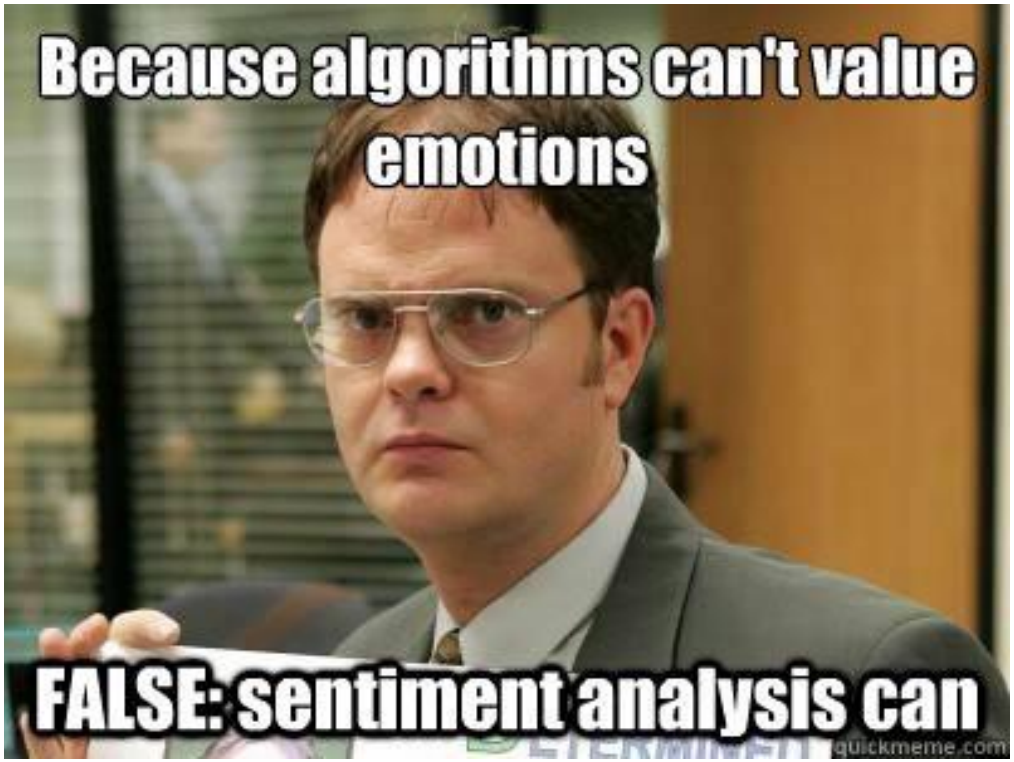
CISC 372
Text Analytic II
NLP Cont'd



Last lectures

- Tuning Method
- Association Rule Mining
- Text Analysis
  - Classification
  - Preprocessing
  - Representations
    - BOW
    - N-gram
    - character n-gram
    - Part-Of-Speech Tagging
    - Dependency Tree
  - Vanilla RNN

Today

- Text Analysis
  - Vanilla RNN
  - Gated Recurrent Unit
  - LSTM
  - Attention Mechanism

Sentiment Analysis

- A sentence -> **positive** or **negative**

This film was just brilliant, casting location, scenery story direction everyone's really suited the part they played.

Sentiment Analysis

1. Start with an empty memory
2. Read next word
3. Interpret its meaning (lookup)
4. Add it to the memory (memorize)
5. Go back to 2

This film was just brilliant, casting location, scenery story direction everyone's really suited the part they played.

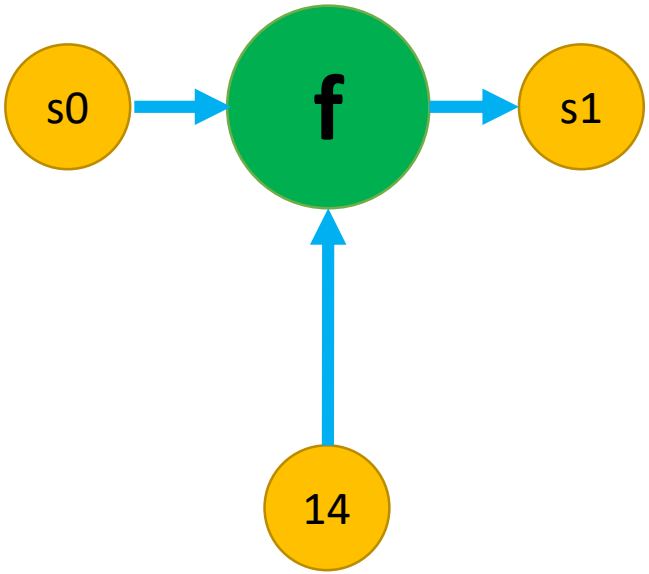
Building vocabulary

Transform tokens to their corresponding IDs (ranked by frequency).

this film was just brilliant casting location scenery story

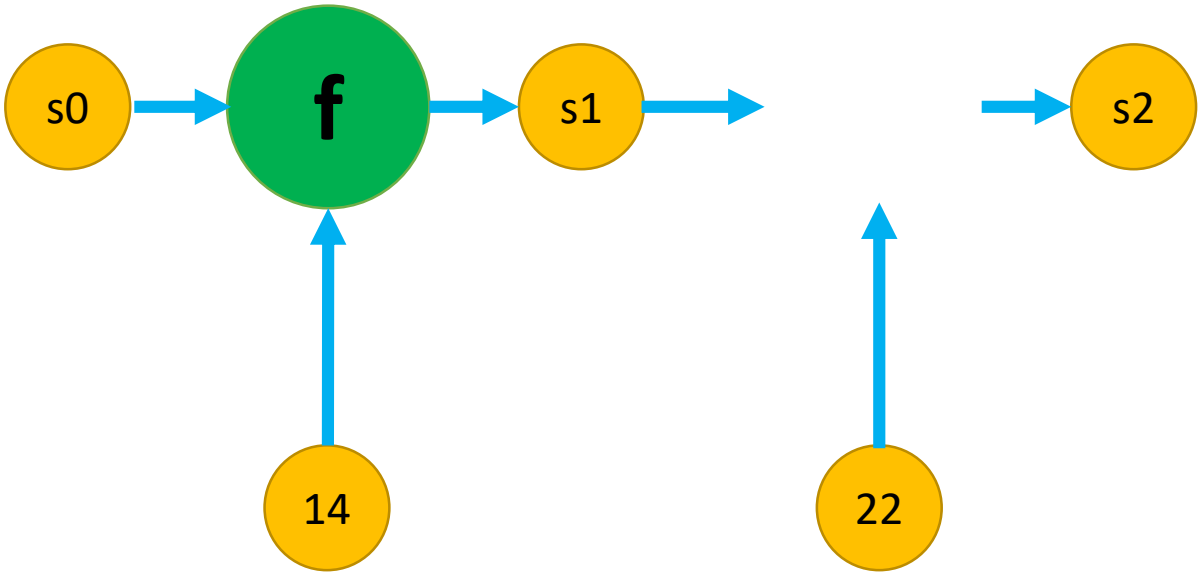
14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 39

Recurrent layer



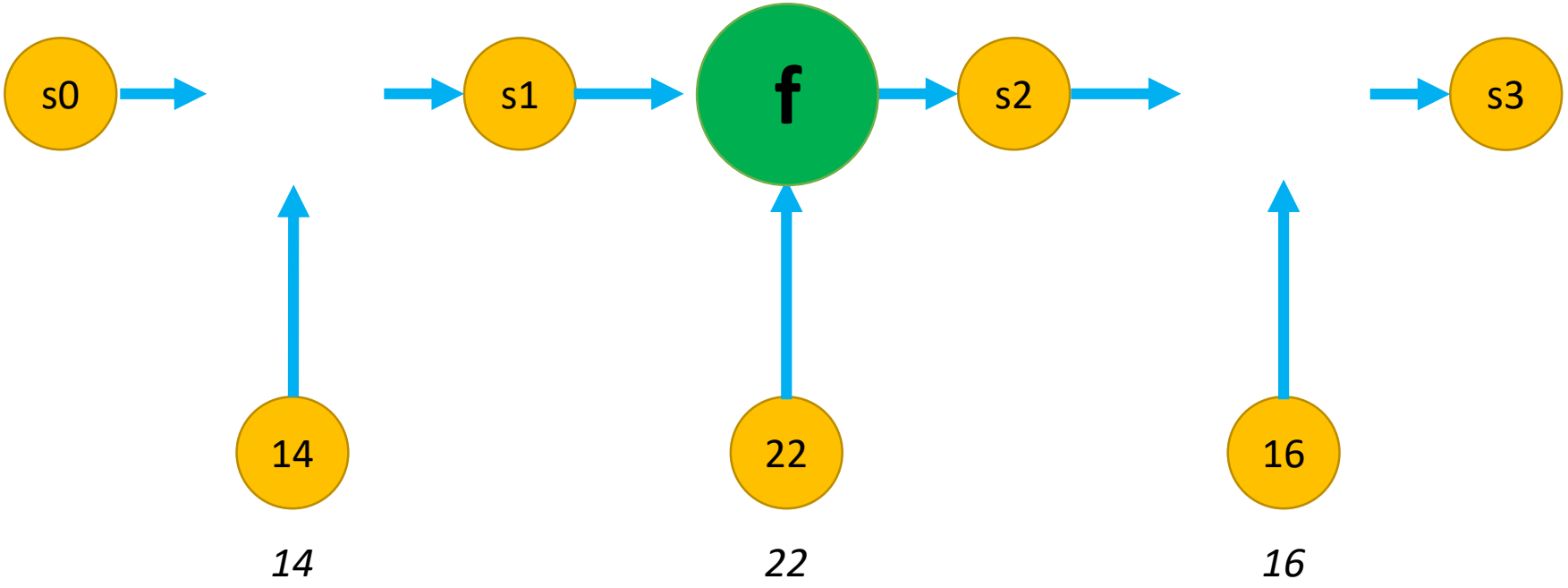
14 *22* *16*

Recurrent layer

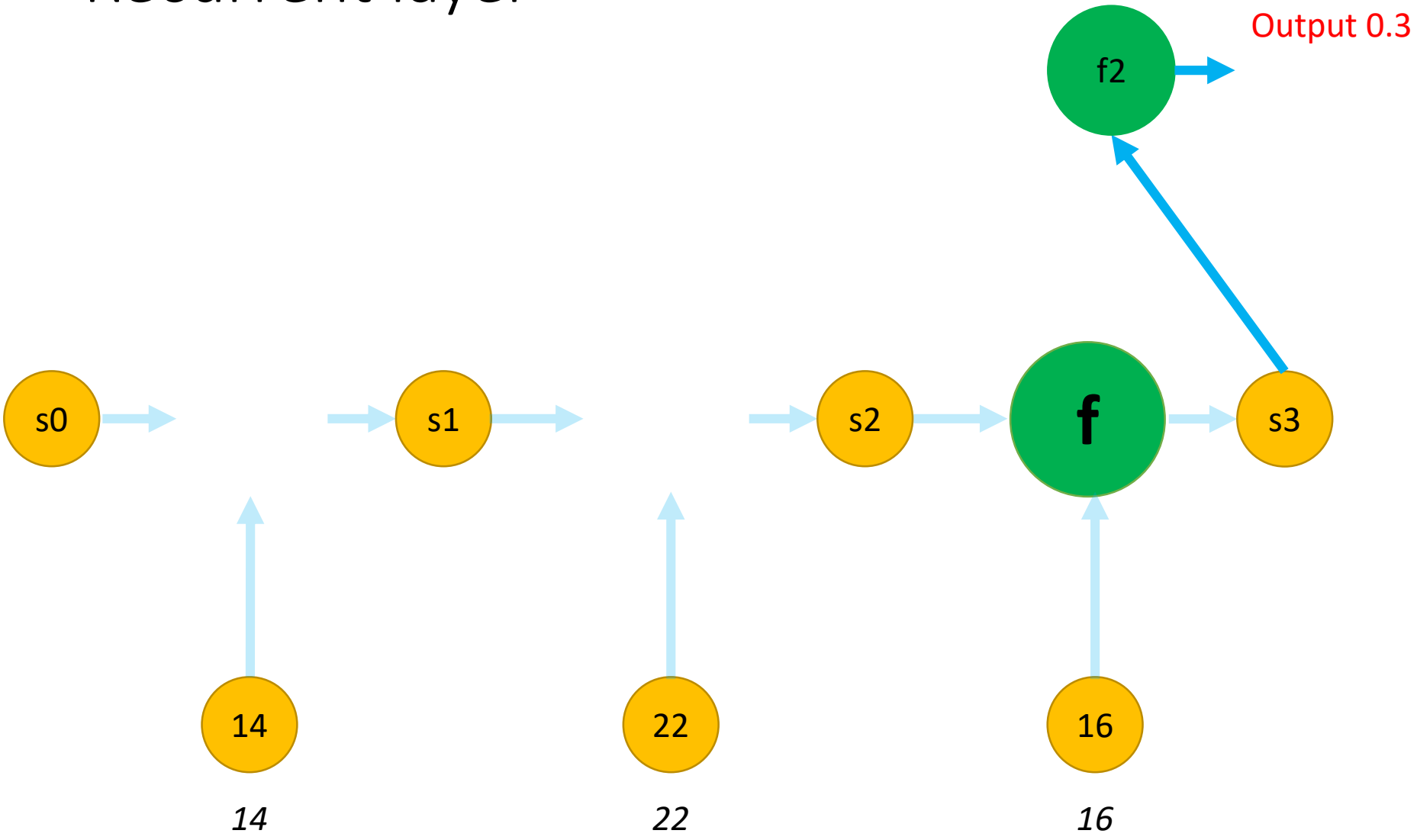


14 *22* *16*

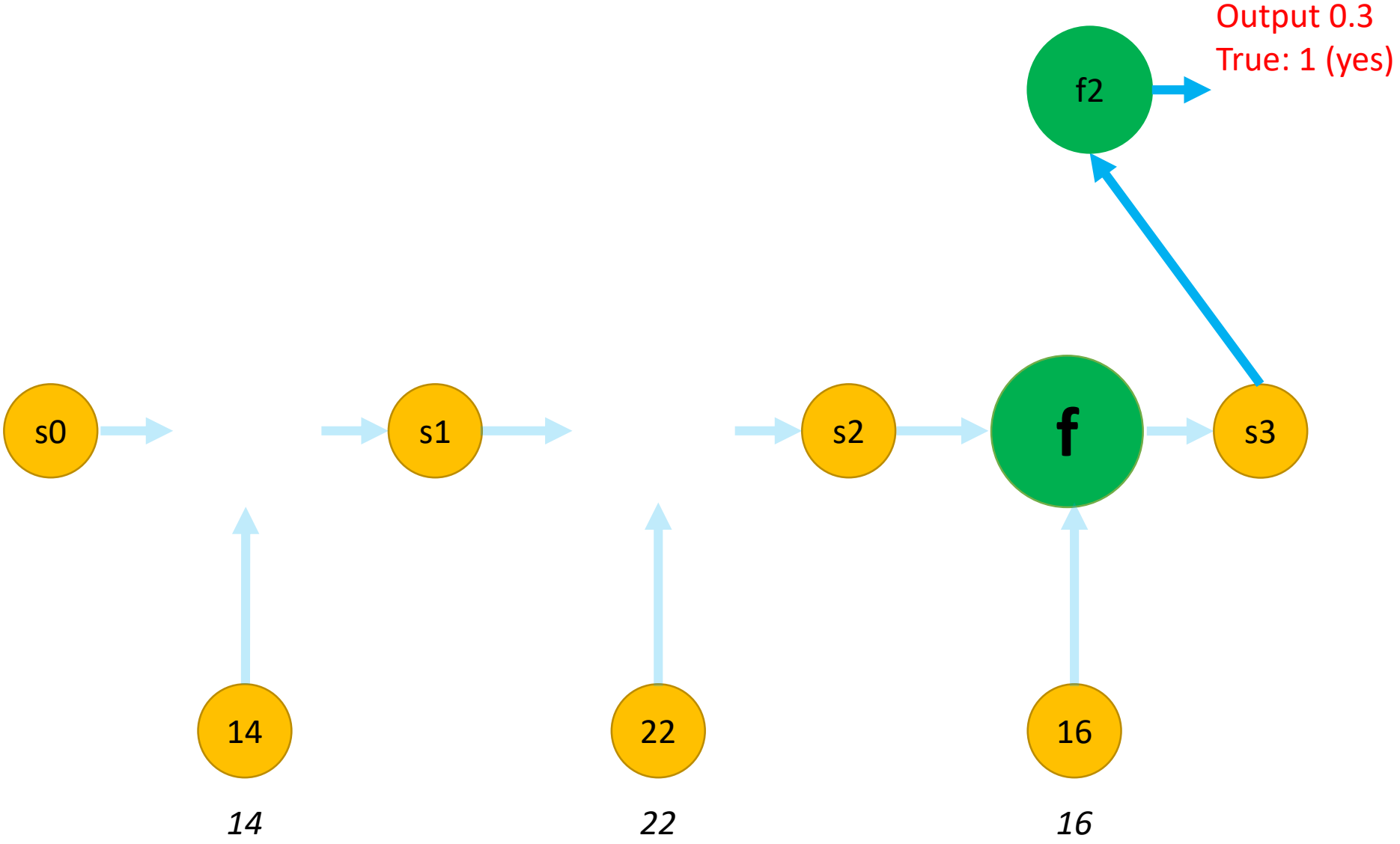
Recurrent layer



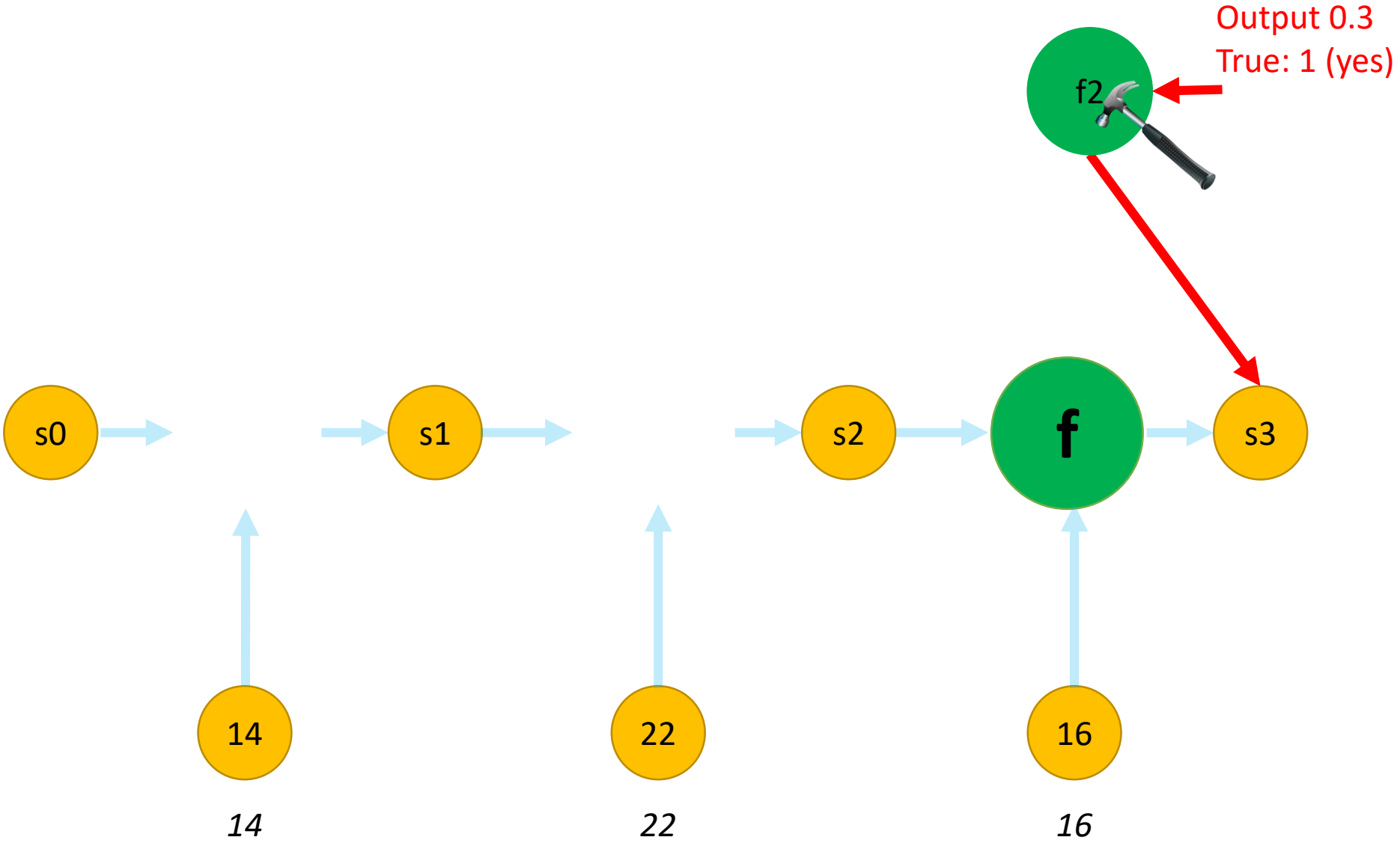
Recurrent layer



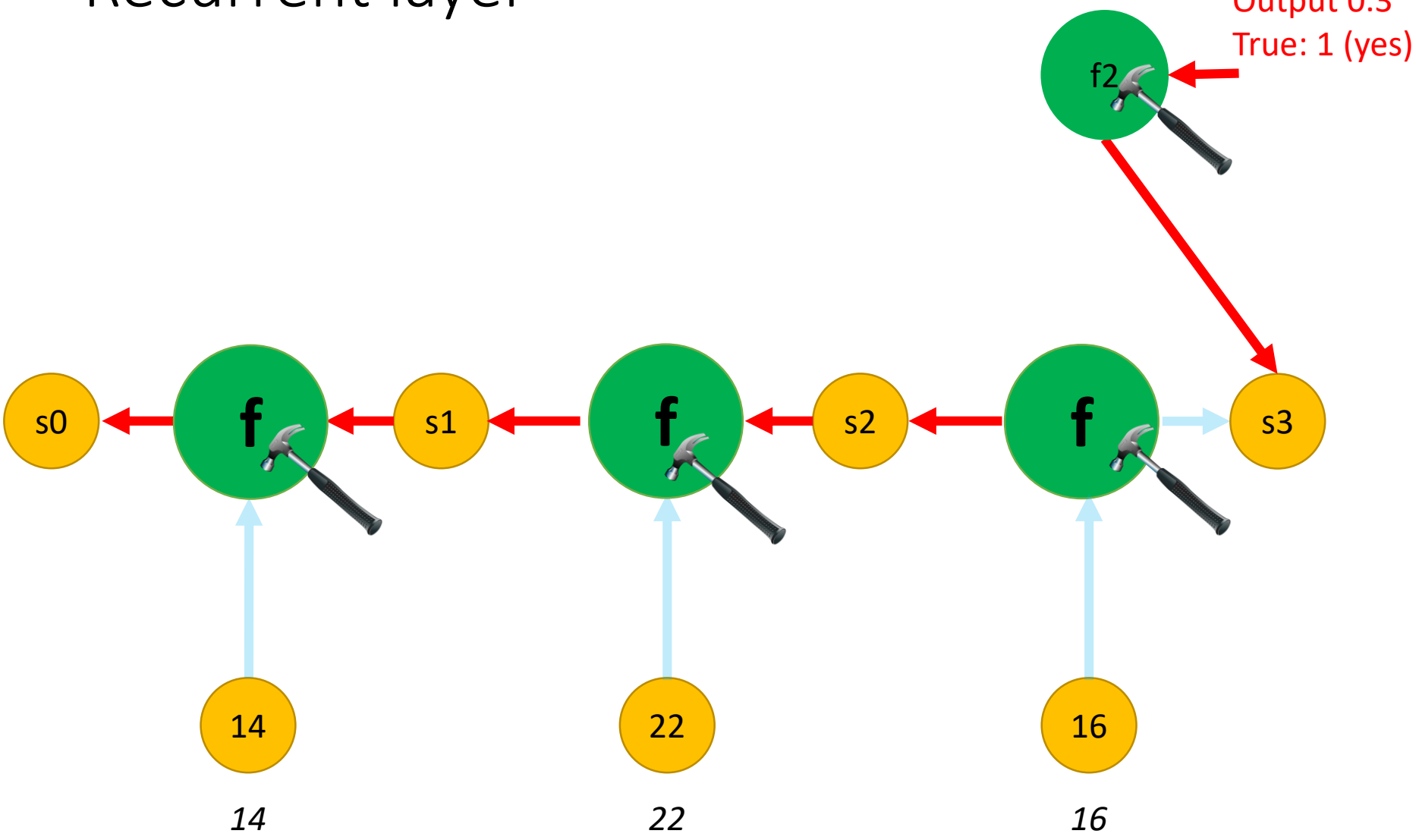
Recurrent layer



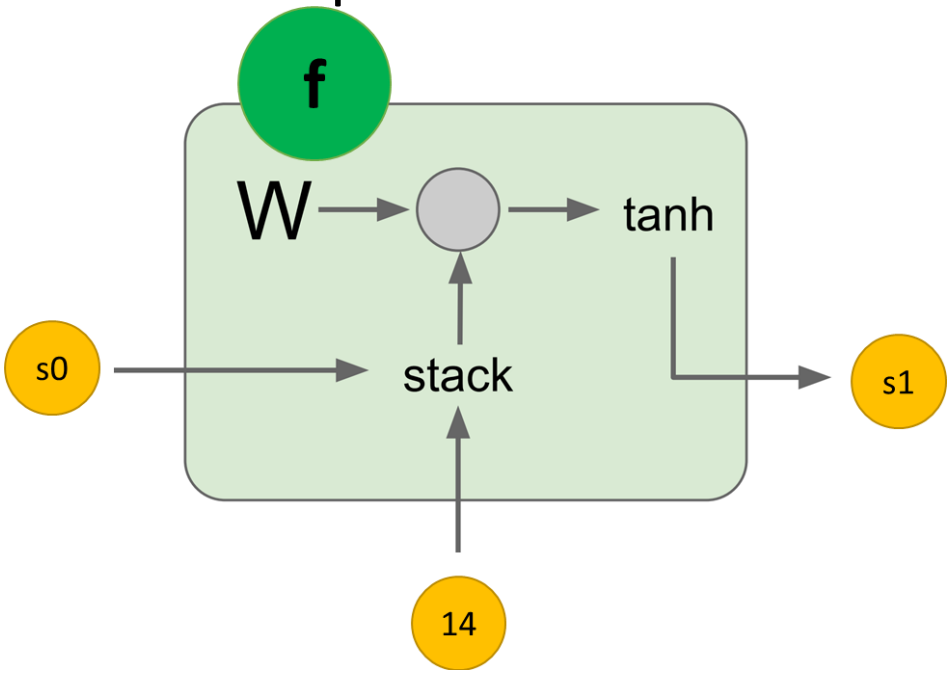
Recurrent layer



Recurrent layer



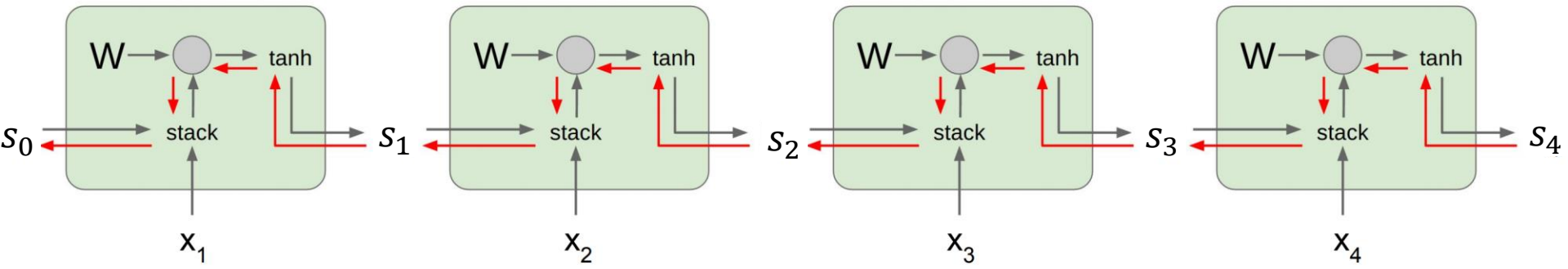
Cell implementation – Vanilla RNN




```
def call(self, inputs, state):
  """Most basic RNN: output = new_state = act(W * input + U * state + B)."""
  _check_rnn_cell_input_dtypes([inputs, state])
  gate_inputs = math_ops.matmul(
      array_ops.concat([inputs, state], 1), self._kernel)
  gate_inputs = nn_ops.bias_add(gate_inputs, self._bias)
  output = self._activation(gate_inputs)
  return output, output
```

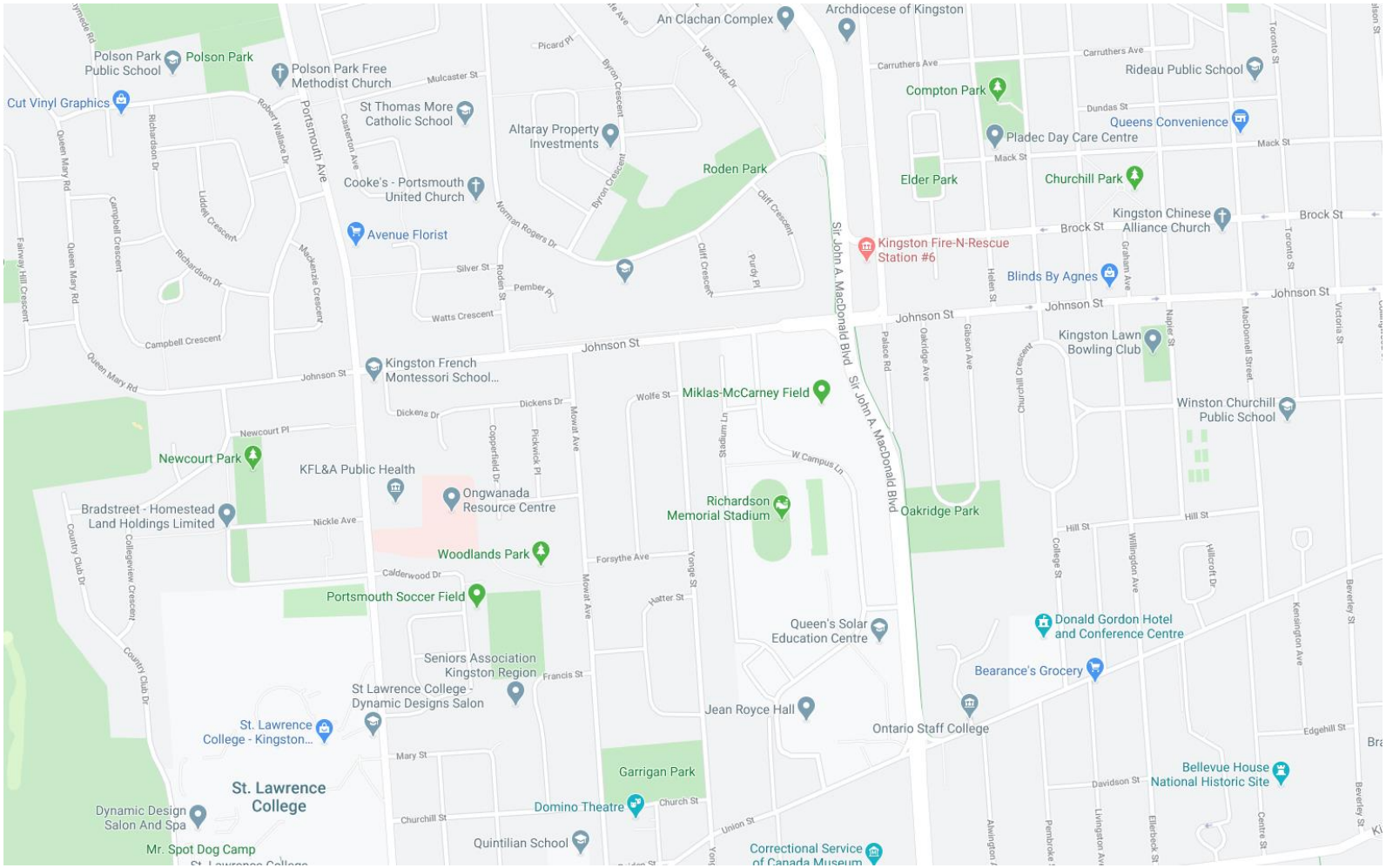
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Cell implementation – Vanilla RNN



Cell implementation – Vanilla RNN

- Problem: like driving in a driveway/roadway

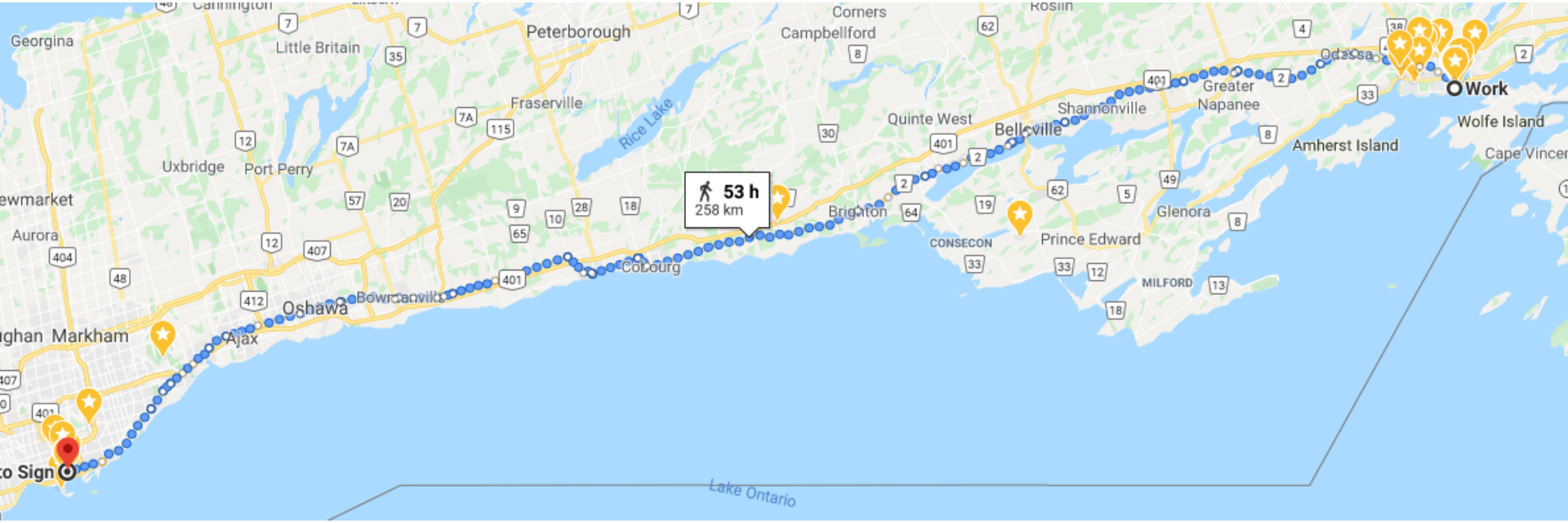


Cell implementation – Vanilla RNN

- Problem
  - Unable to capture dependencies across a long timestamp
  - Gradient Vanishing
    - The longer you travel back, the gradient will be come smaller and smaller due to the effect of chain rule
    - Can only learn (update parameters) from ending items in the sequence
  - Gradient Explode
    - The longer you travel back, the gradient will be come larger and larger due to the effect of chain rule
    - There is no such a memory unit can hold an infinitely increasing value
    - NaN gradient -> NaN update -> NaN parameter
      - NaN – Not a Number (overflow)

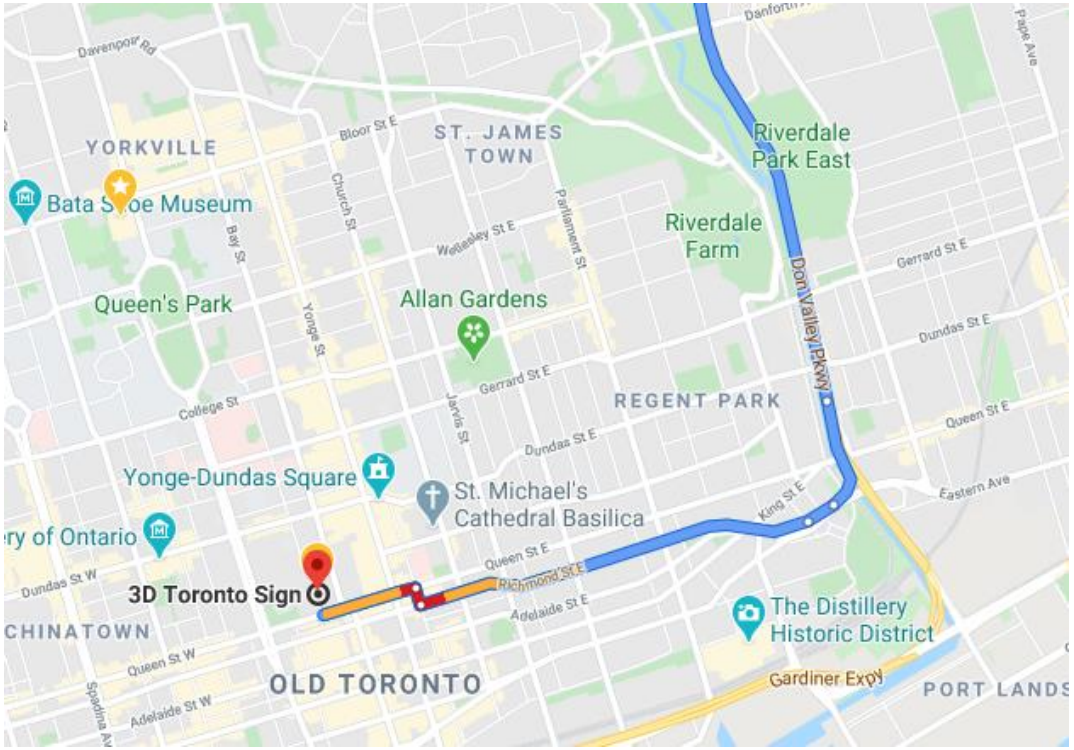
Cell implementation – Vanilla RNN

- Problem: like driving in a driveway/roadway



Cell implementation – Solution

- Highway!!
  - Travel to an area of your destination ultra fast
  - Then switch to the roadway/driveway to your final destination



Cell implementation – GRU

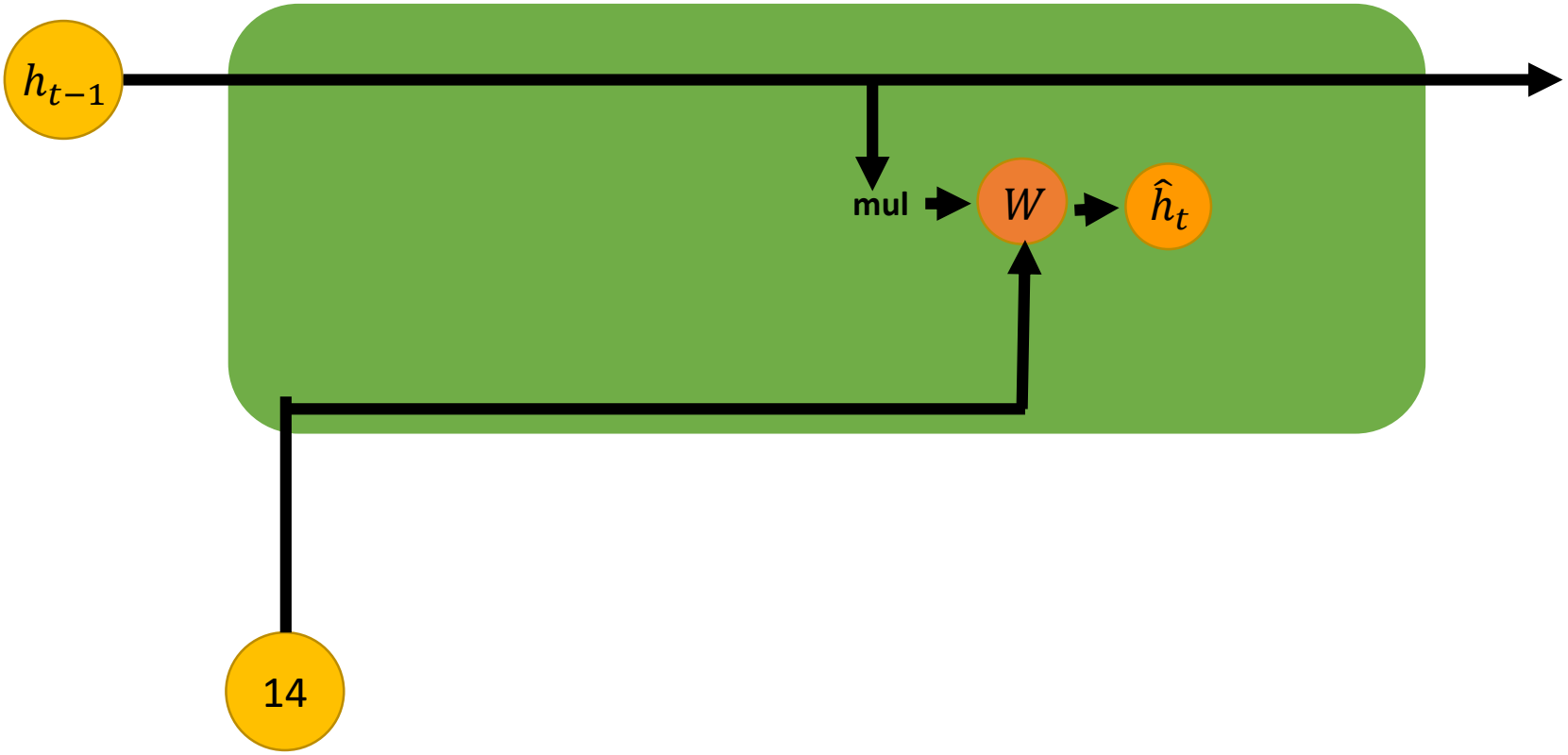
- GRU (similar idea to LSTM)
  - Long-short term memory
  - Long -> high way memory
    - Where to get off the high way
  - Short -> driveway/roadway memory
    - Where to pick up people (signal) of your final destination

Cell implementation – GRU

Cell implementation – GRU (high way)

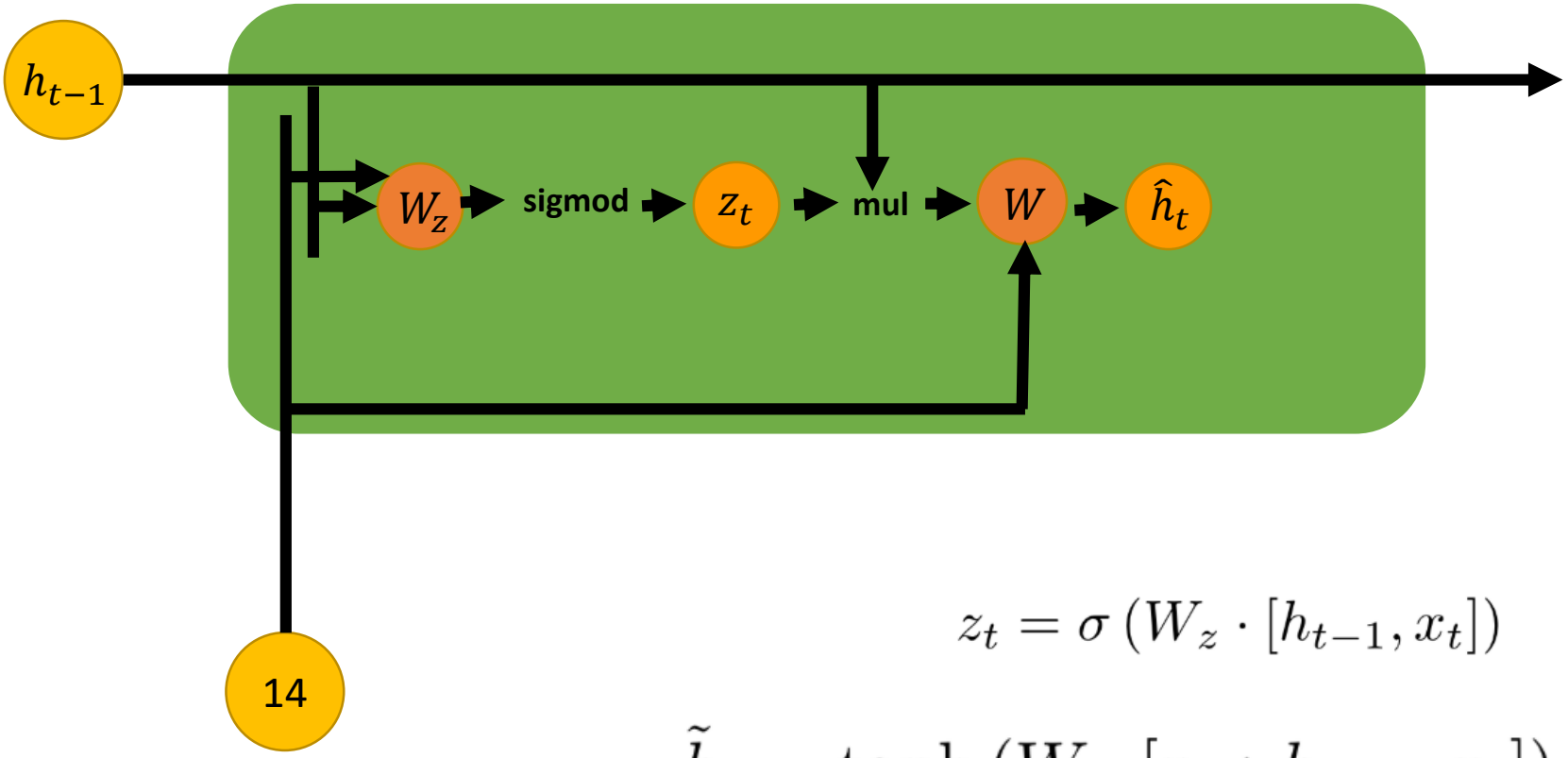
14

Cell implementation – GRU (roadway)



Cell implementation – GRU (roadway - gated)

Not all the `passengers` need to go down the highway
W_z create a gate z_t [0, 1] to determine who is going to go down the highway

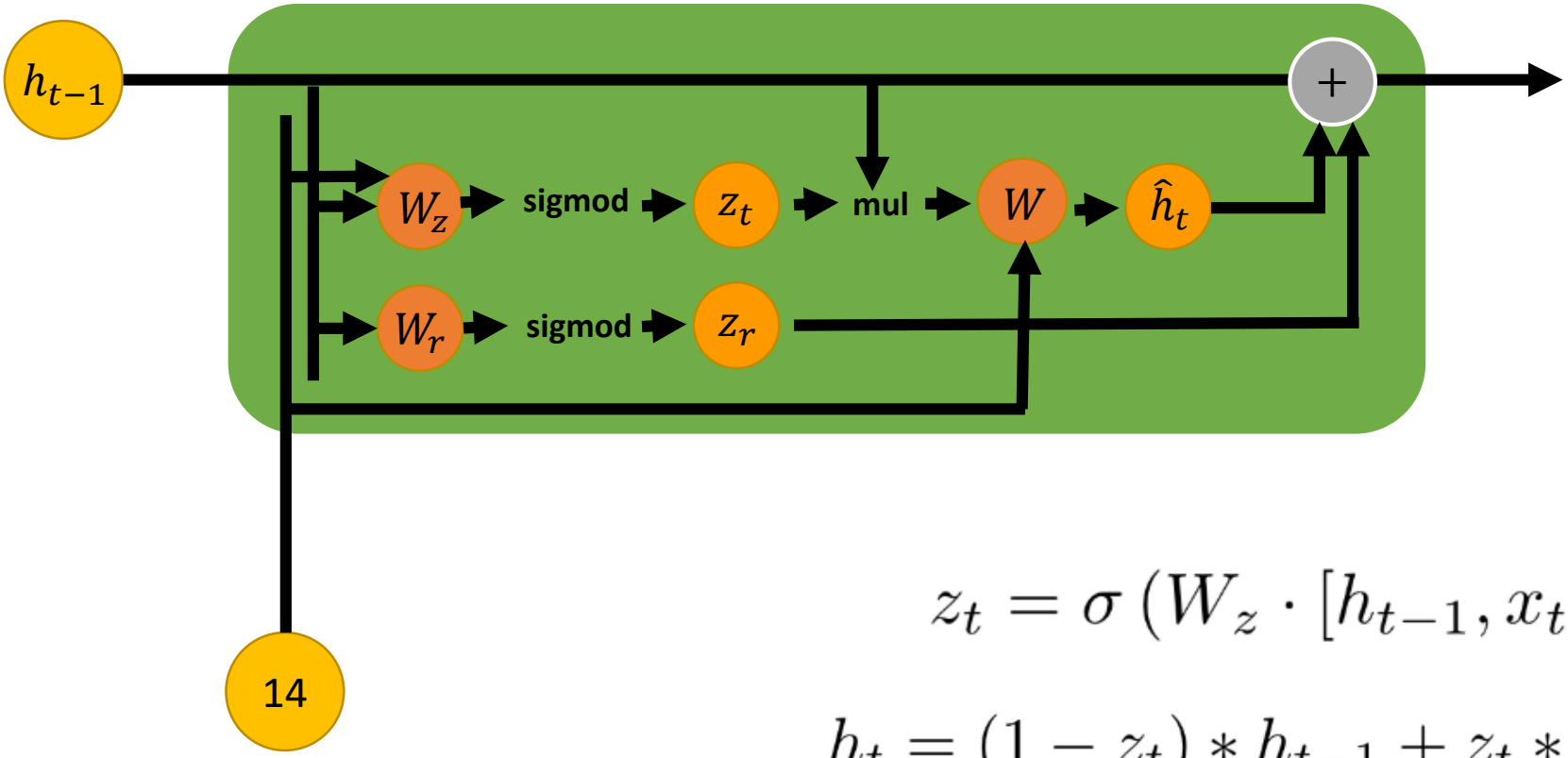


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

Cell implementation – GRU (merged)

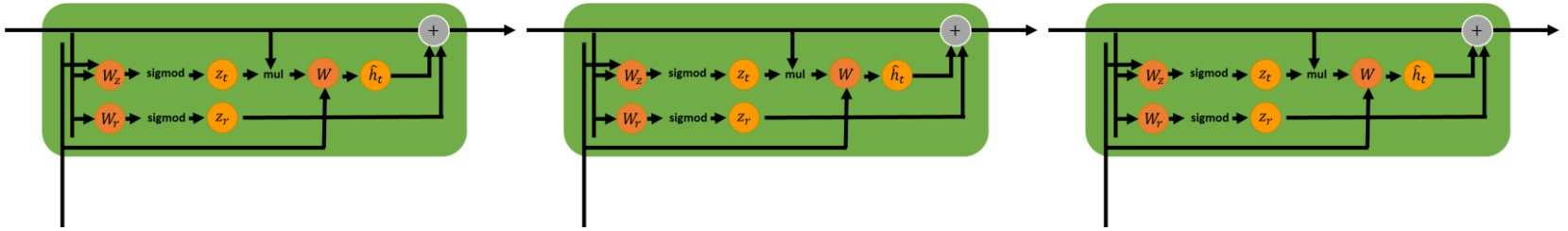
We also need to determine how many `passengers` need to pickup back to the highway. W_r create another gate z_r determine how roadway and high way are merged again



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

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

Cell implementation – GRU (merged)



14

22

16

14

22

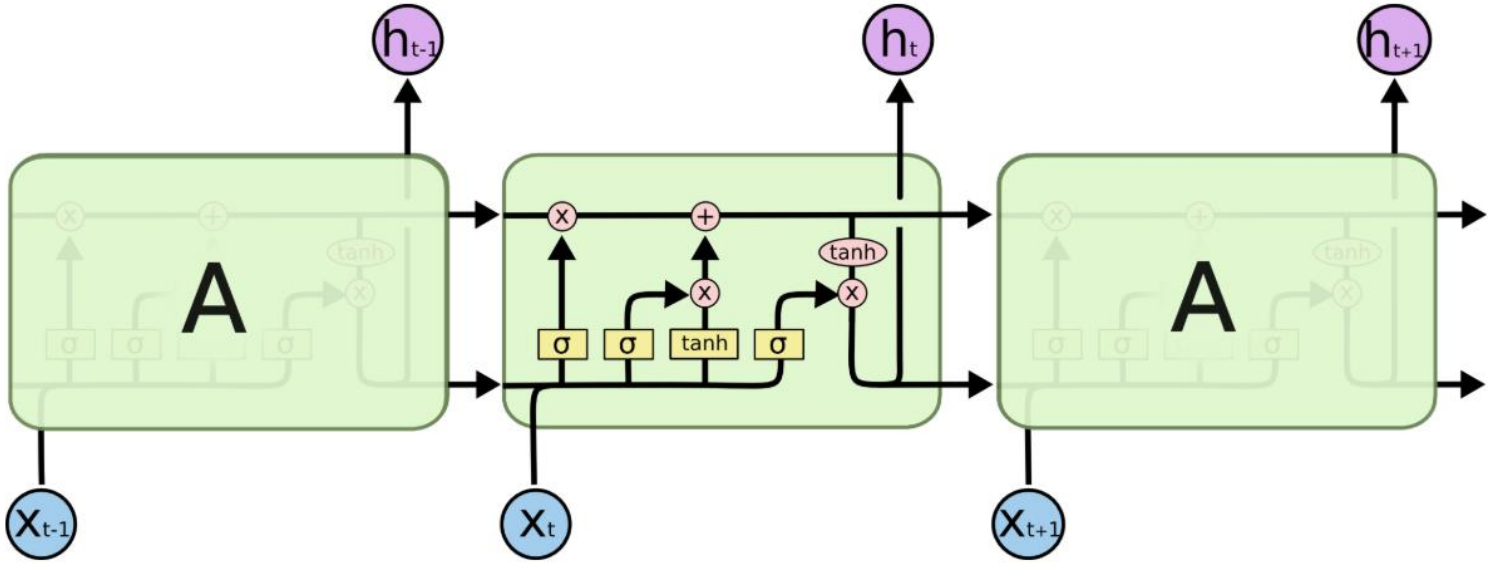
16

Cell implementation – GRU (merged)

```
def call(self, inputs, state):
  """Gated recurrent unit (GRU) with nunits cells."""
  _check_rnn_cell_input_dtypes([inputs, state])

  gate_inputs = math_ops.matmul(
      array_ops.concat([inputs, state], 1), self._gate_kernel)
  gate_inputs = nn_ops.bias_add(gate_inputs, self._gate_bias)

  value = math_ops.sigmoid(gate_inputs)
  r, u = array_ops.split(value=value, num_or_size_splits=2, axis=1)

  r_state = r * state

  candidate = math_ops.matmul(
      array_ops.concat([inputs, r_state], 1), self._candidate_kernel)
  candidate = nn_ops.bias_add(candidate, self._candidate_bias)

  c = self._activation(candidate)
  new_h = u * state + (1 - u) * c
  return new_h, new_h
```

Cell implementation – LSTM

- Created before GRU
- Two hidden states
  - One acts as long-term memory (always going highway)
  - One acts as short-erm memory (always going roadway)
- More gates
  - higher computational overhead
- Similar performance



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Attention Mechanism

- Can I go even faster?!?
  - Directly jump to the destination!!

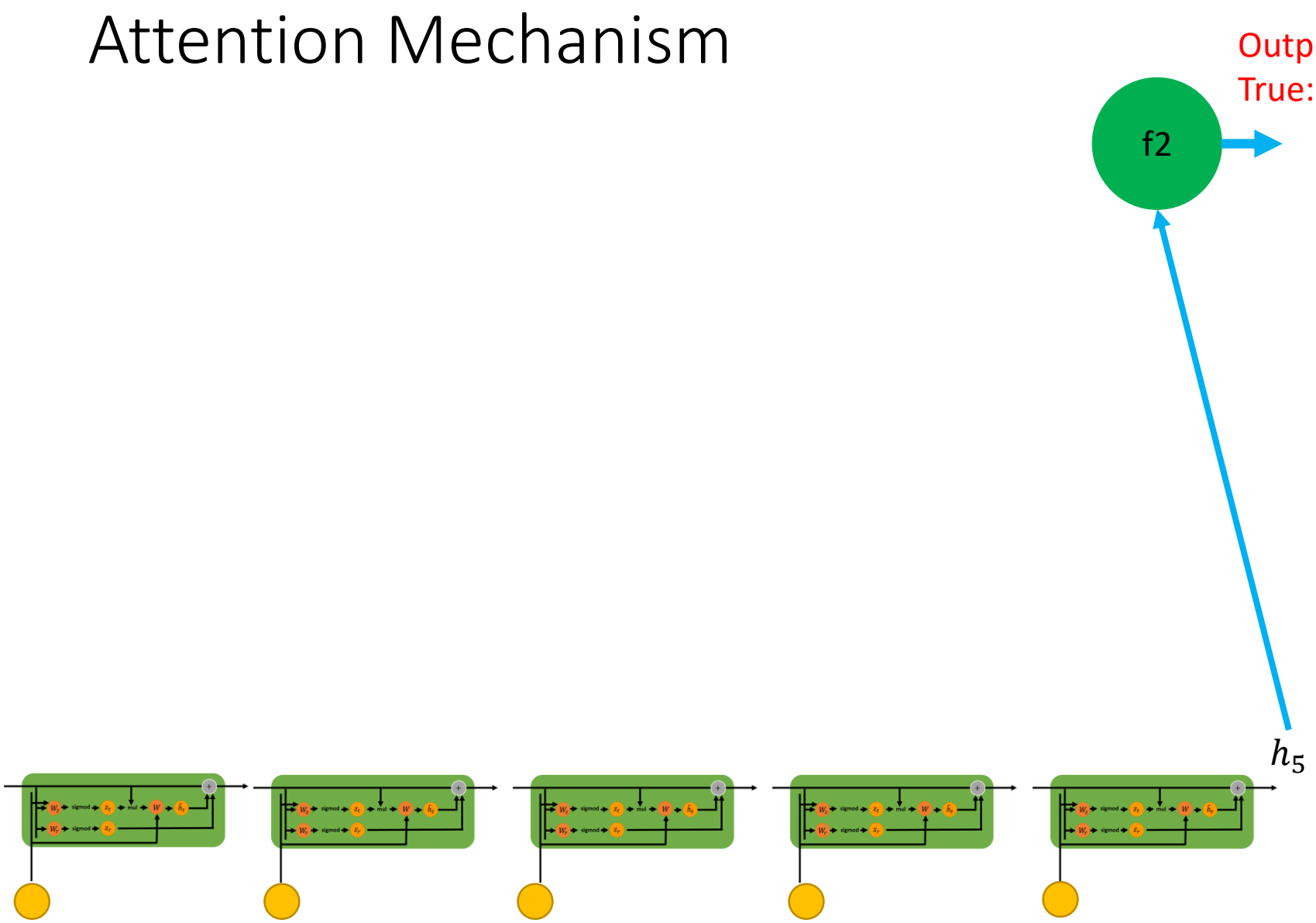
https://tenor.com/view/star-wars-light-speed-gif-4824465

Attention Mechanism

- Can I go even faster?!?
  - Directly jump to the destination!!

https://www.pinterest.ca/pin/726979564827879912/?lp=true

Attention Mechanism



Attention Mechanism