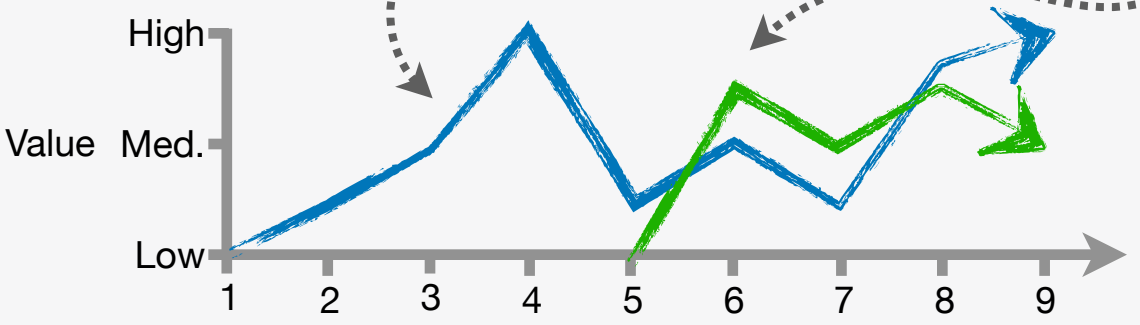
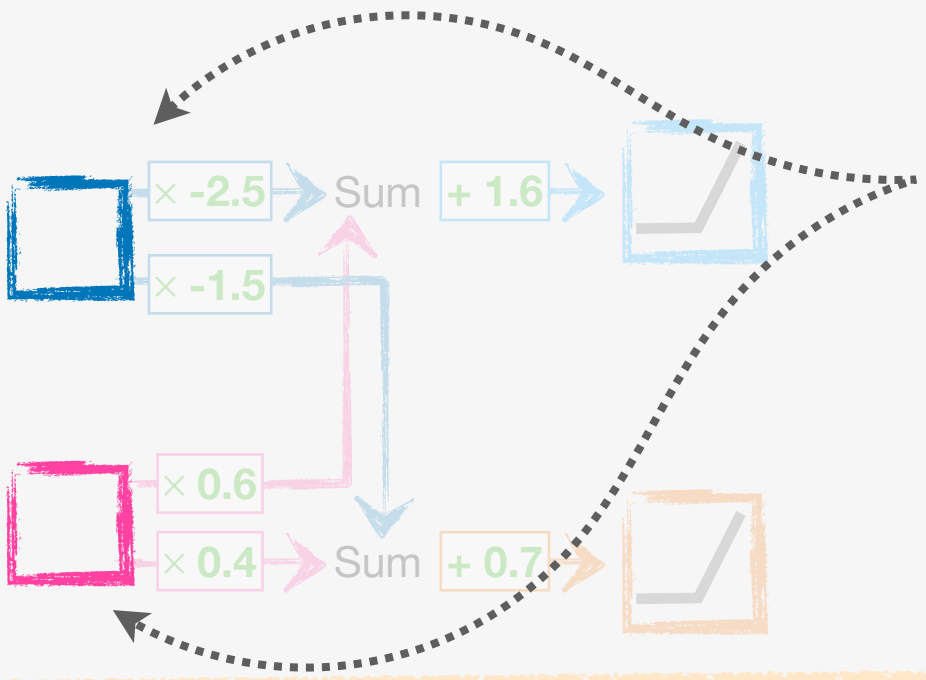
Stock Prediction with Recurrent Neural Networks (RNNs)!!!

Recurrent Neural Networks: Main Ideas

1 **A Problem:** Someone wants to invest some **StatBucks**, but he doesn't know if he should invest them in the **blue company**, which has **9** days of trading data…

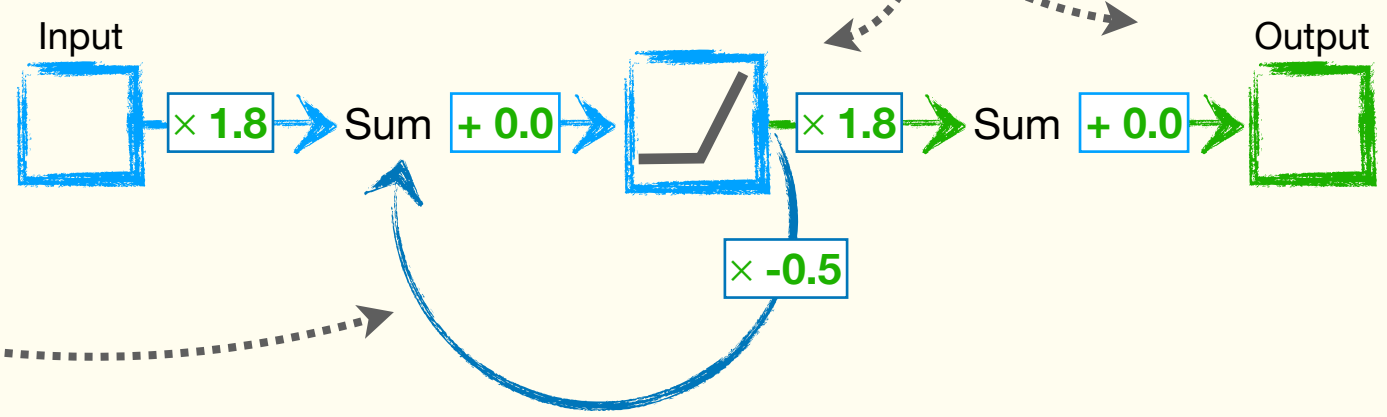
…or the **green company**, which only has **5** days of data.





2 **A Solution:** We can build a **Recurrent Neural Network (RNN)**, which can handle different amounts of input data.

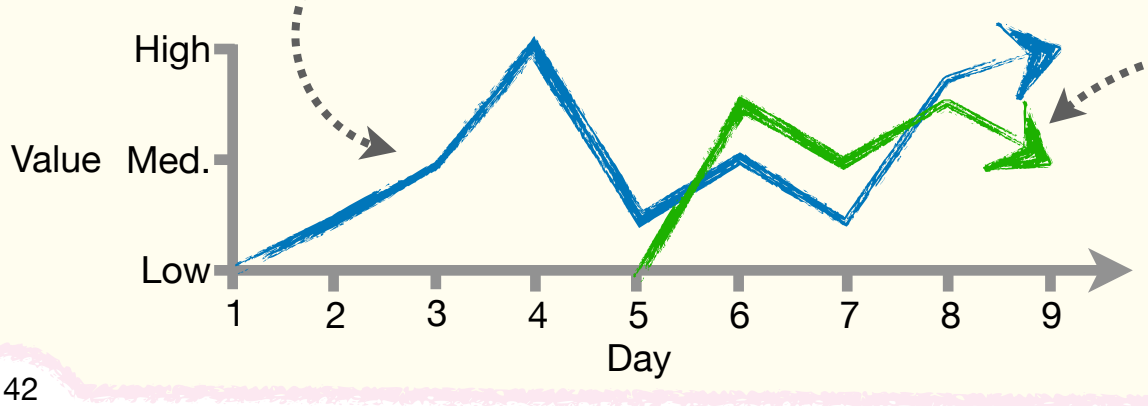
The one thing that makes an RNN different from the neural networks that we've seen so far is that RNNs have a **feedback loop**.



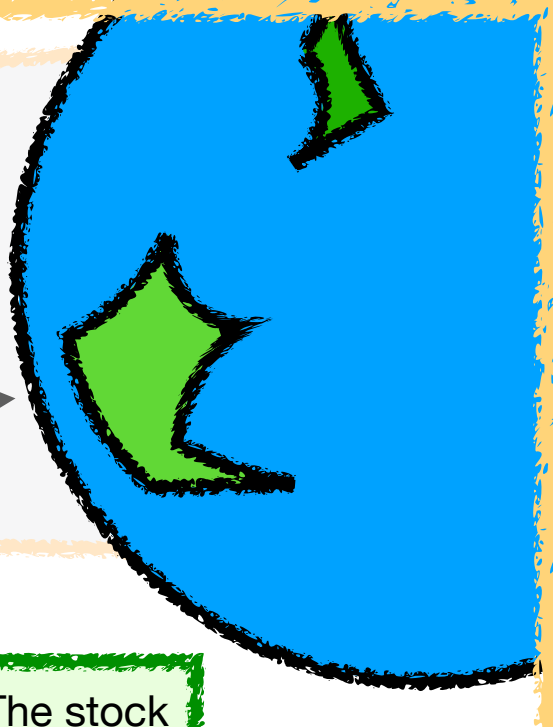
The feedback loop allows the RNN to use sequential input values, so if we have **9** data points, like we do for the **blue company**, we just use the feedback loop **9** times.

And if we have **5** data points, like we do for the **green company**, we just use the feedback loop **5** times.

Now let's see how an RNN can handle different amounts of sequential data, one step at a time!!!

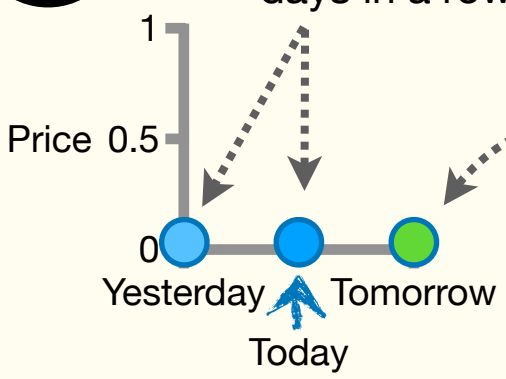


RNNs: Step-by-Step



1 Real stock market datasets are super complicated, and we'd probably get in a lot of trouble if we offered advice on how to make money with them. So, for this example, we're going to use a dataset from **StatLand**, where things are much simpler and there are far fewer lawyers.

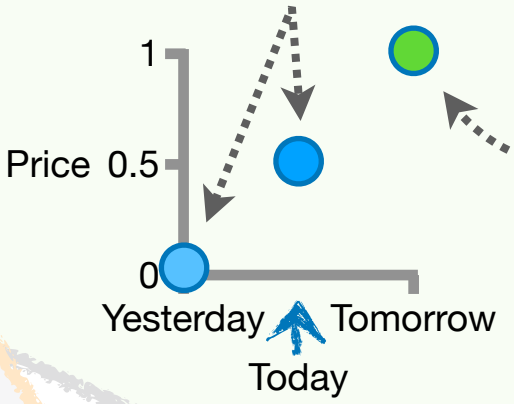
2 In **StatLand**, if the price of a stock is relatively low for two days in a row…



…then, more often than not, the price remains low on the next day.

NOTE: The stock prices are scaled to go from **0** to **1**.

3 In contrast, if yesterday's price was low and today's price is medium…

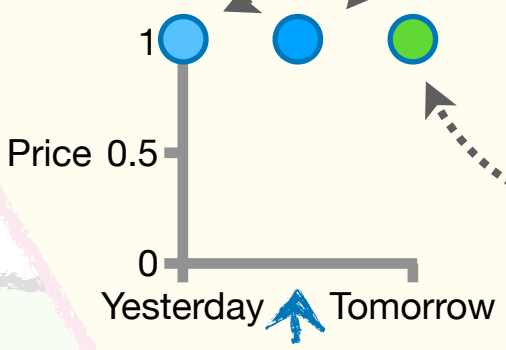


…then tomorrow's price should be even higher.

4 And when the price decreases from high to medium…

…then tomorrow's price will be even lower.

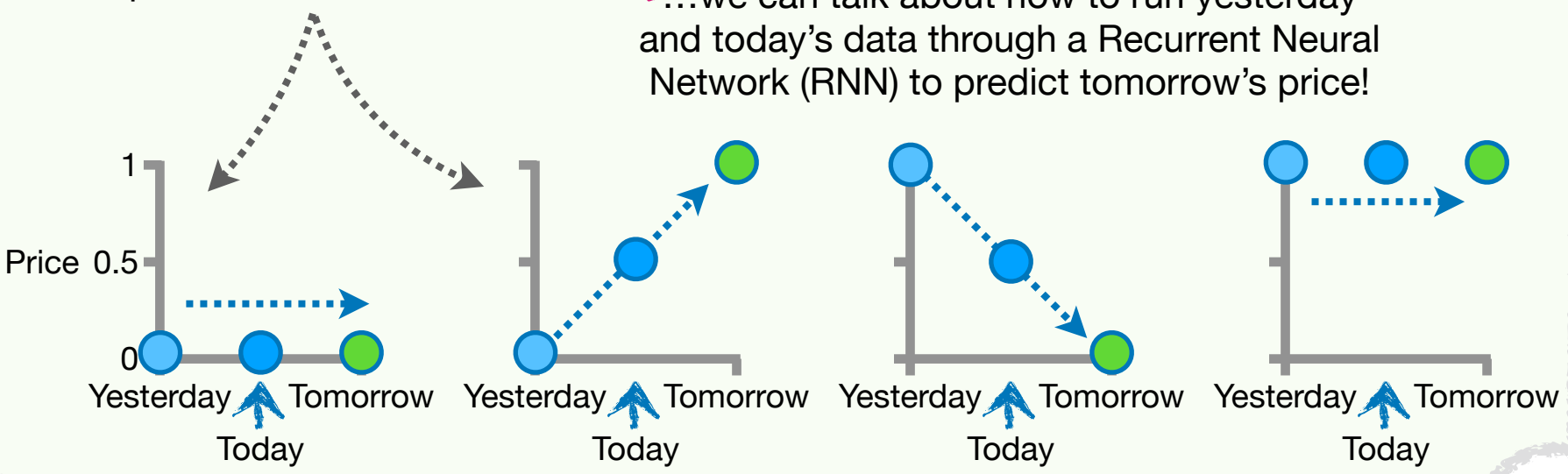
5 Lastly, if the price stays high for two days in a row…



…then the price will be high tomorrow.

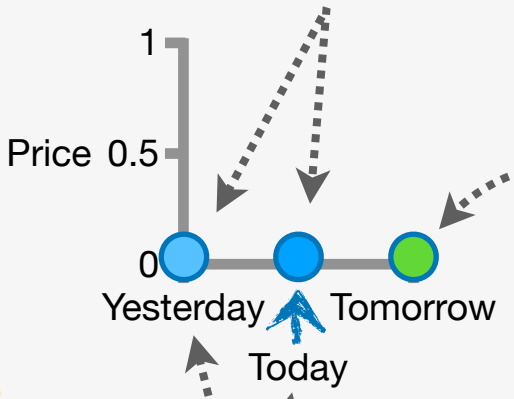
6 Now that we see the general trends in stock prices in **StatLand**…

…we can talk about how to run yesterday and today's data through a Recurrent Neural Network (RNN) to predict tomorrow's price!



RNNs: Step-by-Step

7 We'll start by running two days of low-priced stock through the RNN…

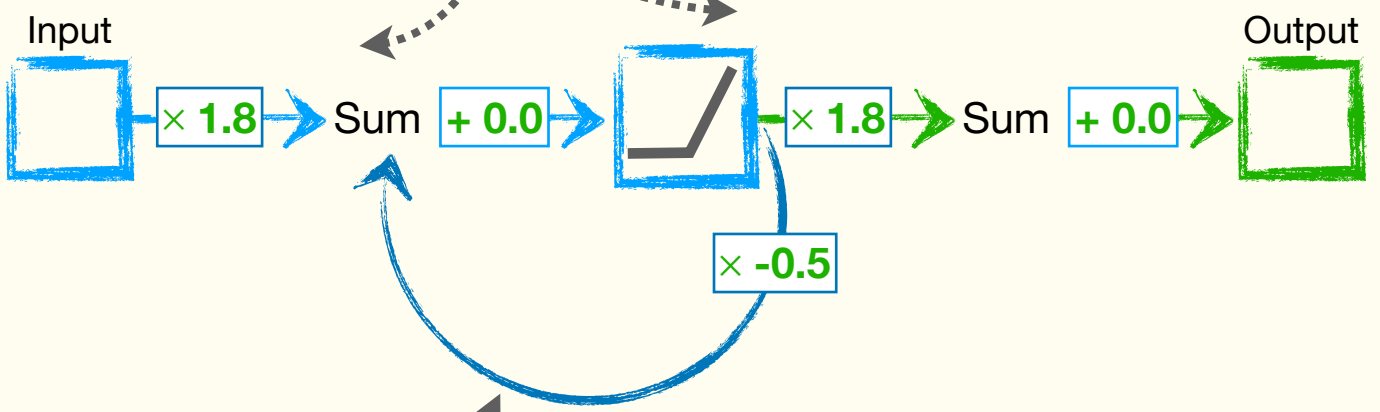


…to see if it can correctly predict tomorrow's low price.

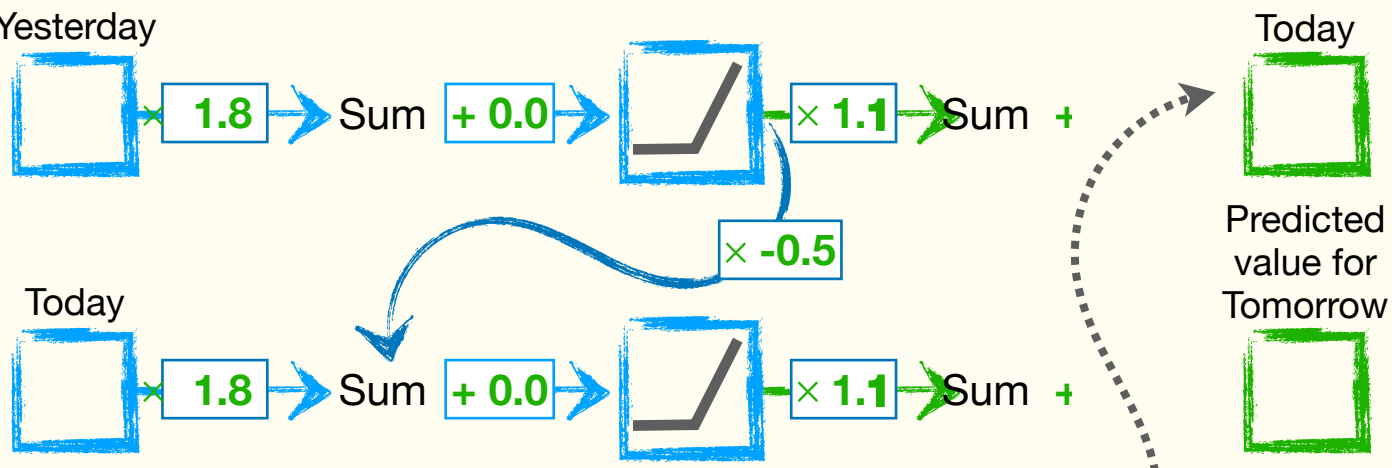
8 Now, because we have two input values, one for yesterday and one for today, we can sequentially run them through this RNN…

…but keeping track of the math with the feedback loop is a real headache.

The good news is that instead of getting a headache, the feedback loop allows us to **unroll** the RNN so that we have a network with two inputs…



…one for yesterday and one for today.



Unrolling this RNN also gives us two outputs, one that predicts today's value, which we don't need since we already know it, and one that predicts tomorrow's value, which is what we're interested in.

RNNs: Step-by-Step

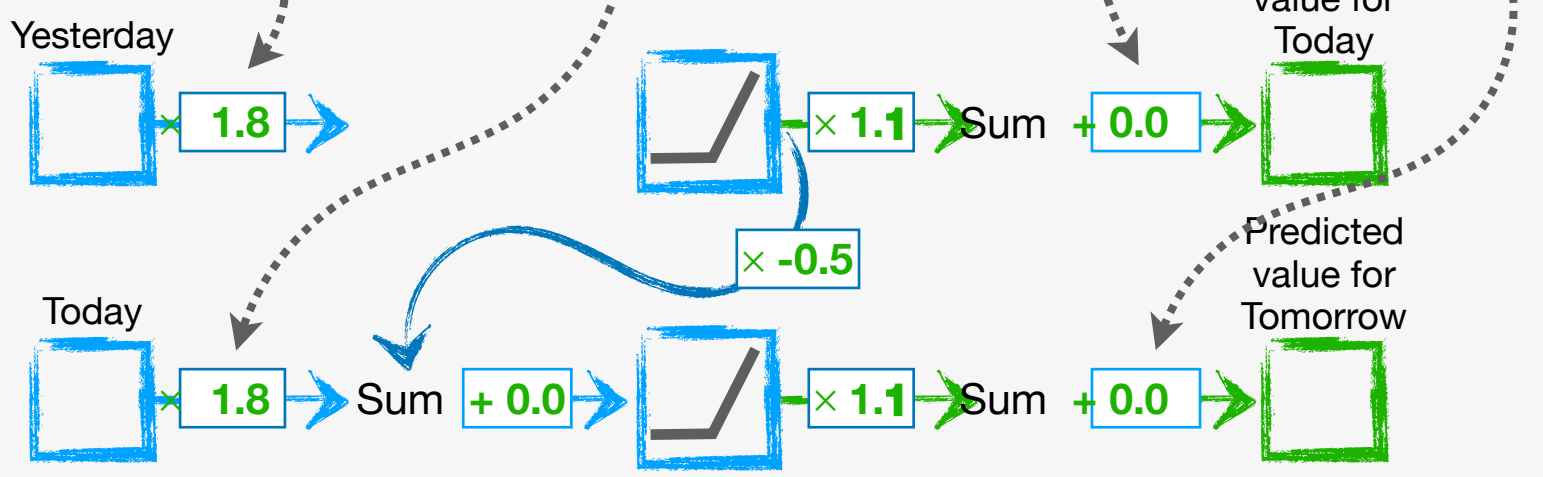
9 When we unroll an RNN, each copy shares the same weights and biases.

In other words, updating this weight with backpropagation…

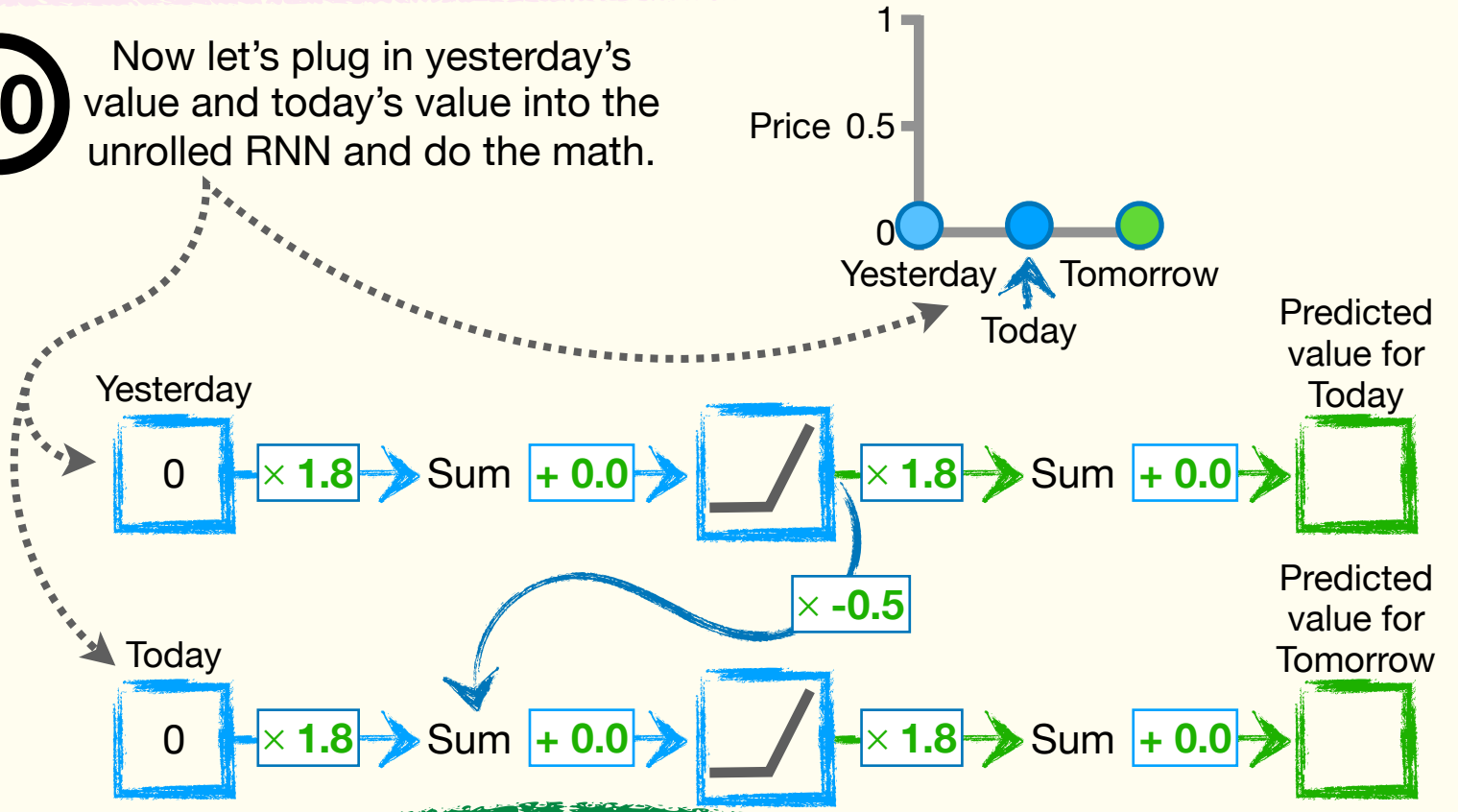
…also updates this weight to the exact same value.

And updating this bias with backpropagation…

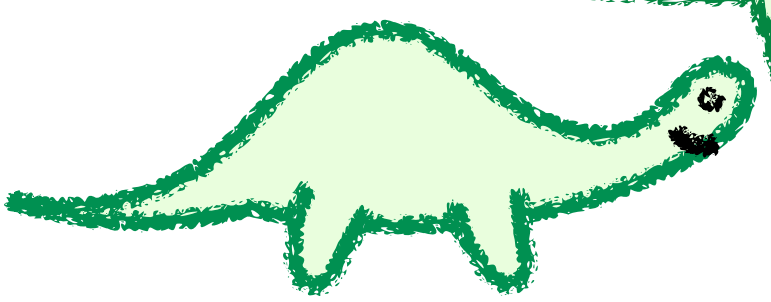
…also updates this bias to the exact same value.



10 Now let's plug in yesterday's value and today's value into the unrolled RNN and do the math.

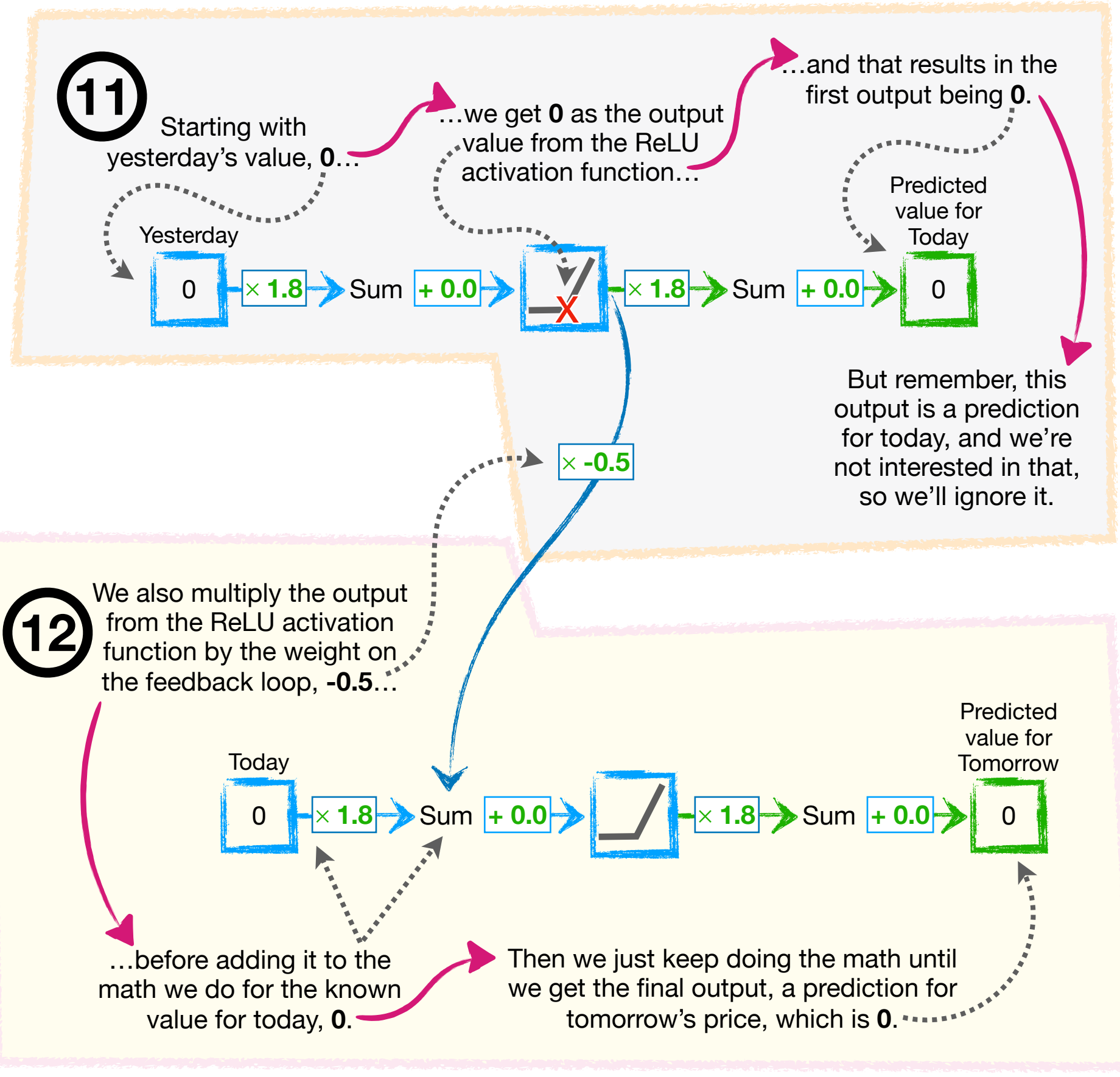


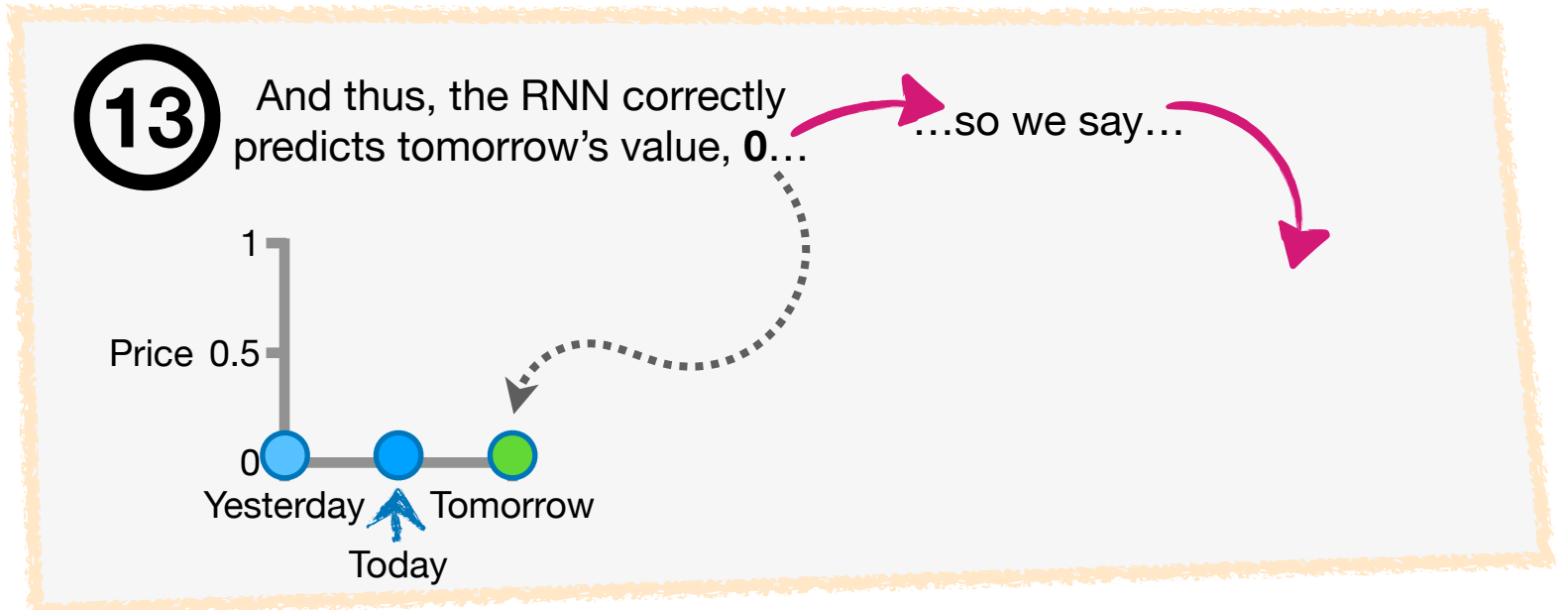
In this example, we're applying an RNN to stock market data, but RNNs can be applied to any sequential data, like spoken words for speech recognition.



Very cool!

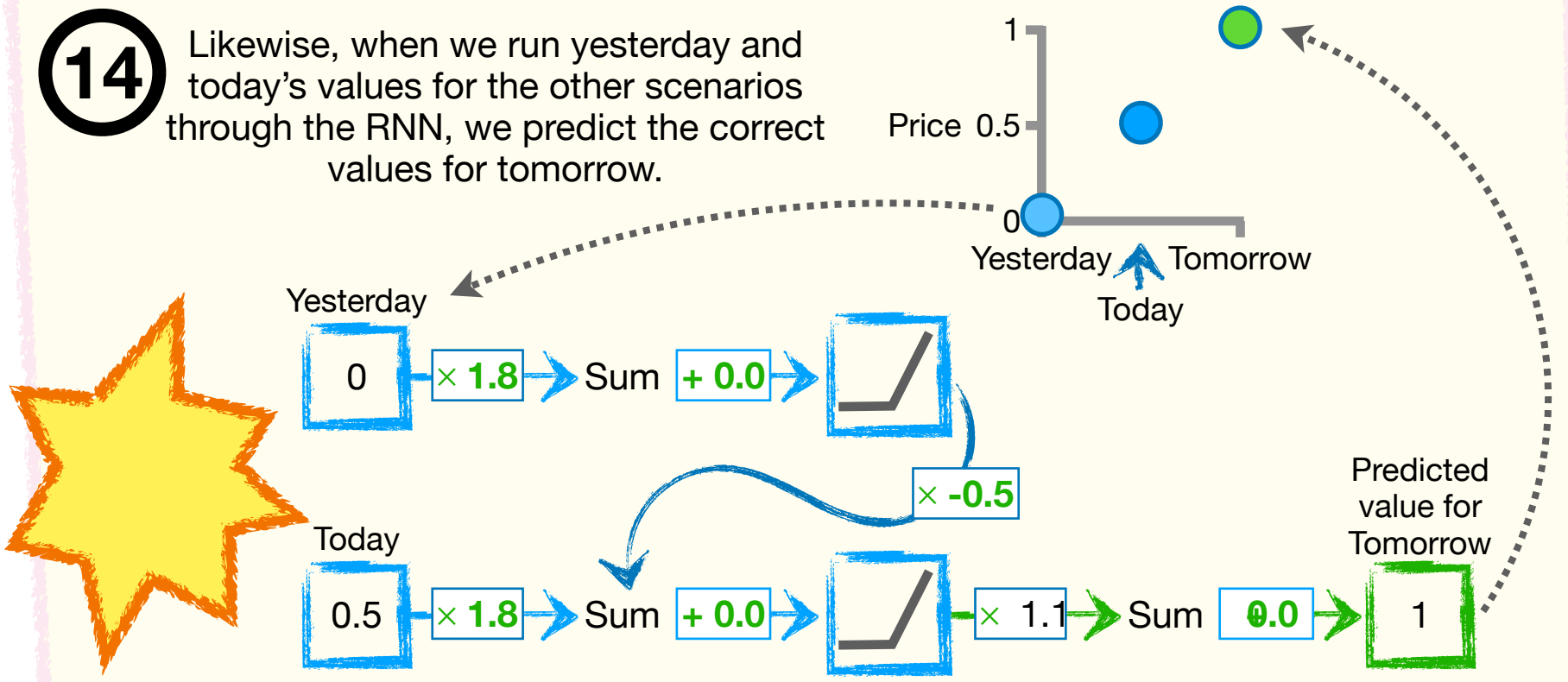
RNNs: Step-by-Step

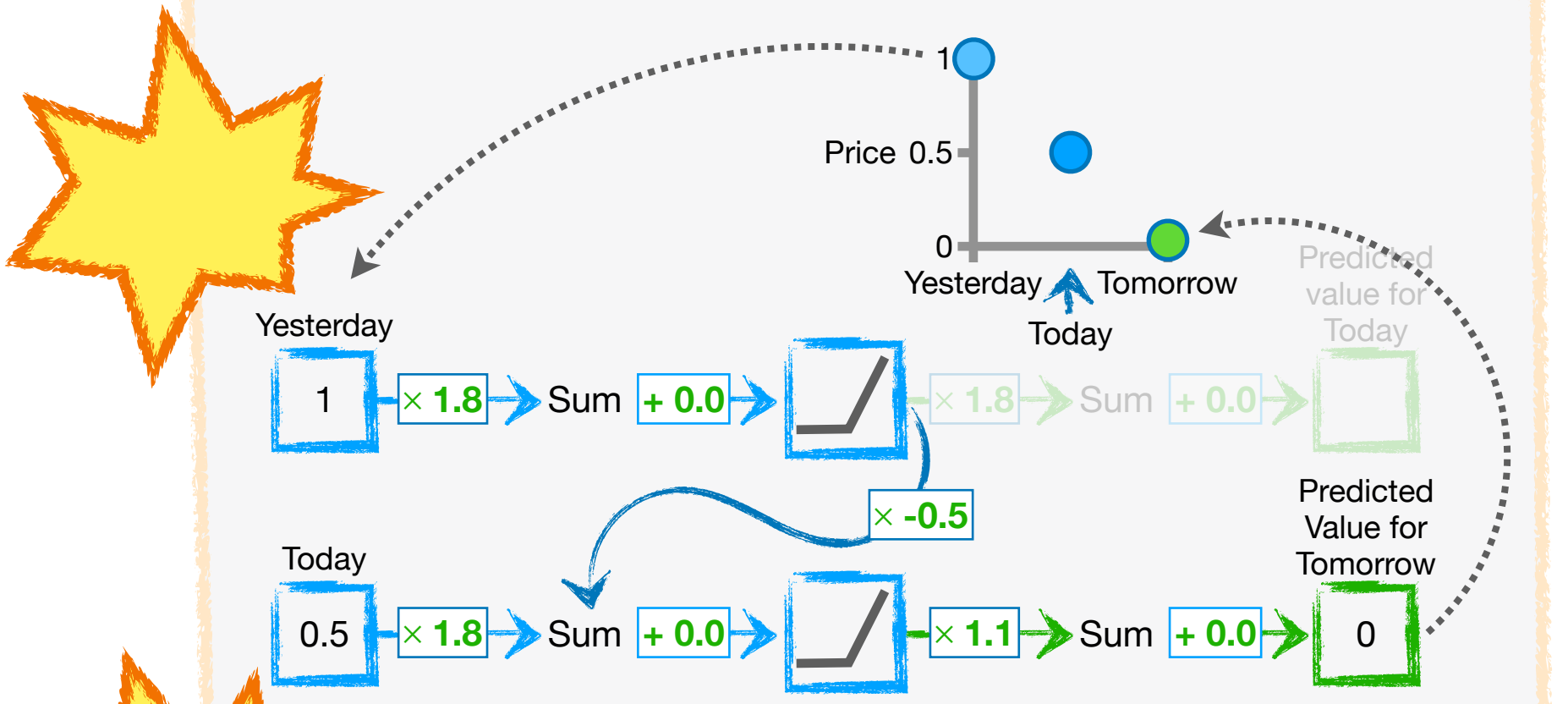


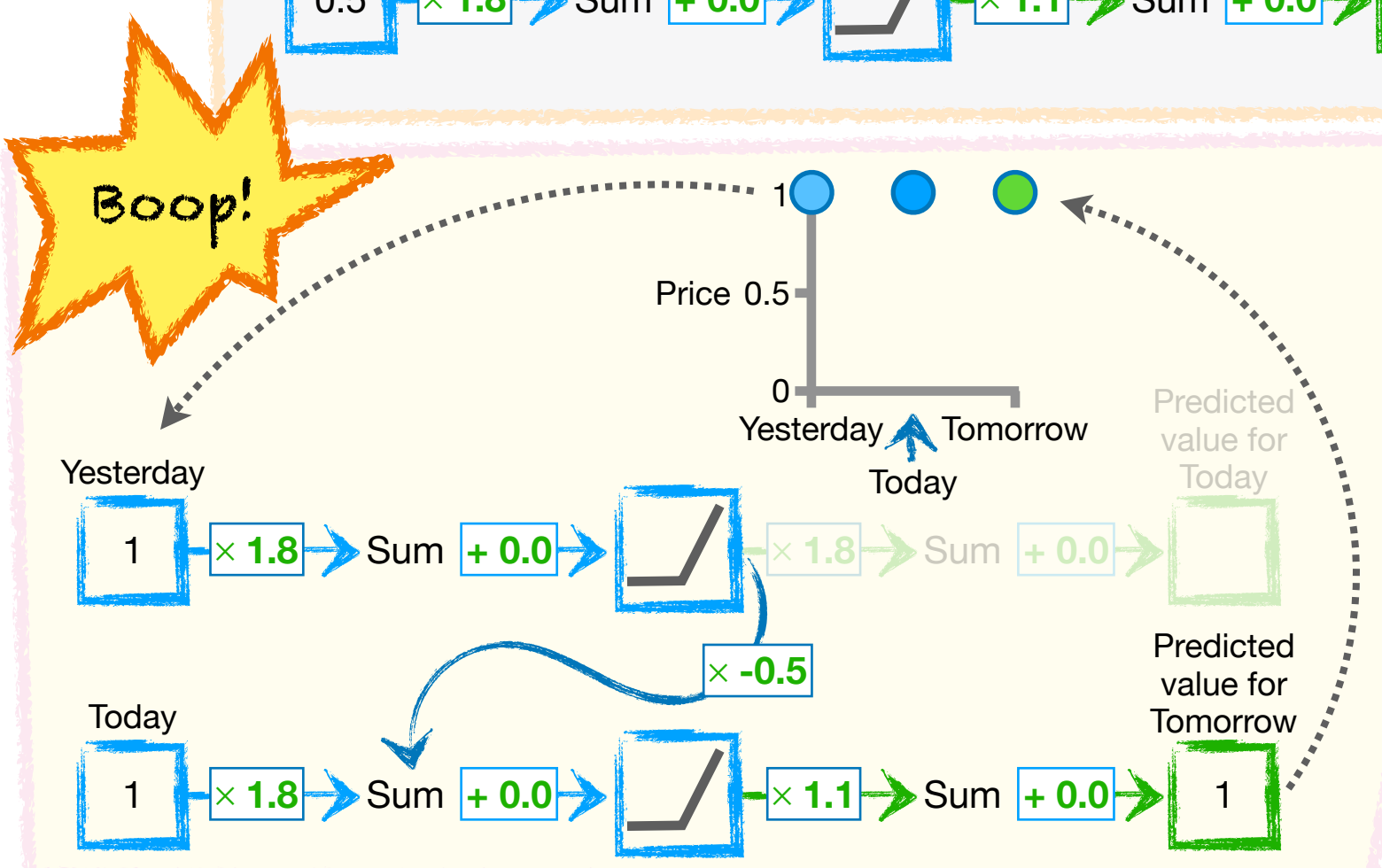


RNNs: Step-by-Step

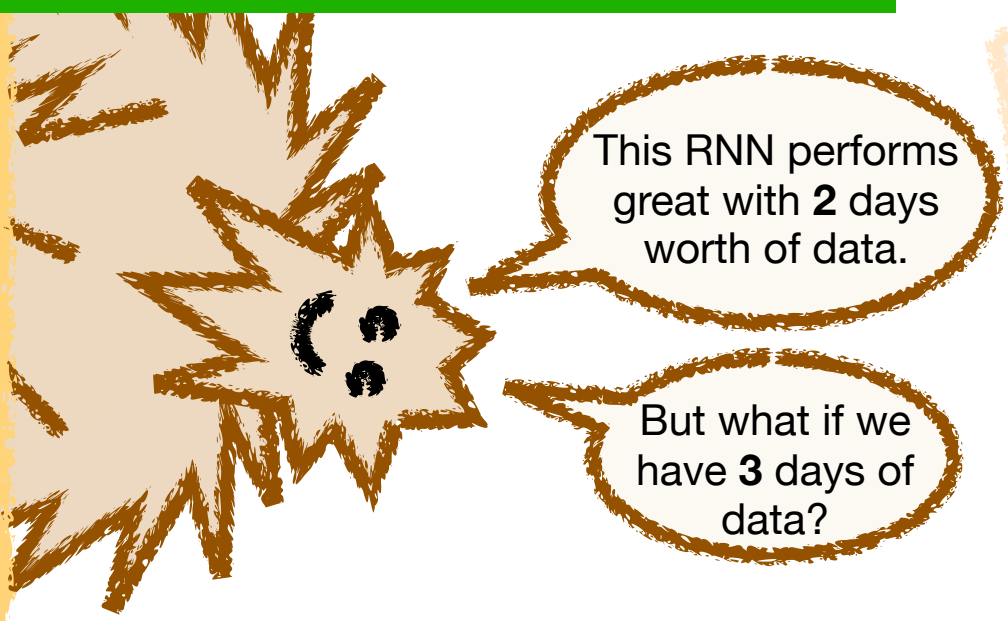
14 Likewise, when we run yesterday and today's values for the other scenarios through the RNN, we predict the correct values for tomorrow.



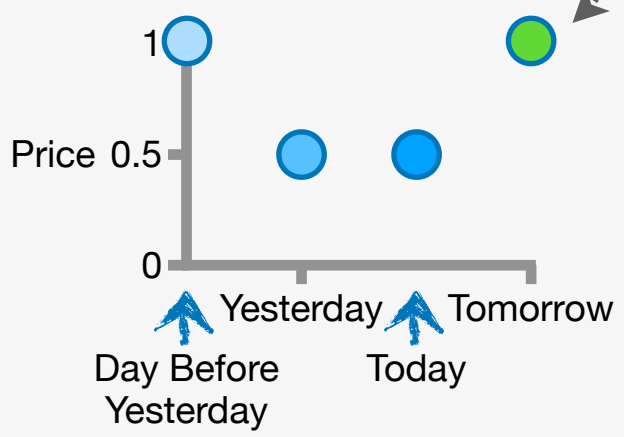




RNNs: Step-by-Step



15 When we want to use **3** days of data to make a prediction about tomorrow's price, like this…



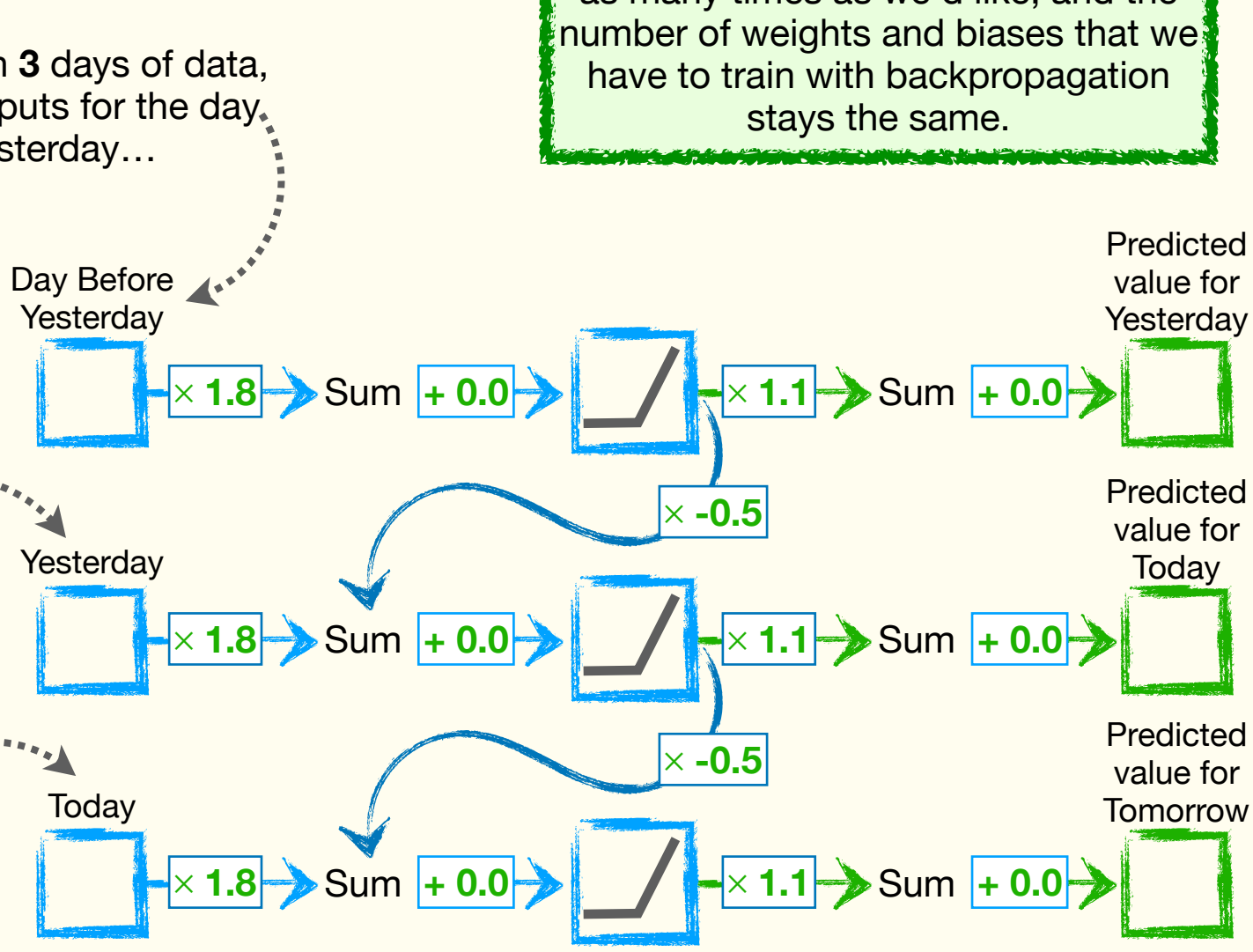
16 …then we just keep unrolling the RNN until we have an input for each day of data.

In this case, with **3** days of data, we now have inputs for the day before yesterday…

…yesterday…

…and today…

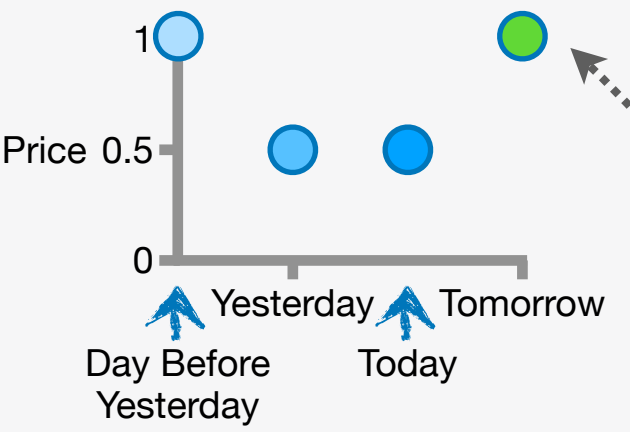
NOTE: Each time we unroll an RNN, we keep sharing the same weights and biases. This means we can unroll as many times as we'd like, and the number of weights and biases that we have to train with backpropagation stays the same.



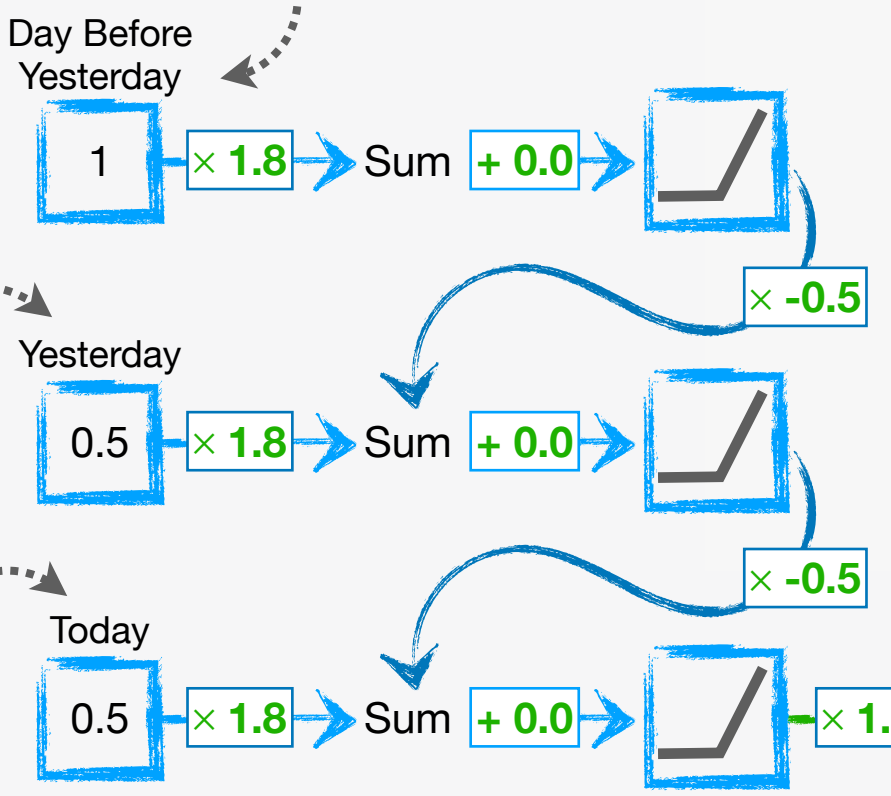
…along with corresponding predictions for yesterday, today, and tomorrow.

RNNs: Step-by-Step

17 Now we just we plug the values into the inputs, always from oldest to newest.



In this case, that means we start by plugging in the value for the day before yesterday…



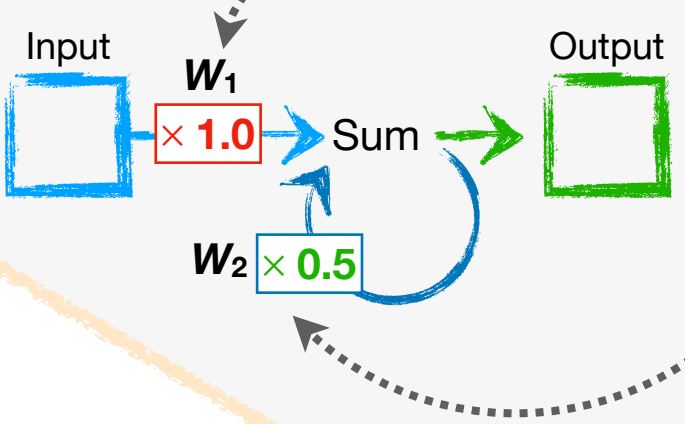
And when we do the math, the last output gives us the predicted value for tomorrow.

18 In summary ***basic*** RNNs are cool because they can expand, by unrolling, to handle any size dataset without increasing the number of weights and biases we need to train.

Now let's talk about how to apply backpropagation to an RNN and the problems it creates for the *basic* versions.

RNN Backpropagation

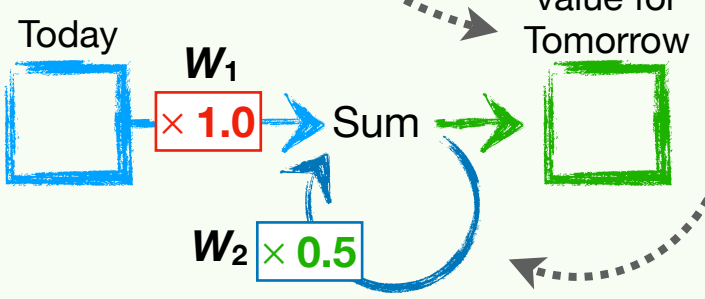
1 To illustrate how to apply backpropagation to an RNN, let's start with the simplest RNN ever. It only has **2** weights, labeled W_1 and W_2.



2 And just like we did in **Chapter 2**, we'll quantify the difference between the Observed and Predicted values with the Sum of the Squared Residuals (SSR).

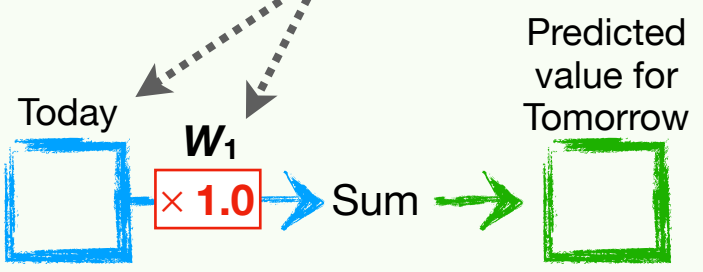
$$\textbf{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

3 Also, to keep things super simple at the start, let's begin by just using **Today's** value to predict **Tomorrow's** value…



…and that means for now, we can ignore the feedback loop, since we only have a single input value, **Today's** value…

…and that leaves us with an even simpler network that calculates a **Predicted** value by multiplying **Today's** value by W_1.



4 That means that the SSR is linked to W_1 by the **Predicted** value…

$$\textbf{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

$$\textbf{Predicted} = \text{Today} \times W_1$$

…and if we wanted to optimize W_1, then **The Chain Rule** tells us that the derivative of the SSR with respect to W_1…

$$\frac{d\,SSR}{d\,W_1} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,W_1}$$

…is equal to the derivative of the SSR with respect to the **Predicted** values…

…multiplied by the derivative of the **Predicted** values with respect to W_1.

RNN Backpropagation

5 Since we already derived the derivative of the SSR with respect for the **Predicted** values way back in **Chapter 2**, we can just plug it in.

$$\frac{d\,SSR}{d\,Predicted} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\frac{d\,SSR}{d\,W_1} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,W_1}$$

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times \frac{d\,Predicted}{d\,W_1}$$

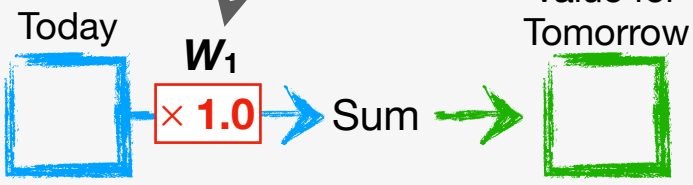
6 And because the **Predicted** values come from multiplying **Today's** value by W_1…

…the derivative of the **Predicted** values with respect to W_1…

…is just **Today's** value.

$$\frac{d\,Predicted}{d\,W_1} = \frac{d}{d\,W_1}\,Today \times W_1 = Today$$

$$Today \times W_1 = Predicted$$



7 So, we can plug in **Today's** value for the derivative of the **Predicted** value with respect to W_1…

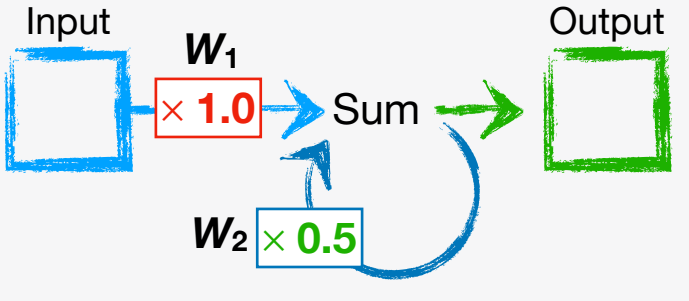
…to get the derivative of the SSR with respect to W_1.

Bam.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times \frac{d\,Predicted}{d\,W_1}$$

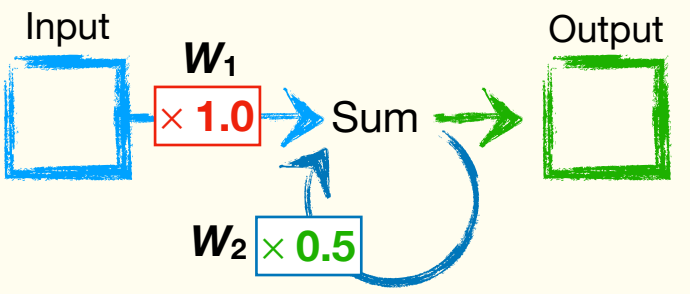
$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times Today$$

8 Now that we've seen how to calculate the derivative of the SSR with respect to W_1 when we only use **Today's** value in the RNN, let's learn how to calculate the derivative of the SSR with respect to W_1 when we use **Yesterday's** value and **Today's** value to predict **Tomorrow's** value.

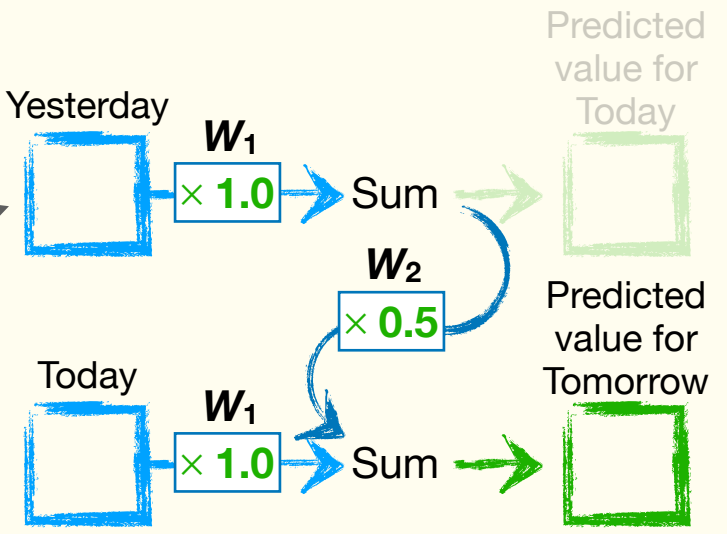


RNN Backpropagation

9 When we use **Yesterday's** value and **Today's** value to make a prediction with this RNN…

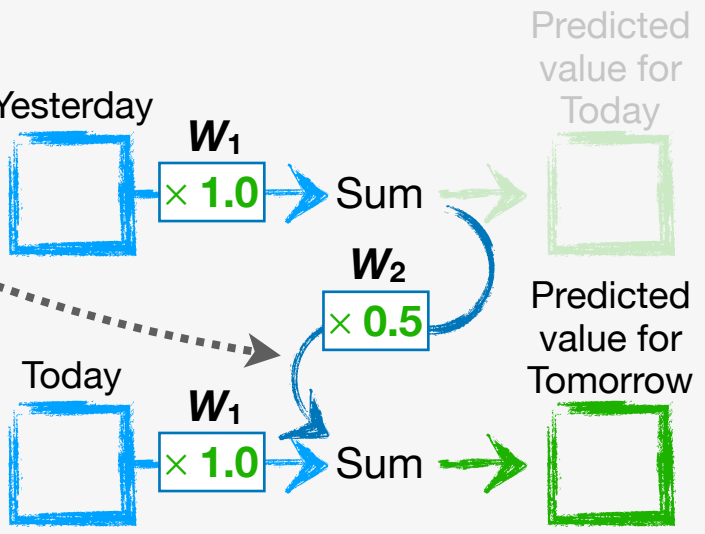


…then we unroll the RNN to allow for the two input values.



10 When we unroll the RNN once, the **Predicted** value is the sum of **Yesterday's** value multiplied by W_1 multiplied by W_2…

…plus **Today's** value multiplied by W_1.



11 Thus, just like before, the SSR is linked to W_1 by the **Predicted** value…

$$\text{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

$$\text{Predicted} = (\text{Yesterday} \times W_1 \times W_2) + (\text{Today} \times W_1)$$

…but now the **Predicted** value is the sum of **Yesterday's** value multiplied by W_1 multiplied by W_2…

…plus **Today's** value multiplied by W_1.

12 So the derivative of the **Predicted** values with respect to W_1 is this…

$$\frac{d\,\text{Predicted}}{d\,W_1} = \frac{d}{d\,W_1}(\text{Yesterday} \times W_1 \times W_2) + (\text{Today} \times W_1)$$

$$= (\text{Yesterday} \times W_2) + \text{Today}$$

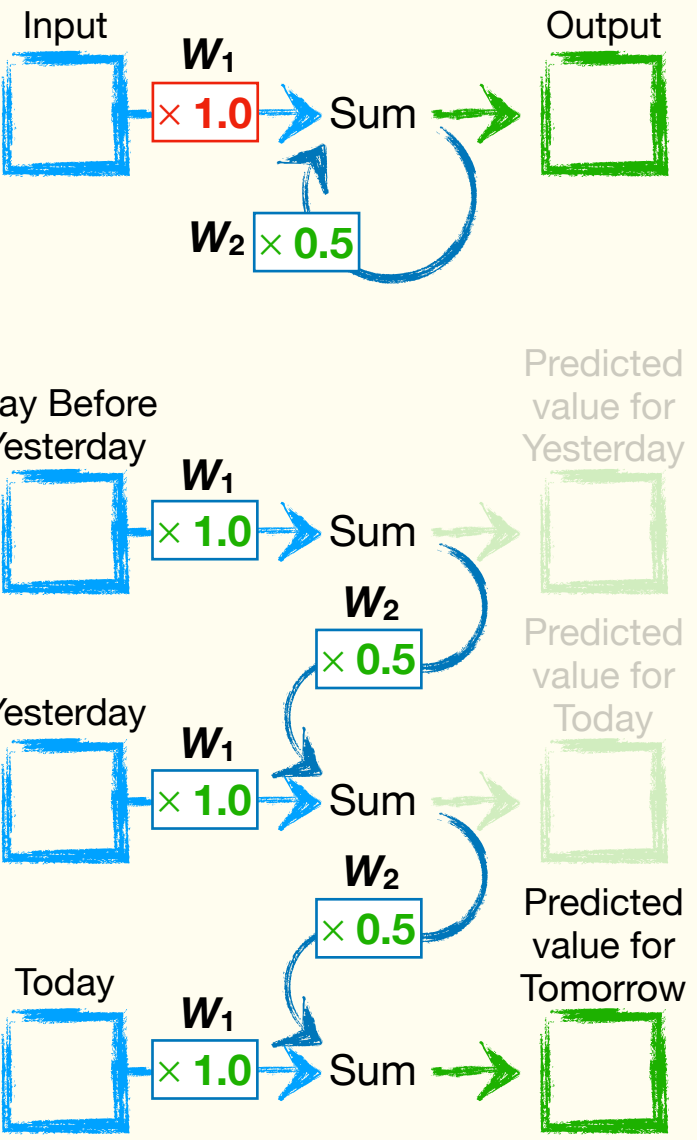
…which makes this the derivative SSR with respect to W_1.

Bam.

$$\frac{d\,\text{SSR}}{d\,W_1} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times [(\text{Yesterday} \times W_2) + \text{Today}]$$

RNN Backpropagation

13 When we use **The Day Before Yesterday's**, **Yesterday's**, and **Today's** values to make a prediction with this RNN…



…then we unroll the RNN to allow for the **3** input values.

14 Now we multiply the **Day Before Yesterday's** value by W_1 and W_2…

…then add it to **Yesterday's** value multiplied by W_1…

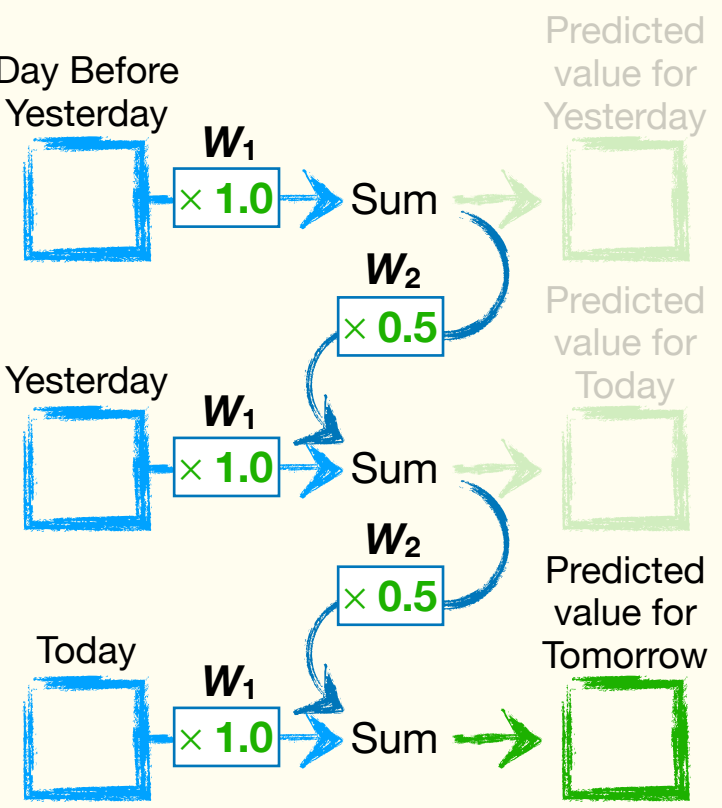
…and then multiply that sum by W_2…

…and then add that product to **Today's** value multiplied by W_1…

…to calculate the **Predicted** value.

Predicted = {[(Day Before Yesterday × W_1 × W_2) + (Yesterday × W_1)] × W_2} + (Today × W_1)

RNN Backpropagation



15 OK, so we just saw how this unrolled RNN…

…results in this equation for the **Predicted** value.

$$\textbf{Predicted} = \{[(\text{Day Before Yesterday} \times W_1 \times W_2) + (\text{Yesterday} \times W_1)] \times W_2\} + (\text{Today} \times W_1)$$

Now, if we multiply the stuff inside the square brackets…

…by this W_2…

…we get this!

$$\textbf{Predicted} = (\text{Day Before Yesterday} \times W_1 \times W_2^2) + (\text{Yesterday} \times W_1 \times W_2) + (\text{Today} \times W_1)$$

16 And now we determine the derivative of the **Predicted** values with respect to W_1…

TERMINOLOGY ALERT!!!

When backpropagation is applied to RNNs, it's called **Backpropagation Through Time**.

$$\frac{d\,\text{Predicted}}{d\,W_1} = \frac{d}{d\,W_1}[(\text{Day Before Yesterday} \times W_1 \times W_2^2) + (\text{Yesterday} \times W_1 \times W_2) + (\text{Today} \times W_1)]$$

$$= (\text{Day Before Yesterday} \times W_2^2) + (\text{Yesterday} \times W_2) + \text{Today}$$

…and plug it into the derivative of the SSR with respect to W_1.

$$\frac{d\,SSR}{d\,W_1} = \frac{d\,SSR}{d\,\text{Predicted}} \times \frac{d\,\text{Predicted}}{d\,W_1}$$

$$= \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times [(\text{Day Before Yesterday} \times W_2^2) + (\text{Yesterday} \times W_2) + \text{Today}]$$

RNN Backpropagation

17 Now that we understand how to determine derivatives for RNNs, I want to point out that in the derivative of the SSR with respect to W_1…

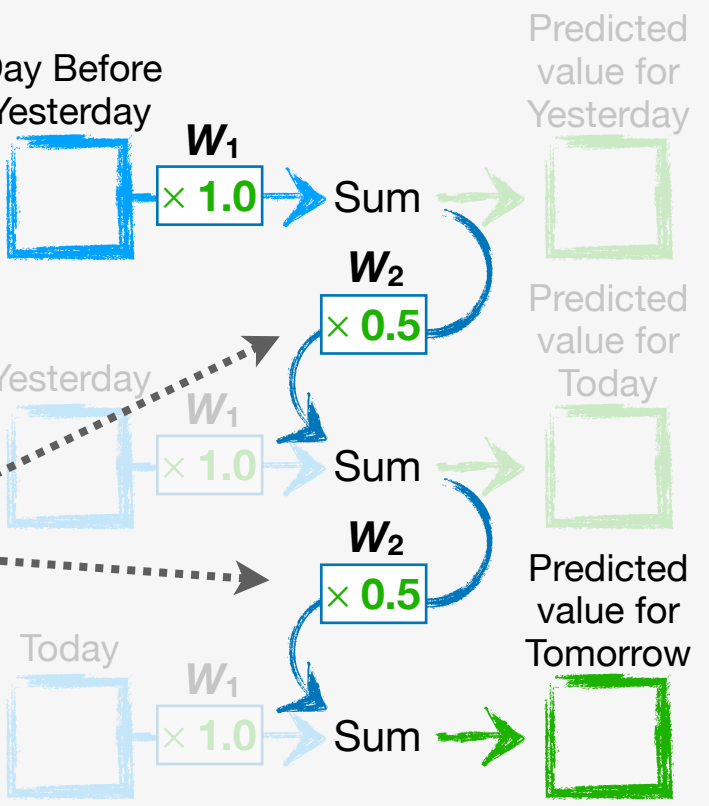
…the number of times we multiply each input value by W_2…

…corresponds to the number of times we had to unroll the RNN to include that value.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times [(Day\ Before\ Yesterday \times W_2^2) + (Yesterday \times W_2) + Today]$$

18 For example, we unrolled the RNN twice to include **The Day Before Yesterday's** value, and, in the derivative, we multiplied **The Day Before Yesterday's** value by W_2 squared.

And the multiplication in the derivative makes sense because when we look at the unrolled RNN, we can see how **The Day Before Yesterday's** value was multiplied by W_2 twice on its way to the **Predicted** value.



$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times [(Day\ Before\ Yesterday \times W_2^2) + (Yesterday \times W_2) + Today]$$

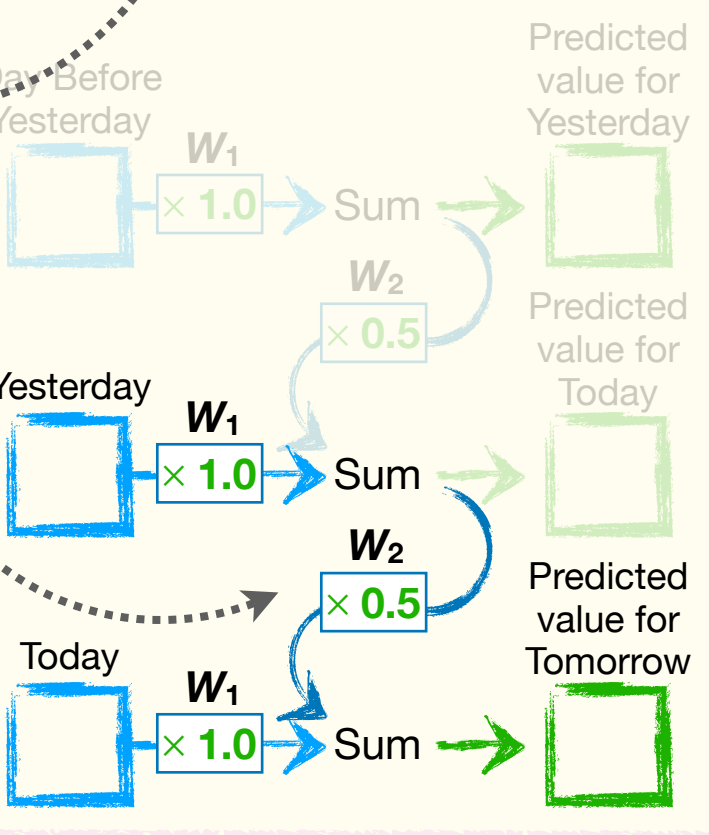
19 Likewise, we unrolled the RNN once to include **Yesterday's** value, and, in the derivative, we multiplied **Yesterday's** value by W_2 one time, which is the same as W_2 raised to the power of **1**.

$$W_2 = W_2^1$$

Lastly, we didn't unroll the RNN to include **Today's** value, and, in the derivative, we didn't multiply **Today's** value by W_2, which is the same as multiplying it by W_2 raised to the **0th** power.

$$W_2^0 = 1$$

Now let's talk about how this pattern leads to trouble!!!

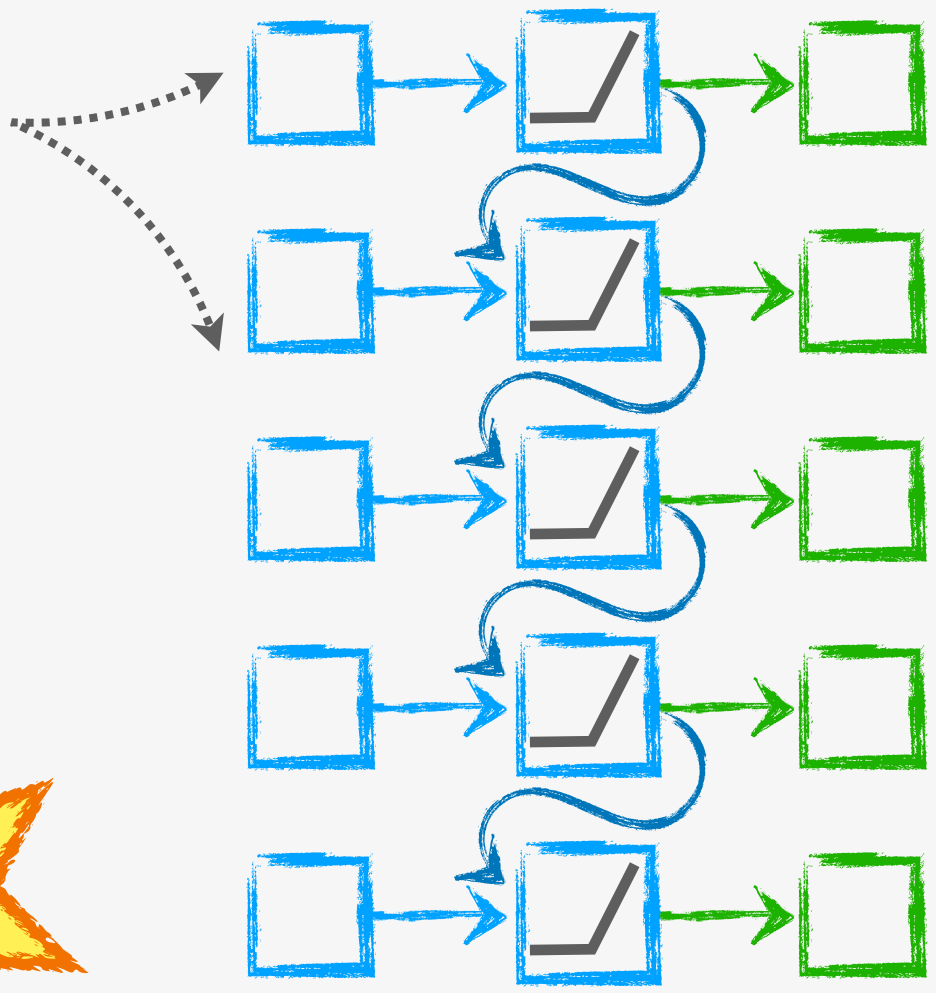


The Vanishing/Exploding Gradient Problems

1 Because input values can be multiplied by the same weight a bunch of times, *basic* RNNs, like the ones we've been discussing in this chapter, have problems with training.

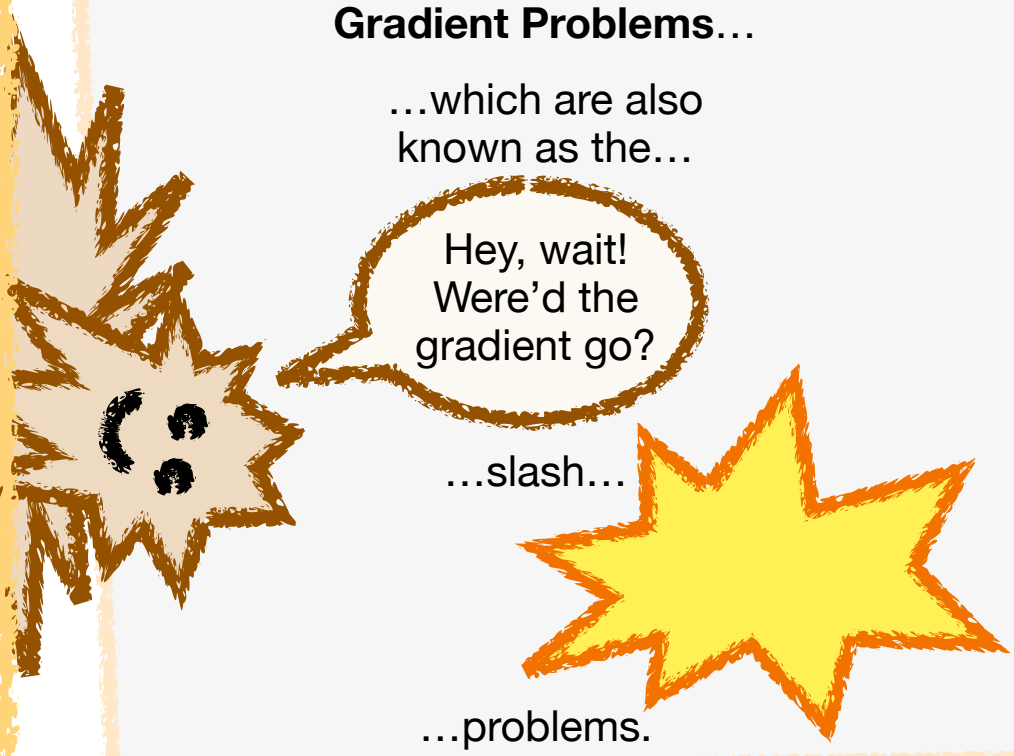
NOTE: In this context, the term gradient refers to the derivatives we calculate for gradient descent.

The main idea is that *basic* RNNs have limits on how many times they can unroll before older data points start having too little or too much influence on training.



These problems are called **The Vanishing/Exploding Gradient Problems**…

…which are also known as the…



2 **The Vanishing/Exploding Gradient** problems have to do with the weight along the squiggle.

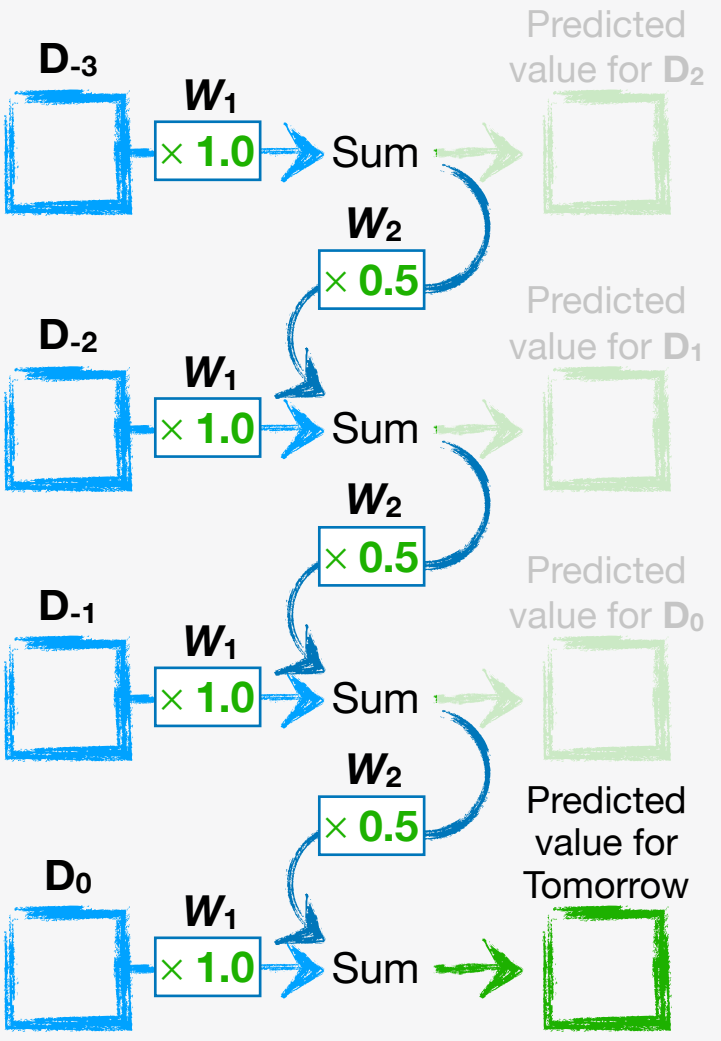
As we'll see in the next few pages, if the weight is between **-1** and **1**, then parts of the derivative, or gradient, that we use for gradient descent will vanish.

And if the weight is less than **-1** or greater than **1**, then parts of the gradient that we use for gradient descent will explode.

We'll start by seeing how parts of the gradient can vanish.

The Vanishing/Exploding Gradient Problems

3 Given the pattern of raising the power of W_2 by the number of times we unrolled the RNN to include a value, if we want to include **4** days of data, labeled D_0, D_{-1}, D_{-2}, and D_{-3}…



…then the derivative of the SSR with respect to W_1 would include a term with D_{-3} multiplied by the cube of W_2 because we needed to unroll the RNN **3** times to include D_{-3}.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\times [(D_{-3} \times W_2^3) + (D_{-2} \times W_2^2) + (D_{-1} \times W_2) + D_0]$$

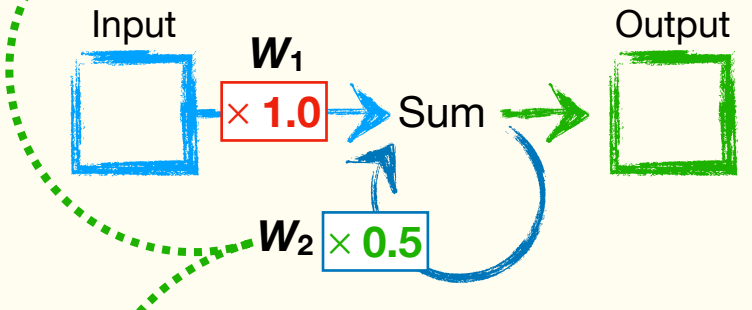
4 Now remember, the derivative of the SSR with respect to W_1 is what we use in gradient descent to optimize W_1…

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\times [(D_{-3} \times W_2^3) + (D_{-2} \times W_2^2) + (D_{-1} \times W_2) + D_0]$$

…and because W_2 scales each input value, D_{-0}, D_{-1}, D_{-2}, and D_{-3}…

…we can plug in the current value for W_2, **0.5**, to see how much each input value will contribute to W_1's optimization.



$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\times [(D_{-3} \times 0.5^3) + (D_{-2} \times 0.5^2) + (D_{-1} \times 0.5^1) + D_0]$$

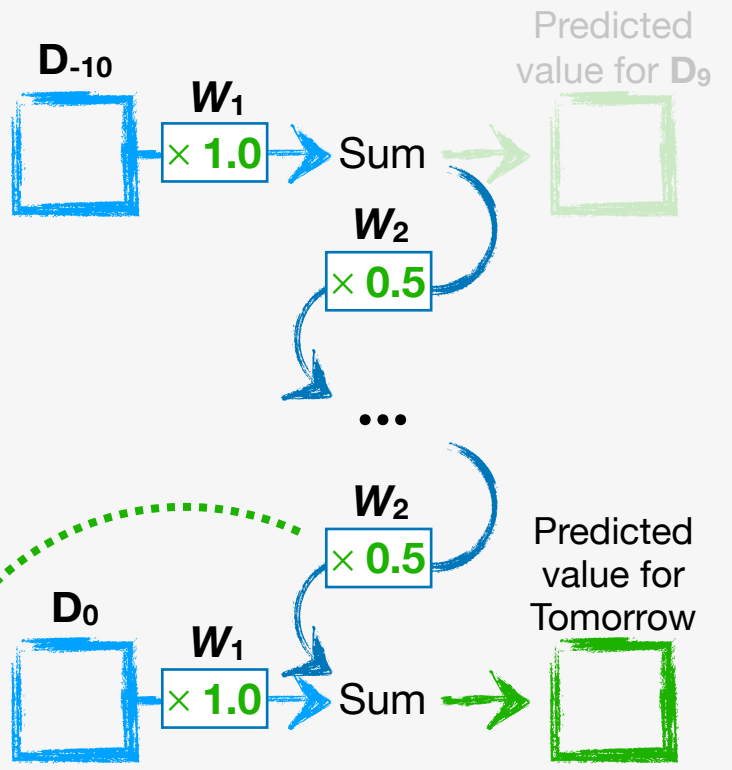
And now we can see that the value for D_{-3} will be multiplied by **0.125**, and thus will contribute a lot less to the derivative than the value for D_0, which won't shrink at all.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\times [(D_{-3} \times 0.125) + (D_{-2} \times 0.25) + (D_{-1} \times 0.5) + D_0]$$

The Vanishing/Exploding Gradient Problems

5 Likewise, if we had **11** days of data, then we would unroll the RNN **10** times to include the oldest data point, D_{-10}…

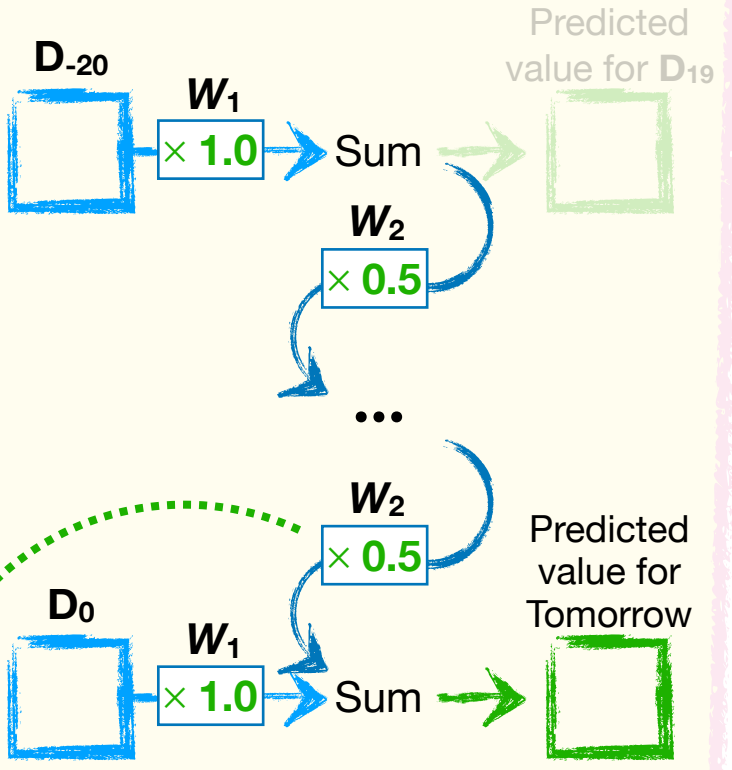


…and D_{-10} would be scaled by **0.001**, and thus, play an even smaller role in training than D_0, which would not be scaled at all.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times [(D_{-10} \times 0.5^{10}) + \cdots + D_0]$$

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times [(D_{-10} \times 0.001) + \cdots + D_0]$$

6 Lastly, if we had **21** days of data, which isn't that much data, and we unrolled the RNN **20** times to include the oldest data point, D_{-20}…



…then D_{-20} would be scaled by **0.000000001** and basically be reduced to **0** and thus would play no significant role in training.

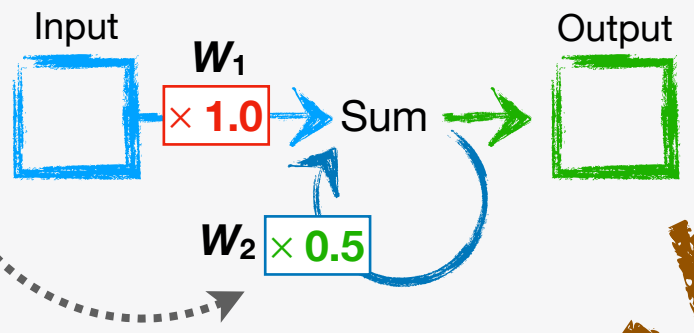
In other words, the part of the gradient for D_{-20} would vanish.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times [(D_{-20} \times 0.5^{20}) + \cdots + D_0]$$

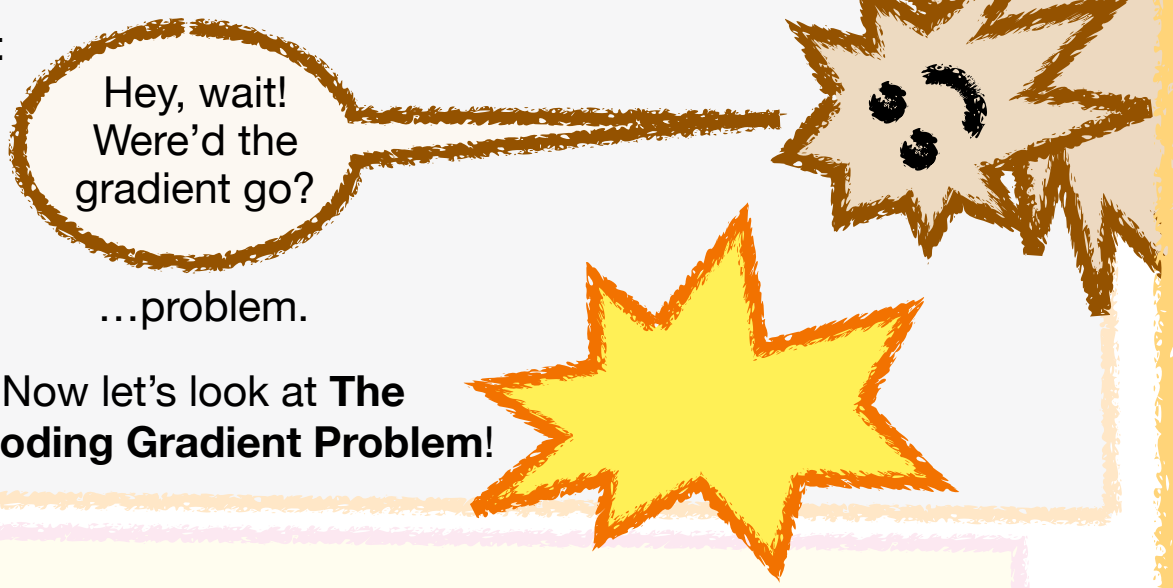
$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times [(D_{-20} \times 0.000000001) + \cdots + D_0]$$

The Vanishing/Exploding Gradient Problems

7 In summary when W_2 is between **-1** and **1** we can't actually use that much data to train the RNN because W_2 will scale the older data points and prevent them from contributing to the derivative for W_1.



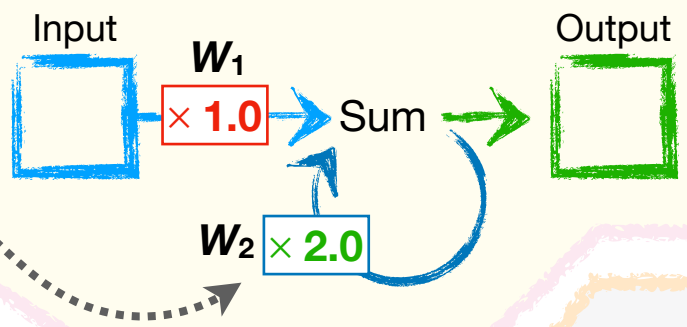
This is **The Vanishing Gradient Problem**. Also known as the…



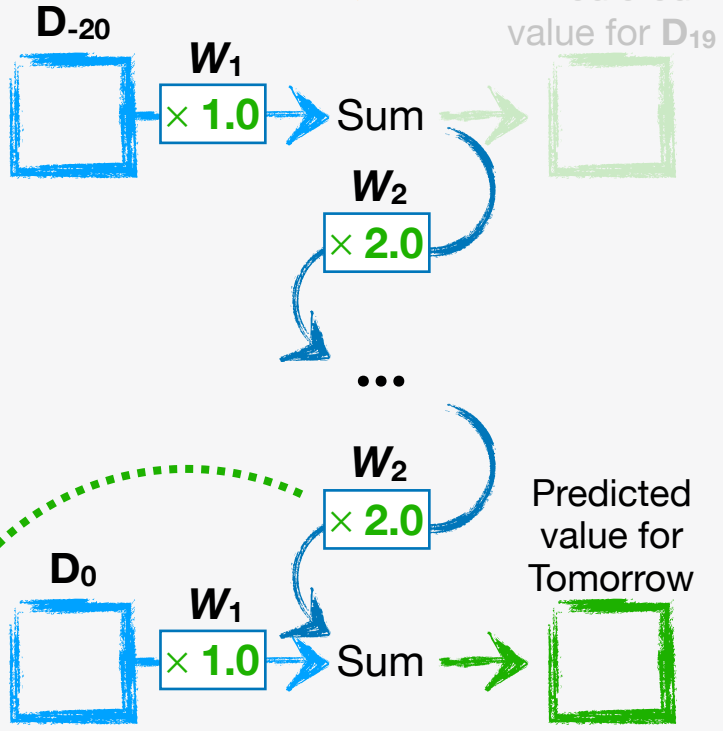
…problem.

Now let's look at **The Exploding Gradient Problem**!

8 **The Exploding Gradient Problem** shows up when W_2 is less than **-1** or greater than **1**. So, to see it in action, let's set $W_2 = 2$.



9 Like in the previous example, if we had **21** days of data and unrolled the RNN **20** times to include the oldest data point, D_{-20}…



…then D_{-20} would be scaled by **1,048,576** and would completely dwarf any contribution to training that might be given by D_0, which isn't scaled at all.

In other words, the part of the gradient associated with D_{-20} would explode.

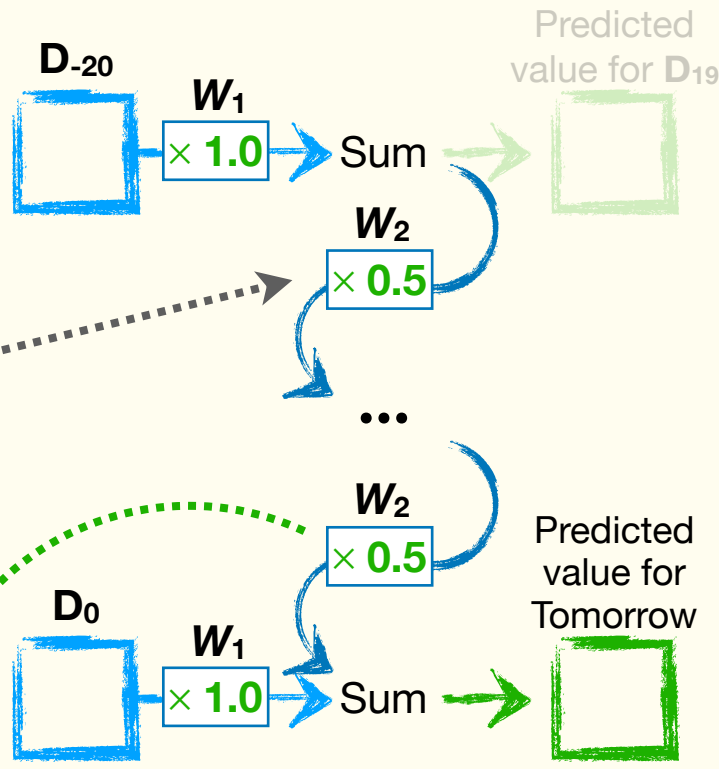
$$\frac{d\,SSR}{d\,W_1} = [(D_{-20} \times 2^{20}) + \cdots + D_0]$$

$$\frac{d\,SSR}{d\,W_1} = [(D_{-20} \times 1{,}048{,}576) + \cdots + D_0]$$

The Vanishing/Exploding Gradient Problems

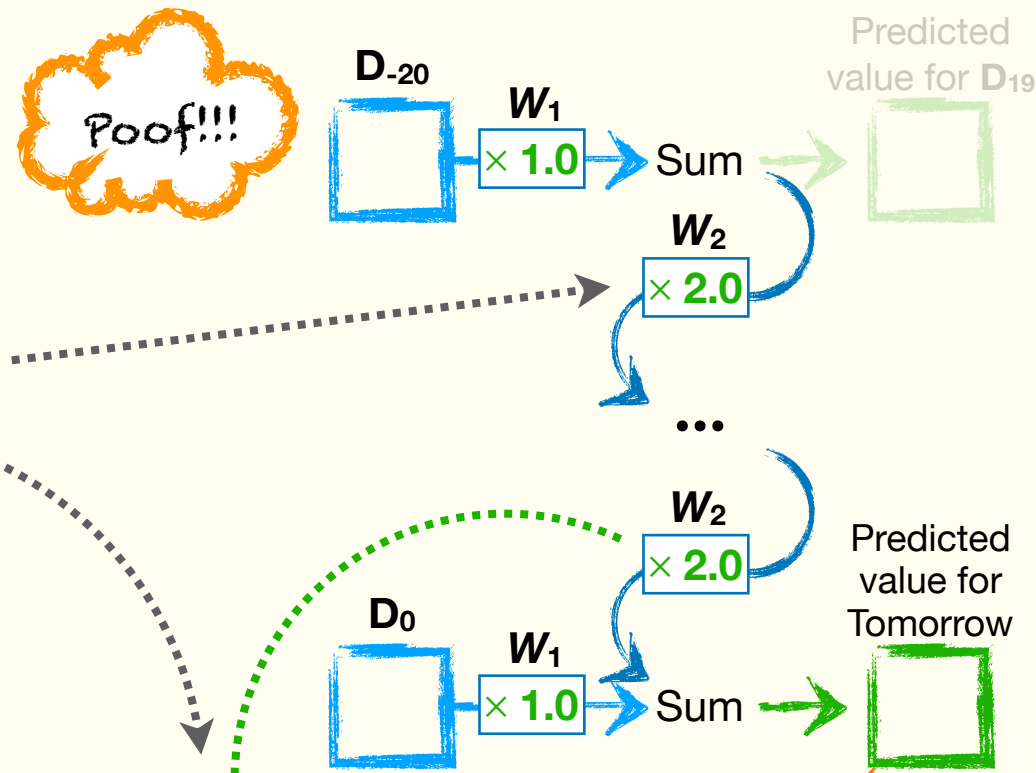
10 So, in the past few pages we've seen **The Vanishing/Exploding Gradient Problems** in action.

When the value for W_2 is between **-1** and **1**, it doesn't take much before older data points become insignificant.



$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times [(\mathbf{D_{-20}} \times \mathbf{0.5^{20}}) + \cdots + \mathbf{D_0}]$$

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times [(\mathbf{D_{-20}} \times \mathbf{0.000001}) + \cdots + \mathbf{D_0}]$$



And when the value for W_2 is less than **-1** or greater than **1**, it doesn't take much before older data points completely overwhelm the newer data points.

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times [(\mathbf{D_{-20}} \times \mathbf{2^{20}}) + \cdots + \mathbf{D_0}]$$

$$\frac{d\,SSR}{d\,W_1} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times [(\mathbf{D_{-20}} \times \mathbf{1{,}048{,}576}) + \cdots + \mathbf{D_0}]$$

11 The good news is that it doesn't take much to reduce the effects of **The Vanishing/Exploding Gradient Problems**, and we'll learn about one way to deal with it in the next chapter when we talk about **Long Short-Term Memory**.

NOTE: Long Short-Term Memory networks are a type of RNN, so we'll save the coding tutorial for the next chapter.

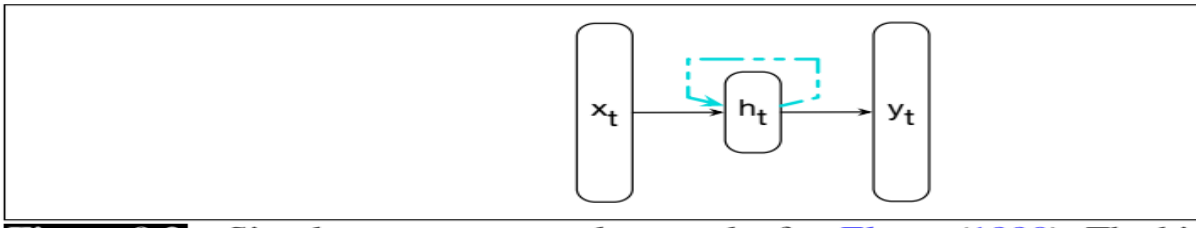


Figure 9.2 Simple recurrent neural network after Elman (1990). The hi
a recurrent connection as part of its input. That is, the activation value
depends on the current input as well as the activation value of the hid
previous time step.

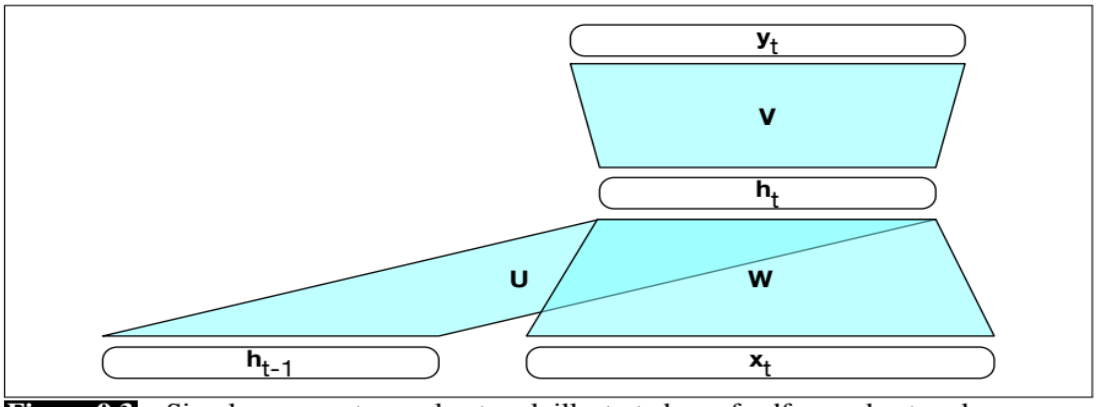


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

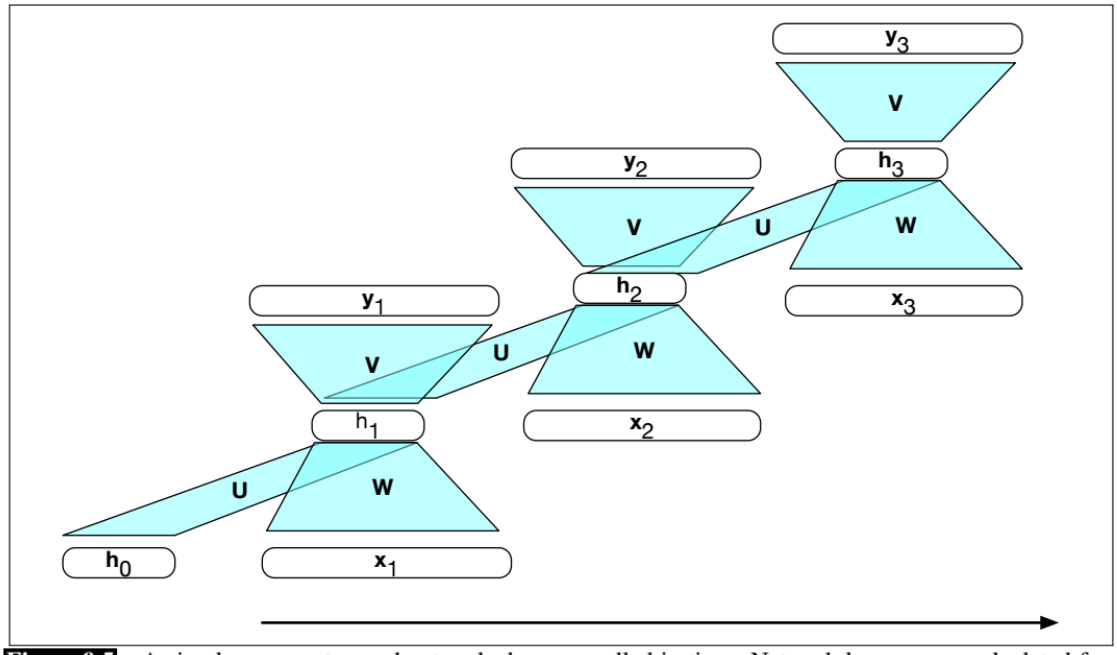


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are recalculated for each time step, while the weights **U**, **V** and **W** are shared in common across all time steps.

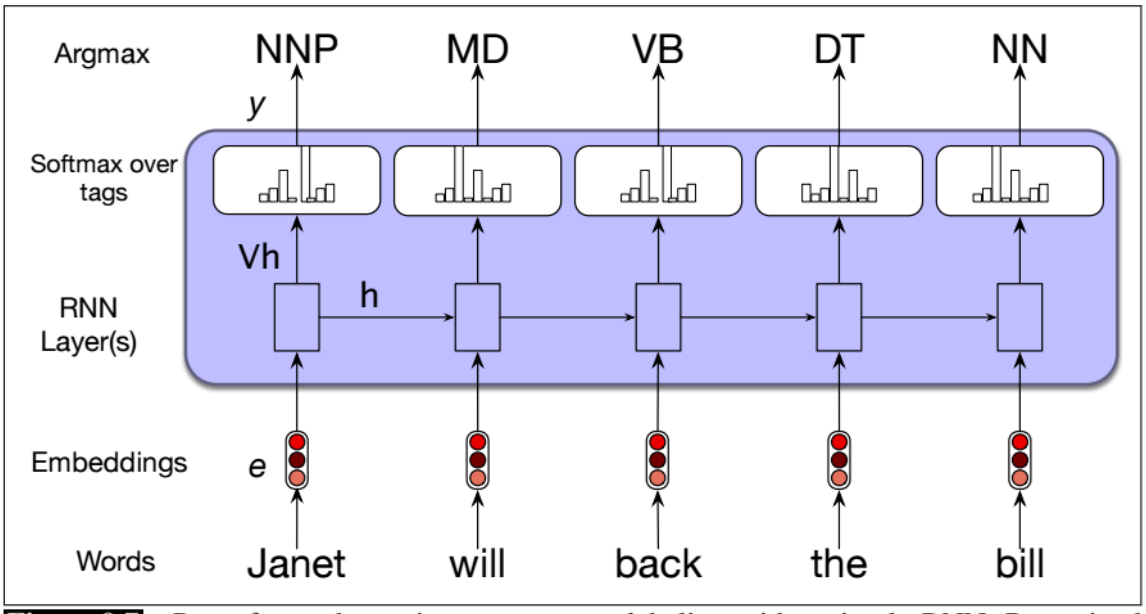


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

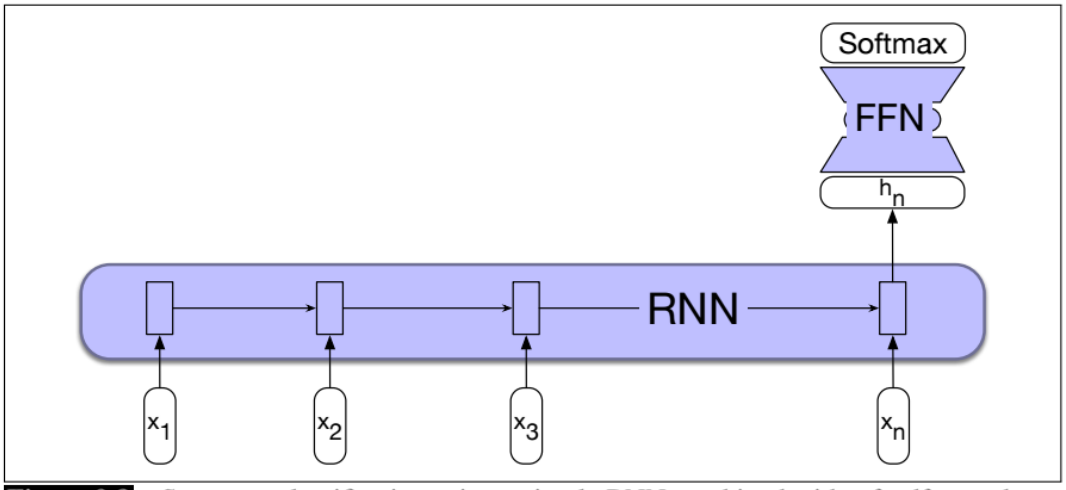


Figure 9.8 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

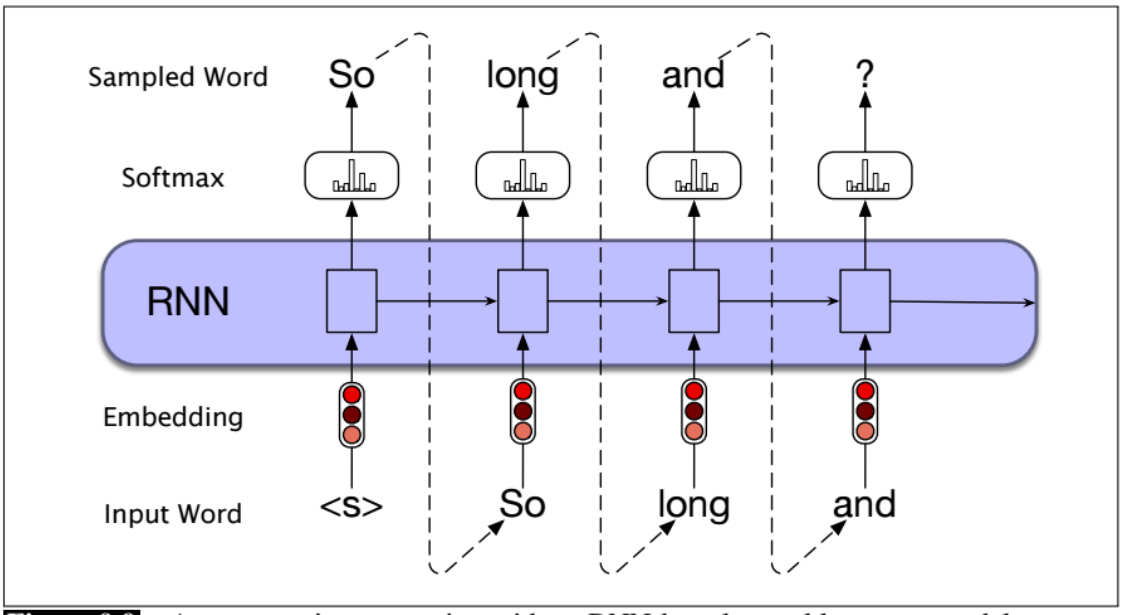


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

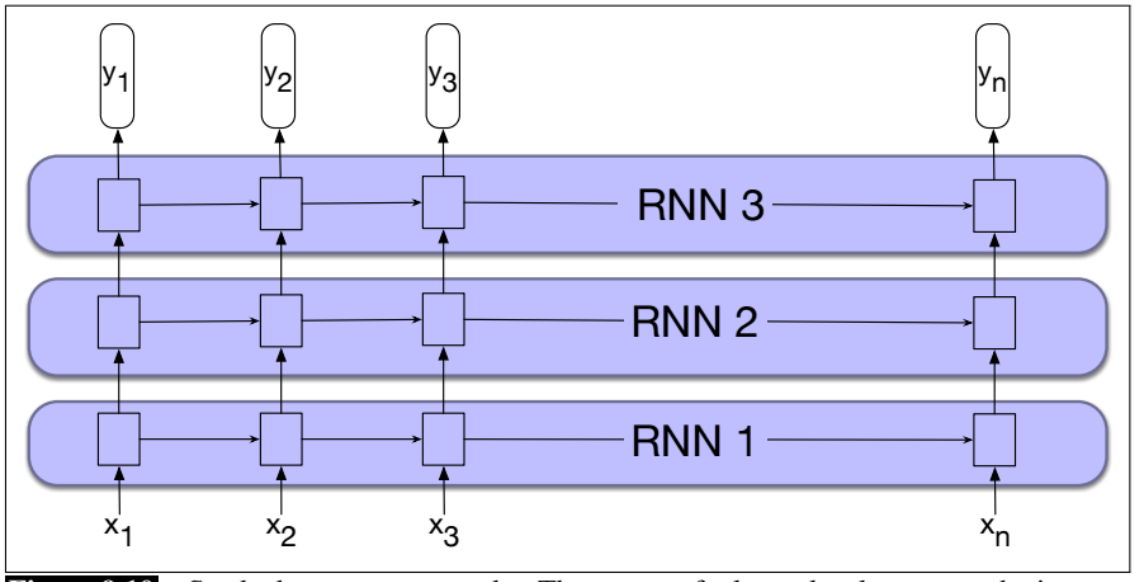


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

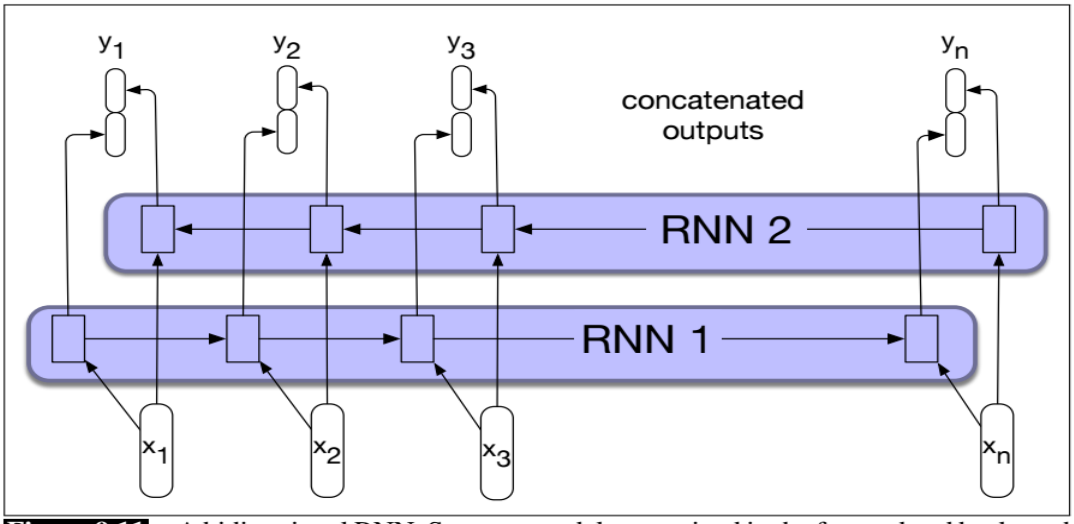


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

$$\begin{aligned}\mathbf{h}_t &= [\mathbf{h}_t^f ; \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b\end{aligned}$$

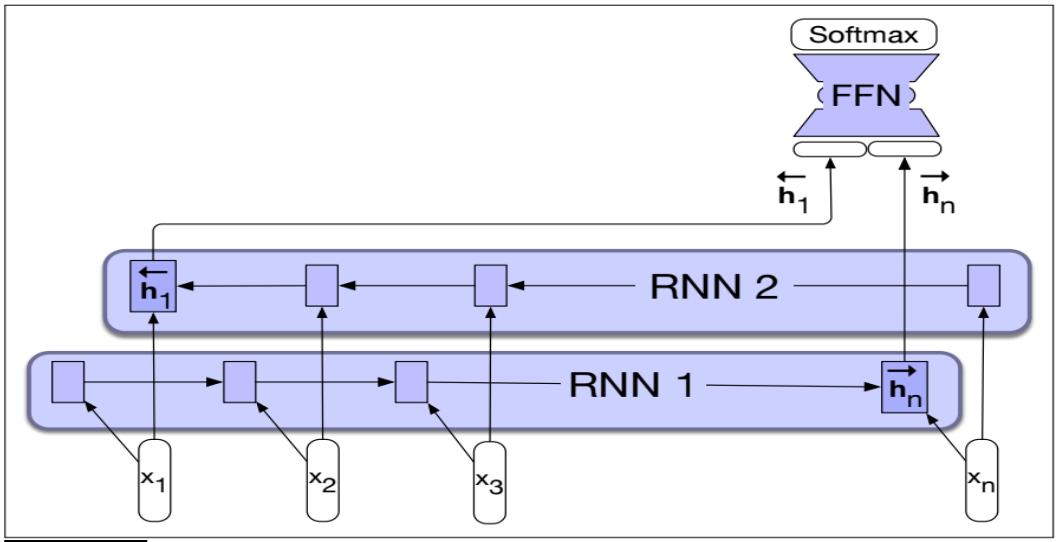


Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.