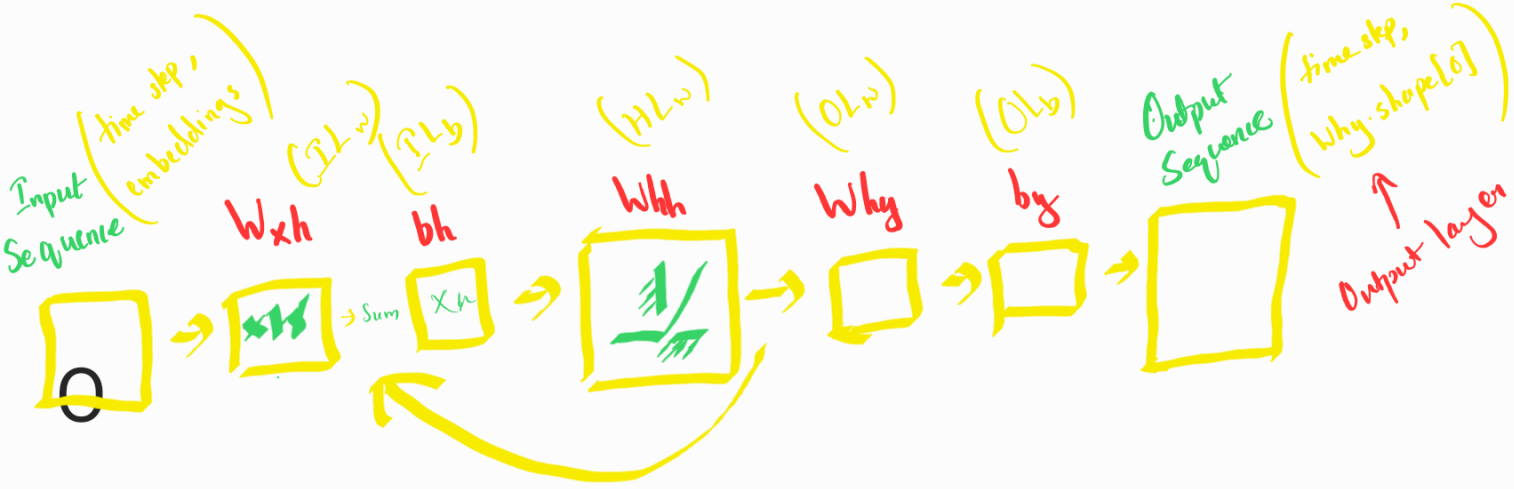
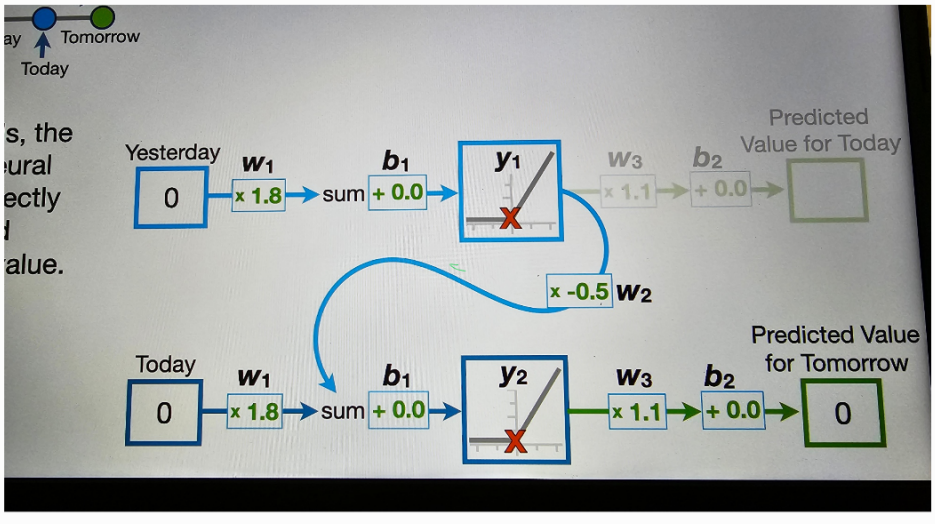
- RNN have weights, biases layers, activation functions

- The difference is that it has a feedback loop.

- The feed back loop makes it possible to use sequential inputs one by one. over time





- Instead of using the Input of the 1st for prediction it is given through the feedback loop and makes a prediction with the sequence of Inputs.

- Initialize hidden state (1, Whh.shape[0])

- When each input comes in we have to unroll the network. (results in new neural networks in background)

(- looping through the time step
- Finding input at current step
- Compute hidden state at current time step)

$h\_s = tf.matmul(xt, Wxh) + tf.matmul(h\_s, Whh) + bh$

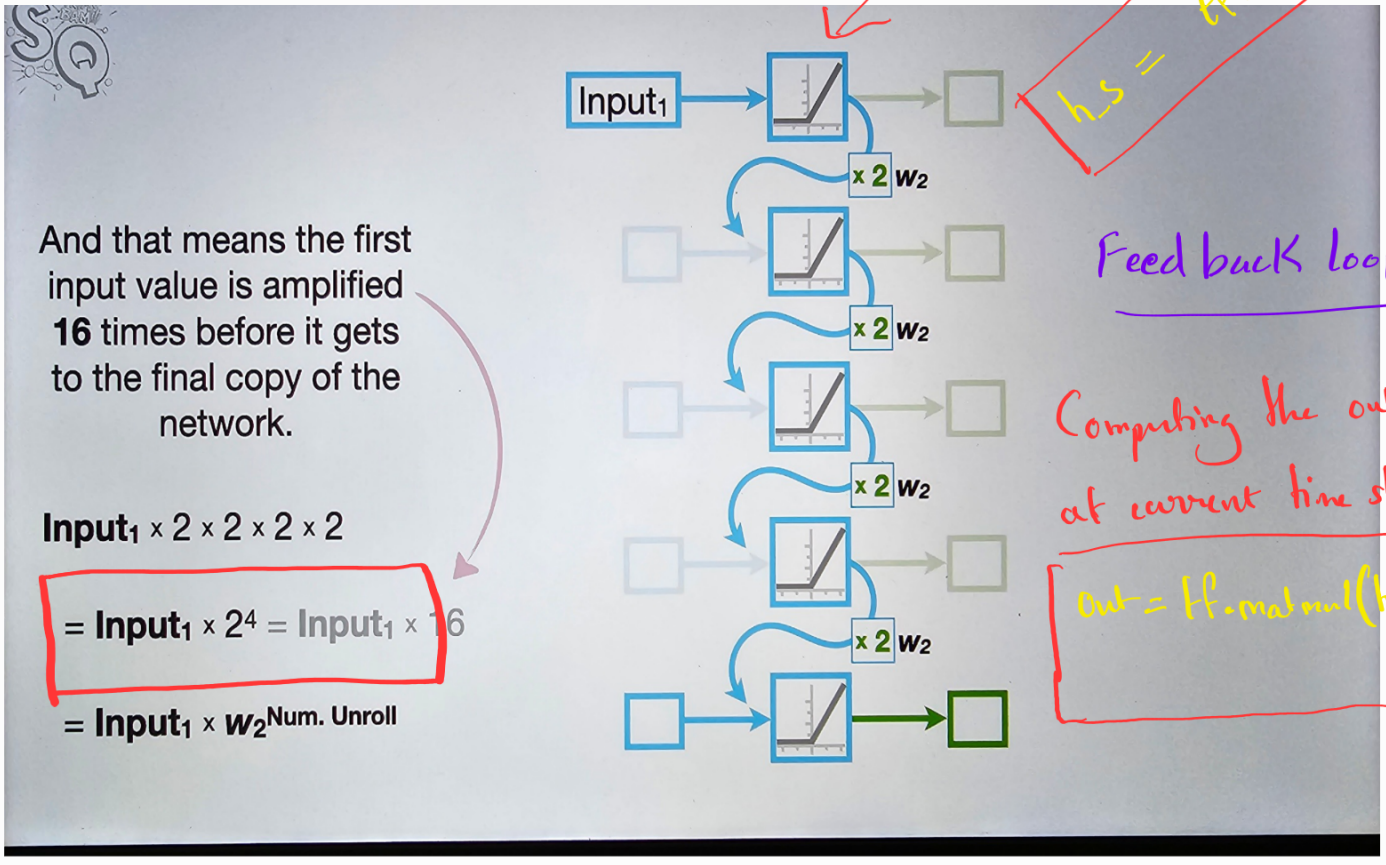
- Even though we unroll the RNN multiple times the weights and biases are shared and there is no new addition of extra weights.

- The more we unroll RNN the more harder it becomes to

Main. (Vanishing / Exploding Gradient Descent problems)

- where did the Gradient go?

Exploding Problem



Applying Sigmoid on hidden state

h_s = ff.math.sigmoid(h_s)

Feed back loop

Computing the output at current time step

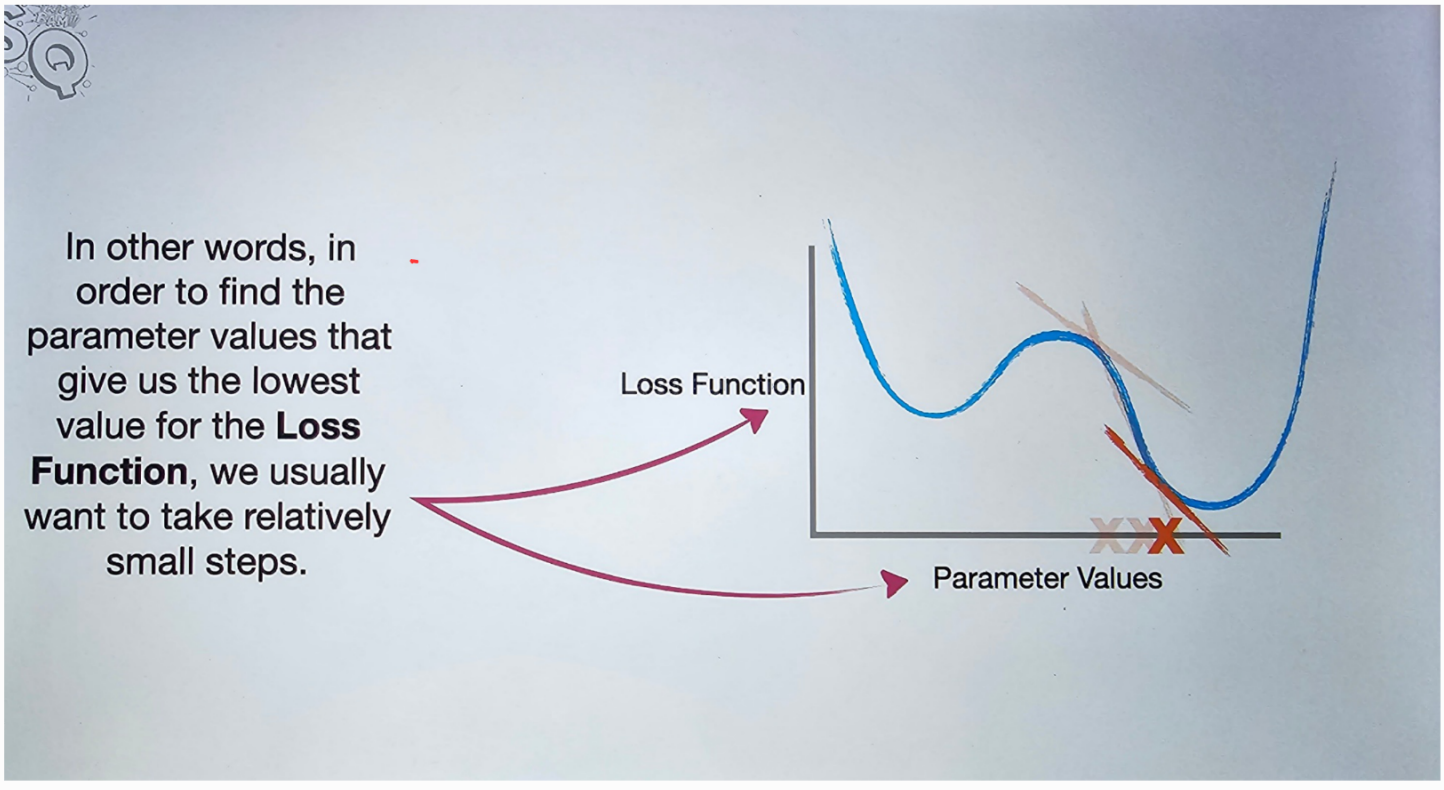
out = ff.matmul(h_s, why) + by

- The Input gets multiplied

as many times as we unroll the RNN.

- If we have 50 data 1×2^{50}

$\Rightarrow$! HUGE NUMBER



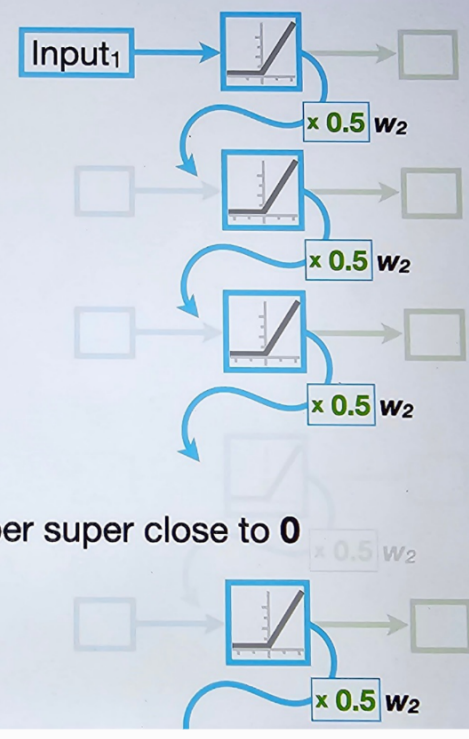
- It neglects us from taking

small steps and we end up in Exploding GD. We just bounce around.

Vanishing Gradient problem

- When we limit w_2 to values < 1
- A solution to avoid Exploding Gradient Descent.

Because this number is **super close to 0**, this is called the **Vanishing Gradient Problem.**

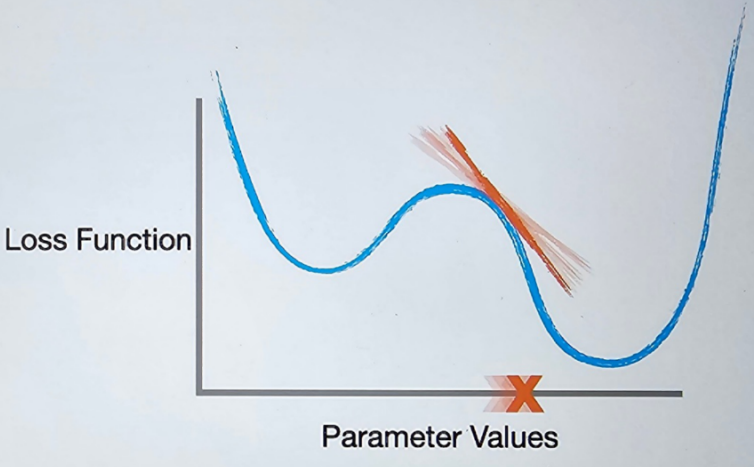


Input$_1$ × 0.5^{50} = **Input$_1$** × a number super close to **0**

Input$_1$ × $w_2^{\text{Num. Unroll}}$

- Results in Vanishing GD

And as a result, we end up hitting the maximum number of steps we are allowed to take before we find the optimal value.



- Solution to Exploding / Vanishing GD is LSTM (Long-short-term Memory-Networks)

Training a RNN

Parameters Wxh, Whh, Why, bh, by,

input Sequence, targets,
alpha, epochs.

Steps

1) Defining the Optimizer

Optimizer = tf.optimizers.Adam
(learning_rate = alpha)

2) Defining the loss fn

loss = tf.losses.MeanSquaredError()

3) Looping through Epochs.

4) Iterating through Each batch of Input (Input_sequence.shape[0])

5) with tf.GradientTape() as tape:

6) Finding y_pred

output_sequence, h_s

= rnn_f_p(Wxh, whh, why

bh, by, batch_input)

7) finding loss

loss = loss_fn(batch_target,

Output Sequence)

8) Finding Gradients in the Scope of with GradientTape() as tape:

```
grads = tape.gradient(loss,
    [Wxh, Whh, Why, bh,
    by])
```

9) Applying Optimizer

On Gradients.

Optimizer.apply_gradients
(zip(grads, [Wxh, Whh, Why,
bh, by]))

10) returning Wxh, Whh,
Why, bh, by