

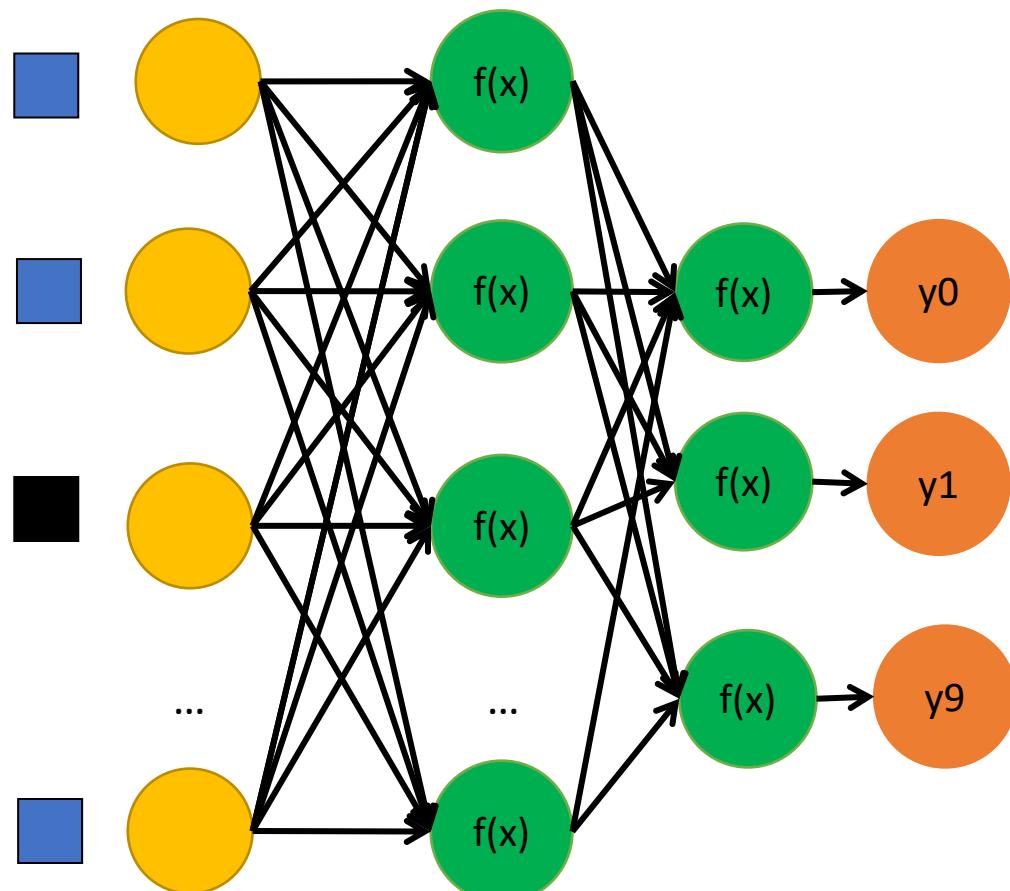
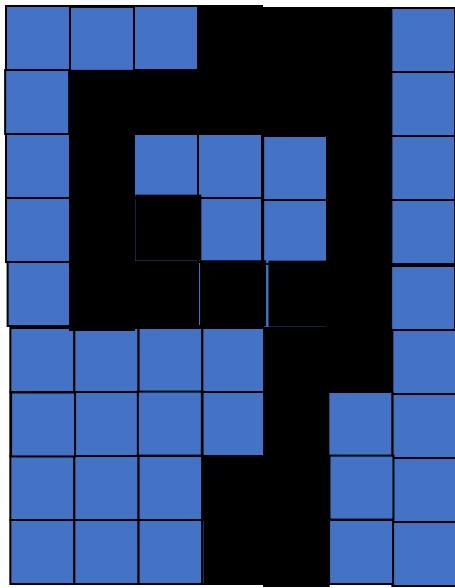
DATA MINING

/CISC 873 - Steven Ding
/Week #6
Spatial

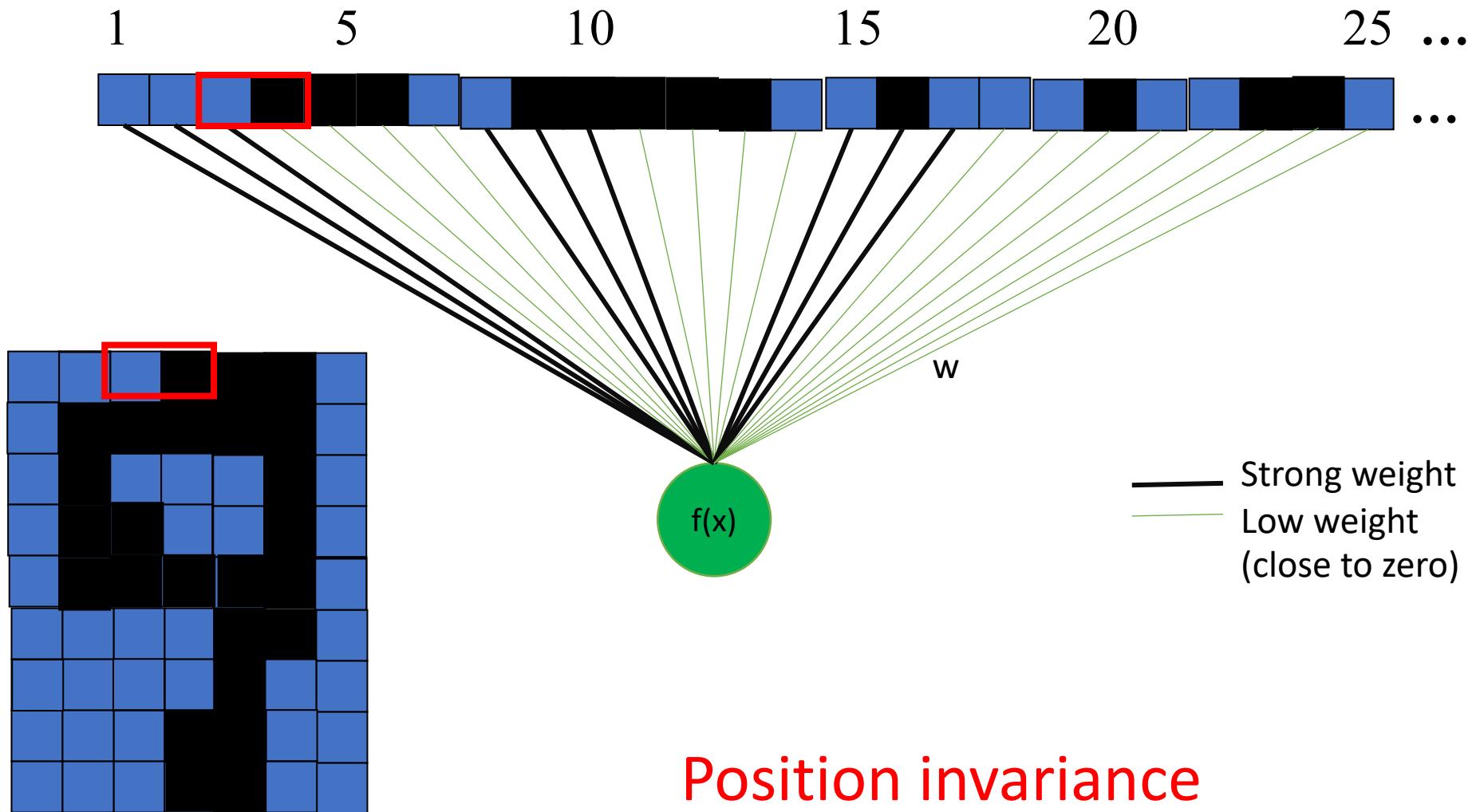
Neural Network cont'd



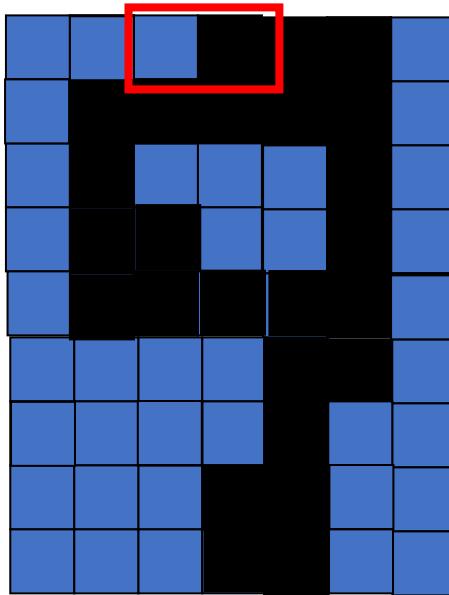
Neural Network cont'd



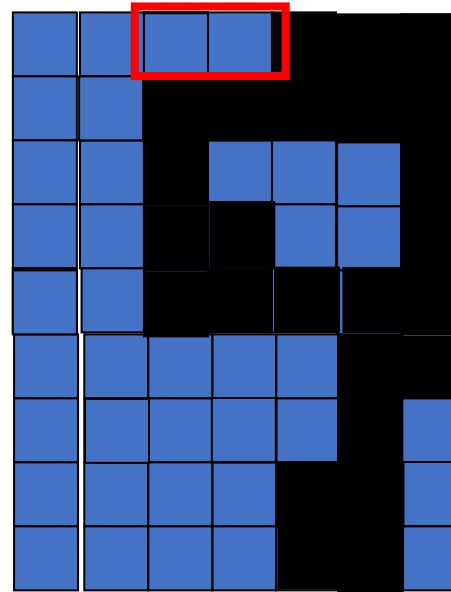
Neural Network cont'd



Neural Network cont'd



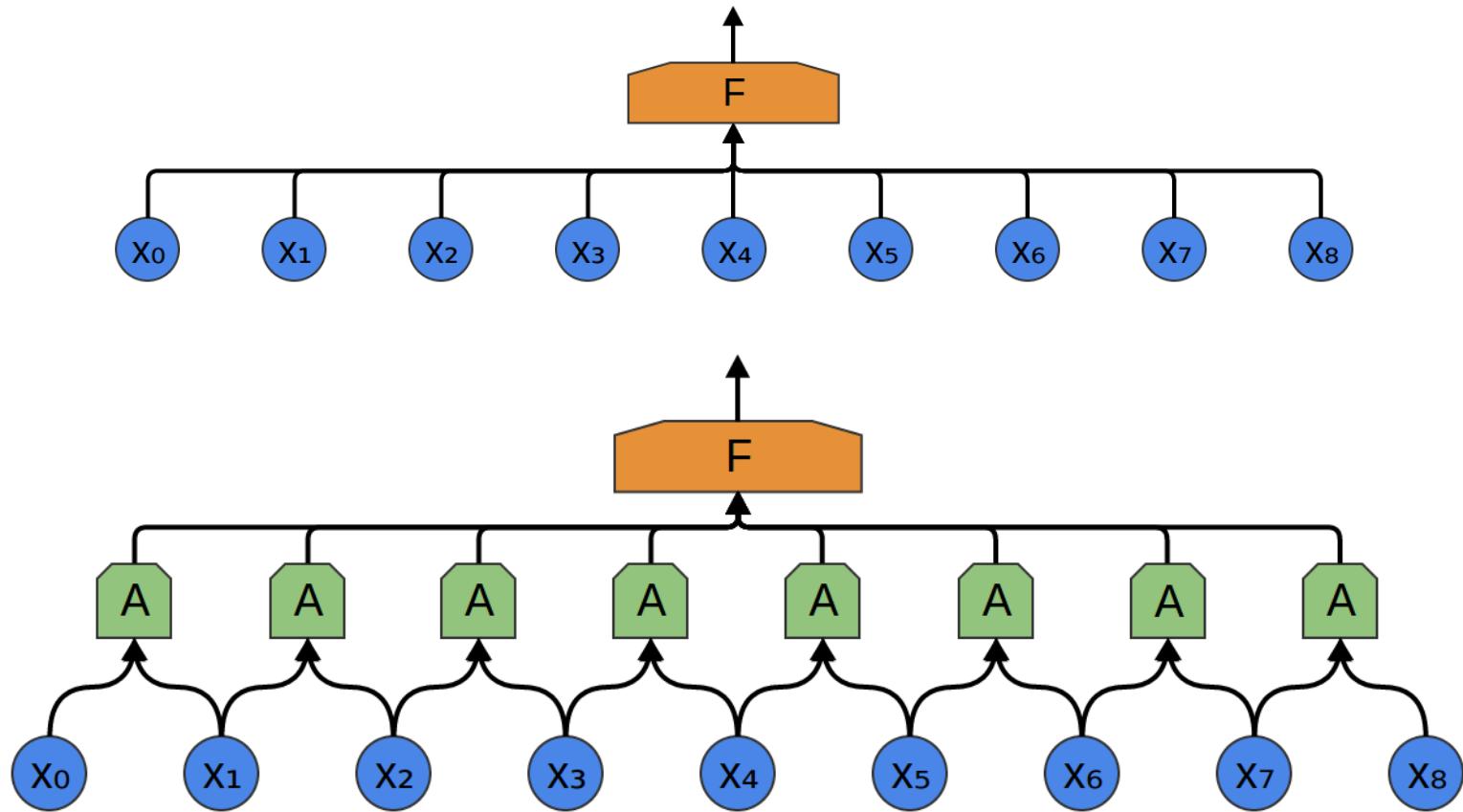
A training sample.



A testing sample. The old hidden unit failed to find the curve.

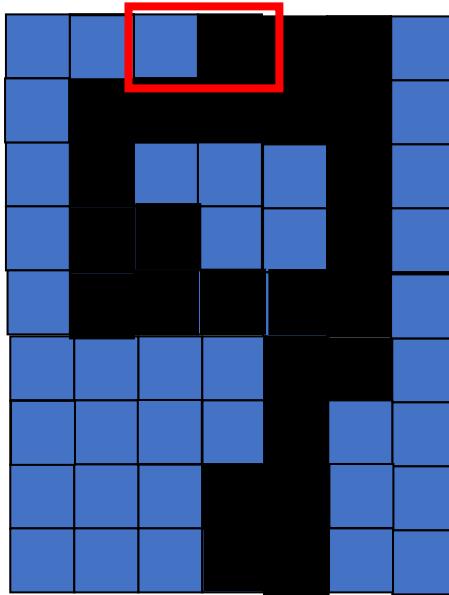
Position invariance

Fully Connected vs Conv1D

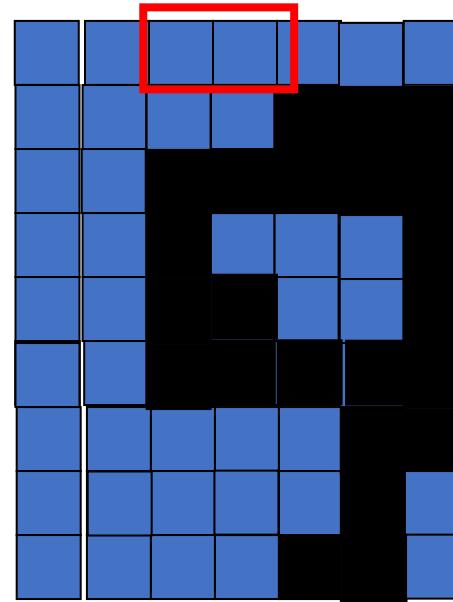


<http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>

Neural Network cont'd



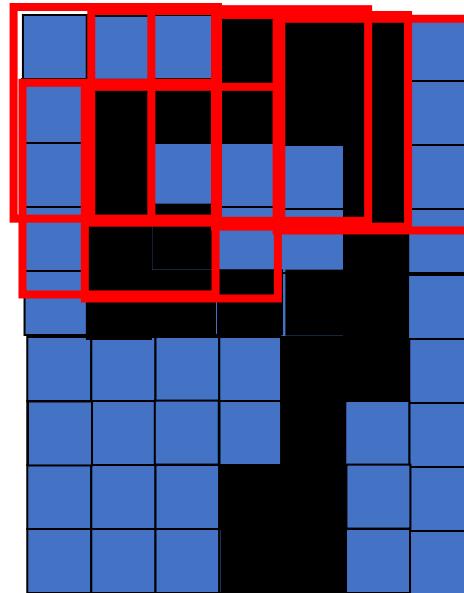
A training sample.



A testing sample. The old hidden unit failed to find the curve.

Position invariance

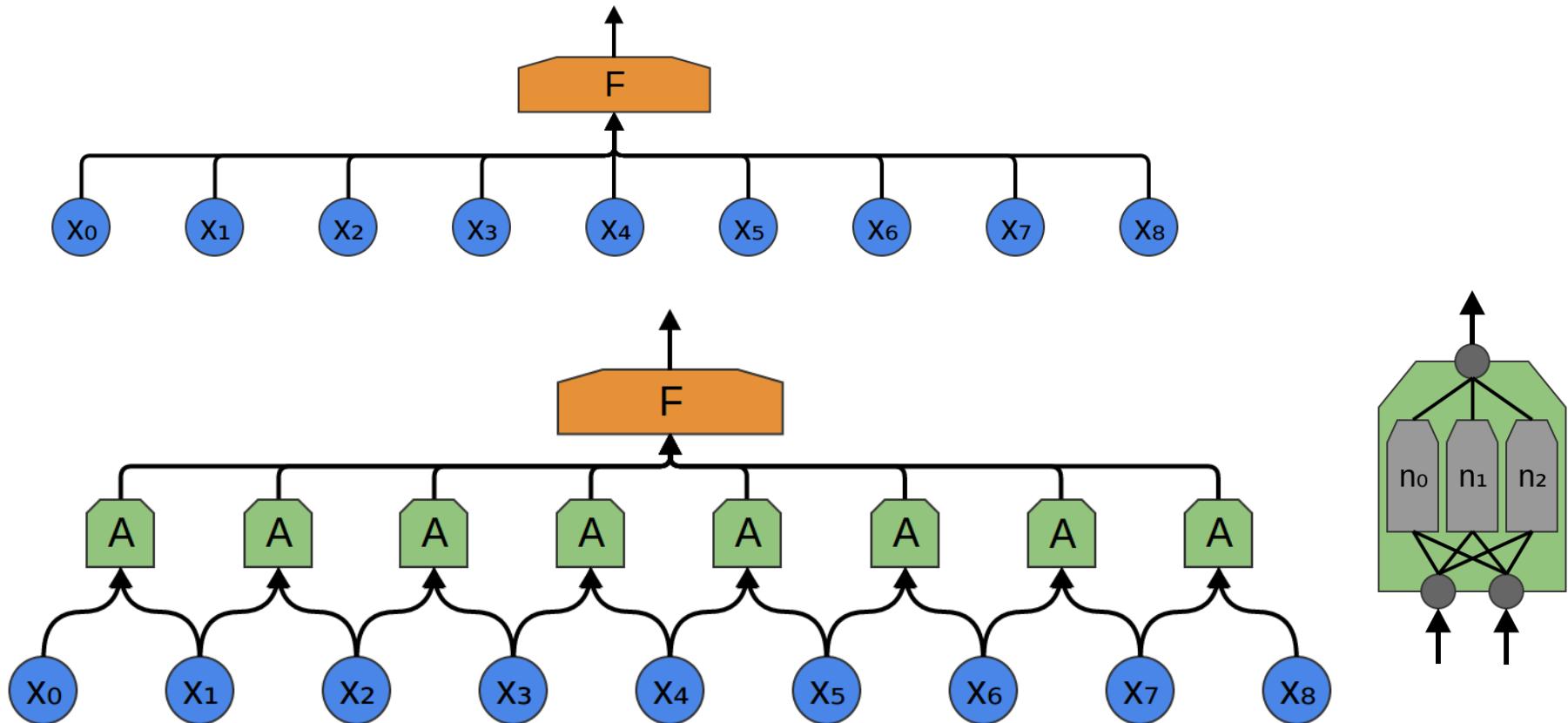
Conv2D



A training sample.

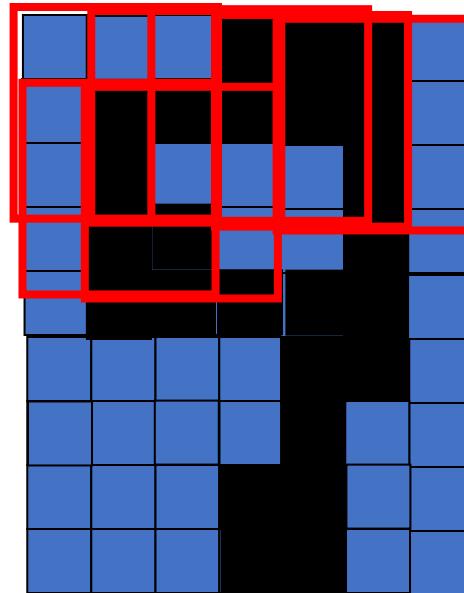
<https://www.youtube.com/watch?v=f0t-OCG79-U>

Fully Connected vs Conv1D



<http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>

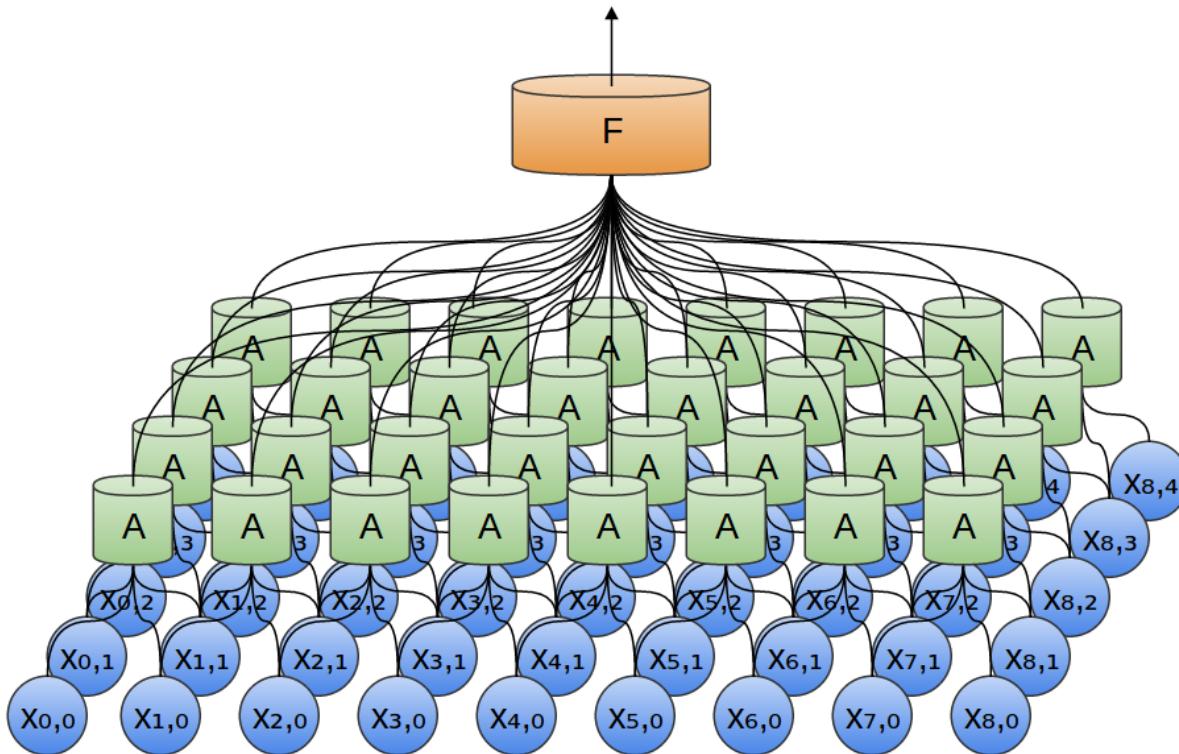
Convolutional Neural Network



A training sample.

<https://www.youtube.com/watch?v=f0t-OCG79-U>

Conv2D



<http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>

Conv2D – Single Filter

- A single kernel = filter = feature detector
 - Mapping window

1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1 <small>$\times 1$</small>	0	0
0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1	0
0 <small>$\times 1$</small>	0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1 <small>$\times 1$</small>	0	0
0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1	0
0 <small>$\times 1$</small>	0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

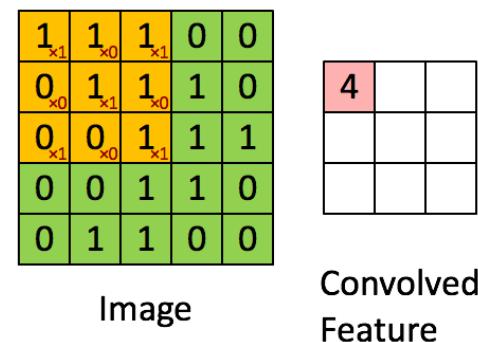
Convolved Feature

Kernel size

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- kernel size
 - Single integer or A tuple/list of two integers
 - Specify height/width of your sliding window
 - E.g. 3

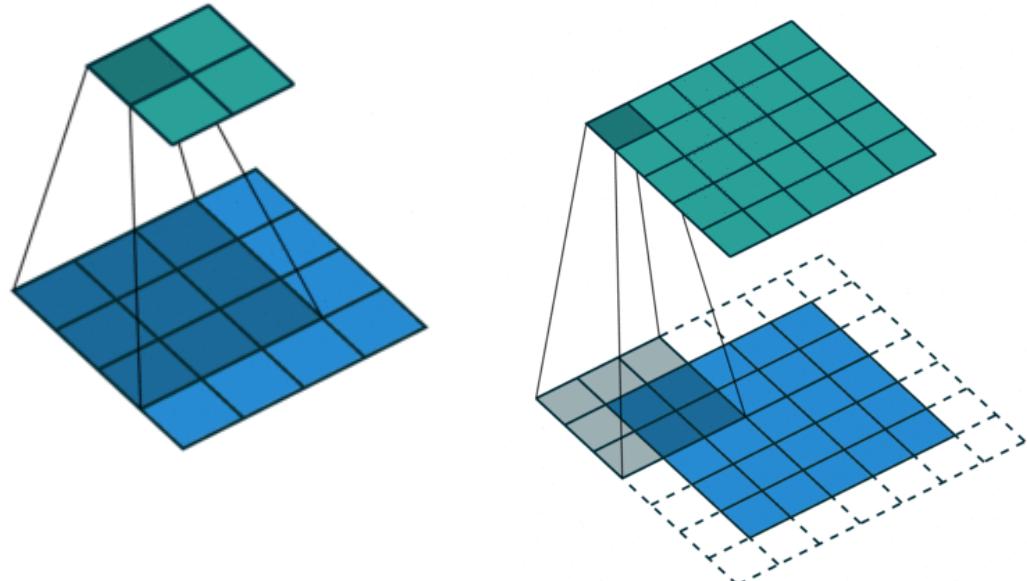
- Input:
 - [None, 5, 5, 1]
- Output:
 - [None, 3, 3, 1]



Padding

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- Padding
 - Valid/same
- Input:
 - [None, 5, 5, 1]
- Output:
 - [None, 5, 5, 1]



https://github.com/vdumoulin/conv_arithmetic

Filters

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- filters
 - Number of filters to be applied

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Filters

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- filters
 - Number of filters to be applied
 - E.g. 6
- Input:
 - [None, 5, 5, 1]
- Output:
 - [None, 3, 3, 6]

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Strides

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- Number of steps to go

- (2, 2)

- Input:

- [None, 5, 5, 1]

- Output:

- [None, 2, 2, 6]

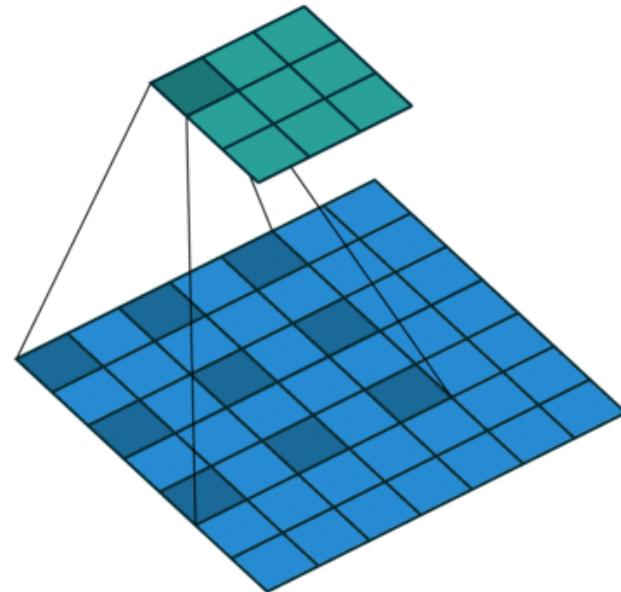
1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

Dilation Rate

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- Expanding the scanner



Filters – Single Chanel

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- filters
 - Number of filters to be applied
 - E.g. 6
- Input:
 - [None, 5, 5, 1]
- Output:
 - [None, 3, 3, 6]

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Filters – Multi Chanel

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

- filters
 - Number of filters to be applied
 - E.g. 6
- Input:
 - [None, 5, 5, 3]
- Output:
 - [None, 3, 3, 6]

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

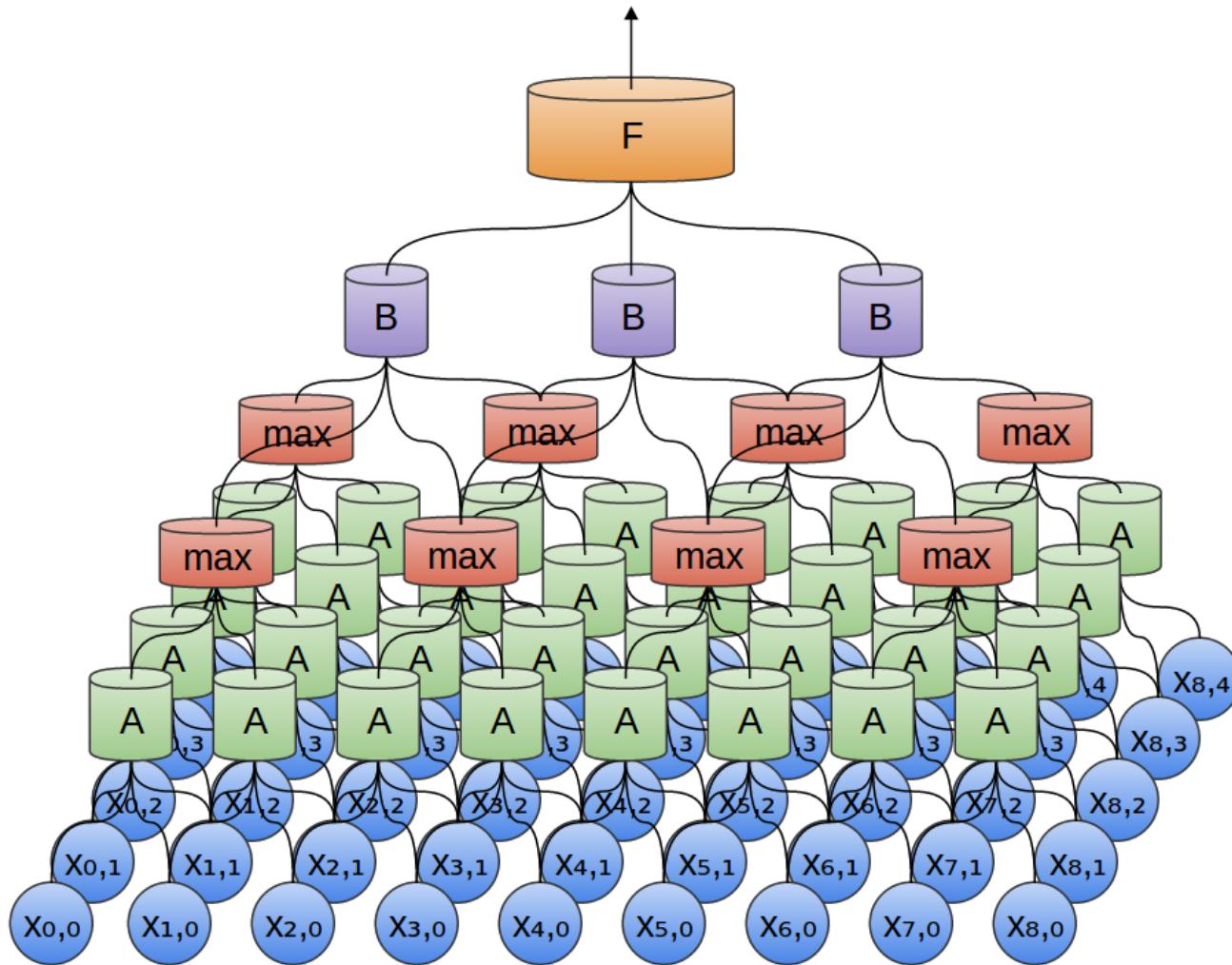
Convolved Feature

Convolutional Neural Network

- Features:
 - Position Invariance
 - Receptive field of different granularity
- Try it out:

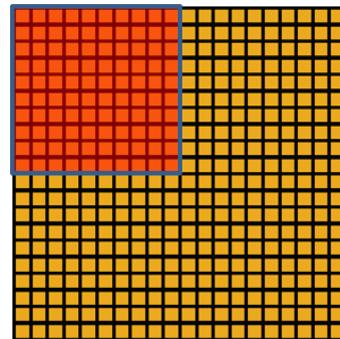
```
4 import tensorflow as tf
5 x = tf.random.normal(shape=(4, 5, 5, 1))
6
7 y = tf.keras.layers.Conv2D(2, 3, activation='relu', strides=2, padding='valid'))(x)
8 print(y.shape)
```

Pooling Layer

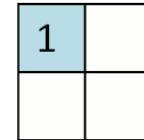


Convolutional Neural Network

- Pooling
 - Position Invariance Feature Extraction
 - Reduce the window
 - Average, Max, etc.



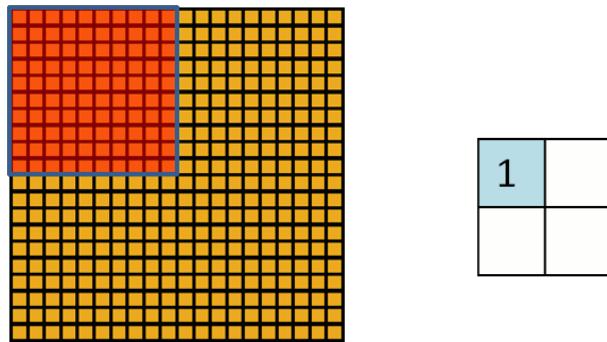
Convolved
feature



Pooled
feature

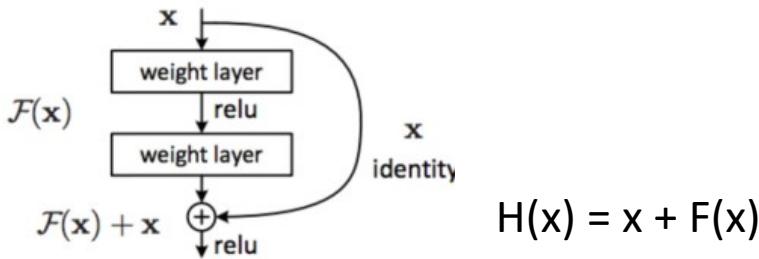
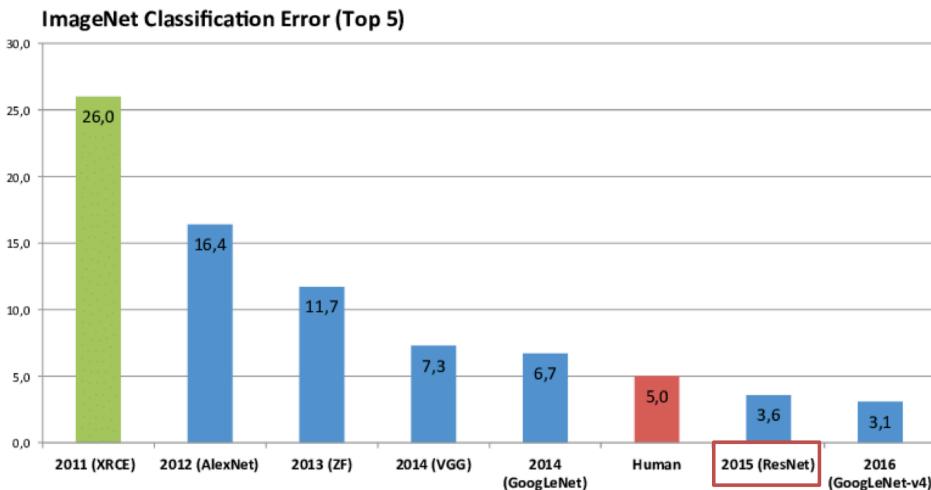
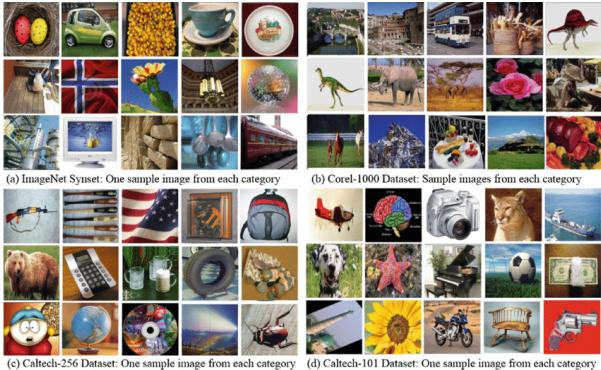
Pooling Layer

```
tf.keras.layers.MaxPool2D(  
    pool_size=(2, 2), strides=None, padding='valid', data_format=None, **kwargs  
)
```



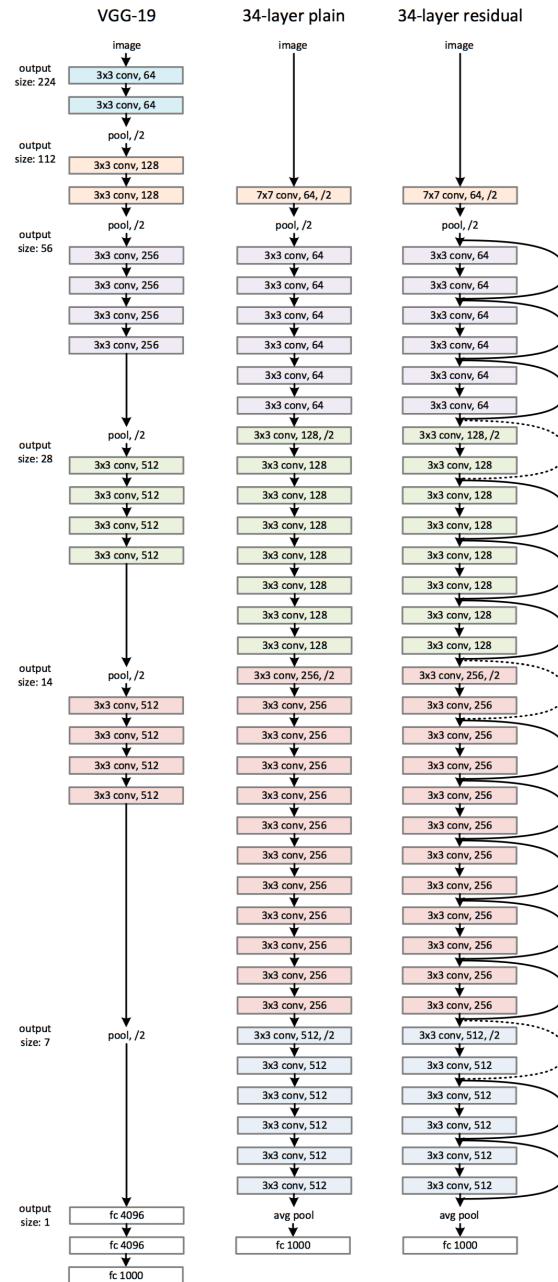
Convolved
feature Pooled
feature

Application



October 23, 2020

CISC 873 - Data Mining



25

DATA MINING

/CISC 873 - Steven Ding
/Week #6
Temporal

Next

- Sequential Analysis
 - Classification
 - Preprocessing
 - Representations
 - BOW
 - N-gram
 - character n-gram
 - Part-Of-Speech Tagging
 - Dependency Tree
 - Vanilla RNN

The Web contains a LOT of text.

- What can we do with it?
 - read it, with the help of search engines
 - communicate (email, IM, social media, ...)
 - learn about how humans inform and communicate, by reversing engineering language use to infer properties of the writer
- Text Analytics
 - Sentiment Analysis
 - Authorship Analysis
 - Socio-economic characteristics inference
 - As a 'reverse-engineering' problem

Sentiment Analysis

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

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



Preprocessing

- Case normalization
 - All lower case (or all higher case)
 - Case may carry critical information depending on the problem
 - E.g. Cases carry writing style
- Stop words & Punctuation removal
 - i, me, my, myself, we, our, ours, ourselves, you, your, yours, yourself, yourselves, he, him, his, himself, she, her, hers, herself, it, its, itself, they, them, their, theirs, ...
 - Domain-driven
 - carries writing style
 - carries semantic information
 - "few" vs "a few"

Preprocessing

- Stemming
 - Stemmers remove morphological affixes from words, leaving only the word stem. (nltk)
 - `print(stemmer.stem("running"))`
 - run
 - `print(stemmer.stem("generously"))`
 - generous

Representation

- How can we represent sequential text data as a numeric vector (from **unstructured** data to **structured** data)

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



Bag-of-Words model

- Represent each unique word as a **feature**, and the value can be
 - Term frequency (TF)
 - Term_frequency / document_frequency (TF-IDF)
- "This film was just brilliant"
 - > {"this": 1, "film":1, "was": 1, "just": 1, "brilliant":1}
 - > [1,1,1,1]

Document-Word Matrix

- doc1: "this is an apple "
- doc2: "this is an orange"

	an	this	is	apple	orange
doc1	1	1	1	1	0
doc2	1	1	1	0	1

Document-Word Matrix

- doc1: "The dog bit the man"
- doc2: "The man bit the dog"

	the	dog	bit	man
doc1	2	1	1	1
doc2	2	1	1	1

Bag-of- n -gram model

- Represent unique *word sequence of length n as feature*, value can be:
 - Term frequency (TF)
 - Term_frequency / document_frequency (TF-IDF)
- "This film was just brilliant"
 - > {"this_film": 1, "film_was":1, "was_just": 1, "just_brilliant": 1}
 - > [1,1,1]

Document-Word Matrix (n-gram model)

- doc1: "The dog bit the man"
- doc2: "The man bit the dog"

	the-dog	dog-bit	bit-the	the-man	man-bit
doc1	1	1	1	1	0
doc2	2	0	1	1	1

Bag-of- n -perm model

- Represent unique *unordered word sequence of length n as feature*, value can be:
 - Term frequency (TF)
 - Term_frequency / document_frequency (TF-IDF)
- "It is an apple. Is it?"
 - > {"it-is": 2, ...}
 - > [2, ...]

Bag-of-character- n -gram model

- Represent unique *character sequence of length n as feature*, value can be:
 - Term frequency (TF)
 - Term_frequency / document_frequency (TF-IDF)
- "This film was just brilliant"
 - > {"th": 1, "hi": 1, "is": 1, "s_f": 1, ...}
 - > [1,1,1, ...]

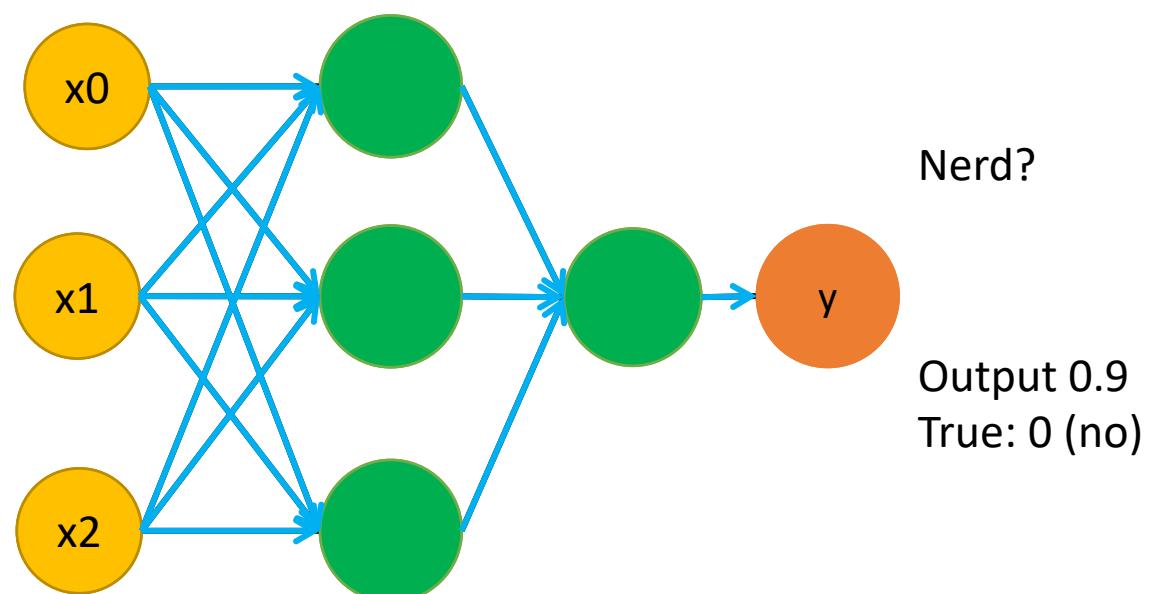
Term weighting

- Similar to feature weighting
- The more frequent a term occur in a document, the less important it is:
 - Term is weighted by IDF (inversed document frequency)
- Information gain
- ...

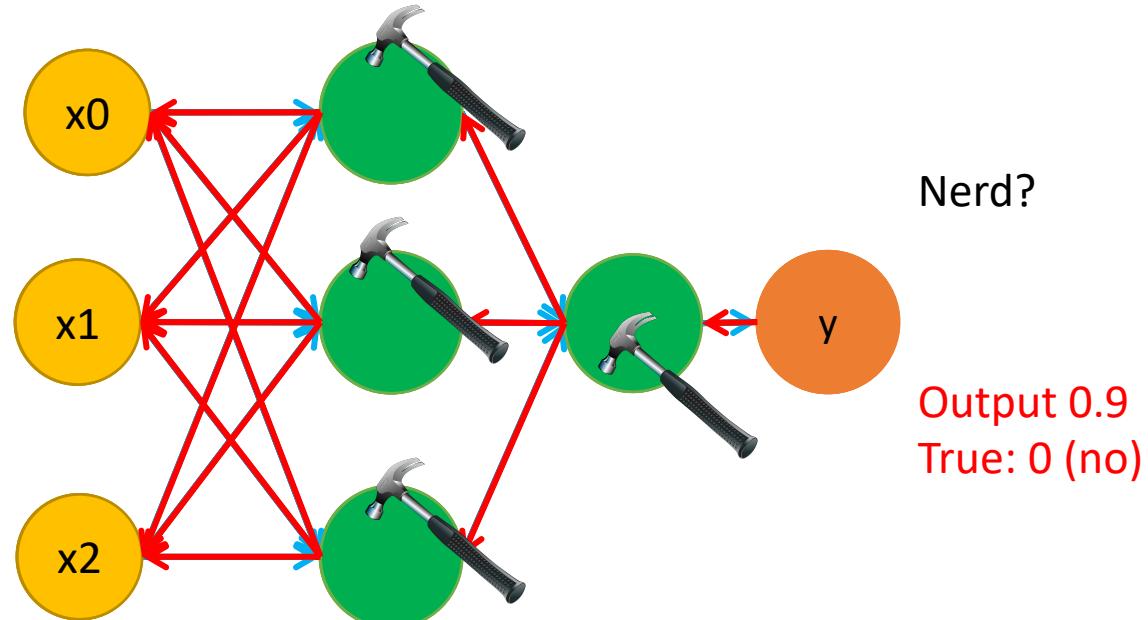
Recurrent Neural Network

- Treating text as a sequence of discrete signals

Neural Network - Forward



Neural Network - Backward



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.

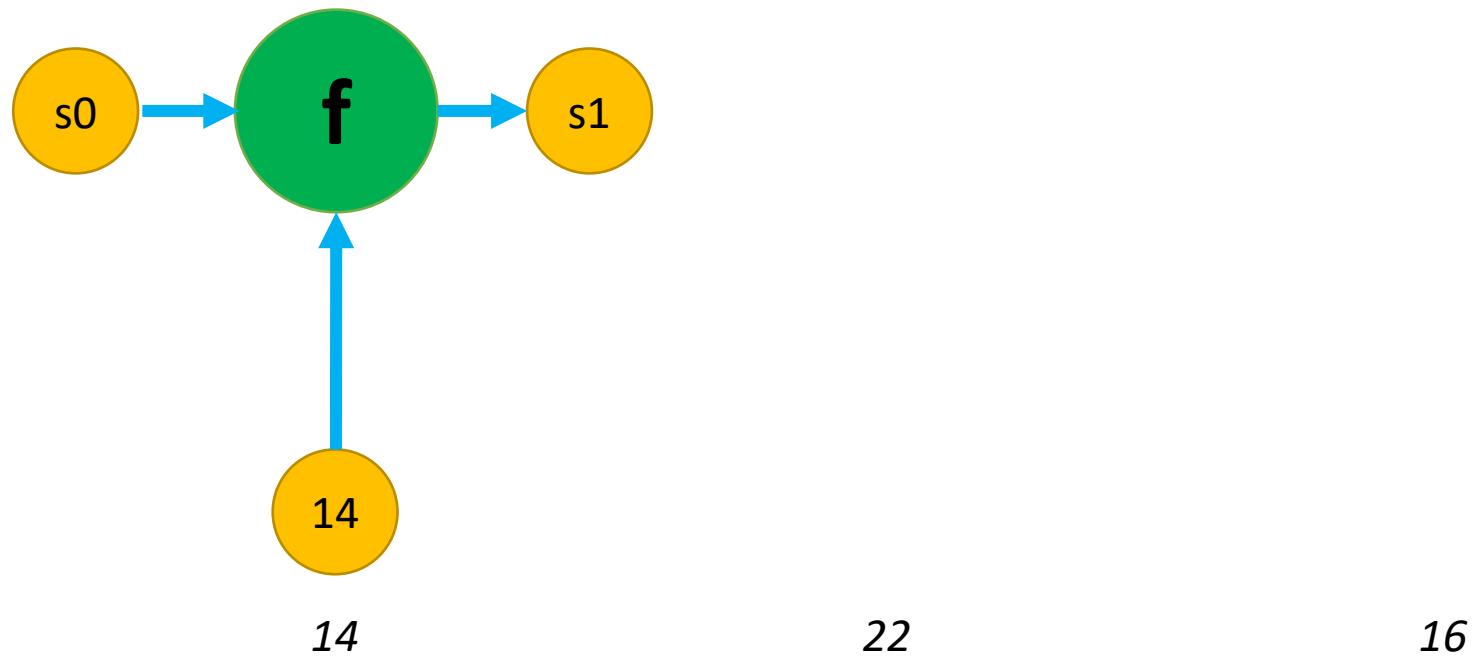
Building vocabulary

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

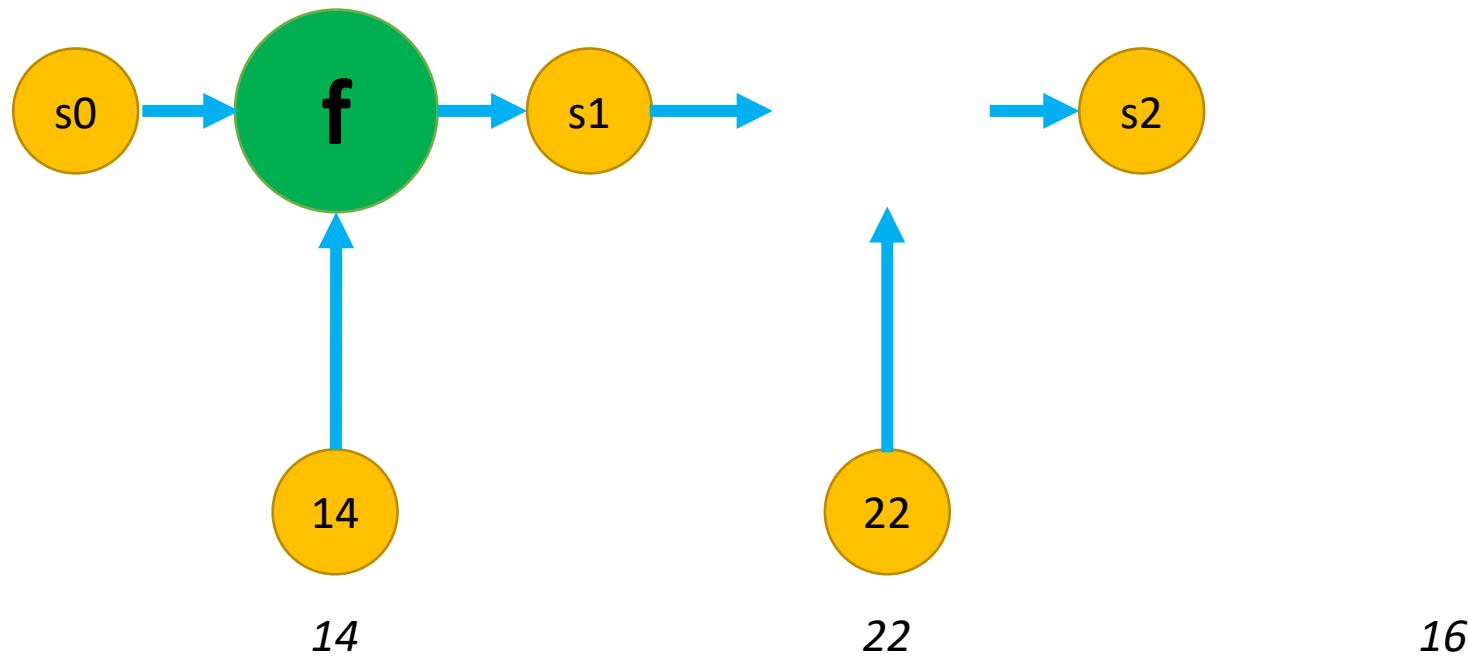
this film was just brilliant casting location scenery story

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

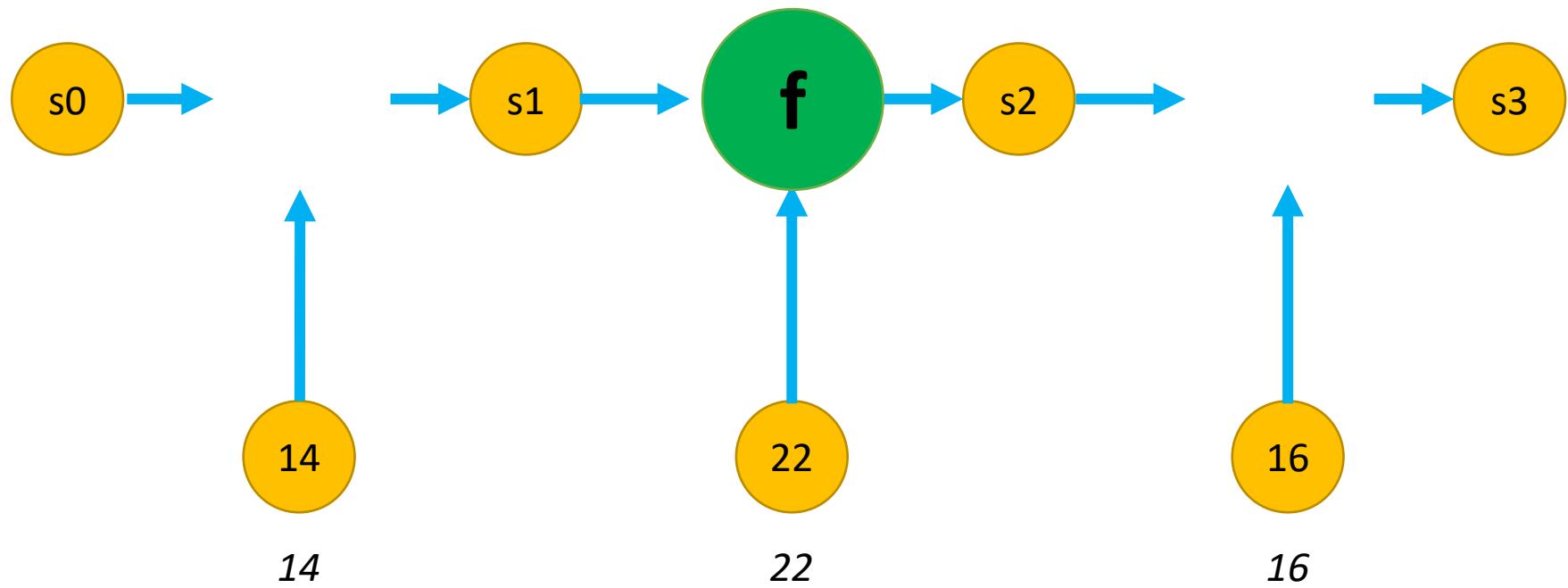
Recurrent layer



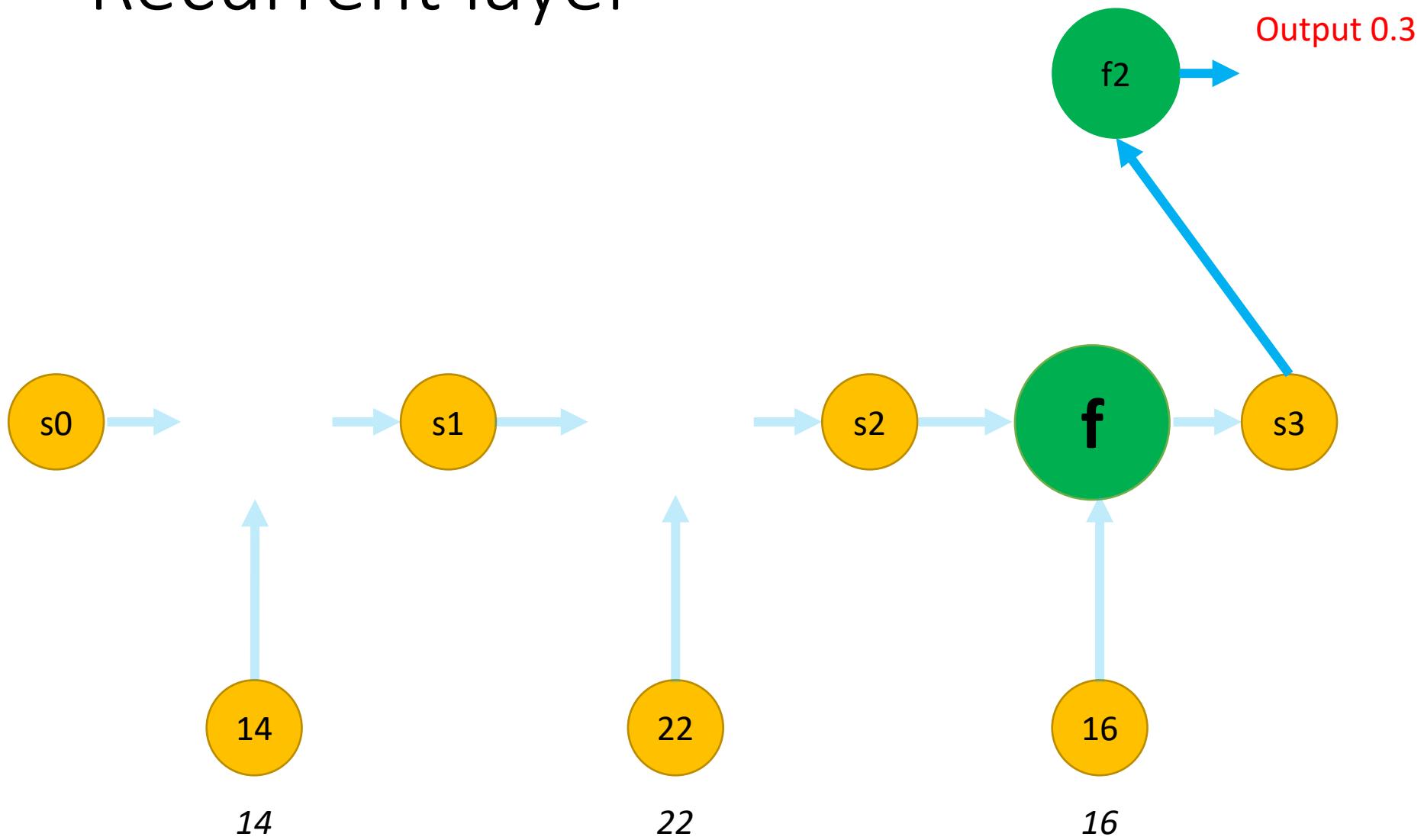
Recurrent layer



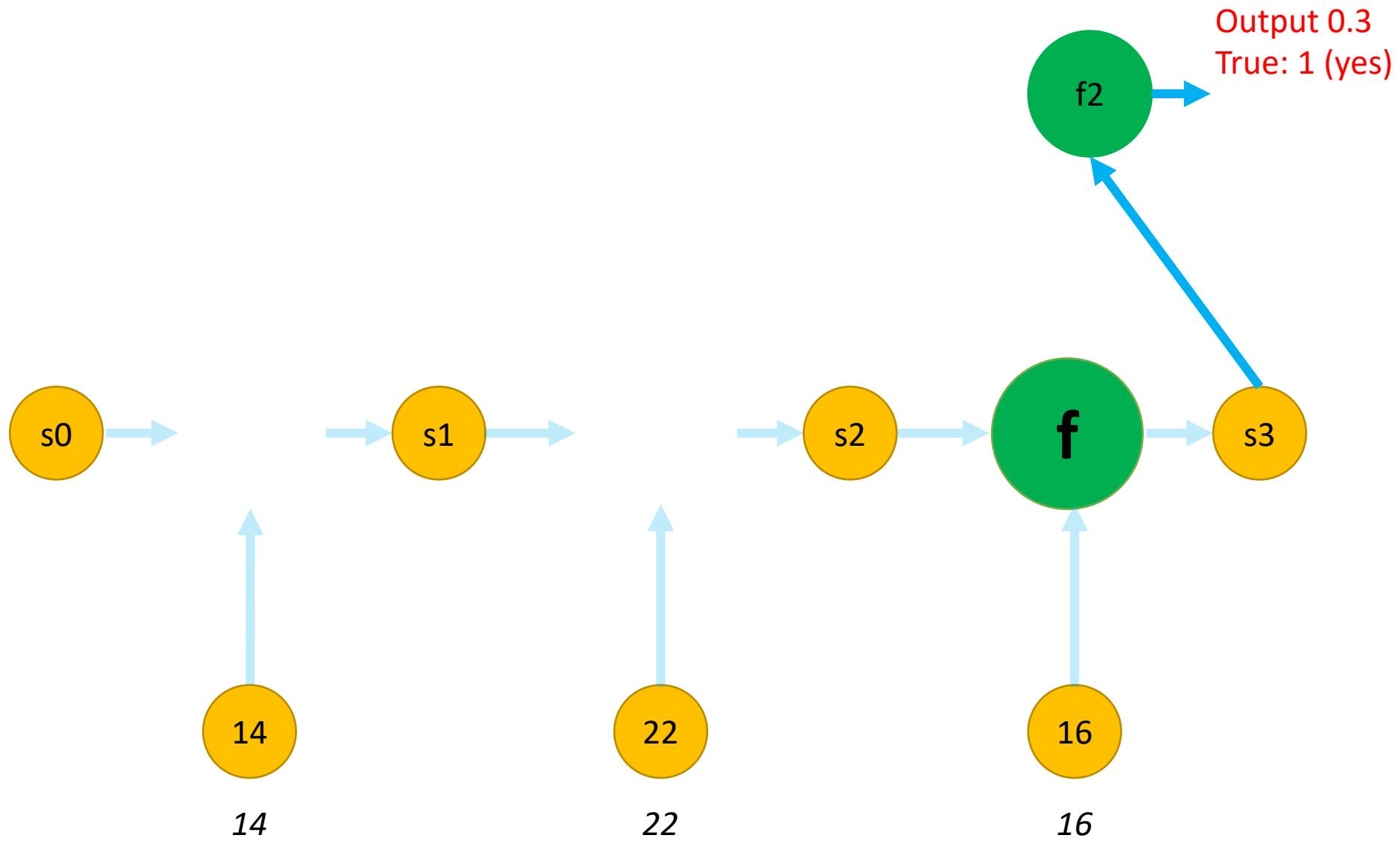
Recurrent layer



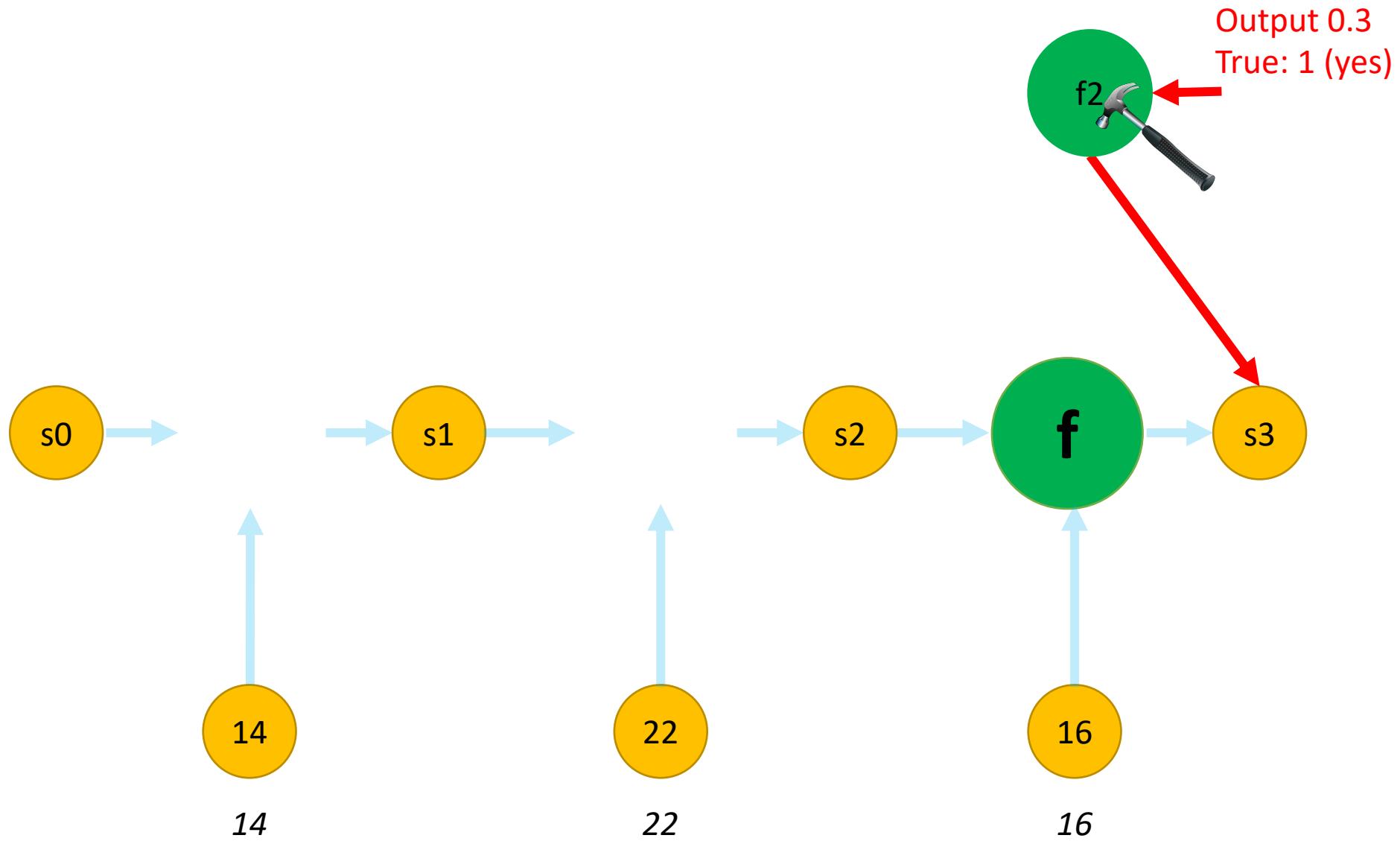
Recurrent layer



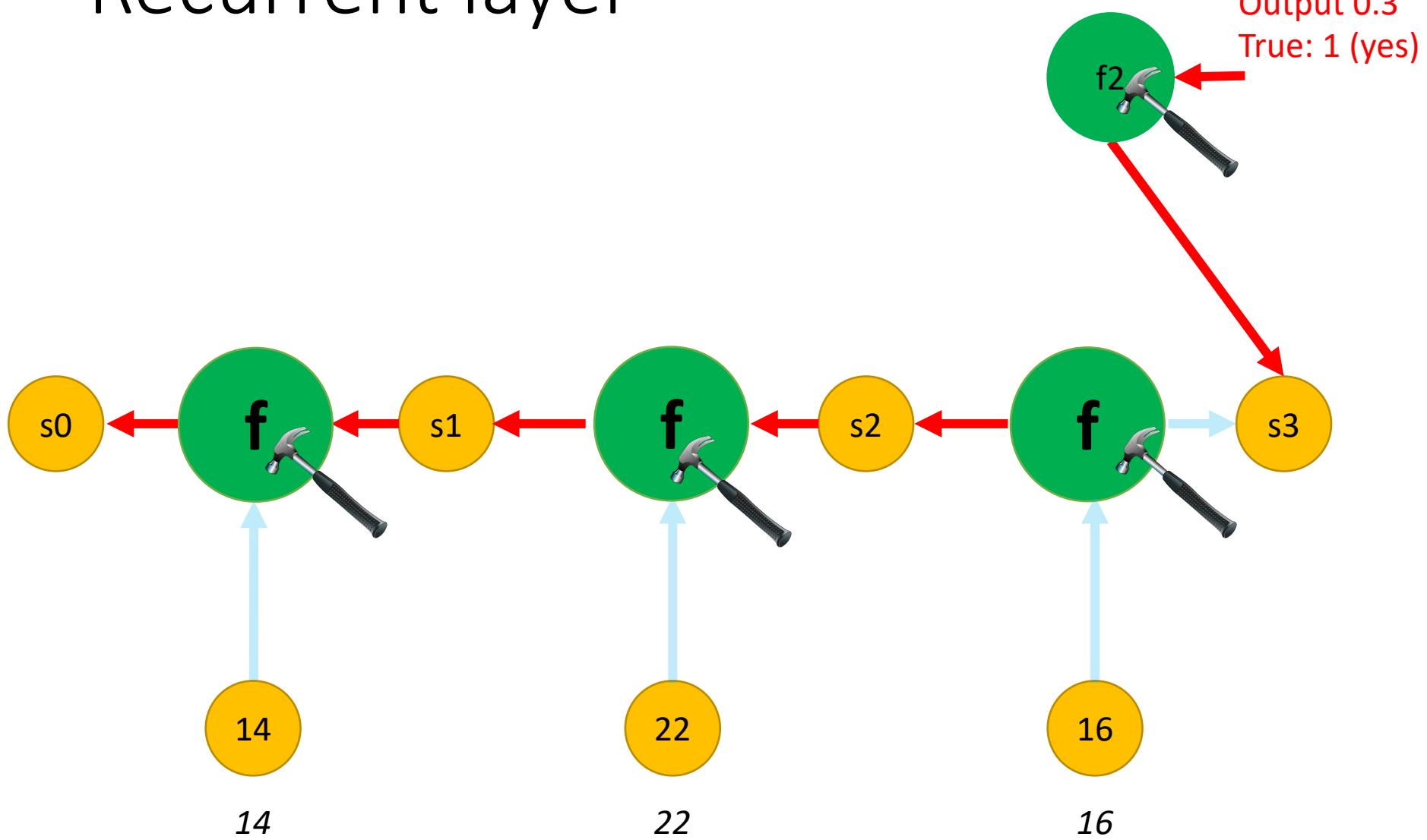
Recurrent layer



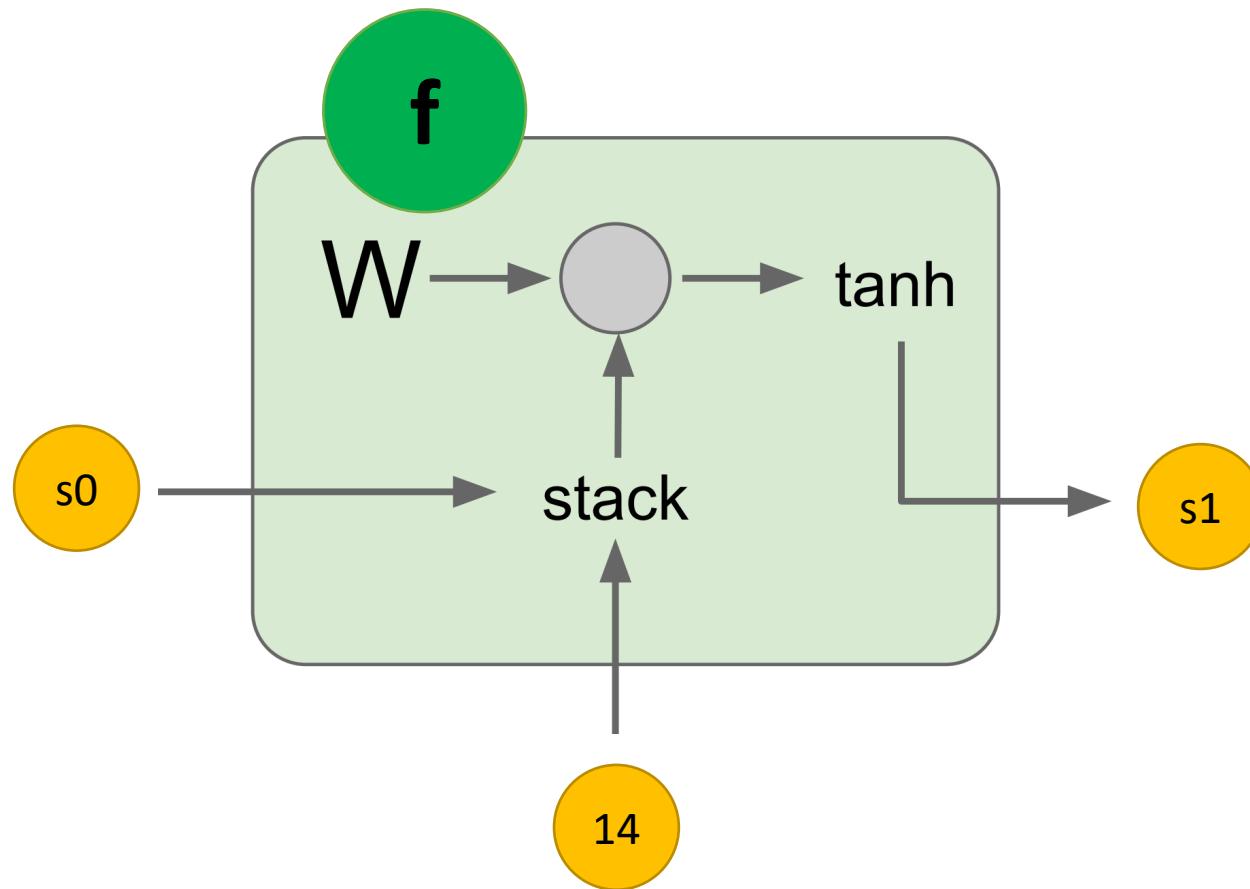
Recurrent layer



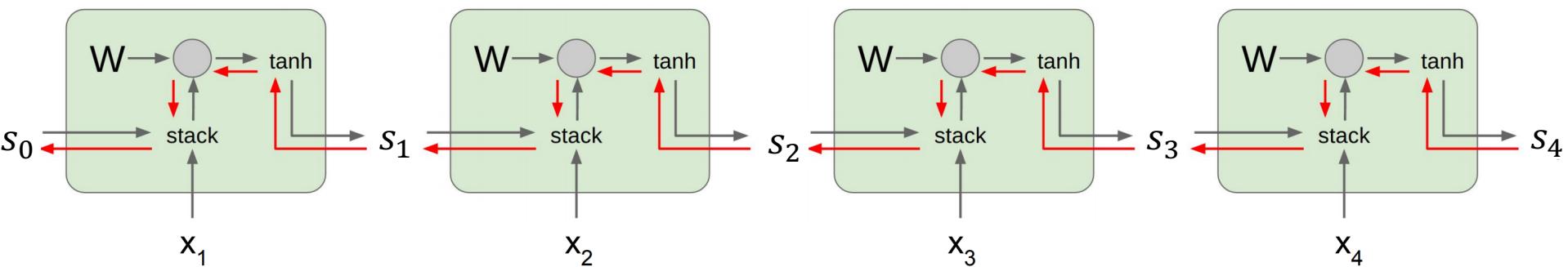
Recurrent layer



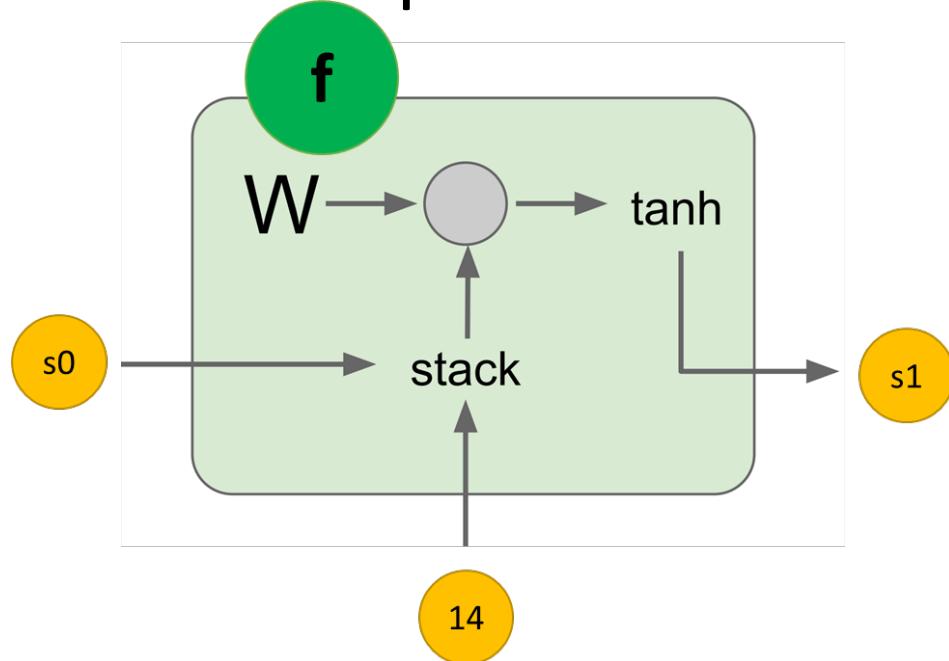
Cell implementation – Vanilla RNN



Cell implementation – Vanilla RNN

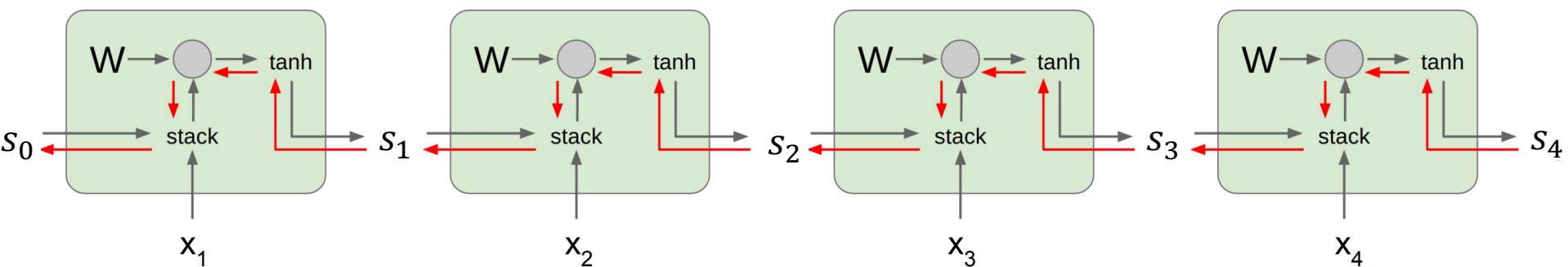


Cell implementation – Vanilla RNN



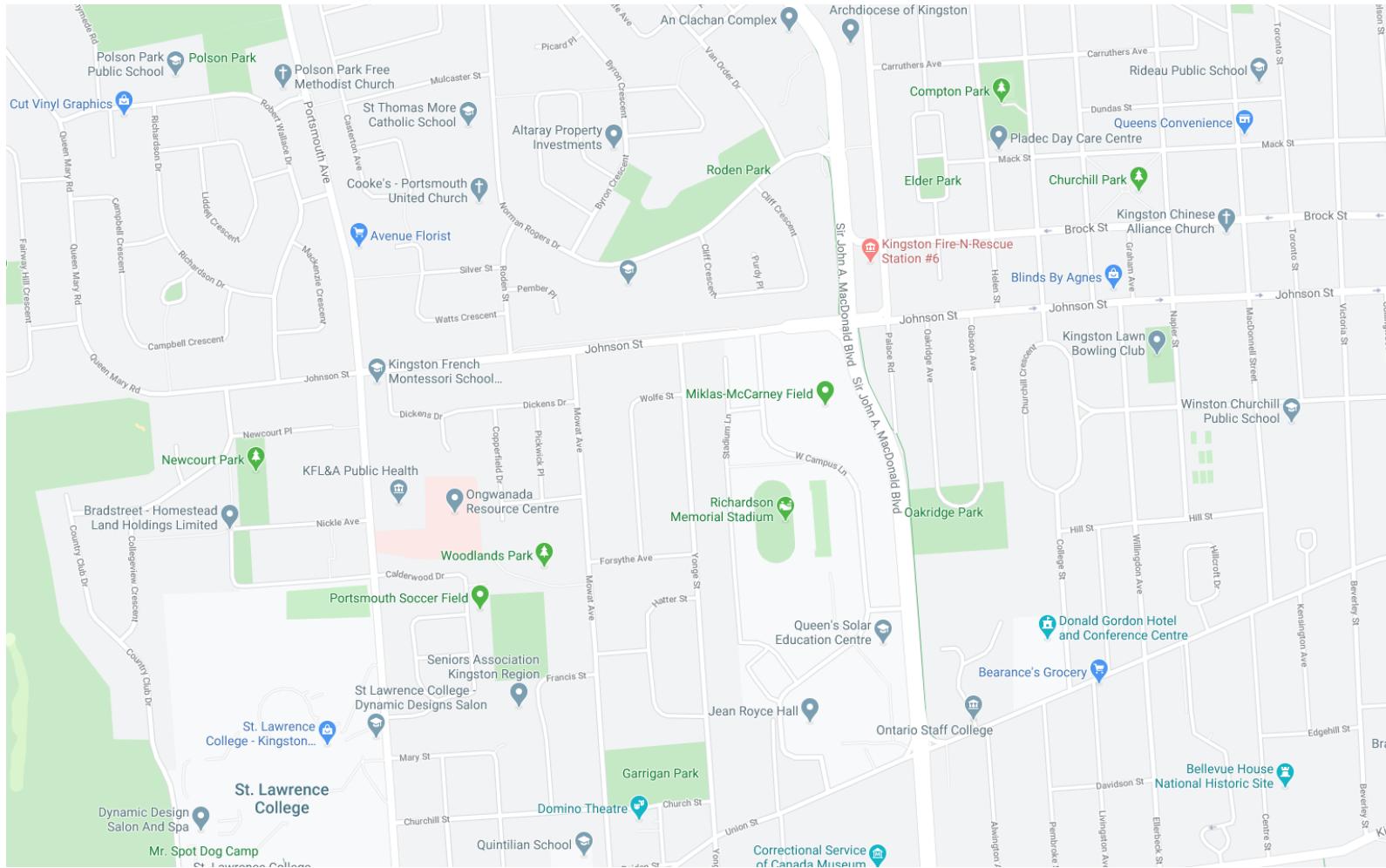
```
def call(self, inputs, state):
    """Most basic RNN: output = new_state = act(w * input + U * state + B)."""
    _check_rnn_cell_input_dtypes([inputs, state])
    gate_inputs = math_ops.matmul(
        array_ops.concat([inputs, state], 1), self._kernel)
    gate_inputs = nn_ops.bias_add(gate_inputs,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

Cell implementation – Vanilla RNN



Cell implementation – Vanilla RNN

- Problem: like driving in a driveway/roadway

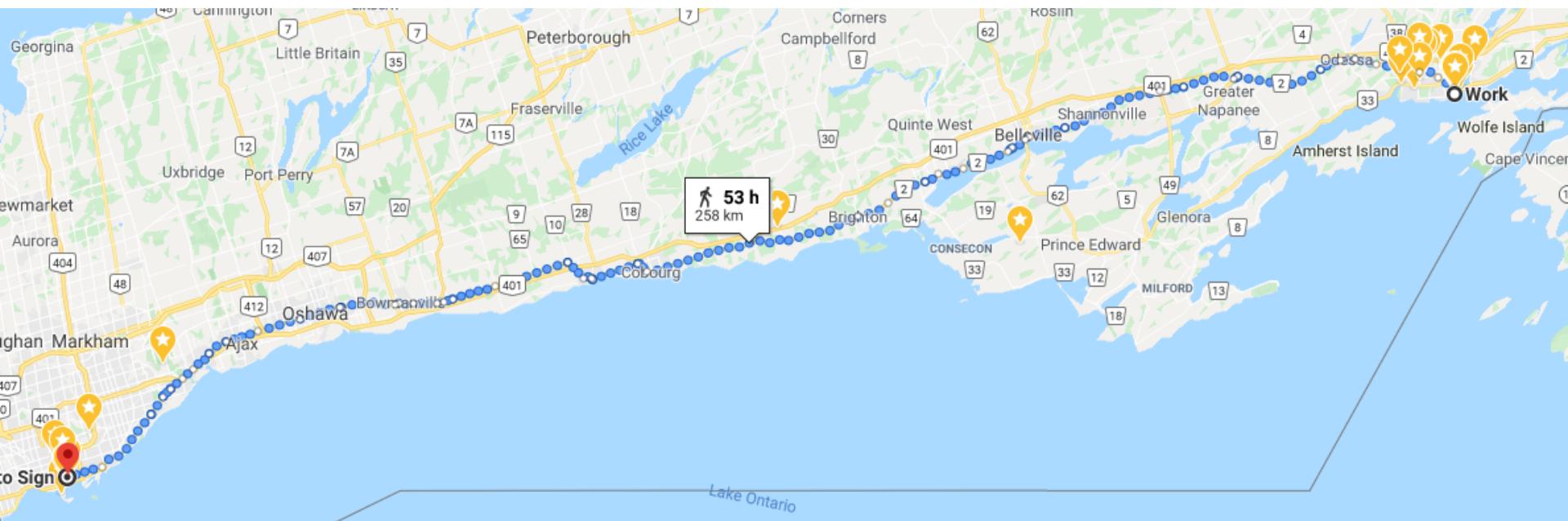


Cell implementation – Vanilla RNN

- 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)

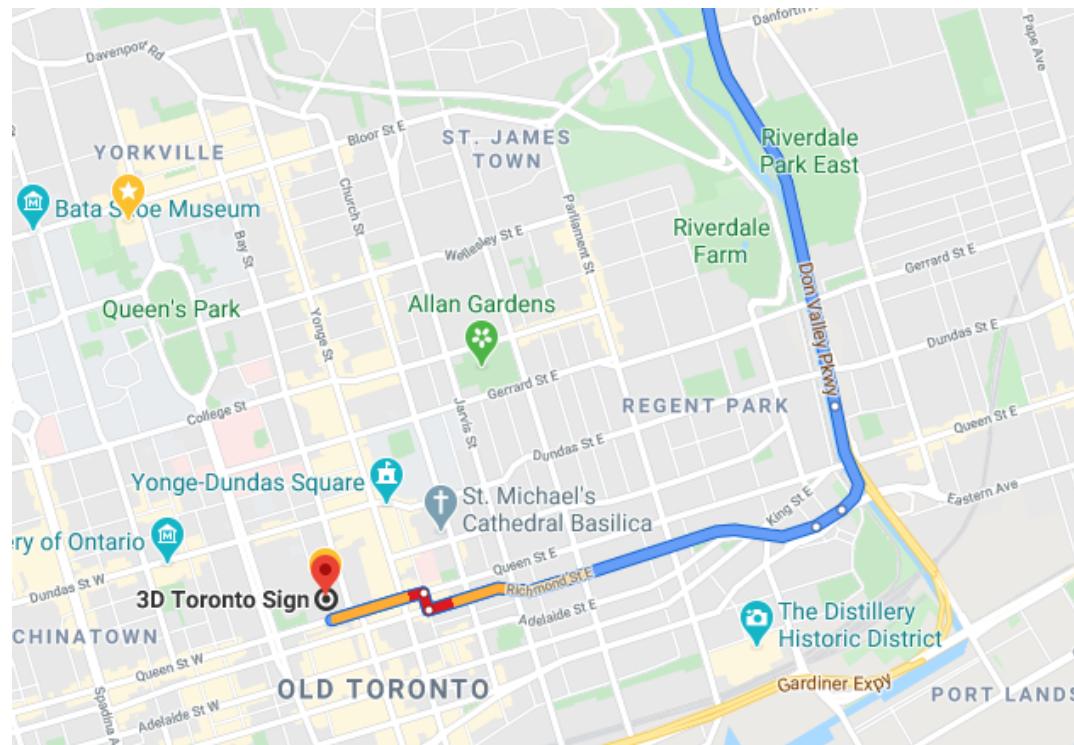
Cell implementation – Vanilla RNN

- 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

h_{t-1}



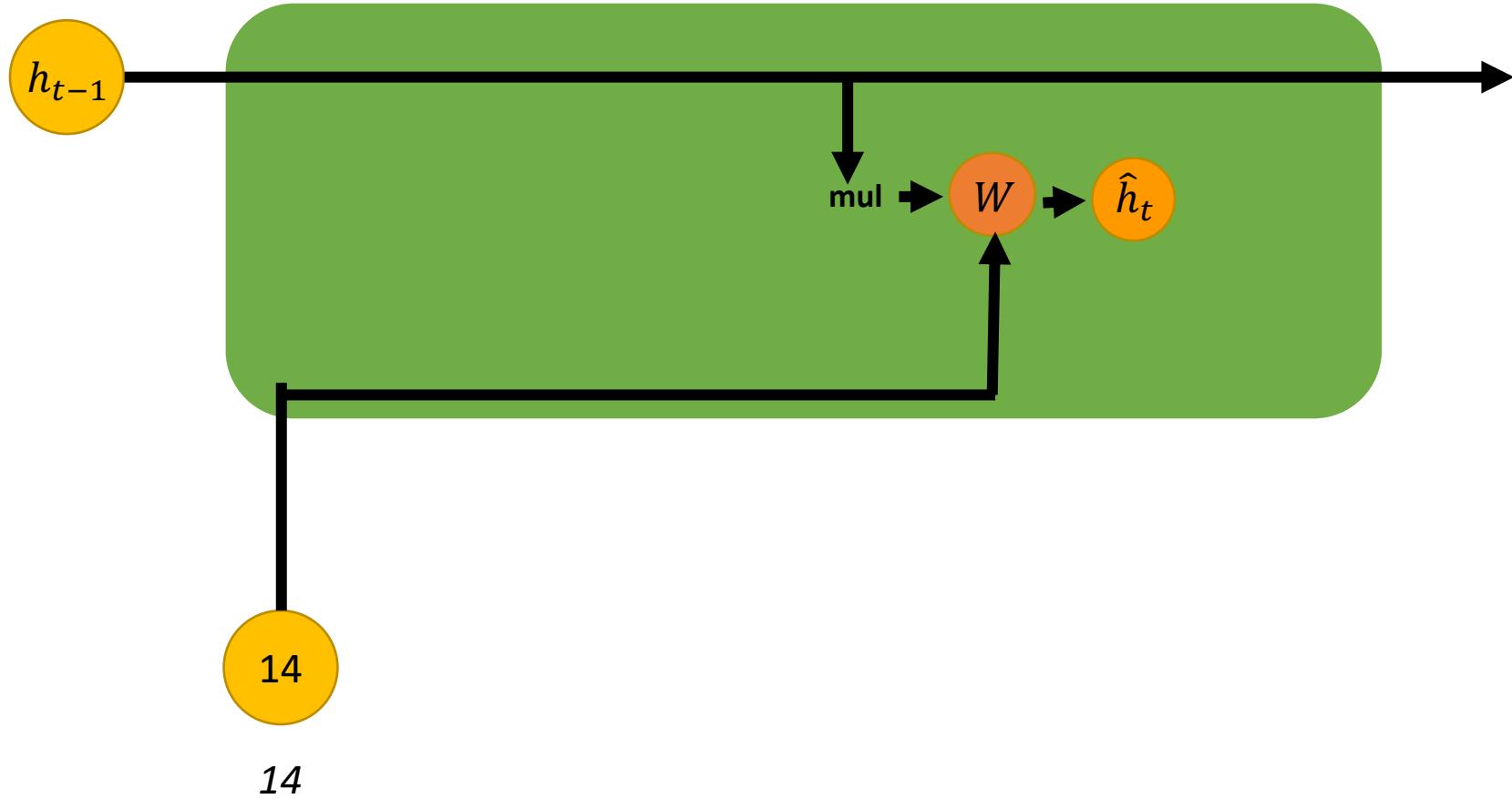
14

14

Cell implementation – GRU (high way)



Cell implementation – GRU (roadway)

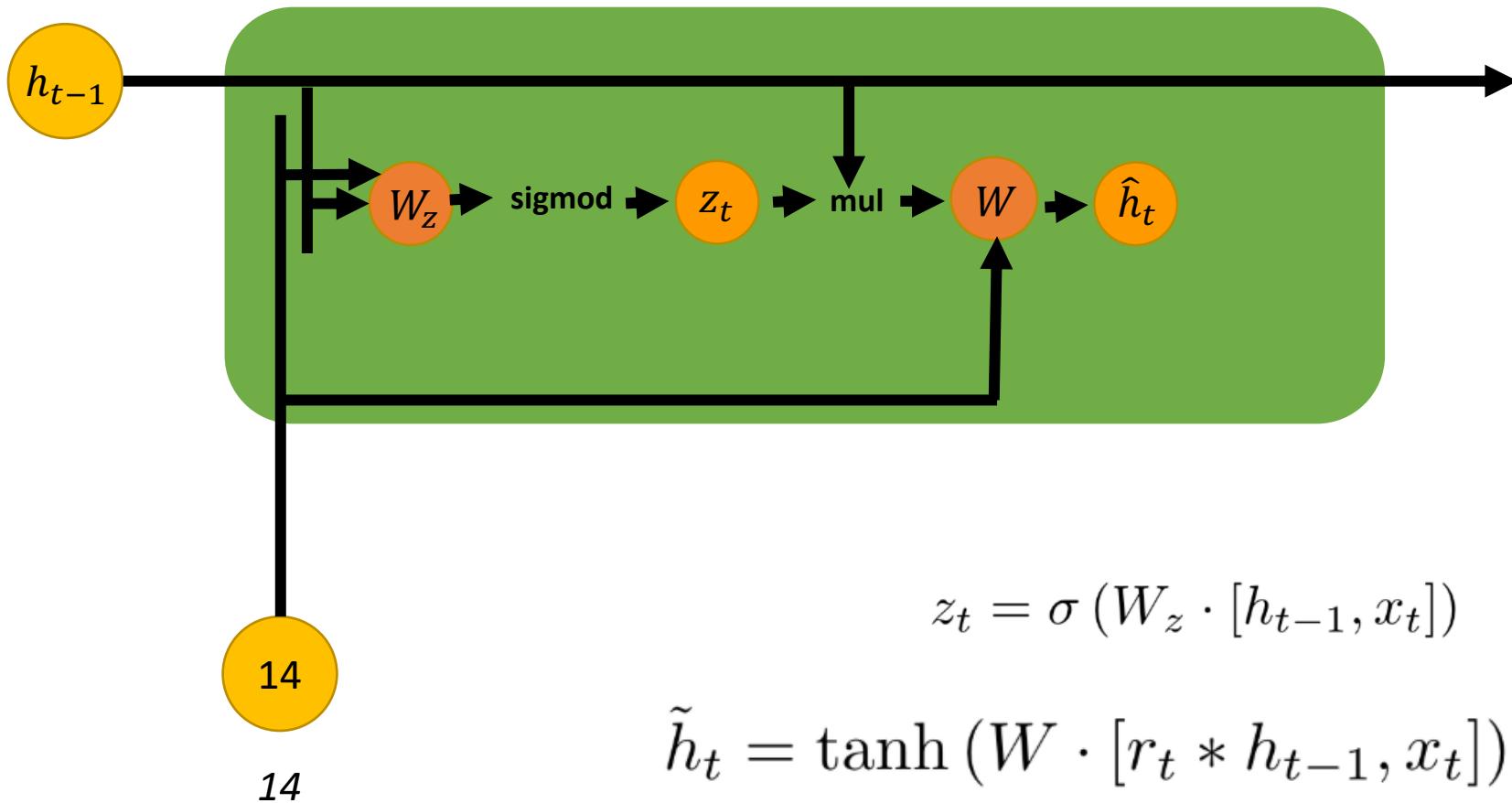


14

Cell implementation – GRU (roadway - gated)

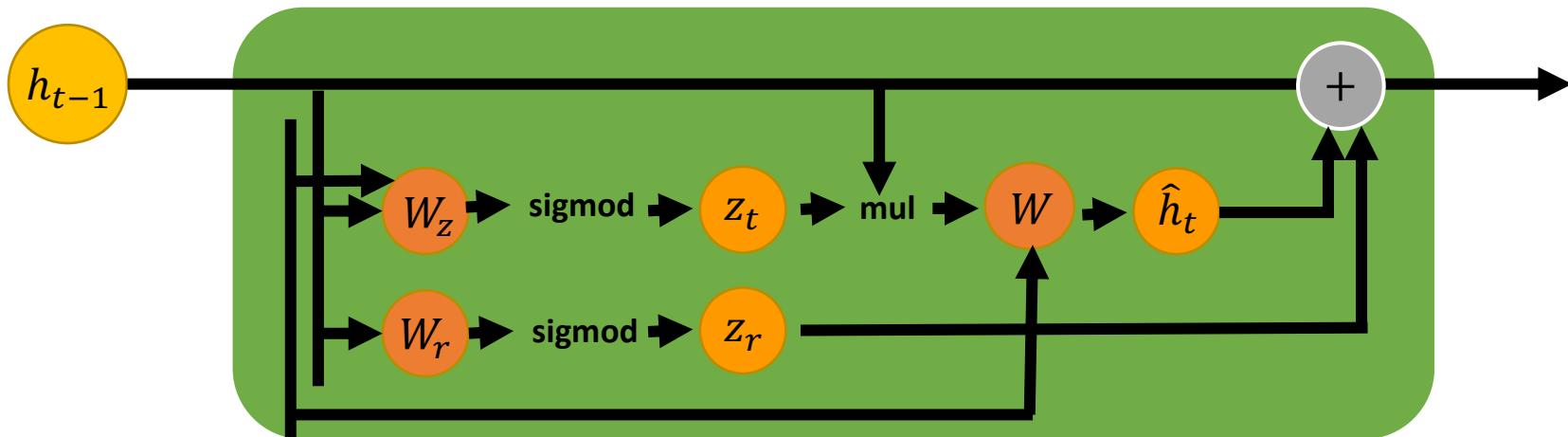
Not all the `passengers` need to go down the highway

W_z create a gate z_t [0, 1] to determine who is going to go down the highway



Cell implementation – GRU (merged)

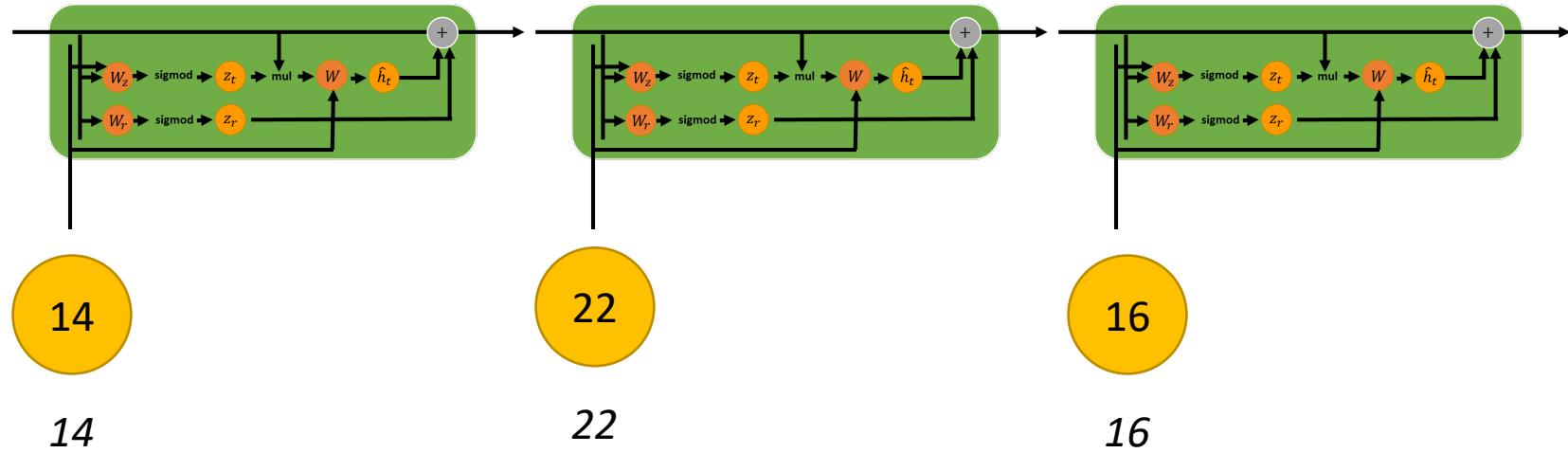
We also need to determine how many `passengers` need to pickup back to the highway. W_r create another gate z_r determine how roadway and highway are merged again



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

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

Cell implementation – GRU (merged)



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)

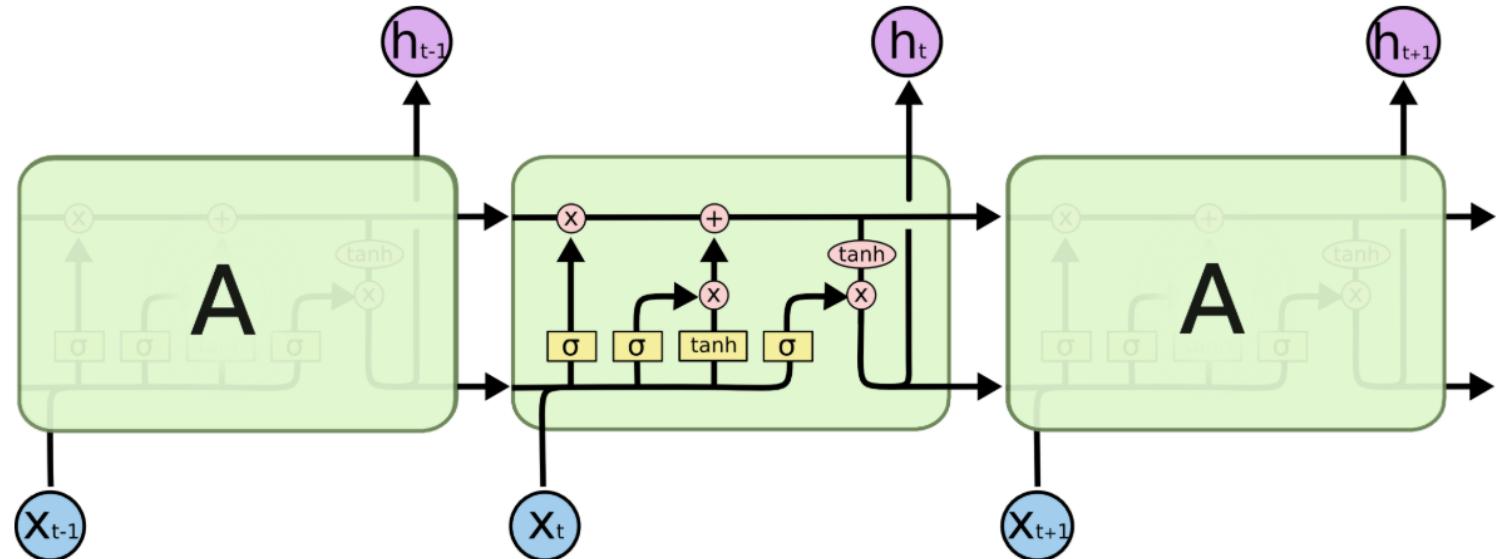
    r_state = r * state

    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-term memory (always going roadway)
- More gates
 - higher computational overhead
- Similar performance



Attention Mechanism

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

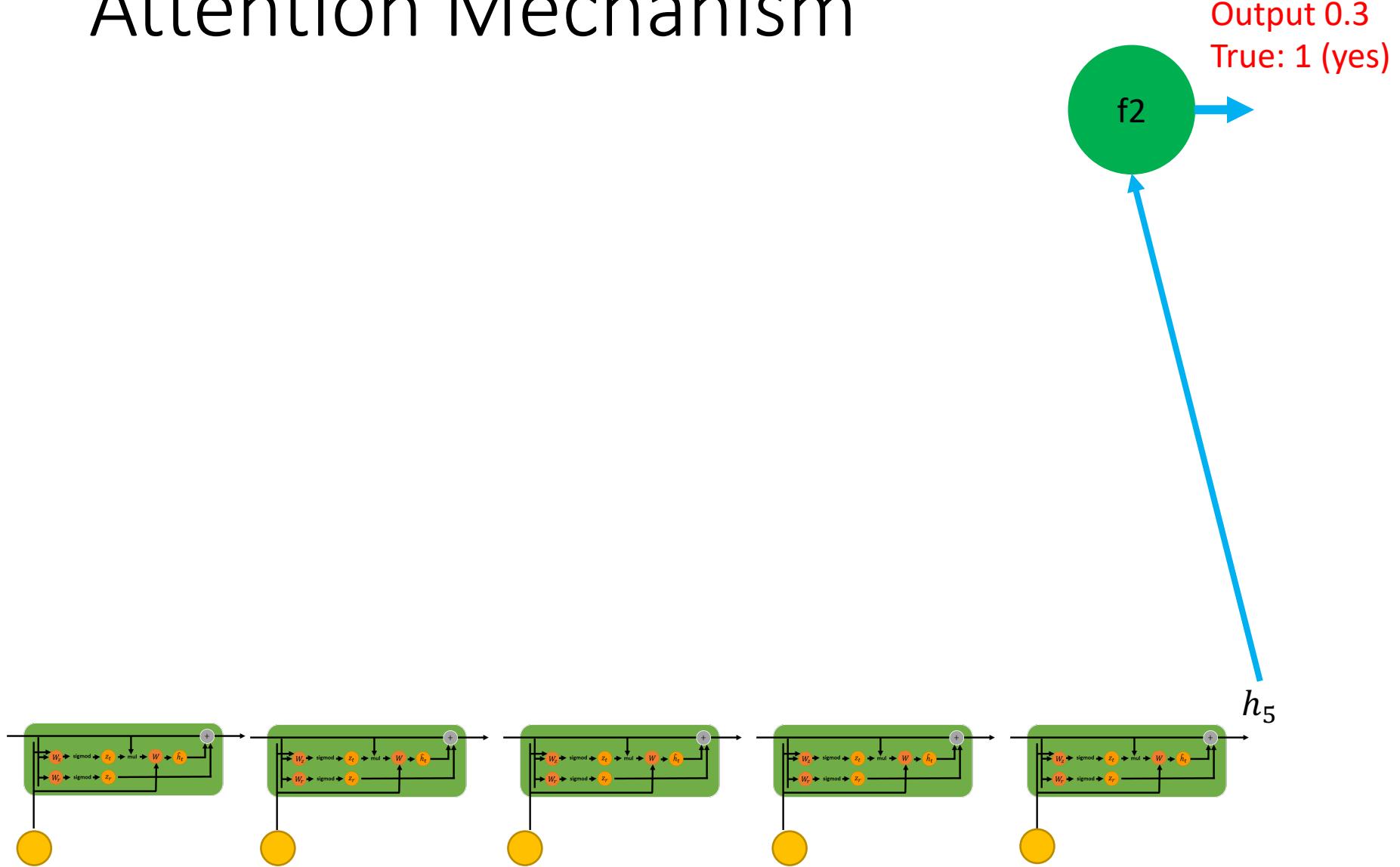


Attention Mechanism

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



Attention Mechanism



Attention Mechanism

