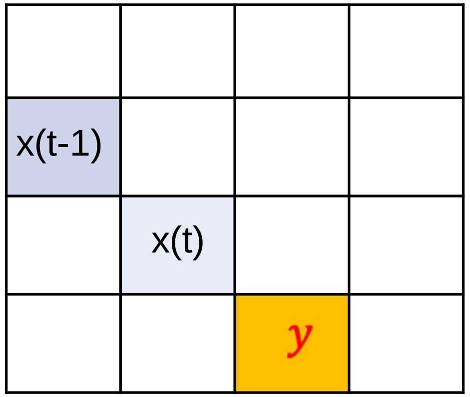
RNN: simple RNN

Why RNN

Why RNN ?

There are two 2 radar points obtained at (t-1) and (t), we want to predict the radar point at (t+1)

Training

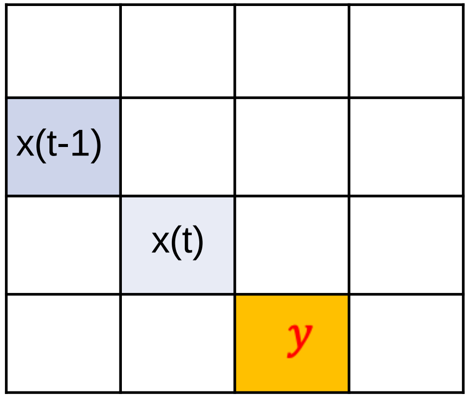


- x(t-1): the radar point obtained at (t-1)
- x(t): the radar point obtained at (t)
- y: the truth in the training data at (t+1)

Why RNN ?

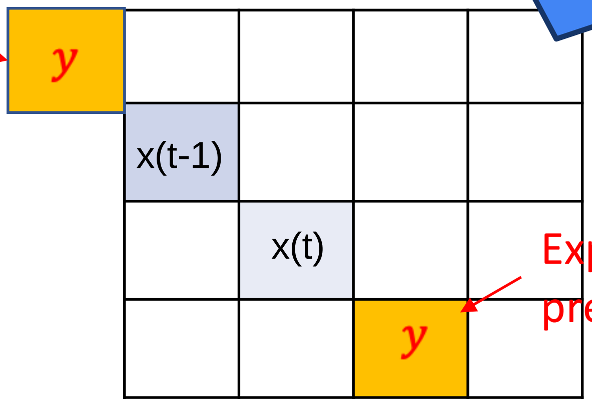
There are two 2 radar points obtained at (t-1) and (t), we want to predict the radar point at (t+1)

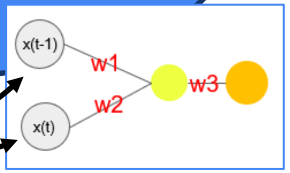
<u>Training</u>



- x(t-1): the radar point obtained at (t-1)
- x(t): the radar point obtained at (t)
- y: the truth in the training data at (t+1)

For a ***DNN***, x(t-1) and x(t) are treated independently, just like x1 and x2 ~ and we are trying to predict y





Does not matter where x(t-1) and x(t) sits (e.g., which node is x(t-1)/x(t))

If we use a neural network such as Densely connected NN, we are not able to tell the sequential information for "x", therefore, the prediction might end up the wrong direction

How RNN works: concept

How RNN works?

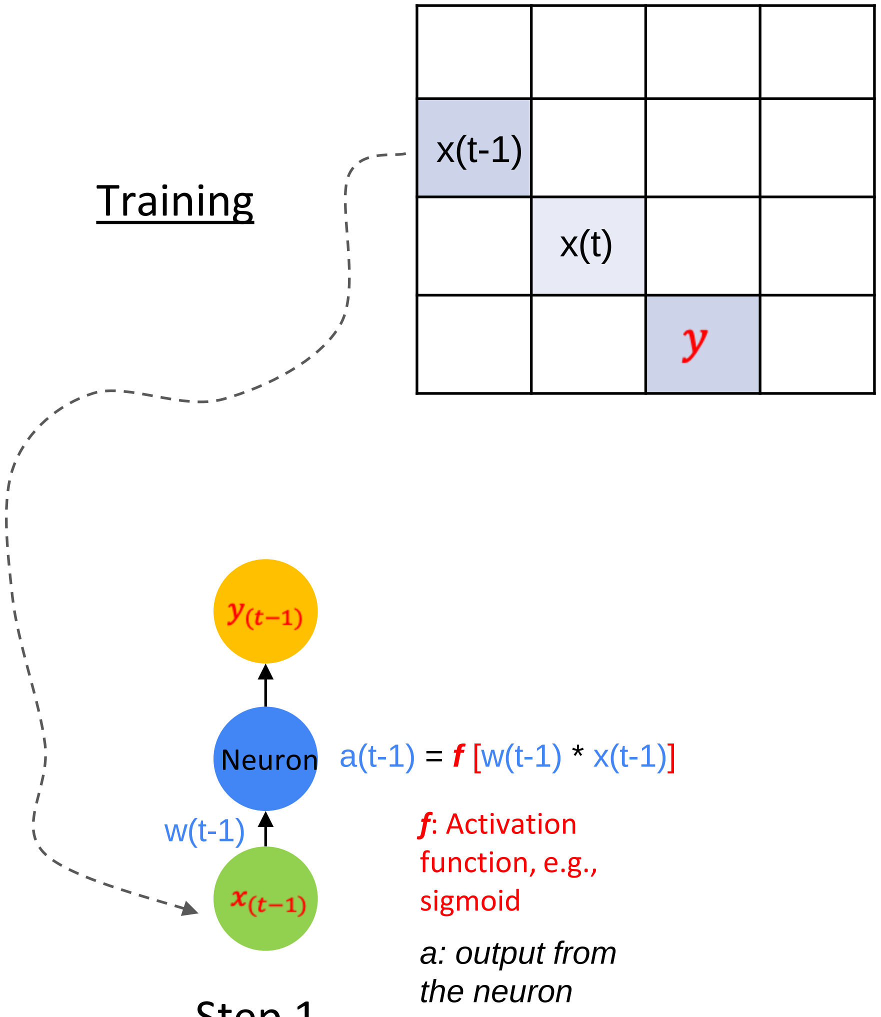
In order to make such a prediction right, we need a sequential of training dataset and model, e.g.,

<u>Training</u>

x(t-1)			
	x(t)		
		y	

How RNN works?

In order to make such a prediction right, we need a sequential of training dataset and model, e.g.,

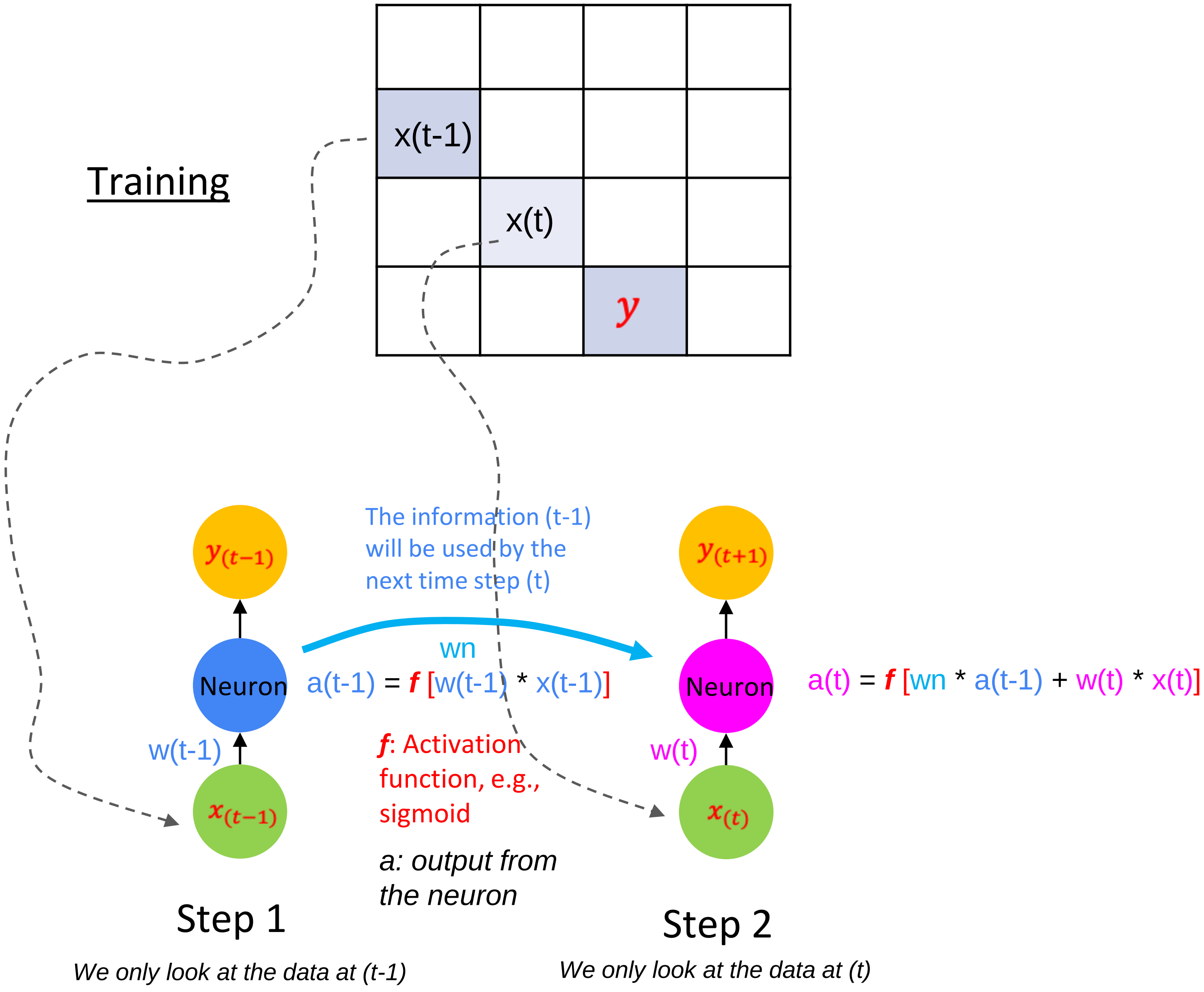


Step 1

We only look at the data at (t-1)

How RNN works?

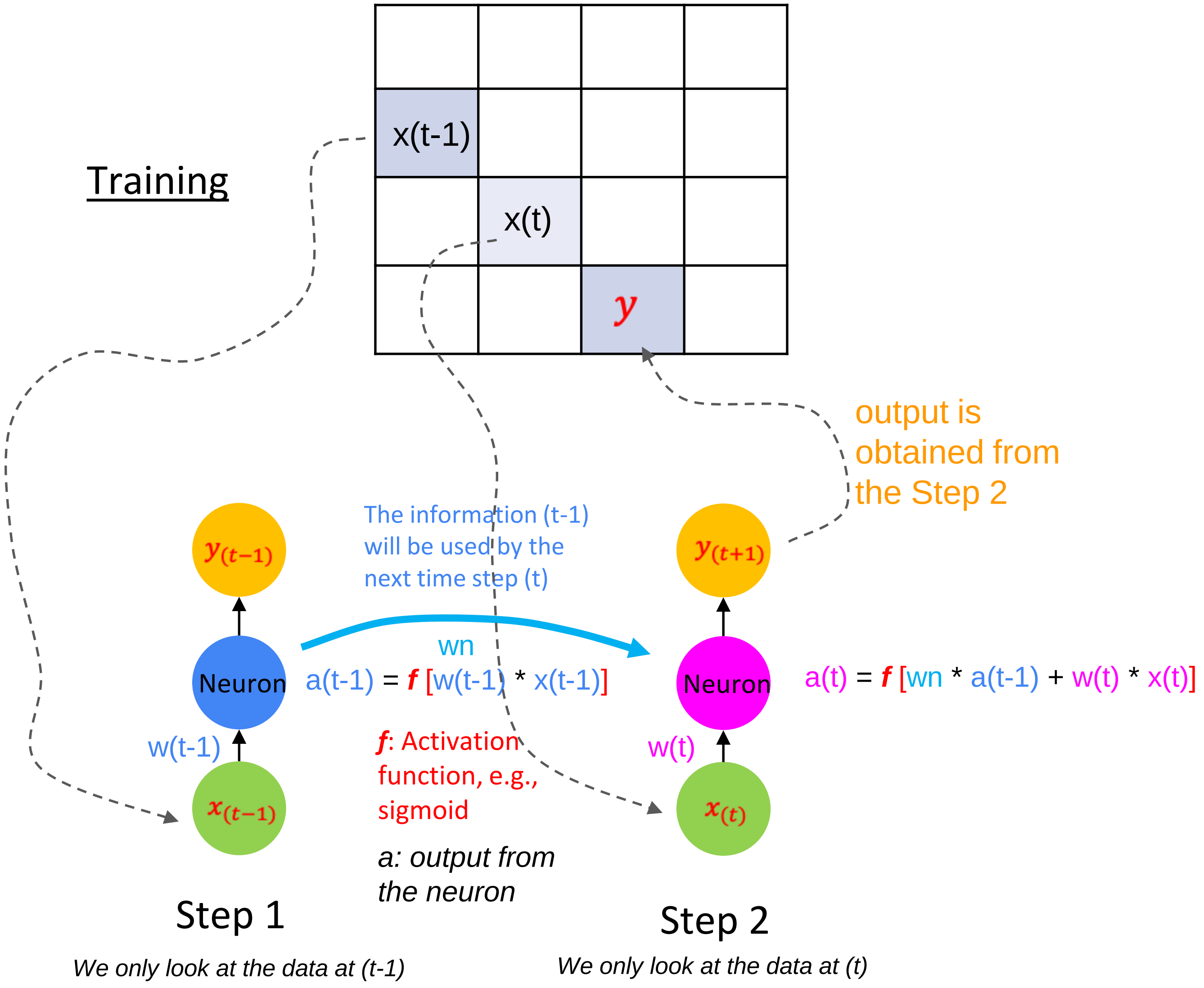
In order to make such a prediction right, we need a sequential of training dataset and model, e.g.,



By doing this, the neuro information from (t-1) is passed to (t)

How RNN works?

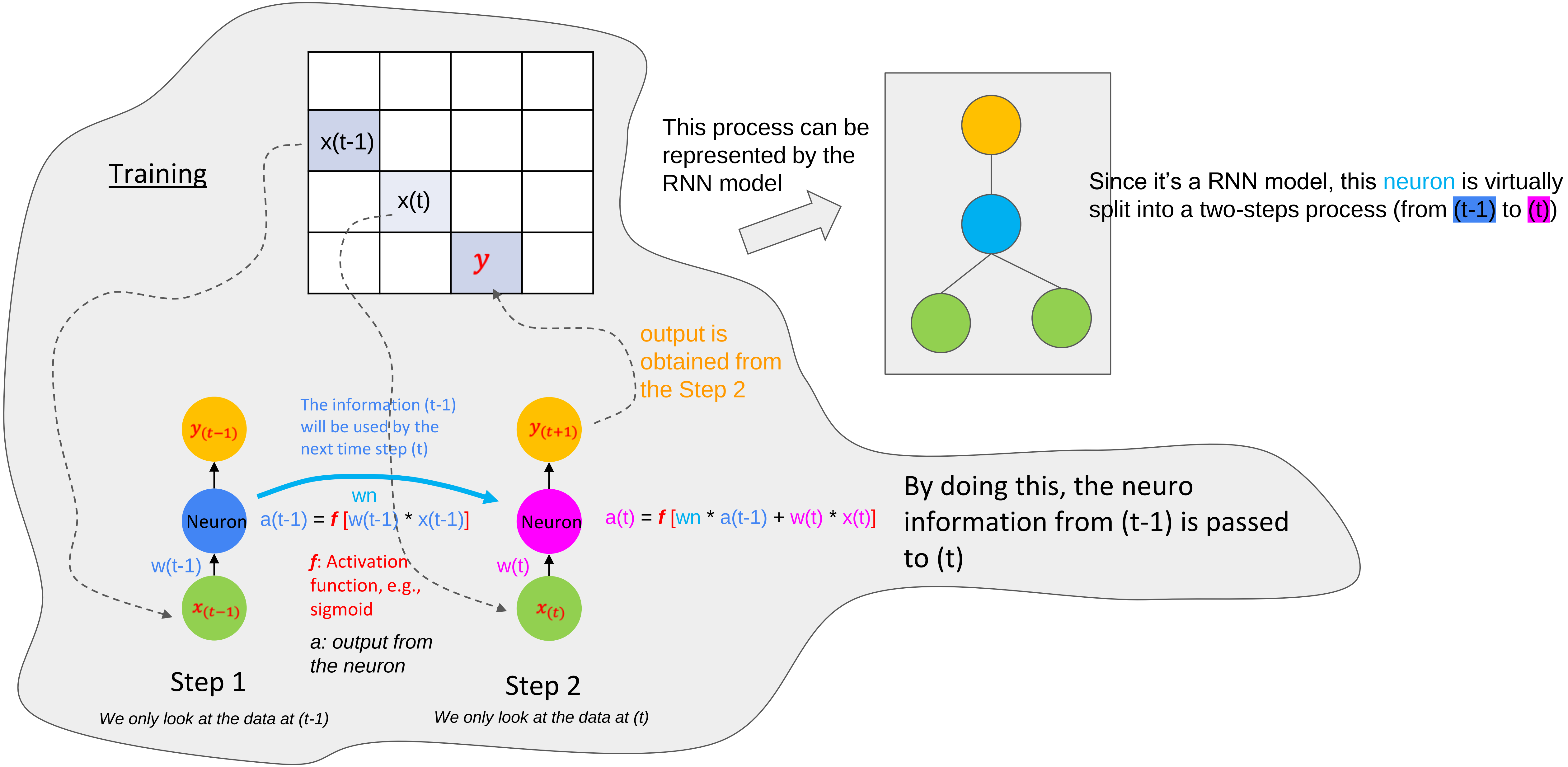
In order to make such a prediction right, we need a sequential of training dataset and model, e.g.,



By doing this, the neuro information from (t-1) is passed to (t)

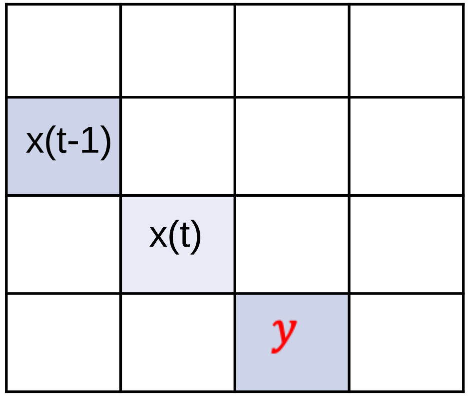
How RNN works?

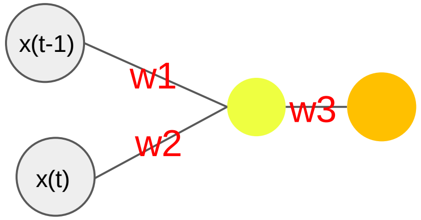
In order to make such a prediction right, we need a sequential of training dataset and model, e.g.,



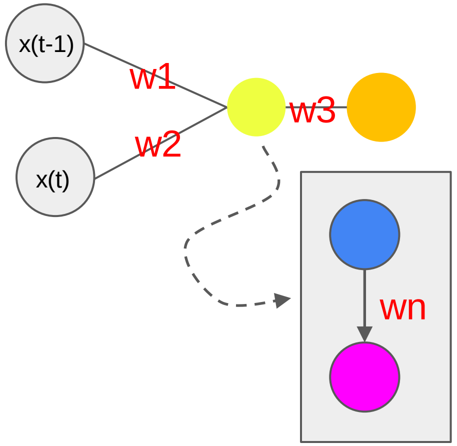
How RNN works?

Compared to an ANN model, RNN requires more parameters to be trained. To make it simpler, let's we only have 1 neuron here:





If it is an ANN model, we will have to train 3 parameters (w1, w2, and w3)



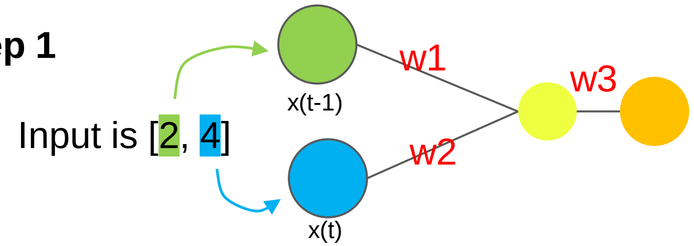
If it is a RNN model, in addition to the 3 parameters (w1, w2, and w3), there is an additional parameter to be trained (wn) to represent the information passing from (t-1) to (t)

How RNN works: a simple example

How RNN works?

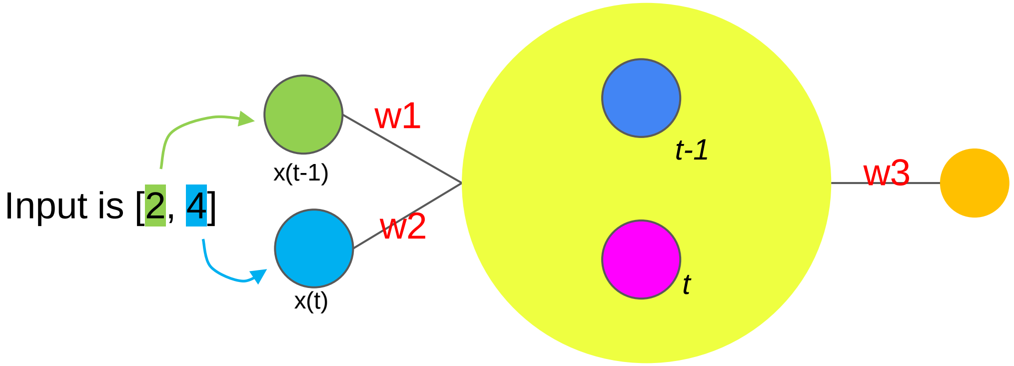
Let’s look at a real example, to make it simple, let’s say we only have one neuron

Step 1



Step 2

Since it’s an RNN, the neuron is split into a 2 steps process for (t-1) and (t)

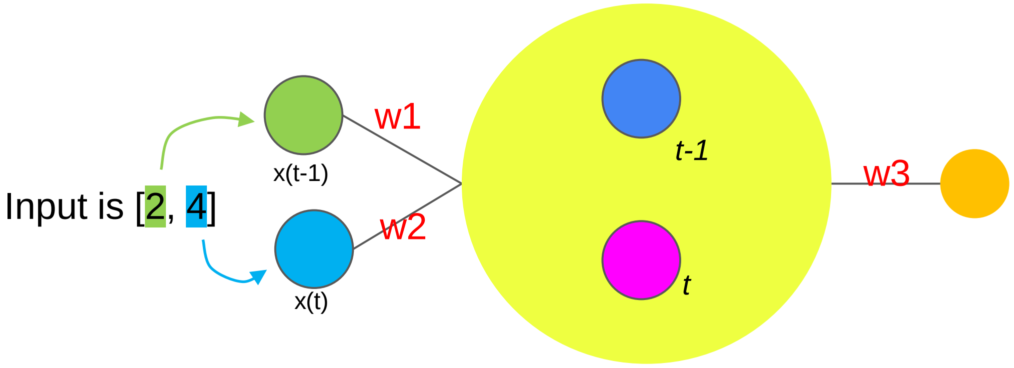


How RNN works?

Let’s look at a real example, to make it simple, let’s say we only have one neuron

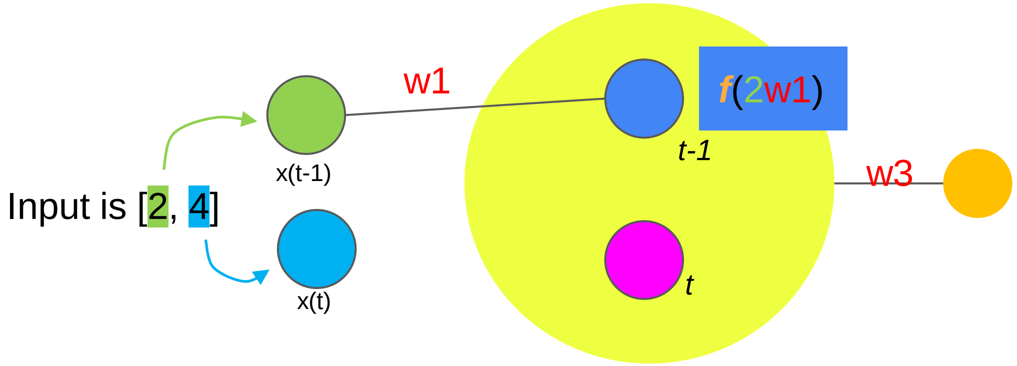
Step 2

Since it’s an RNN, the neuron is split into a 2 steps process for (t-1) and (t)



Step 3

The neuron output for (t-1) is *f*(2w1), where *f* is the activation function

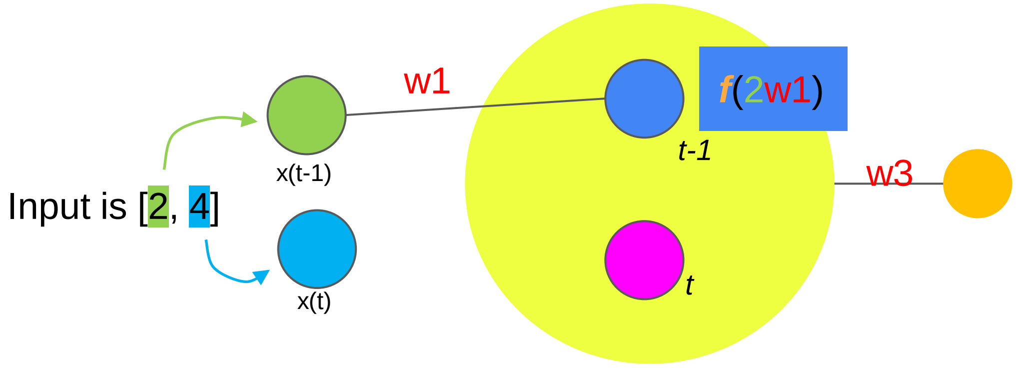


How RNN works?

Let’s look at a real example, to make it simple, let’s say we only have one neuron

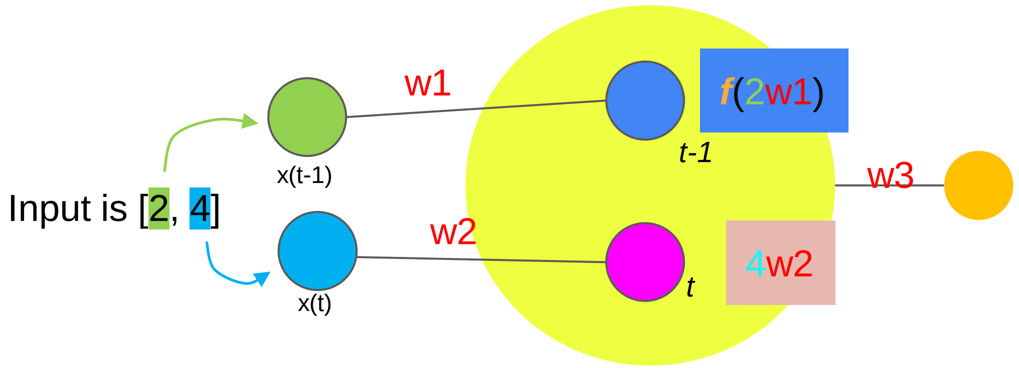
Step 3

The neuron output for (t-1) is *f*(2w1), where ***f*** is the activation function



Step 4

The intermediate neuron output (*without activation function and the information from (t-1)*) for (t) is

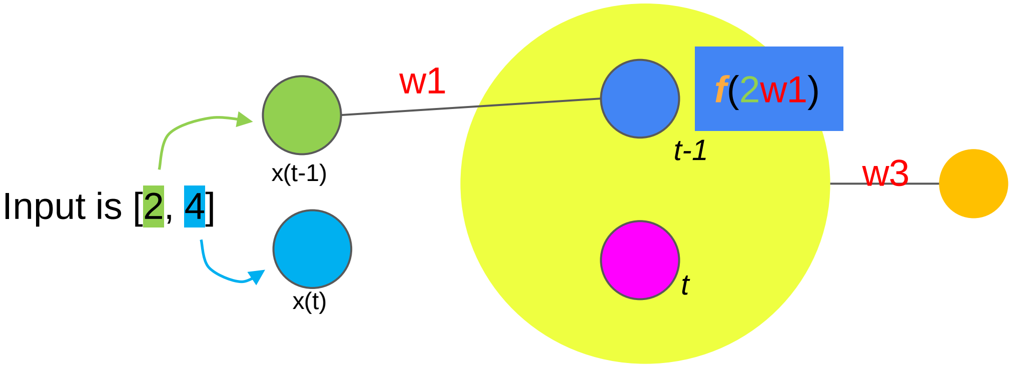


How RNN works?

Let’s look at a real example, to make it simple, let’s say we only have one neuron

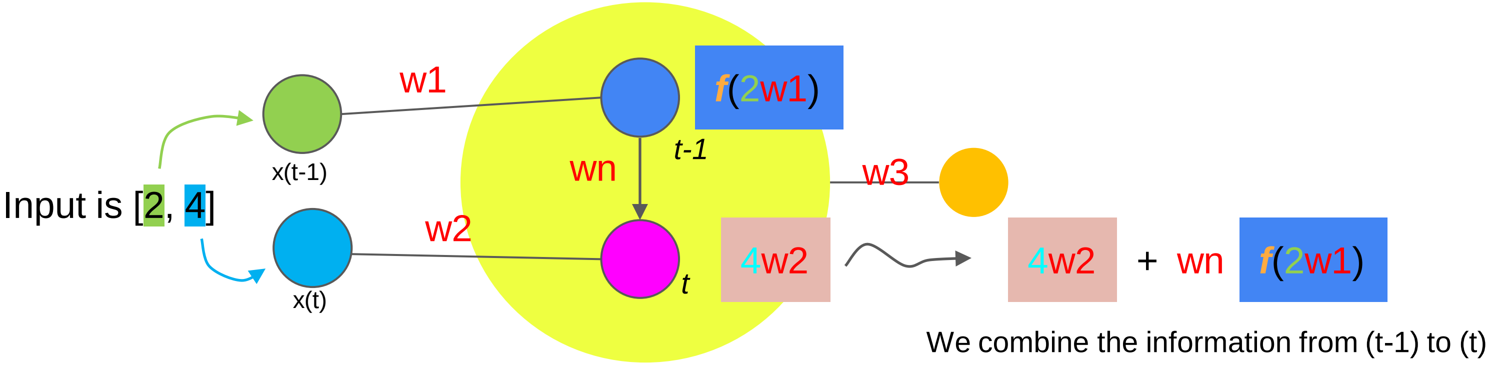
Step 3

The neuron output for (t-1) is *f*(2w1), where *f* is the activation function



Step 4

The intermediate neuron output (*without activation function and the information from (t-1)*) for (t) is

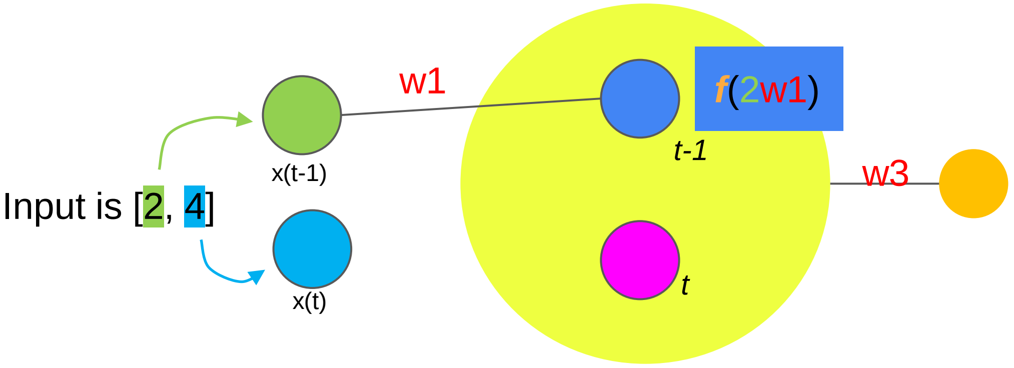


How RNN works?

Let's look at a real example, to make it simple, let's say we only have one neuron

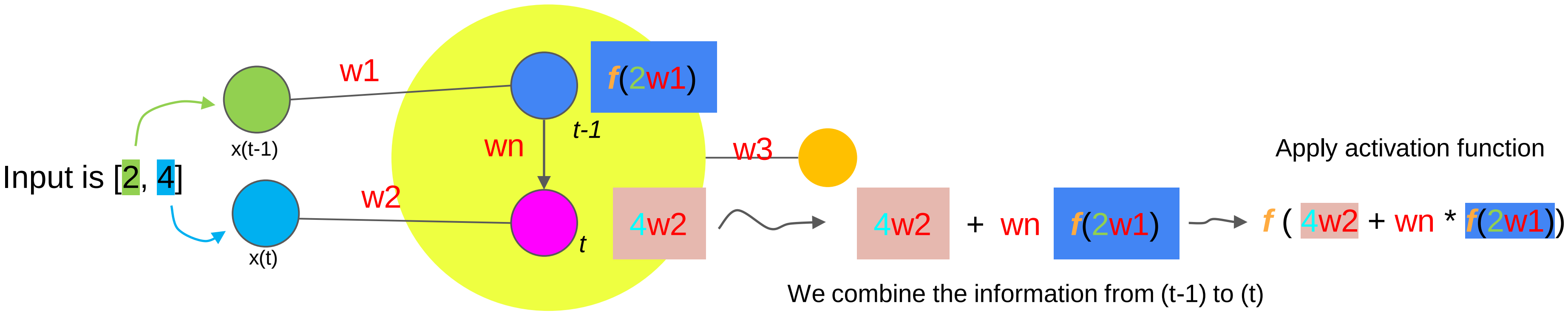
Step 3

The neuron output for (t-1) is *f*(2w1), where *f* is the activation function



Step 4

The intermediate neuron output (*without activation function and the information from (t-1)*) for (t) is

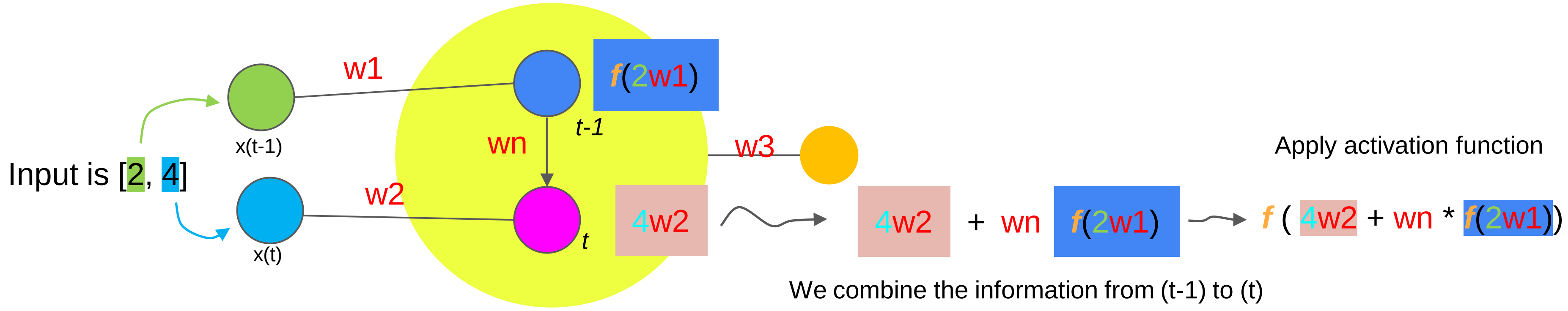


How RNN works?

Let's look at a real example, to make it simple, let's say we only have one neuron

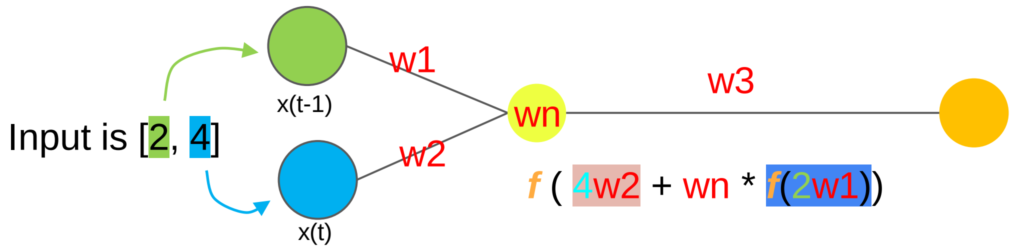
Step 4

The intermediate neuron output (*without activation function and the information from (t-1)*) for (t) is



Step 5

The neuron output there can be represented as

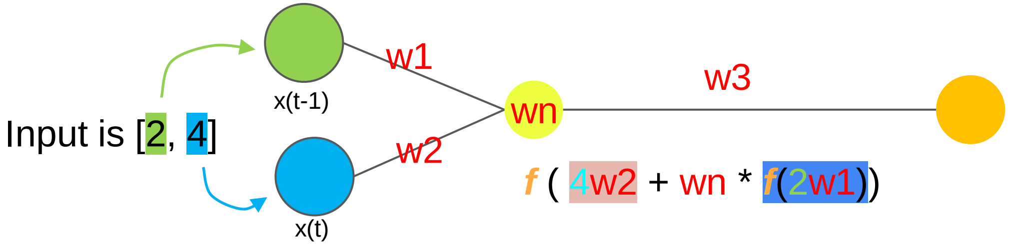


How RNN works?

Let’s look at a real example, to make it simple, let’s say we only have one neuron

Step 5

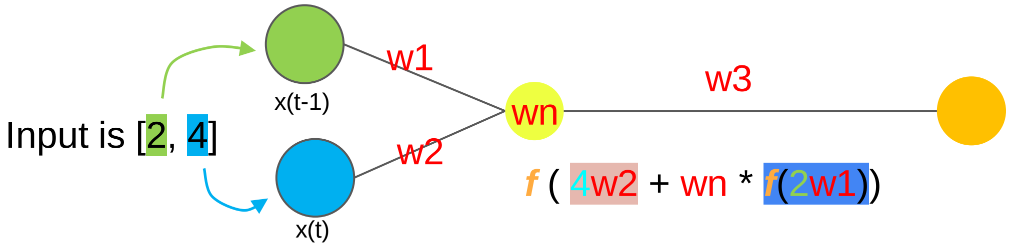
The neuron output there can be represented as



Step 6

So the final output can be calculated as: w3 * f (4w2 + wn * f(2w1))

Note: All the weights, w1, w2, w3 and wn are to be trained by the model

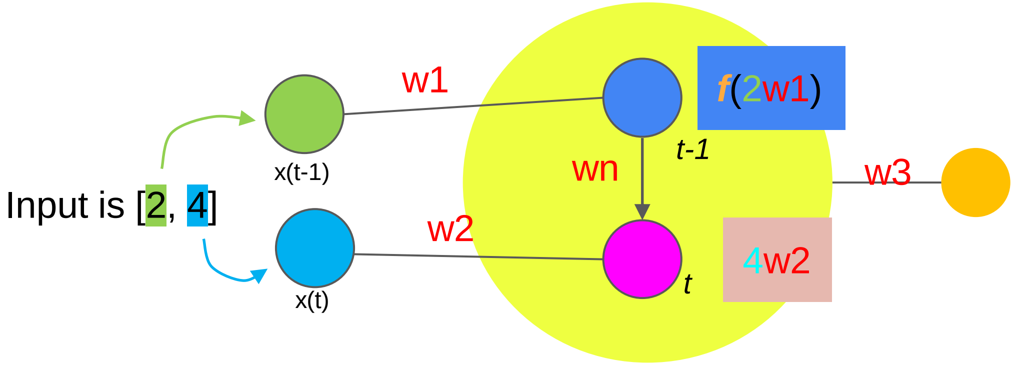


w3 * f (4w2 + wn * f(2w1))

Issues in RNN

The RNN issues

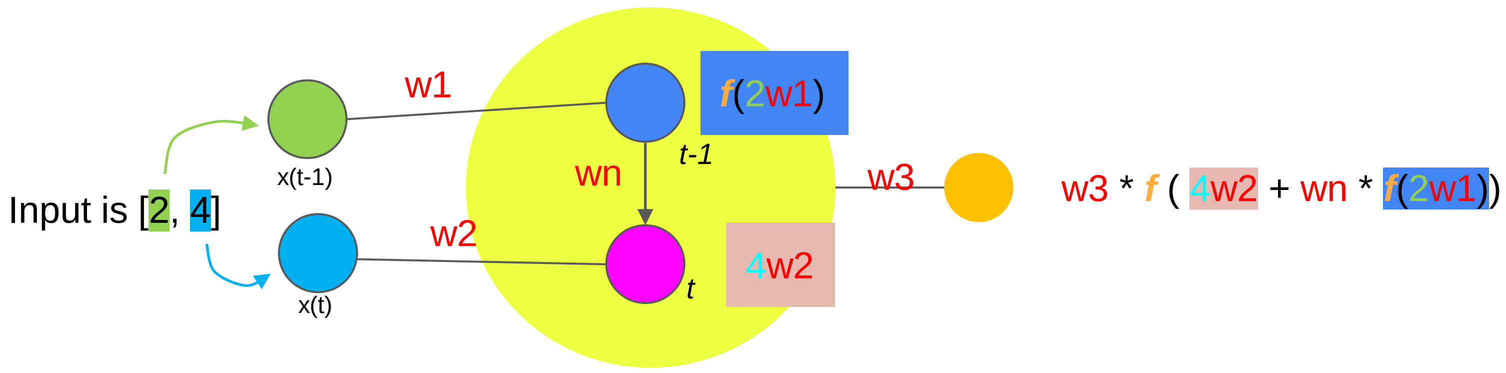
In the above example, we have two inputs (t-1) and (t)



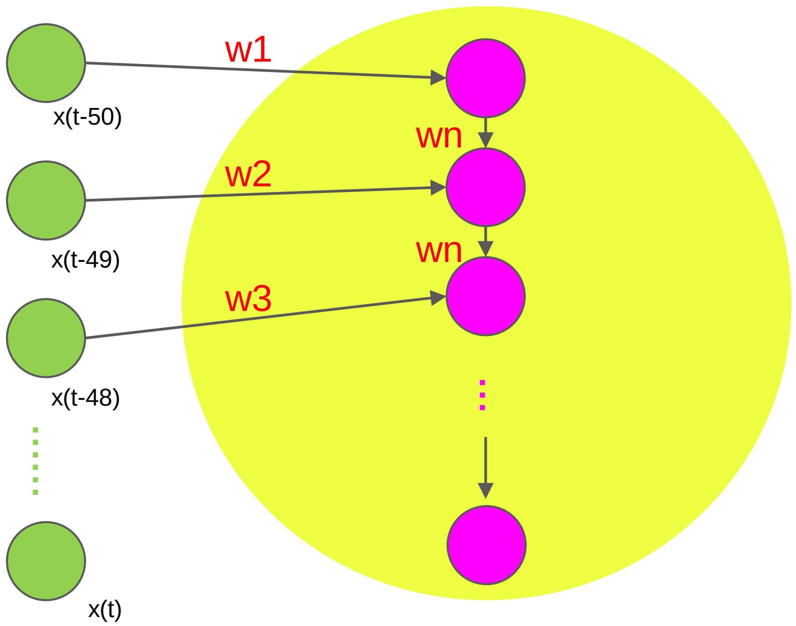
$w3 * f\,(4w2 + wn * f(2w1))$

The RNN issues

In the above example, we have two inputs (t-1) and (t)

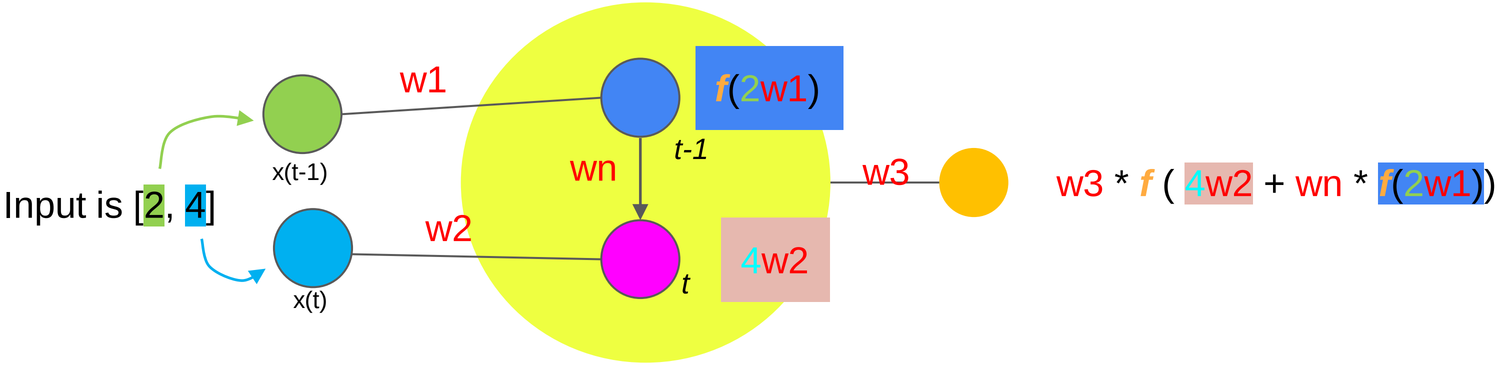


but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:



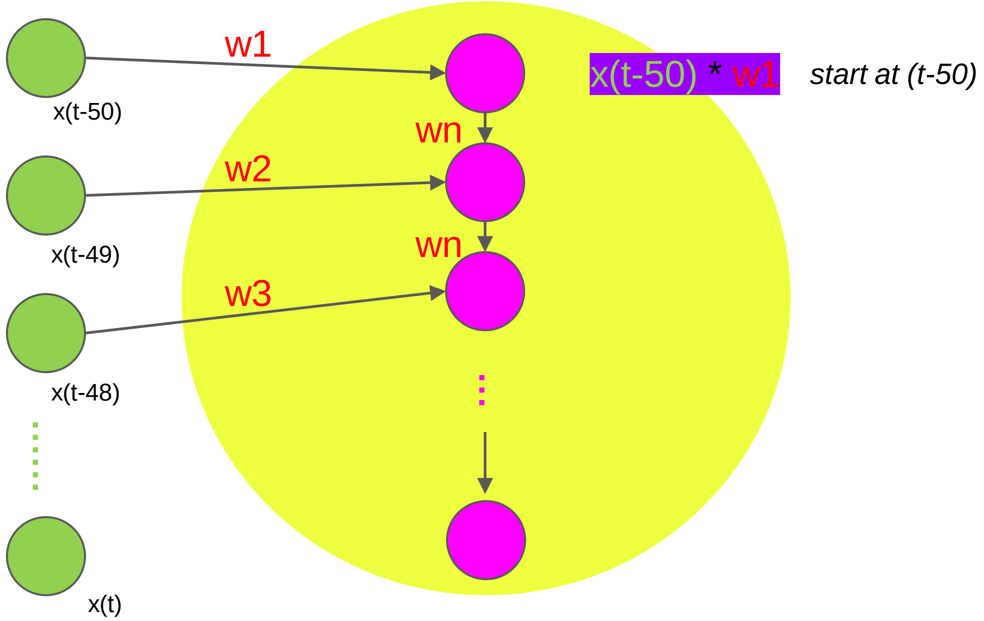
The RNN issues

In the above example, we have two inputs (t-1) and (t)



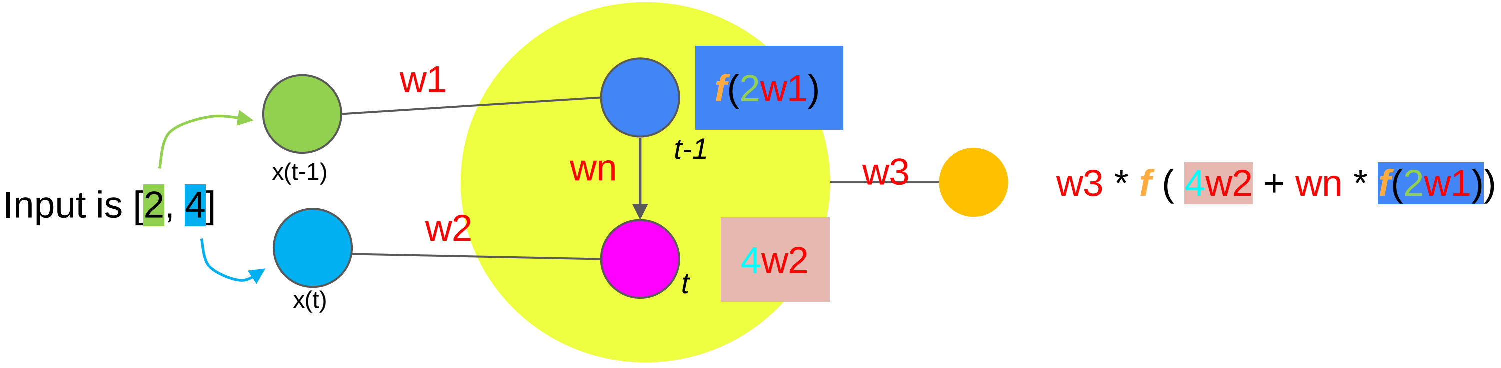
but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let's assume the activation function does not do anything, e.g., ***f(x)=x***, the output of a time step dependant neuron can be written as below:



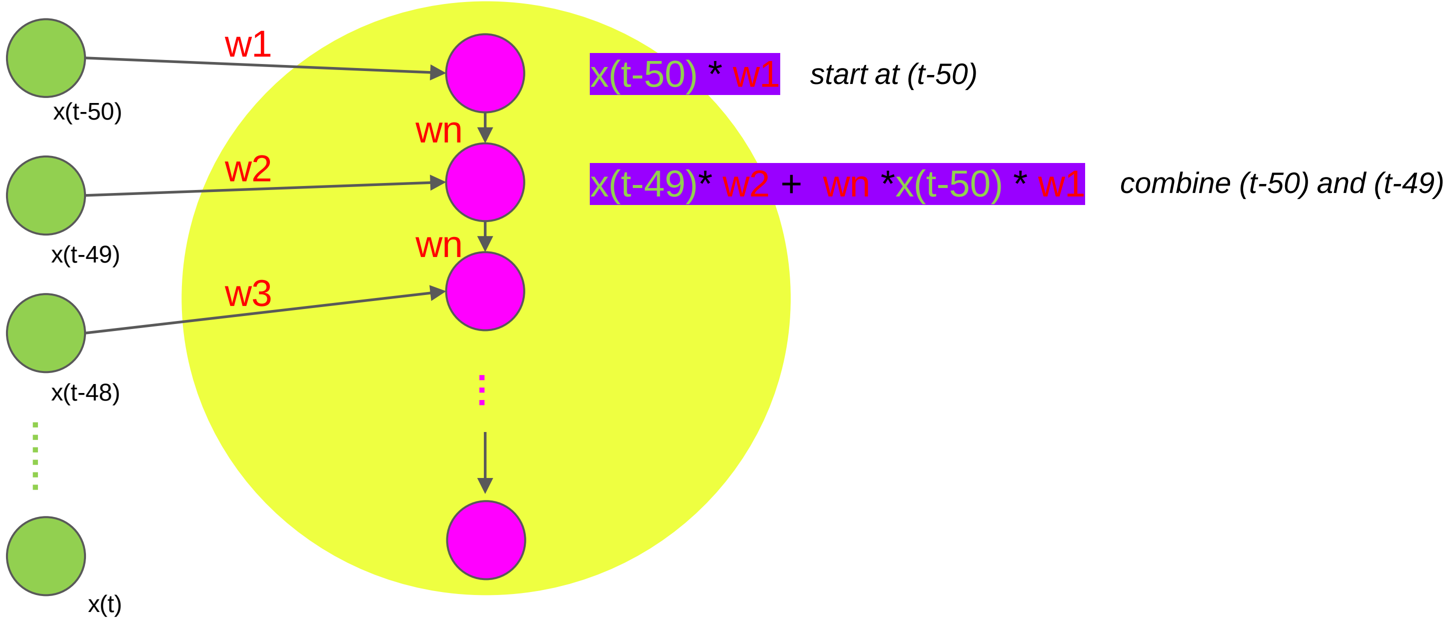
The RNN issues

In the above example, we have two inputs (t-1) and (t)



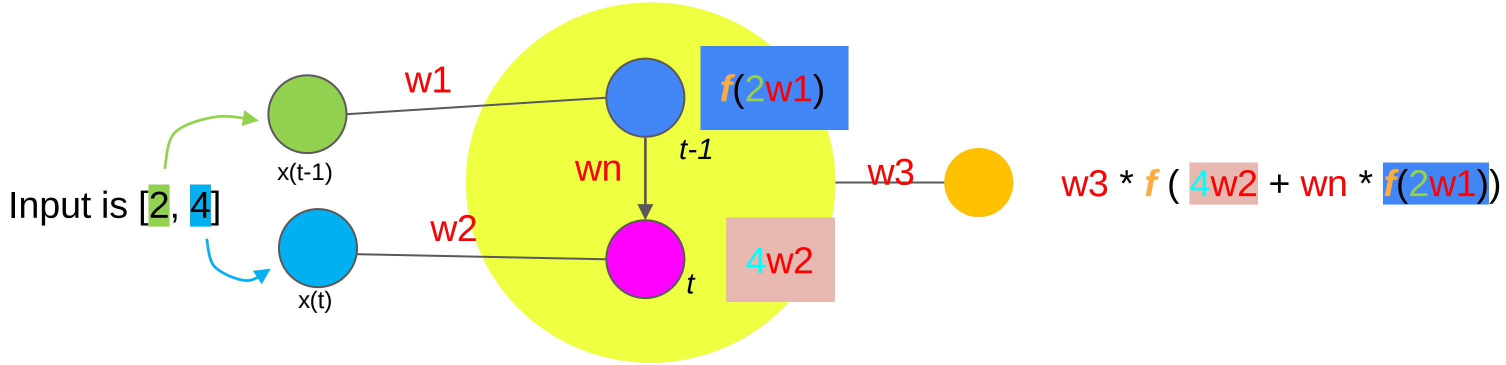
but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let's assume the activation function does not do anything, e.g., *f(x)=x*, the output of a time step dependant neuron can be written as below:



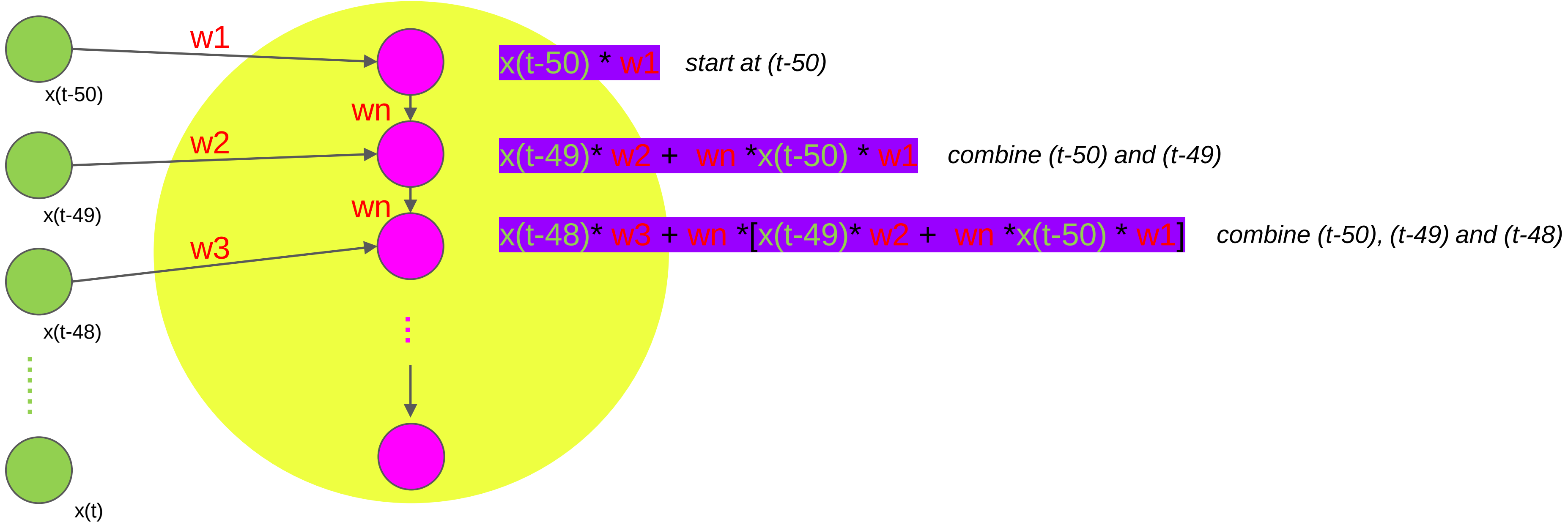
The RNN issues

In the above example, we have two inputs (t-1) and (t)



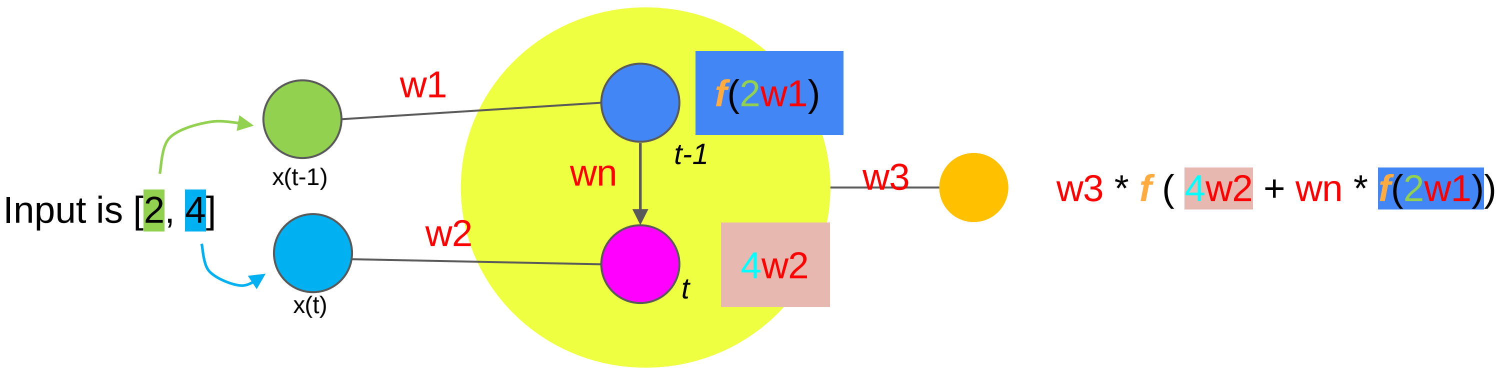
but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let’s assume the activation function does not do anything, e.g., *f(x)=x*, the output of a time step dependant neuron can be written as below:



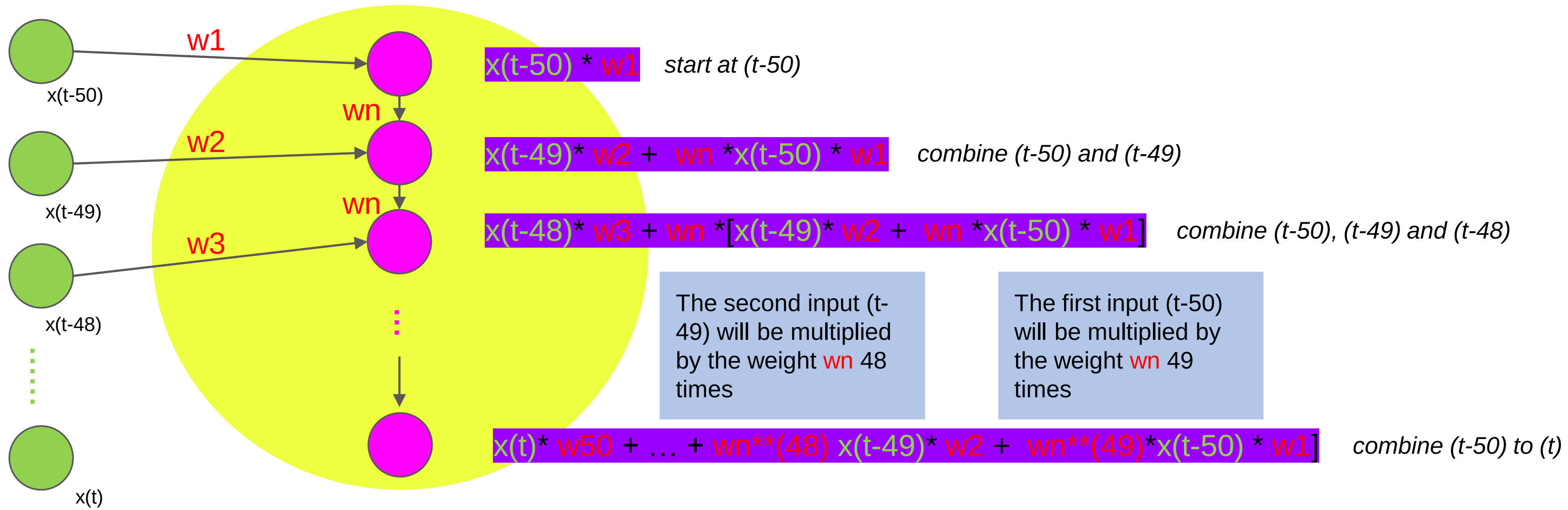
The RNN issues

In the above example, we have two inputs (t-1) and (t)



but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

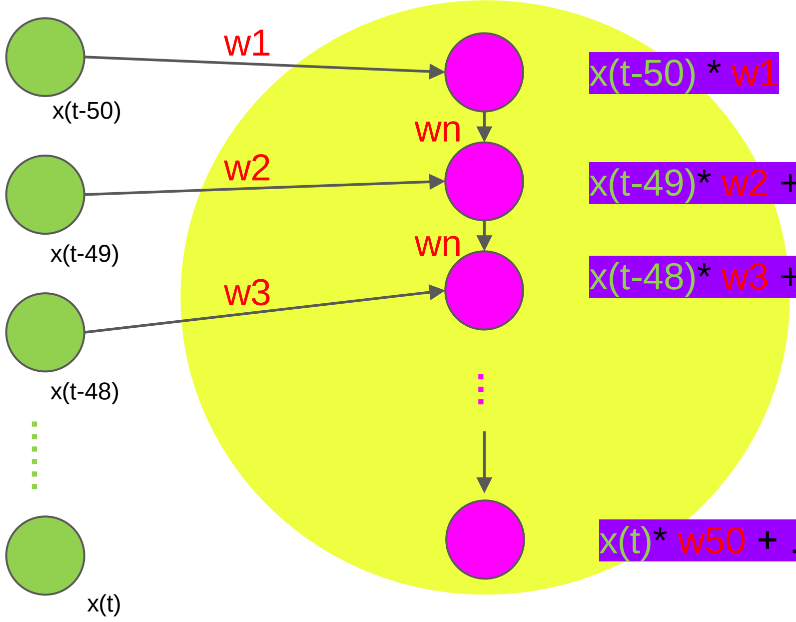
Let’s assume the activation function does not do anything, e.g., *f(x)=x*, the output of a time step dependant neuron can be written as below:



The RNN issues

but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let's assume the activation function does not do anything, e.g., ***f(x)=x***, the output of a time step dependant neuron can be written as below:



x(t-50) * w1 *start at (t-50)*

x(t-49)* w2 + wn *x(t-50) * w1 *combine (t-50) and (t-49)*

x(t-48)* w3 + wn *[x(t-49)* w2 + wn *x(t-50) * w1] *combine (t-50), (t-49) and (t-48)*

The second input (t-49) will be multiplied by the weight wn 48 times

The first input (t-50) will be multiplied by the weight wn 49 times

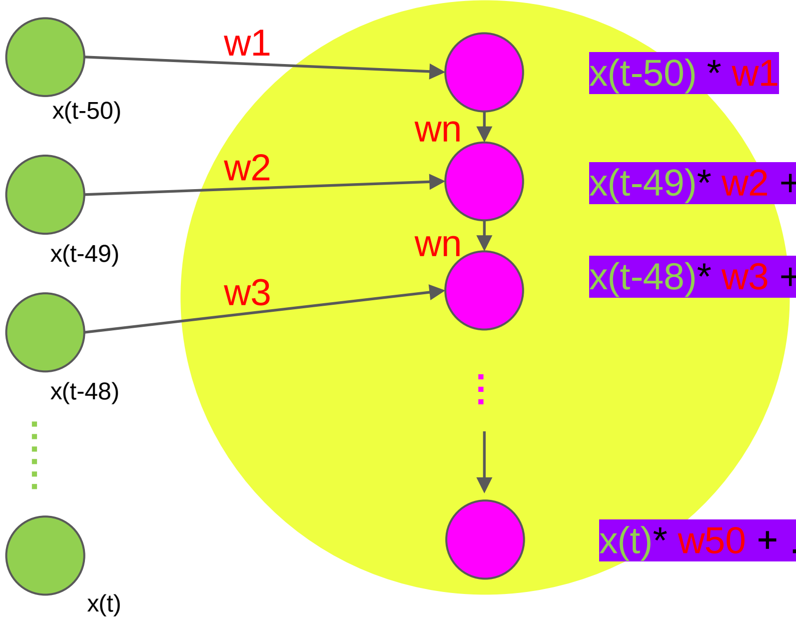
x(t)* w50 + … + wn**(48) x(t-49)* w2 + wn**(49)*x(t-50) * w1] *combine (t-50) to (t)*

When the weight wn is less than one, the older inputs, e.g., x(t-50) and x(t-49), will become extremely small, and therefore their impacts on the backpropagation cost function minimization will be negligible, we call this "***Vanishing gradient***"

The RNN issues

but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let's assume the activation function does not do anything, e.g., ***f(x)=x***, the output of a time step dependant neuron can be written as below:



x(t-50) * w1 *start at (t-50)*

x(t-49)* w2 + wn *x(t-50) * w1 *combine (t-50) and (t-49)*

x(t-48)* w3 + wn *[x(t-49)* w2 + wn *x(t-50) * w1] *combine (t-50), (t-49) and (t-48)*

The second input (t-49) will be multiplied by the weight wn 48 times

The first input (t-50) will be multiplied by the weight wn 49 times

x(t)* w50 + … + wn**(48) x(t-49)* w2 + wn**(49)*x(t-50) * w1] *combine (t-50) to (t)*

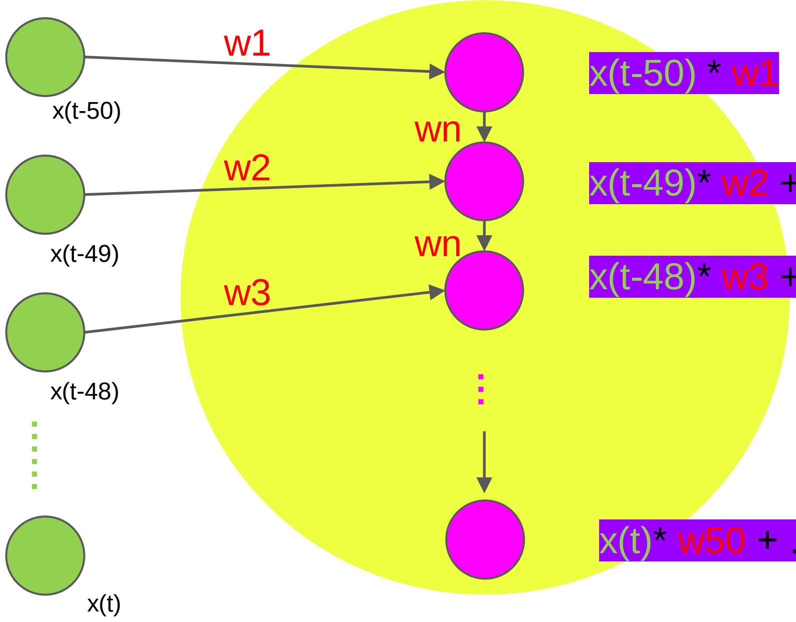
When the weight wn is less than one, the older inputs, e.g., x(t-50) and x(t-49), will become extremely small, and therefore their impacts on the backpropagation cost function minimization will be negligible, we call this "***Vanishing gradient***"

So RNN tends to forget any older inputs and will only remember the recent inputs

The RNN issues

but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let's assume the activation function does not do anything, e.g., ***f(x)=x***, the output of a time step dependant neuron can be written as below:



x(t-50) * w1 *start at (t-50)*

x(t-49)* w2 + wn *x(t-50) * w1 *combine (t-50) and (t-49)*

x(t-48)* w3 + wn *[x(t-49)* w2 + wn *x(t-50) * w1] *combine (t-50), (t-49) and (t-48)*

The second input (t-49) will be multiplied by the weight wn 48 times

The first input (t-50) will be multiplied by the weight wn 49 times

x(t)* w50 + … + wn**(48) x(t-49)* w2 + wn**(49)*x(t-50) * w1] *combine (t-50) to (t)*

When the weight wn is less than one, the older inputs, e.g., x(t-50) and x(t-49), will become extremely small, and therefore their impacts on the backpropagation cost function minimization will be negligible, we call this "***Vanishing gradient***"

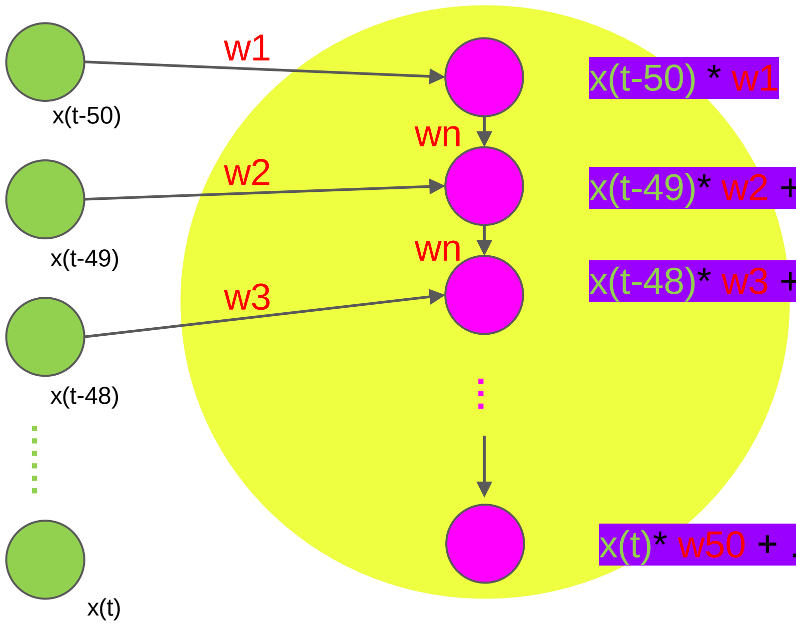
When the weight wn is bigger than one, the older inputs, e.g., x(t-50) and x(t-49), will become extremely big, and therefore their impacts on the backpropagation cost function minimization will dominate, we call this "***Exploding gradient***"

So RNN tends to forget any older inputs and will only remember the recent inputs

The RNN issues

but if we have a very long input lists, e.g., (t-50) to (t), even we still just have one neuro, we will have:

Let's assume the activation function does not do anything, e.g., ***f(x)=x***, the output of a time step dependant neuron can be written as below:



x(t-50) * w1 *start at (t-50)*

x(t-49)* w2 + wn *x(t-50) * w1 *combine (t-50) and (t-49)*

x(t-48)* w3 + wn *[x(t-49)* w2 + wn *x(t-50) * w1] *combine (t-50), (t-49) and (t-48)*

The second input (t-49) will be multiplied by the weight wn 48 times

The first input (t-50) will be multiplied by the weight wn 49 times

x(t)* w50 + … + wn**(48) x(t-49)* w2 + wn**(49)*x(t-50) * w1] *combine (t-50) to (t)*

When the weight wn is less than one, the older inputs, e.g., x(t-50) and x(t-49), will become extremely small, and therefore their impacts on the backpropagation cost function minimization will be negligible, we call this "***Vanishing gradient***"

So RNN tends to forget any older inputs and will only remember the recent inputs

When the weight wn is bigger than one, the older inputs, e.g., x(t-50) and x(t-49), will become extremely big, and therefore their impacts on the backpropagation cost function minimization will dominate, we call this "***Exploding gradient***"

So RNN tends to forget recent inputs and will only remember the old stuff