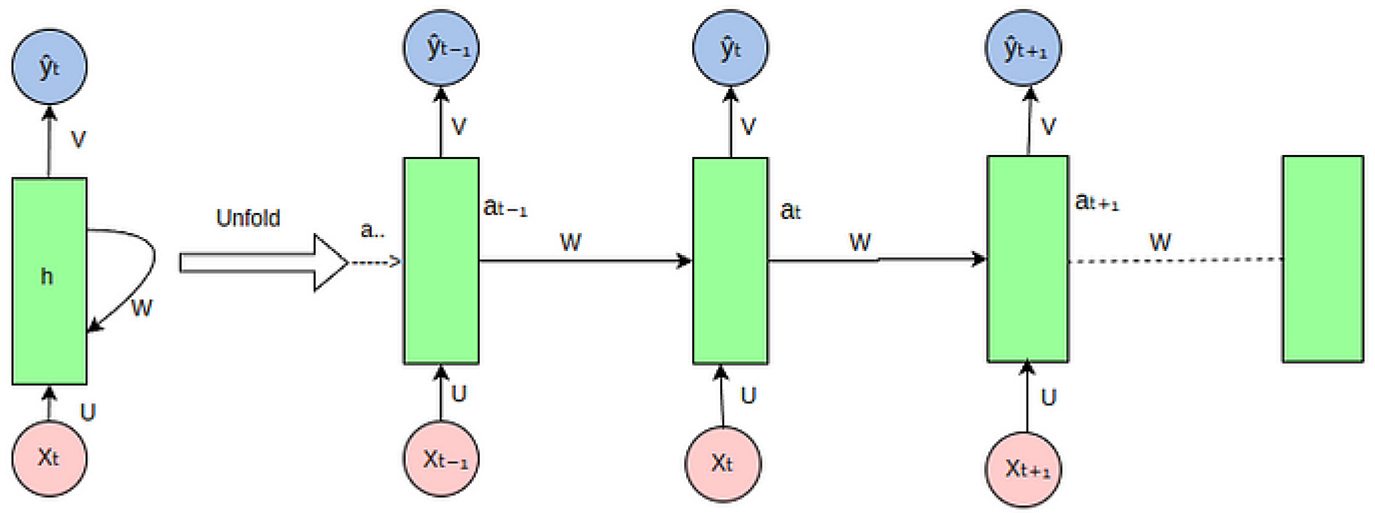
$W_{hh} = 16 \times 16$

$W_{ih} = 16 \times 9$



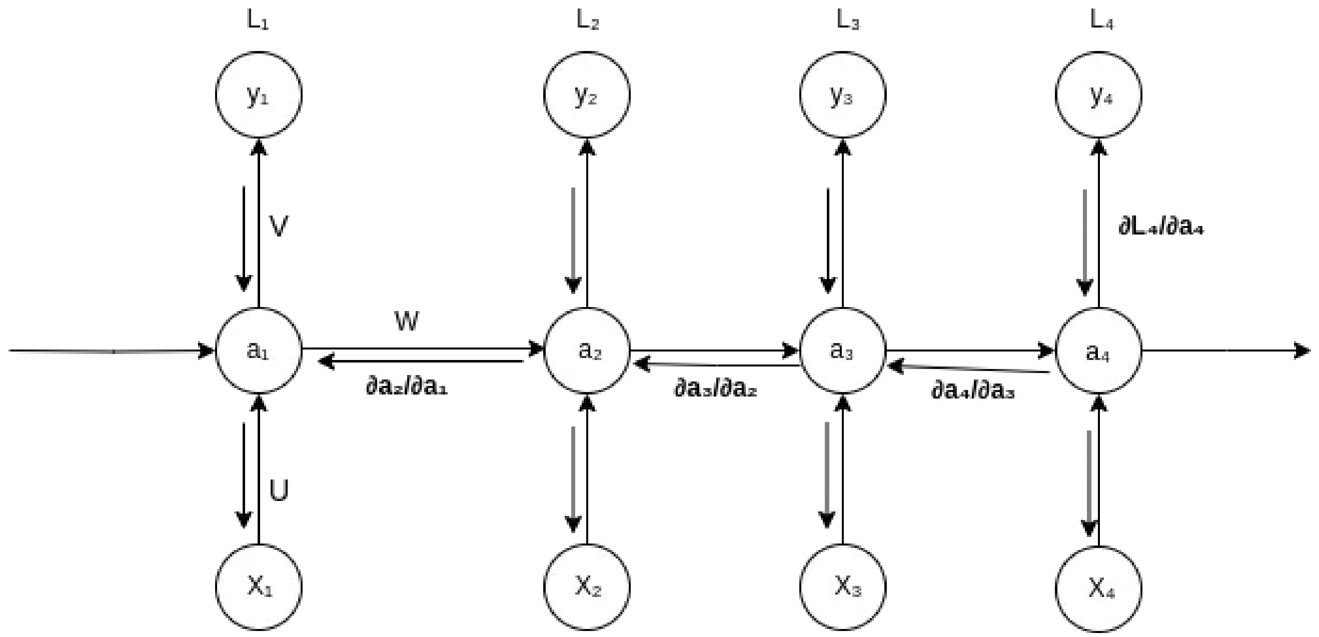
Basic RNN architecture

[Recurrent Neural Network (RNN) Architecture Explained | by Sushmita Poudel | Medium]



Basic RNN architecture

[Recurrent Neural Network (RNN) Architecture Explained | by Sushmita Poudel | Medium]



L_1
L_2
L_3
L_4
y_1
y_2
y_3
y_4
V
$\partial L_4/\partial a_4$
W
a_1
a_2
a_3
a_4
$\partial a_2/\partial a_1$
$\partial a_3/\partial a_2$
$\partial a_4/\partial a_3$
U
X_1
X_2
X_3
X_4

Basic RNN architecture

[Recurrent Neural Network (RNN) Architecture Explained | by Sushmita Poudel | Medium]

$a_t = U * X_t + W * a_{t-1} + b$

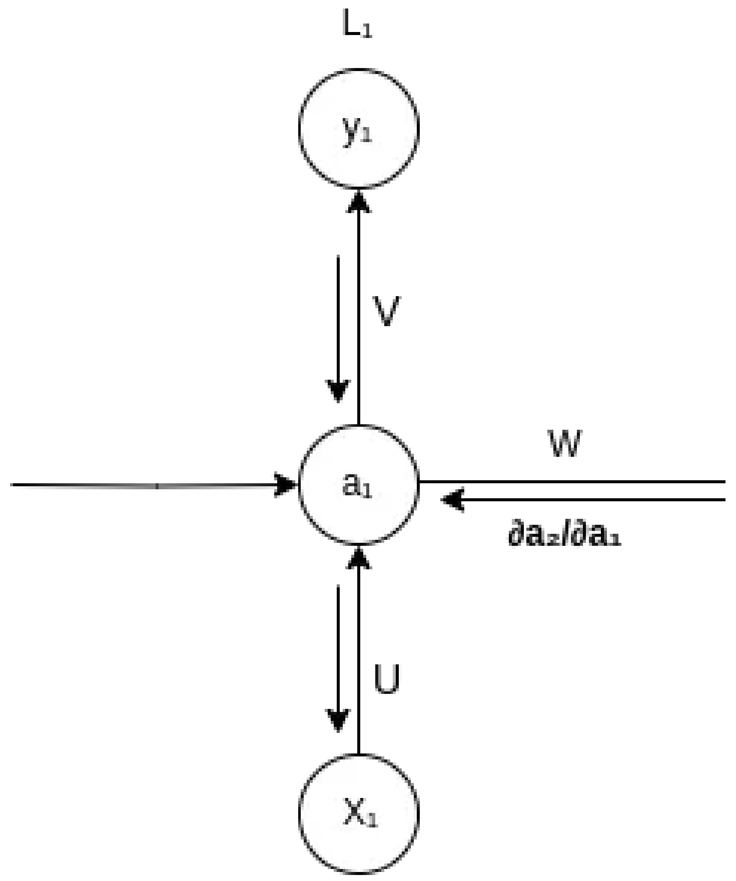
$a_t = \tanh(a_t)$

$\hat{y}_t = \text{softmax}(V * a_t + c)$

$L(y, \hat{y}) = (1/t) * \Sigma(y_t — \hat{y}_t)^2$ — — —— — -> MSE Loss

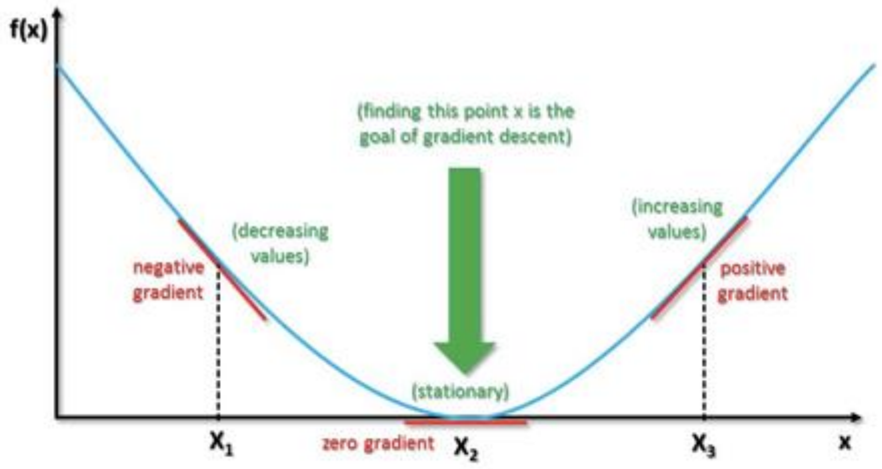
$\partial L_4/\partial W = (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial W) + (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial a_3 * \partial a_3/\partial W) + (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial a_3 * \partial a_3/\partial a_2 * \partial a_2/\partial W) + (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial a_3 * \partial a_3/\partial a_2 * \partial a_2/\partial a_1 * \partial a_1/\partial W)$

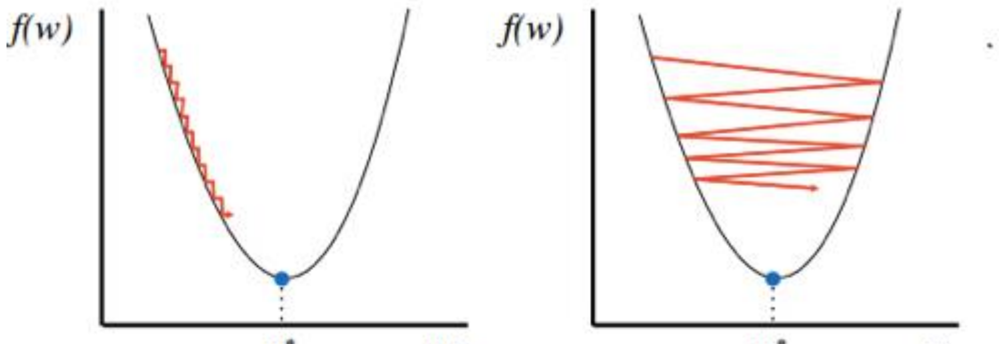
$\partial L_4/\partial U = (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial U) + (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial a_3 * \partial a_3/\partial U) + (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial a_3 * \partial a_3/\partial a_2 * \partial a_2/\partial U) + (\partial L_4/\partial \hat{y}_4 * \partial \hat{y}_4/\partial a_4 * \partial a_4/\partial a_3 * \partial a_3/\partial a_2 * \partial a_2/\partial a_1 * \partial a_1/\partial U)$



Limitation of RNN architecture

[Recurrent Neural Network (RNN) Architecture Explained | by Sushmita Poudel | Medium]()





Too small: converge very slowly

Too big: overshoot and even diverge

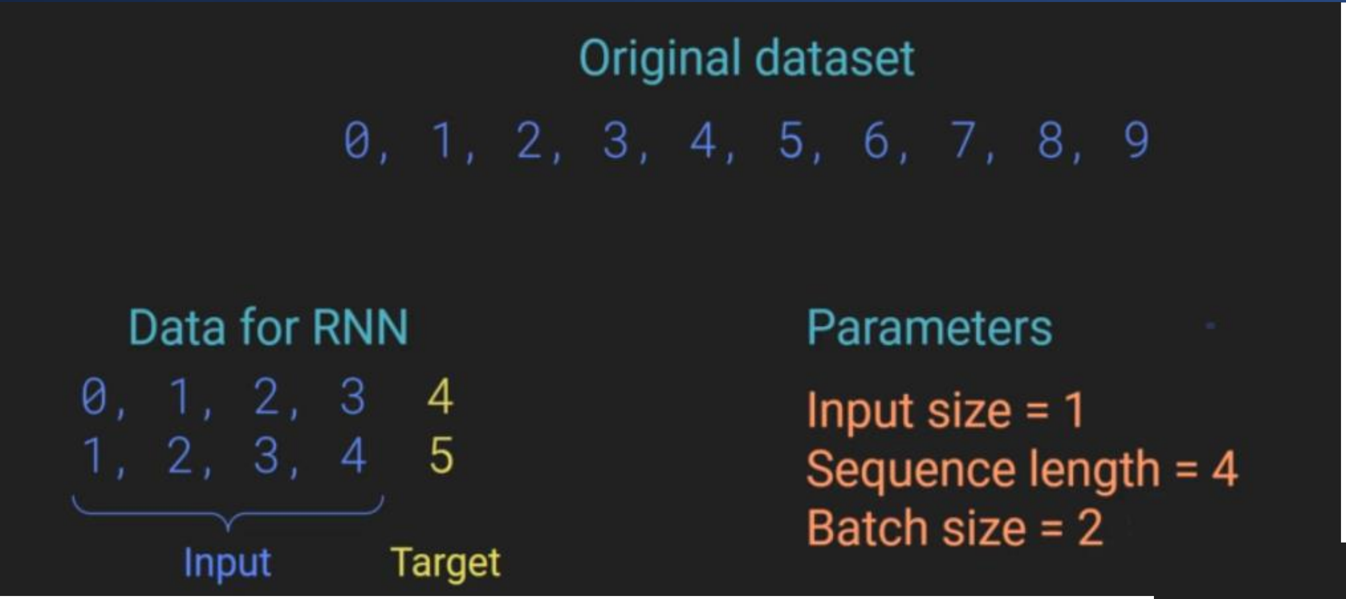
Limitation of RNN architecture

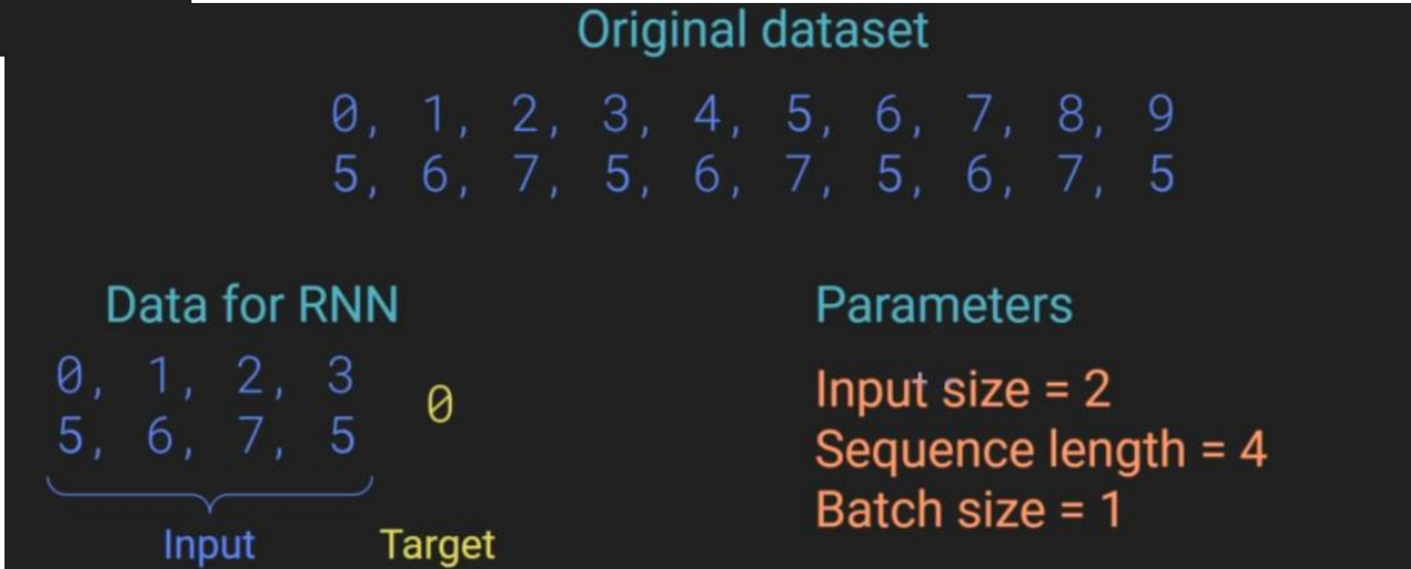
- ❑ During backpropagation, gradients can become too small, leading to the vanishing gradient problem, or too large, resulting in the exploding gradient problem as they propagate backward through time.
- ❑ In the case of vanishing gradients, the issue is that the gradient may become too small where the network struggles to capture long-term dependencies effectively.
  - ❑ It can still converge during training, but it may take a very very long time.
- ❑ In contrast, in exploding gradient problem, large gradient can lead to numerical instability during training, causing the model to deviate from the optimal solution and making it difficult for the network to converge to global minima.

Limitation of RNN architecture

- **Overly sensitive to recent inputs.** - RNNs tend to be more influenced by more recent inputs in a sequence, which can be a limitation when earlier inputs are also important.
- **Low sensitivity to distant inputs.** - Conversely, RNNs can struggle to maintain information about inputs from earlier in the sequence, a problem often referred to as the issue of "long-term dependencies".
- **Risk of vanishing and exploding gradients.** - During training, RNNs are susceptible to these issues where gradients become too small (vanish) or too large (explode) to propagate useful error information through many layers or time steps. This makes it difficult to learn long-range dependencies within a sequence.
- **RNNs are great when important characteristics are a few time-steps back (like an AR model in signal processing).** - RNNs perform well when the important contextual information is located close to the point of prediction, similar to autoregressive (AR) models that predict future points in a time series based on a short window of previous points.
- **RNN-extensions give better performance: LSTM and GRU (later videos!).** - Extensions of basic RNNs, such as Long Short-Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs), have been developed to address these limitations, particularly the issues of long-term dependencies and vanishing gradients. The parenthetical note "later videos!" suggests that these topics will be covered in more detail in subsequent parts of the presentation or educational series.

Organizing inputs into RNNs

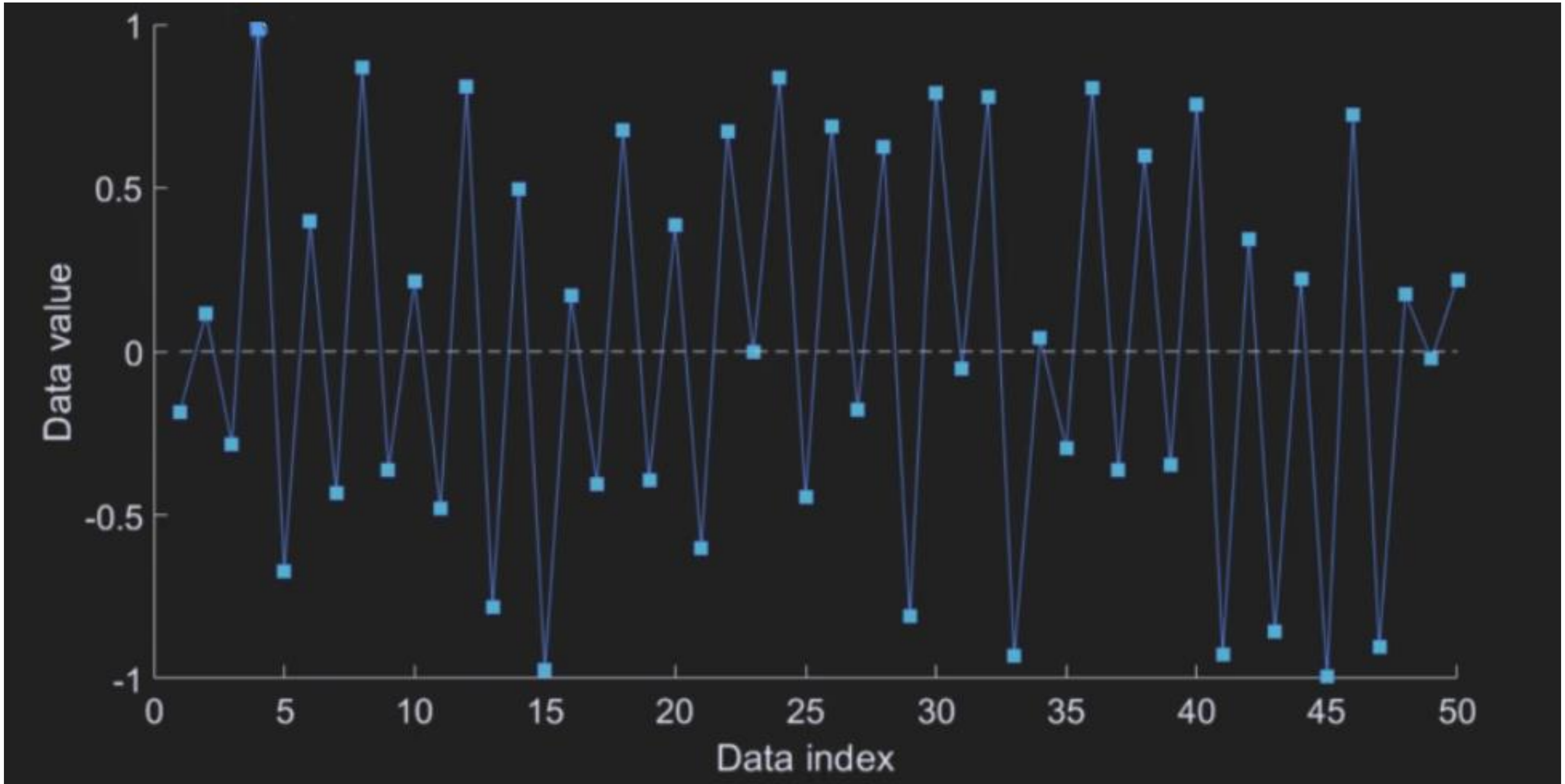




Organizing inputs into RNNs

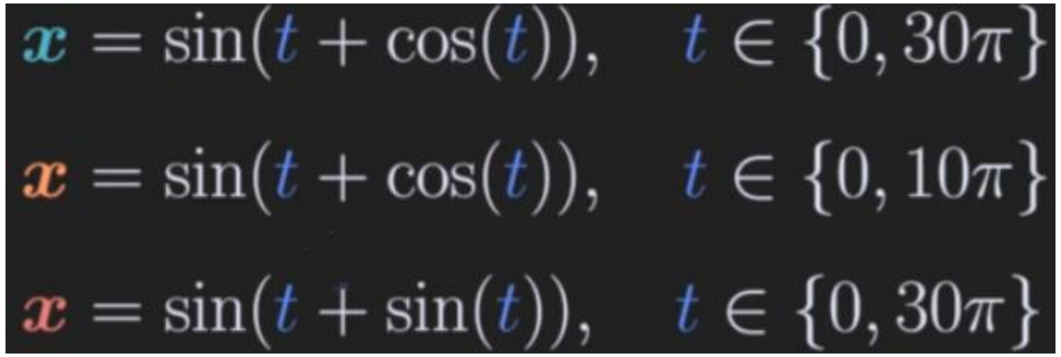
- **Input size:** This is the number of features for each element in the sequence. For example, if the input data is a time series of stock prices, the input size might be the number of different stocks being considered.
- **Sequence length:** This refers to the length of the sequence of data that is fed into the RNN for each iteration of training. In some contexts, this could also be referred to as the 'time steps'. This is important for the RNN to learn from data where the sequence and order matter.
- **Batch size:** This is the number of sequences that are processed in parallel in one pass (one epoch) of training. A larger batch size means more examples are processed simultaneously, which can lead to faster training but also requires more memory.
- **Hidden size:** This denotes the number of units in the hidden layers of the RNN. These are the “memory” units that capture information from the input data as it propagates through the network.
- **Number of layers:** This is the number of hidden layers stacked on top of each other in the RNN architecture. Each layer can capture different levels of abstraction of the data.

Predicting alternating sequences –Python Practice



Predicting alternating sequences –Python Practice #2 & 3

- ❑ **Replace the alternating sequence with sine Function**
- ❑ **Change the seqlength parameter.**
- ❑ **Test on new data with different frequency.**
- ❑ **Test on new data with different function.**

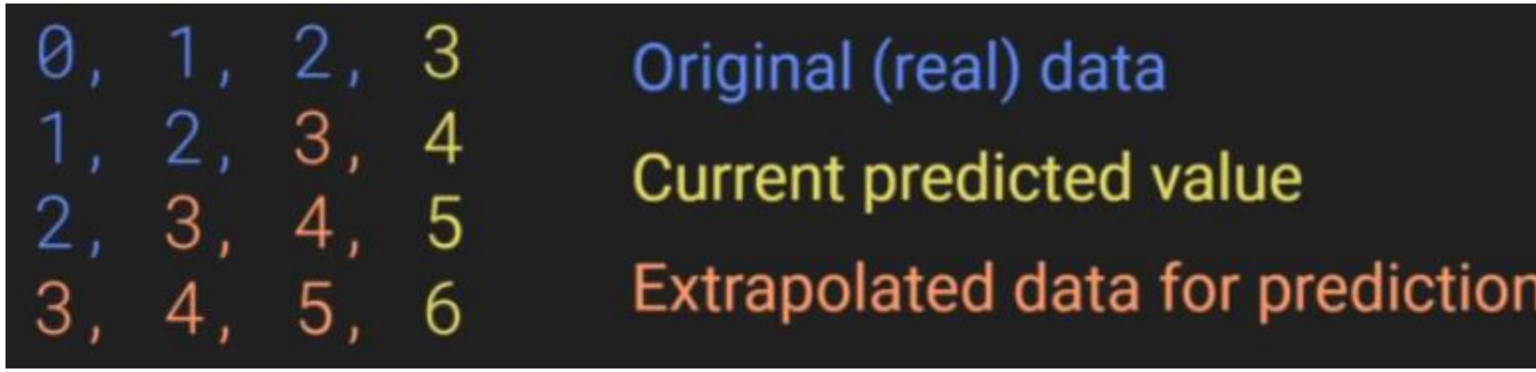


$$x = \sin(t + \cos(t)), \quad t \in \{0, 30\pi\}$$

$$x = \sin(t + \cos(t)), \quad t \in \{0, 10\pi\}$$

$$x = \sin(t + \sin(t)), \quad t \in \{0, 30\pi\}$$

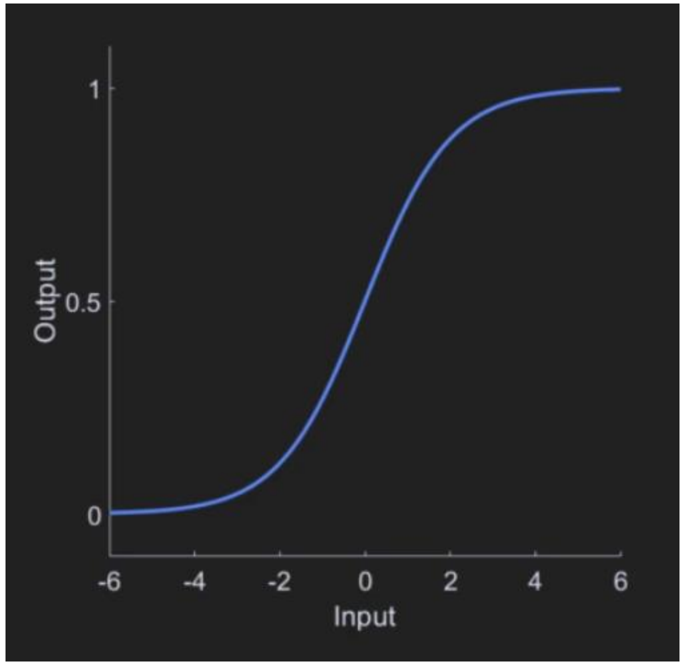
- ❑ **Start with seqlength original data values.**
- ❑ **Predict seqlength+1.**
- ❑ **Append Predicted data point.**
- ❑ **Repeat until 2N data points.**



```
0,  1,  2,  3        Original (real) data
1,  2,  3,  4        Current predicted value
2,  3,  4,  5        Extrapolated data for predictio
3,  4,  5,  6
```

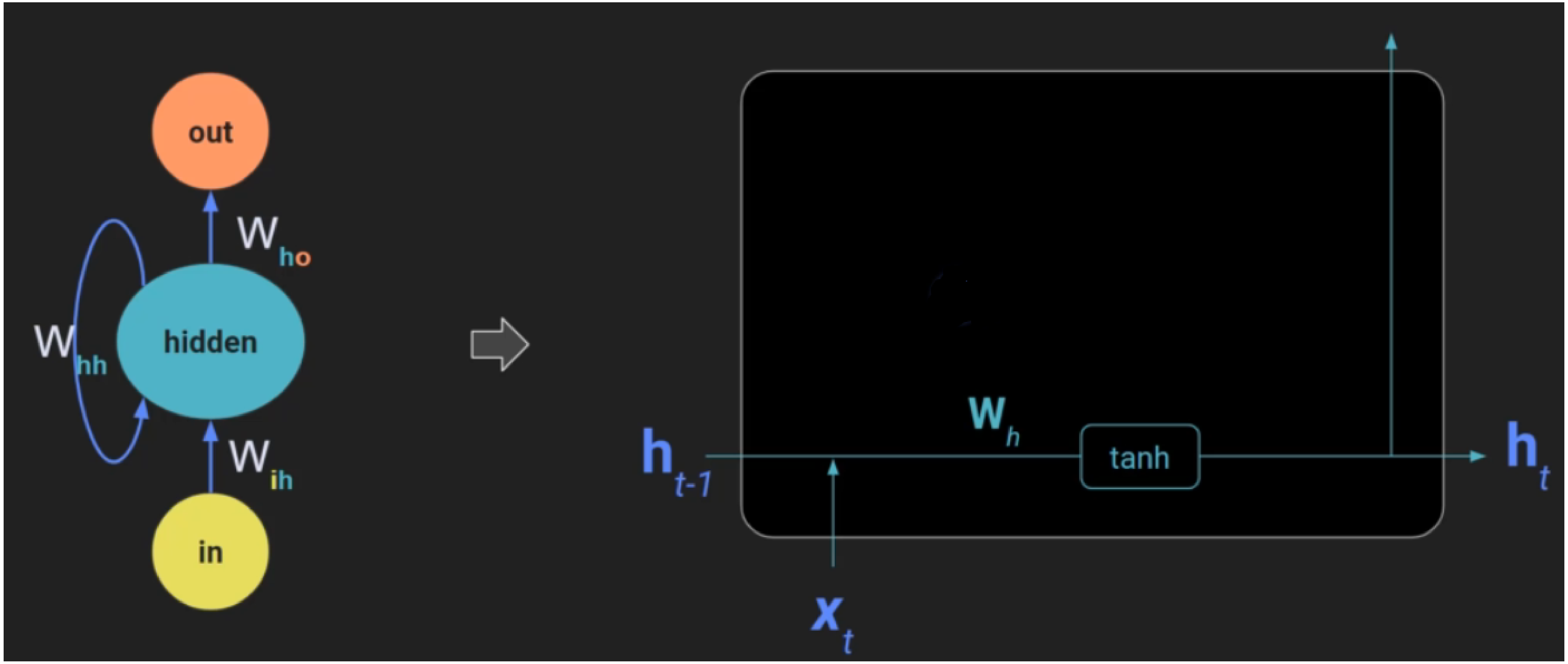
What is a gate?

- ❑ **Sigmoid Function**
- ❑ **Important point: Most values are close to 0 or close to 1.**
- ❑ **When multiplying other variables, the sigmoid can switch off (0) or on (1) the variable, thus gating information routing..**



$$\sigma = \frac{1}{1+e^{-x}}$$

Redrawing the RNN circuit



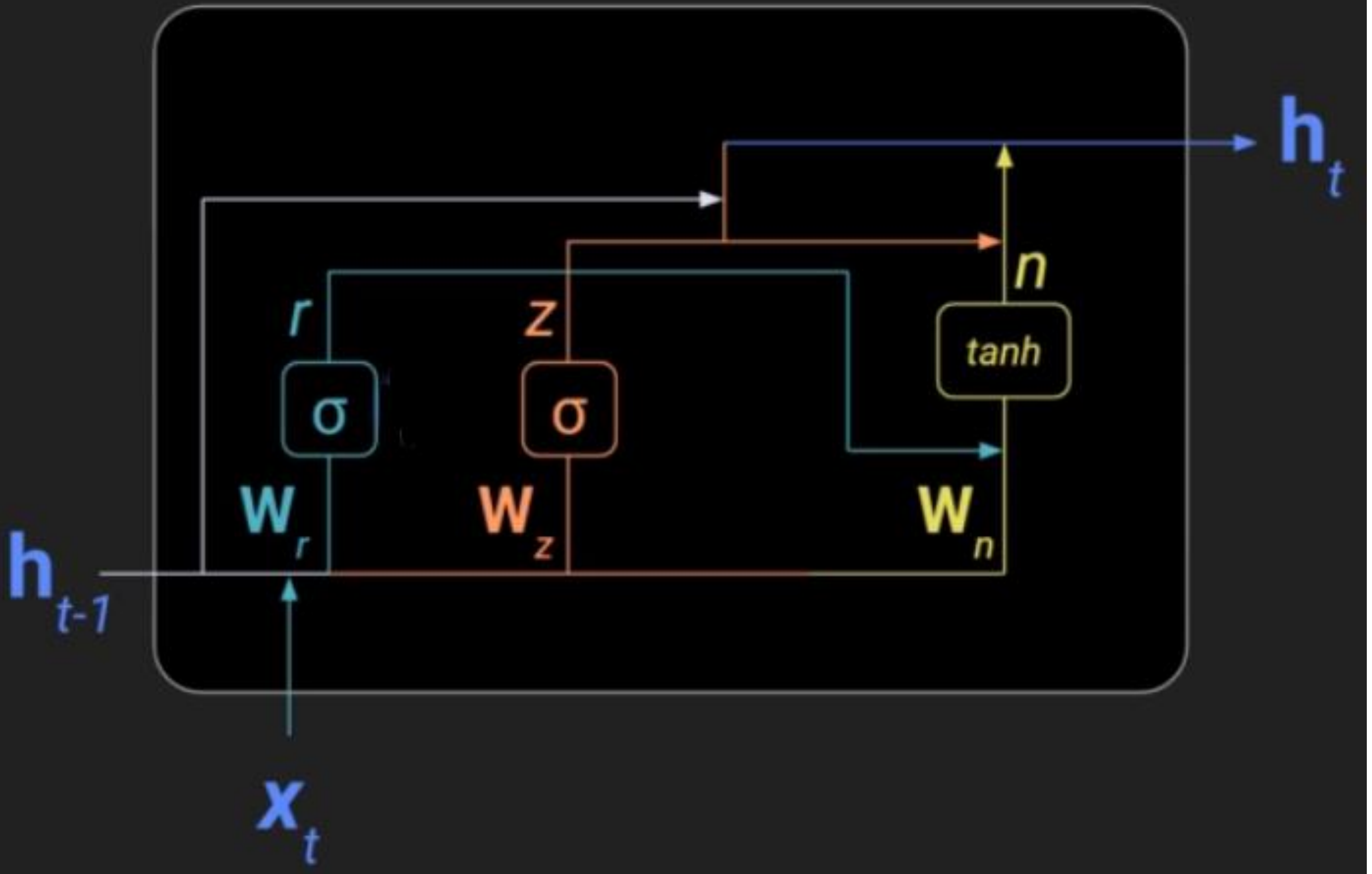
Gated recurrent unit (GRU)

$$r_t = \sigma(\mathbf{W}_r[x_t; h_{t-1}])$$

$$z_t = \sigma(\mathbf{W}_z[x_t; h_{t-1}])$$

$$n_t = \tanh(\mathbf{W}_n[x_t; r_t h_{t-1}])$$

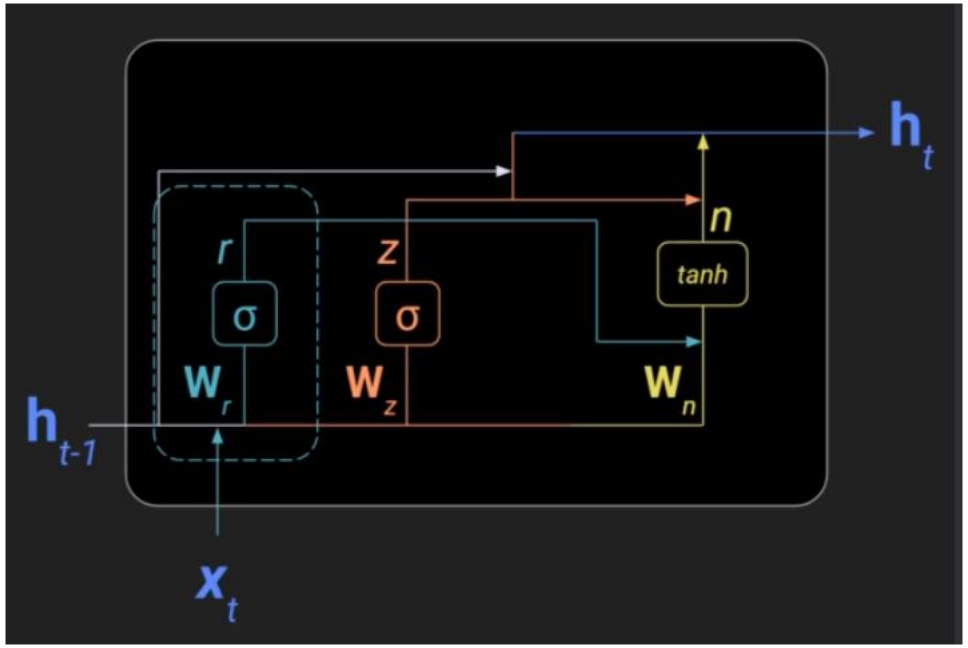
$$h_t = (1 - z_t)n_t + z_t h_{t-1}$$



Gated recurrent unit (GRU)

- ❑ **Reset gate or forget gate**
- ❑ **Combine current input and previous hidden states to decide how of the past to update.**

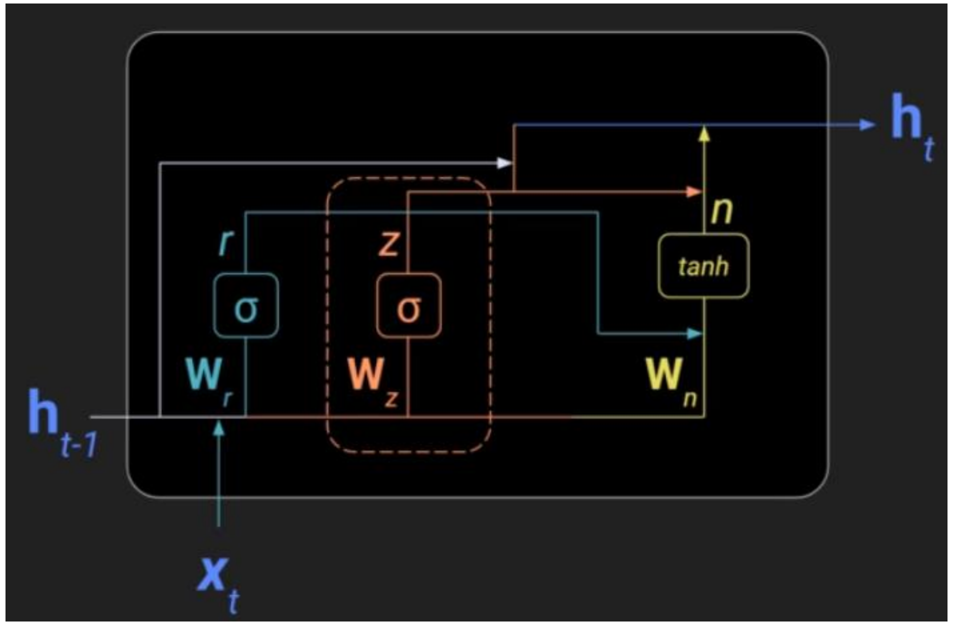
$$r_t = \sigma(\mathbf{W}_r[x_t; h_{t-1}])$$



Gated recurrent unit (GRU)

- **Update Gate**
- **Combine current input and previous hidden states to decide what information to update (the hidden state items).**

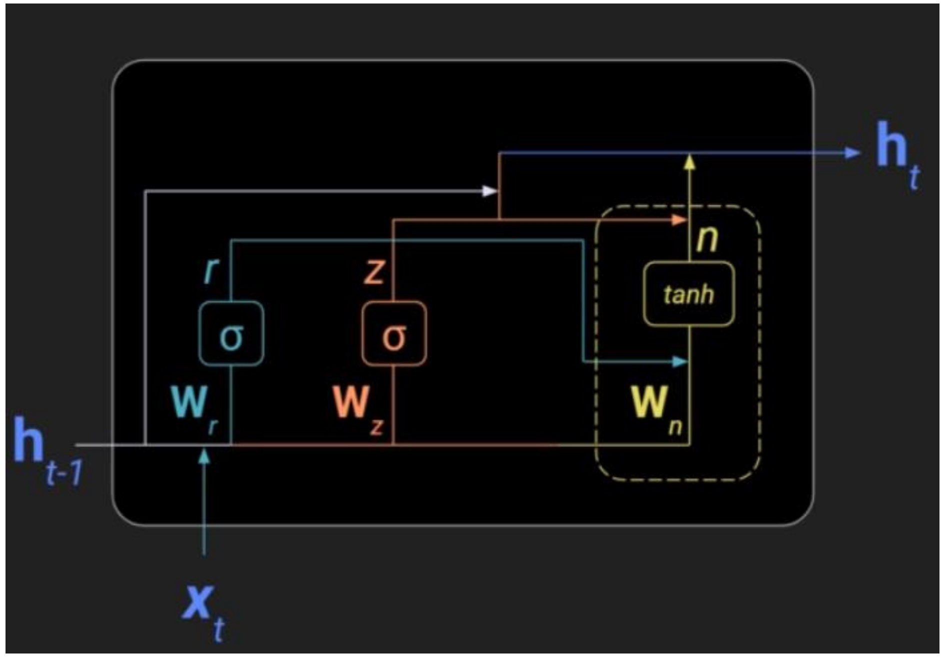
$$z_t = \sigma(\mathbf{W}_z[x_t; h_{t-1}])$$



Gated recurrent unit (GRU)

- ❑ **New Information**
- ❑ **Combine current input and previous hidden states to create new candidate hidden state**

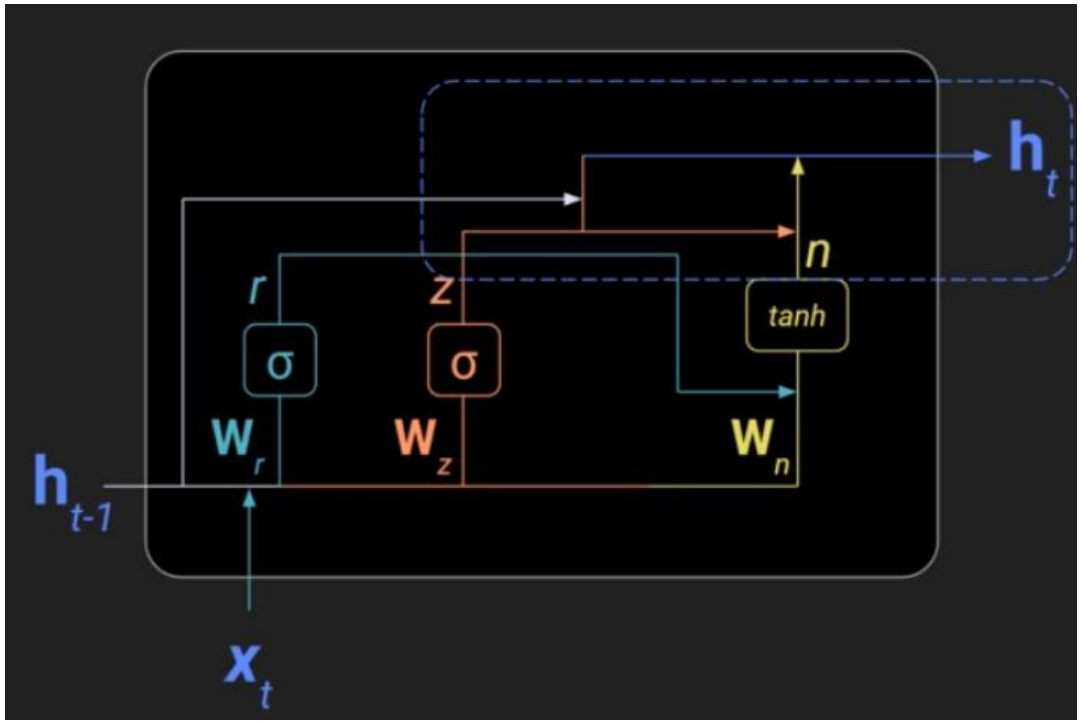
$$n_t = \tanh(\mathbf{W}_n[x_t; r_t h_{t-1}])$$



Gated recurrent unit (GRU)

- **Output (hidden state)**
- **Weighted combination of to_remember new state(n) and to forget old state(h_t-1)**

$$h_t = (1 - z_t)n_t + z_t h_{t-1}$$



Long Short-term memory (LSTM)

$$f_t = \sigma(\mathbf{W}_f[x_t; h_{t-1}])$$

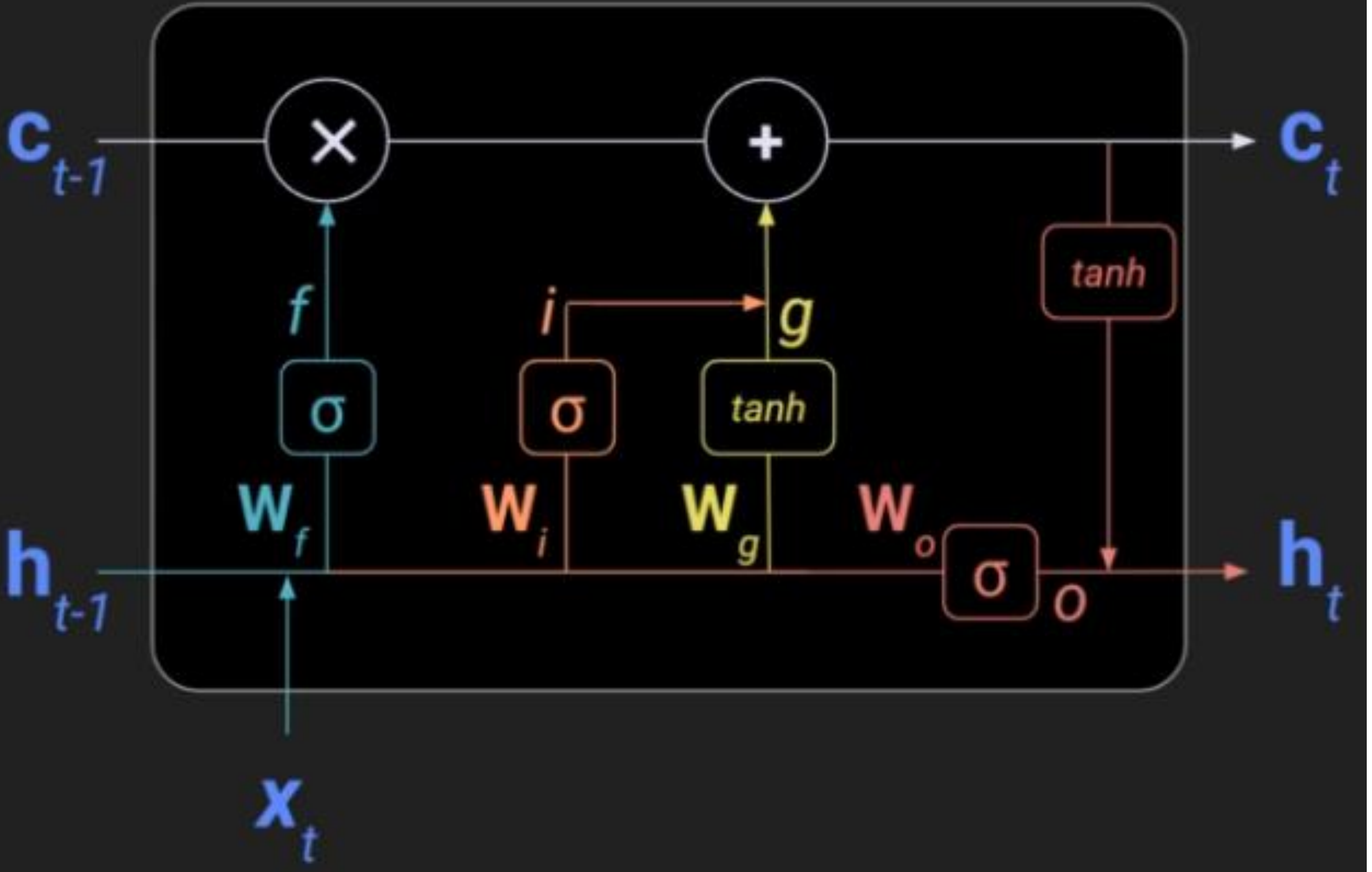
$$i_t = \sigma(\mathbf{W}_i[x_t; h_{t-1}])$$

$$g_t = \tanh(\mathbf{W}_\mathbf{g}[x_t; h_{t-1}])$$

$$o_t = \sigma(\mathbf{W}_\mathbf{o}[x_t; h_{t-1}])$$

$$c_t = f_t\, c_{t-1} + i_t\, g_t$$

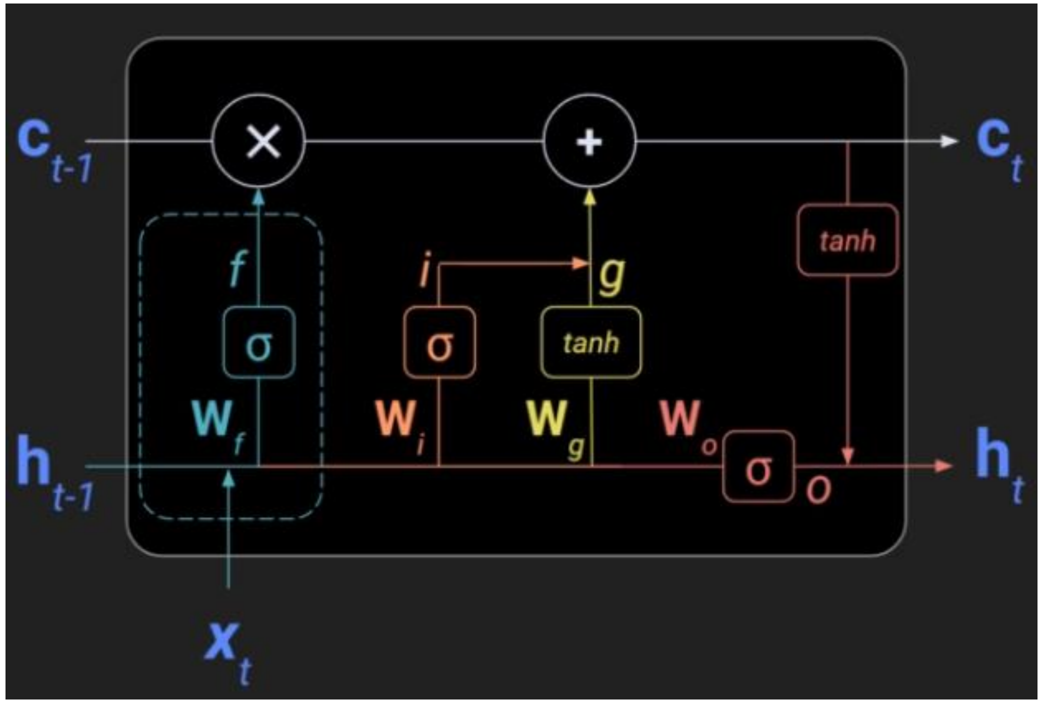
$$h_t = o_t\ \tanh(c_t)$$



Long Short-term memory (LSTM)

- **Forget gate**
- **Combine current input and previous hidden states to decide which previous samples to remember and which to forget.**

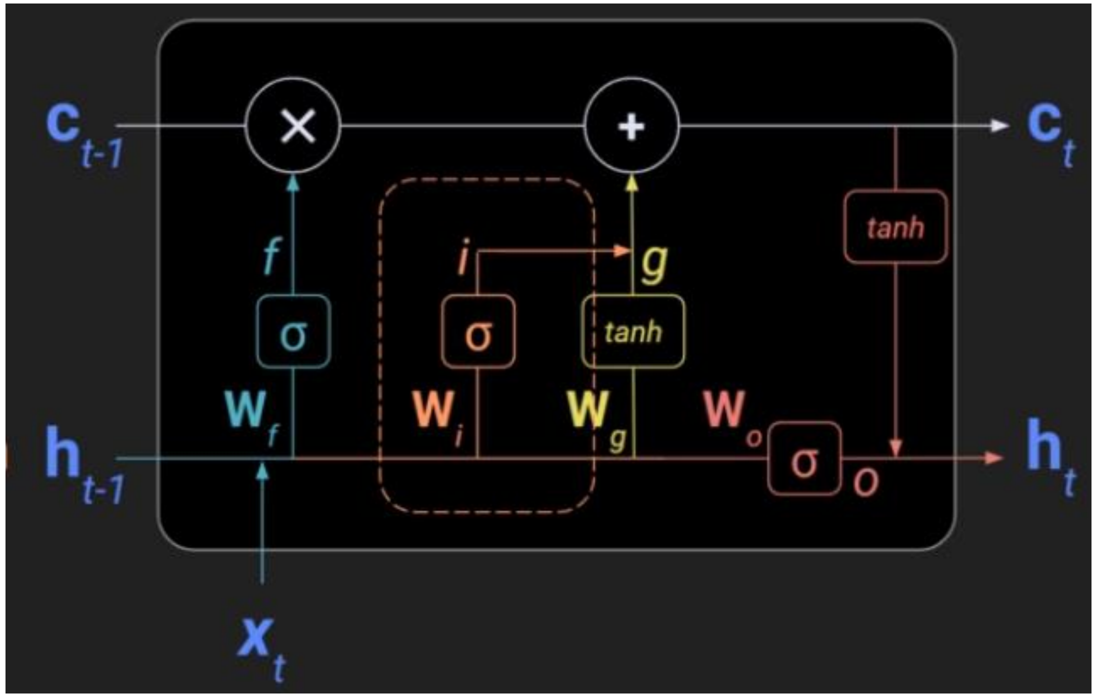
$$f_t = \sigma(\mathbf{W}_f[x_t; h_{t-1}])$$



Long Short-term memory (LSTM)

- **Input gate**
- **Combine current input and previous hidden states to decide which current samples to remember and which to forget.**

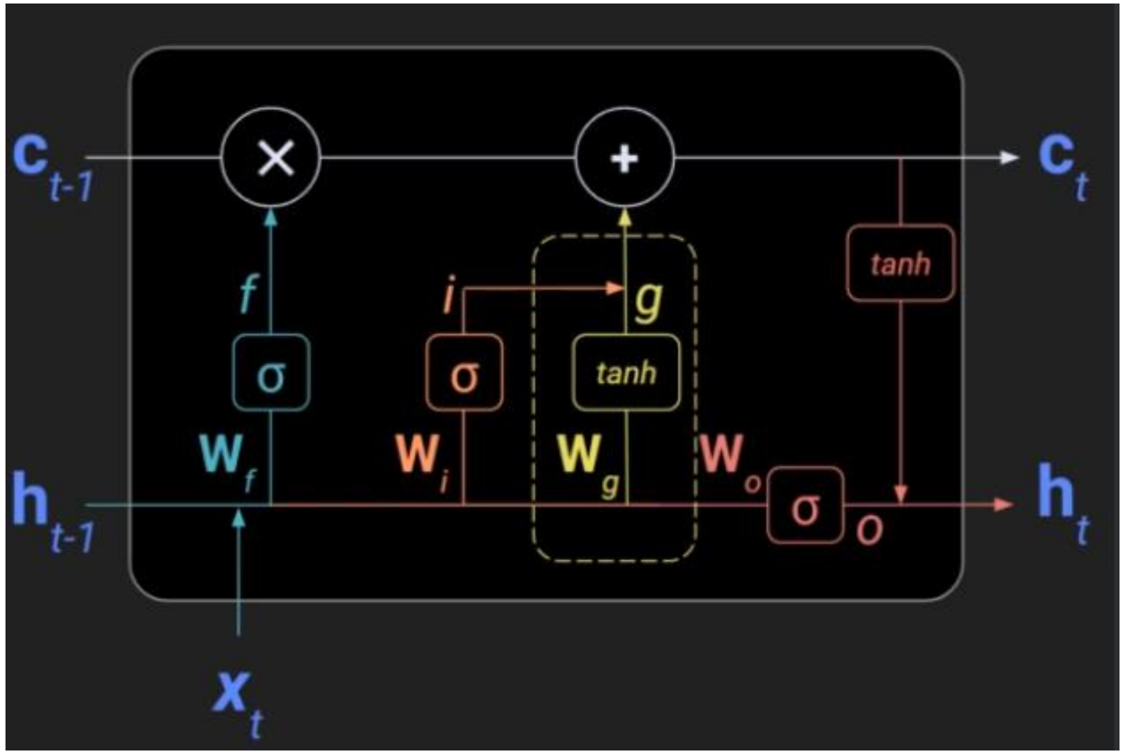
$$i_t = \sigma(\mathbf{W}_i[x_t; h_{t-1}])$$



Long Short-term memory (LSTM)

- **Cell gate**
- **Combine current input and previous hidden states to compute the actual representations.**

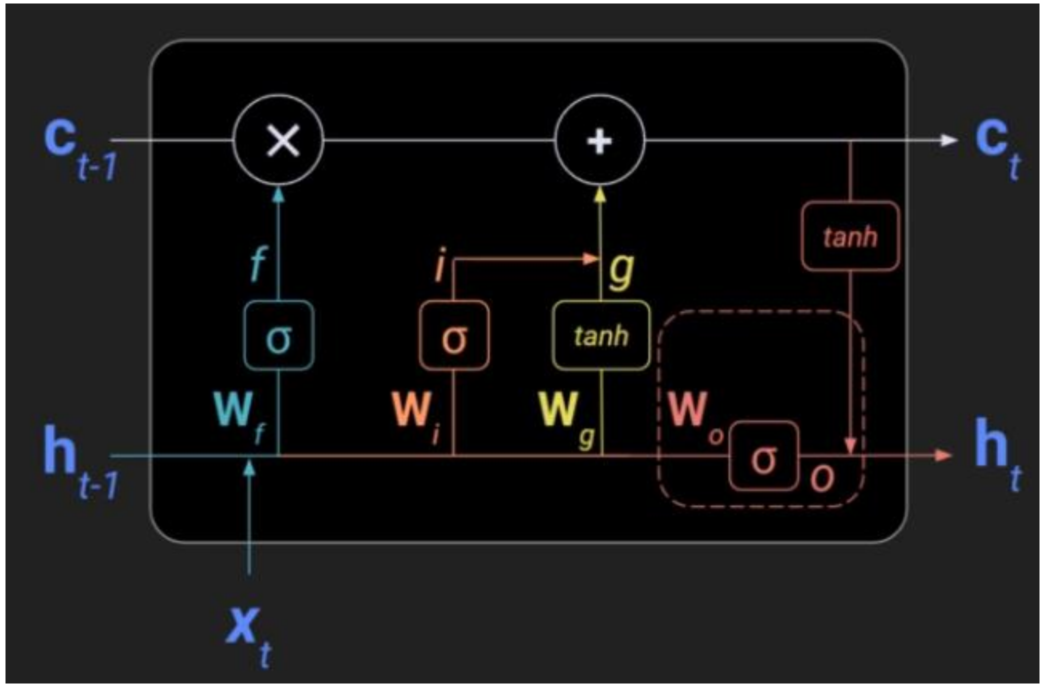
$$g_t = \tanh(\mathbf{W_g}[x_t; h_{t-1}])$$



Long Short-term memory (LSTM)

- Output gate
- Combine current input and previous hidden states and current state to compute the actual LSTM cell output.

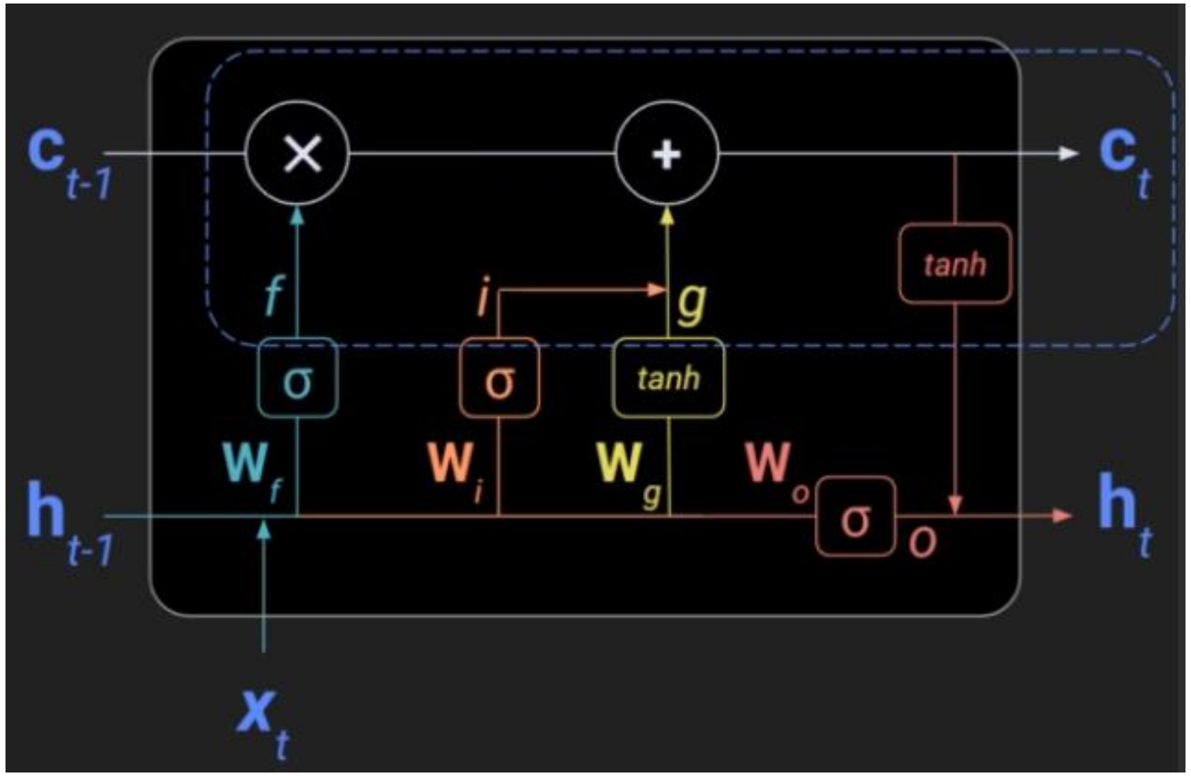
$$o_t = \sigma(\mathbf{W_o}[x_t; h_{t-1}])$$



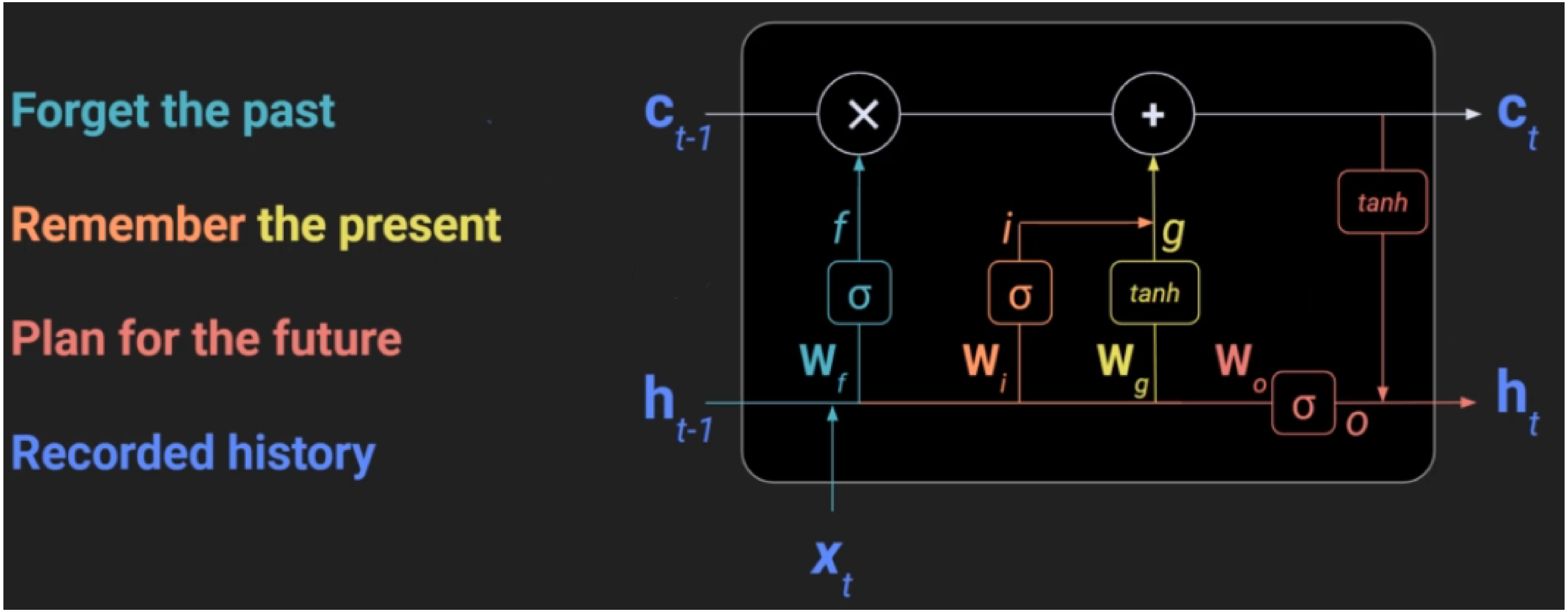
Long Short-term memory (LSTM)

- Cell State
- To remember previous cell state and current cell state

$$c_t = f_t \, c_{t-1} + i_t \, g_t$$

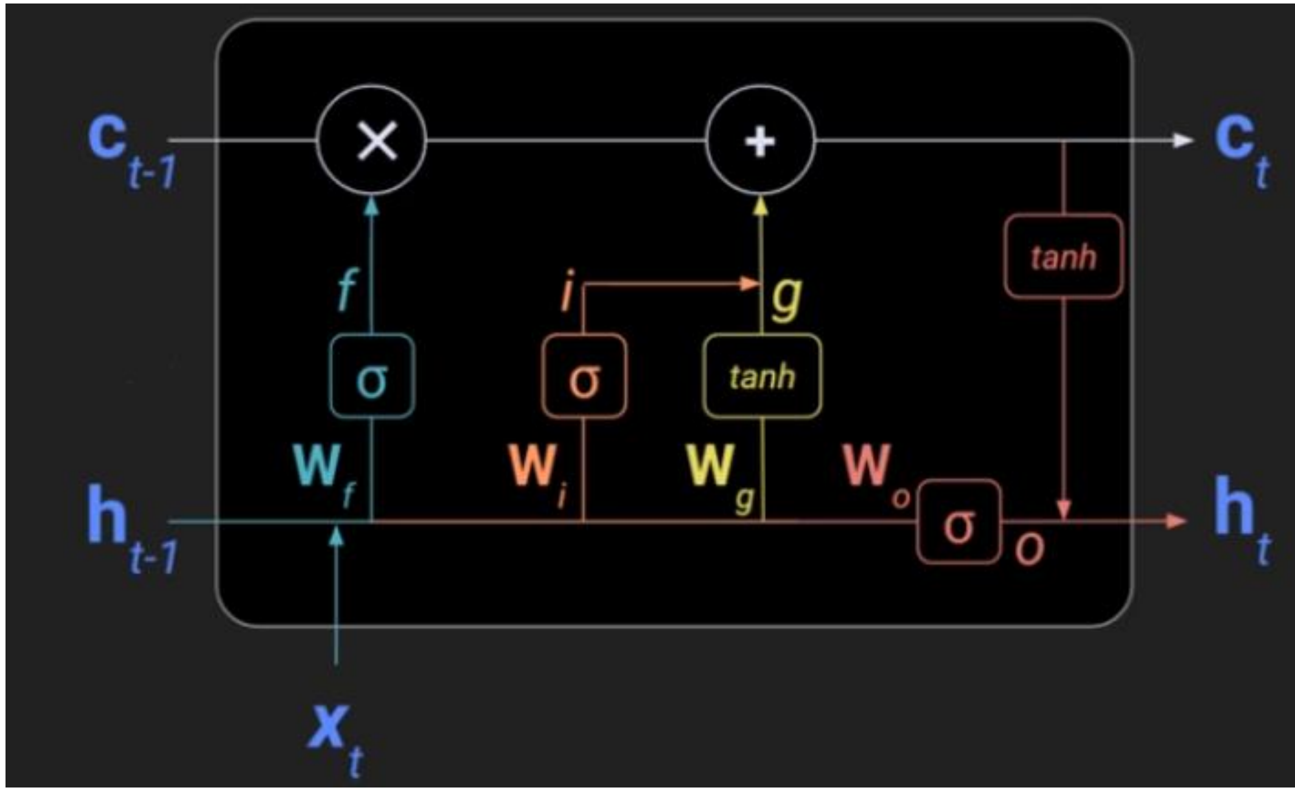


Long Short-term memory (LSTM)



Long Short-term memory (LSTM)

- ❑ Forget the past
- ❑ Remember the present
- ❑ Plan for the future
- ❑ Recorded History



When to use which?

- ❑ **Standard RNN:**
  - ❑ Described as simple with the fewest parameters, making it computationally less intensive.
  - ❑ Good for sequences with short patterns due to its limited ability to maintain information over long sequences, which can be attributed to the vanishing gradient problem.
- ❑ **GRU (Gated Recurrent Unit):**
  - ❑ More complex than standard RNNs, with additional mechanisms to control the flow of information.
  - ❑ Simpler than LSTM (Long Short-Term Memory) networks, often with fewer parameters and potentially less computational load.
  - ❑ Trains faster and is highly scalable, but generally requires more data to perform well compared to standard RNNs.
  - ❑ It strikes a balance between the simplicity of standard RNNs and the complexity of LSTMs.
- ❑ **LSTM (Long Short-Term Memory):**
  - ❑ The most complex among the three, with more parameters and hence requires more data and computational time to train effectively.
  - ❑ Explicitly designed to avoid long-term dependency problems, making it suitable for learning longer sequences or complex patterns.
  - ❑ Includes mechanisms like input, output, and forget gates to maintain and update the memory trace over time.