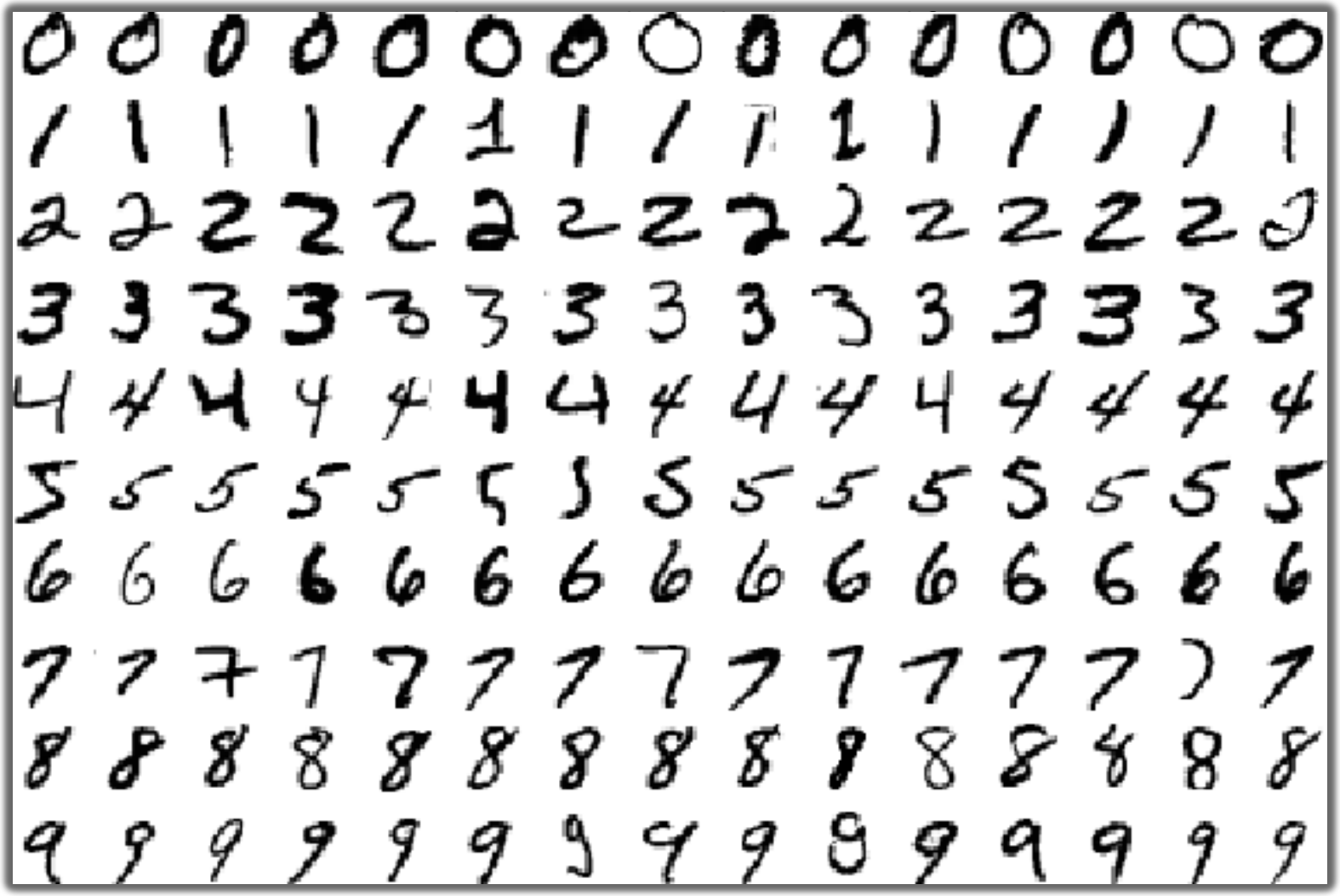
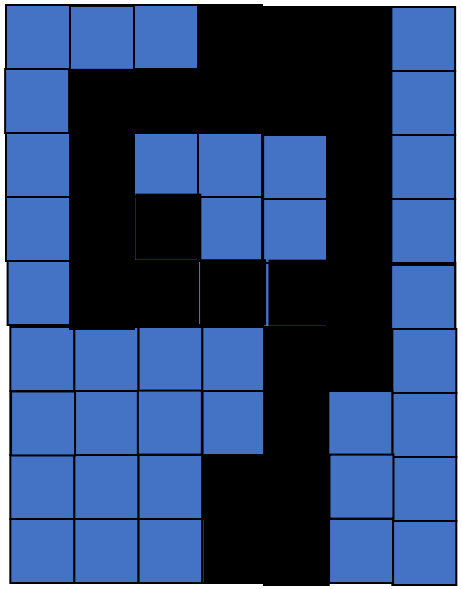
DATA MINING

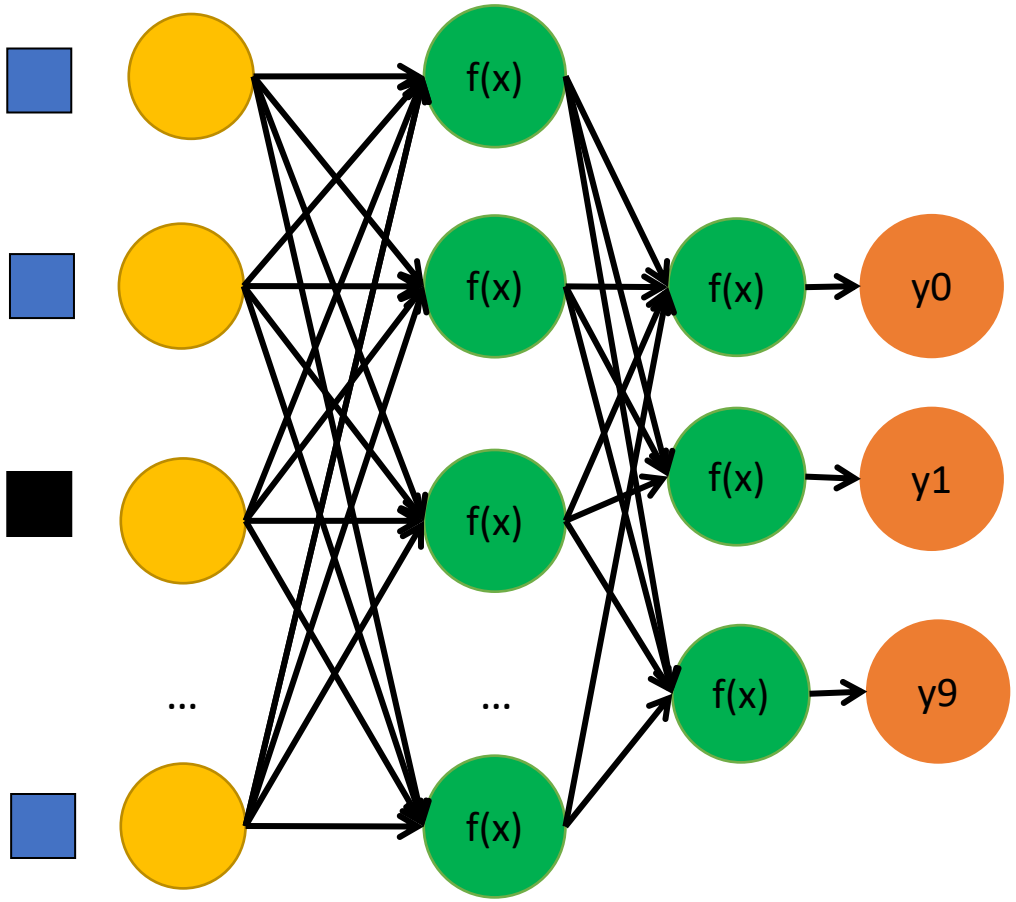
/CISC 873 - Steven Ding
/Week #6
Spatial

Neural Network cont'd

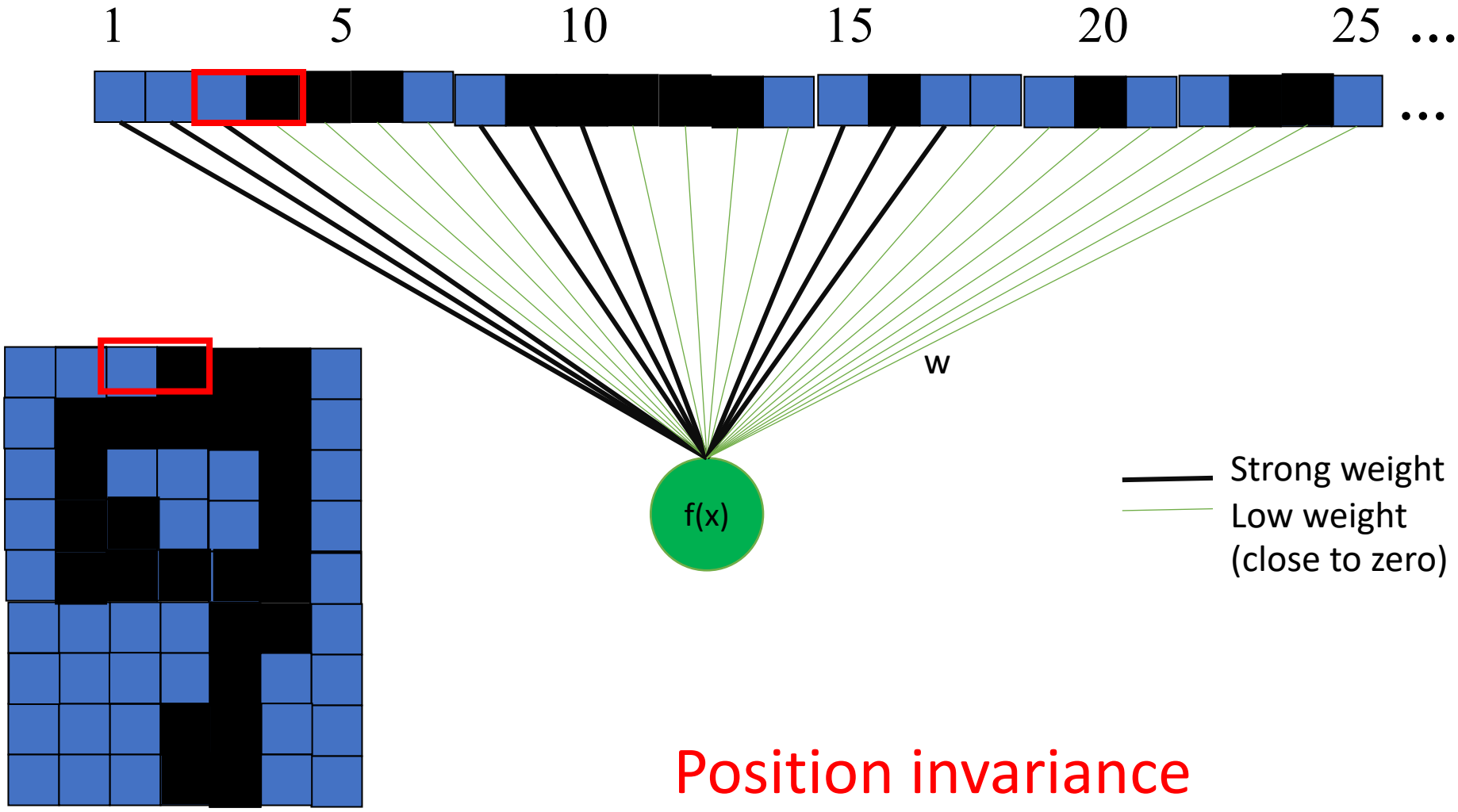


Neural Network cont'd



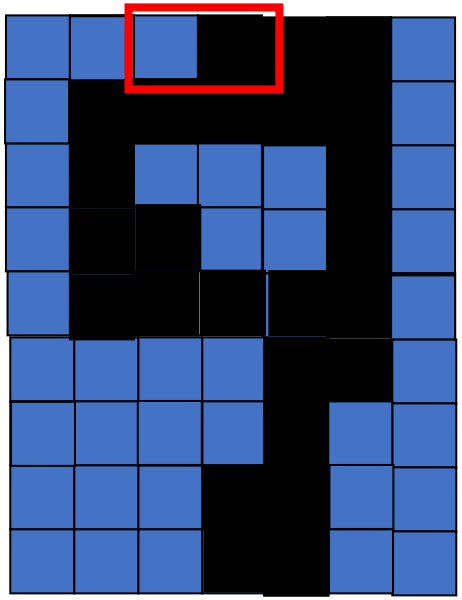


Neural Network cont'd

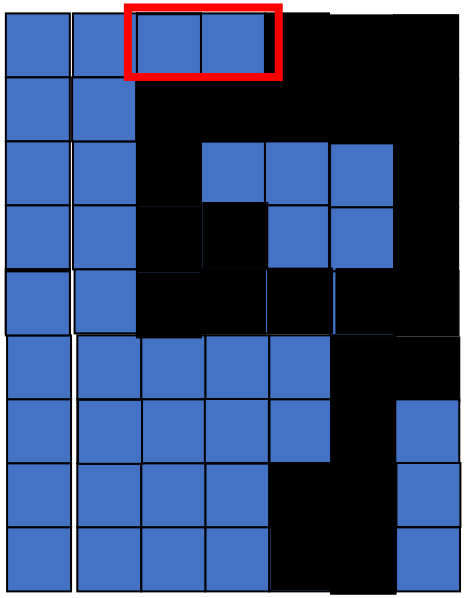


Position invariance

Neural Network cont'd



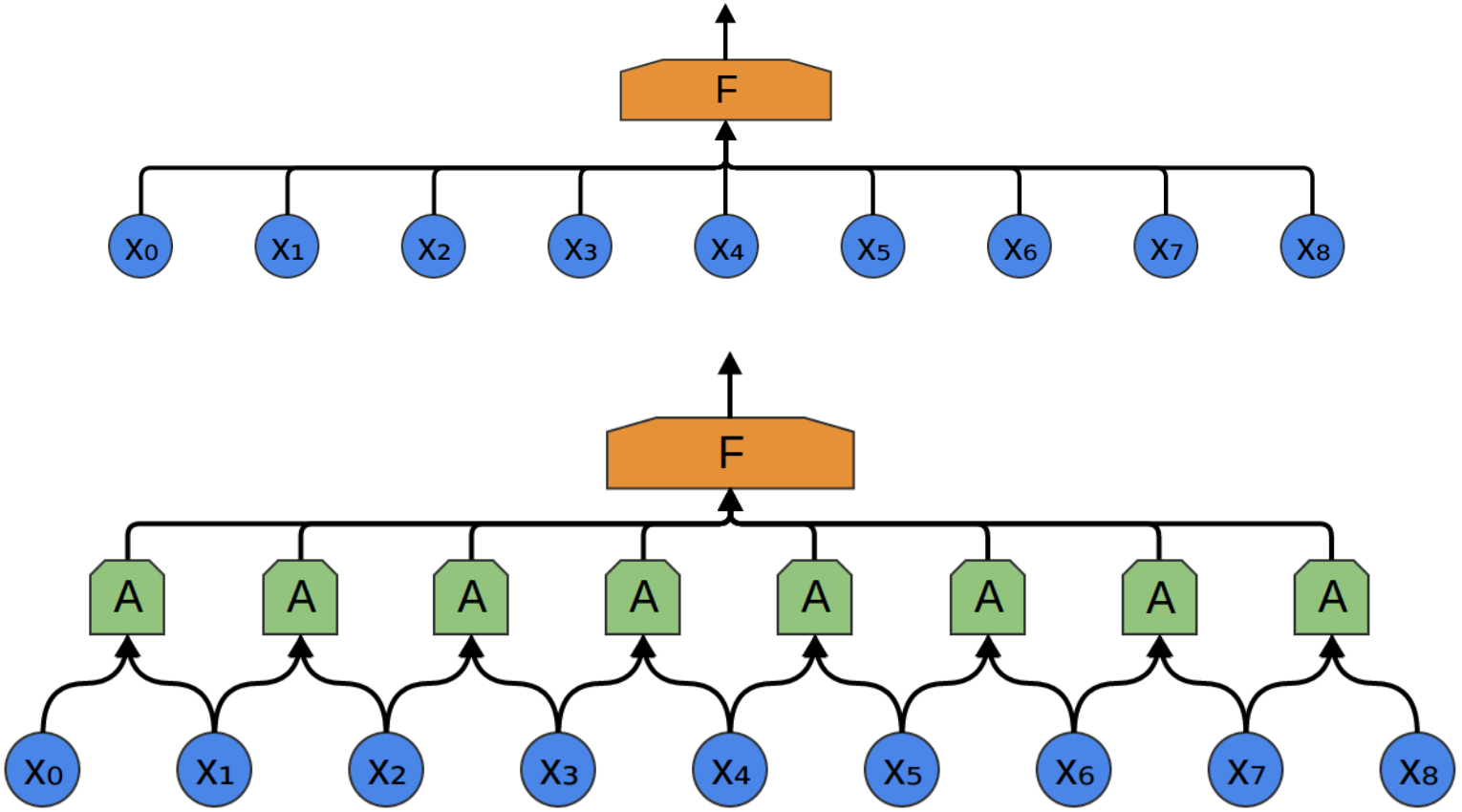
A training sample.



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

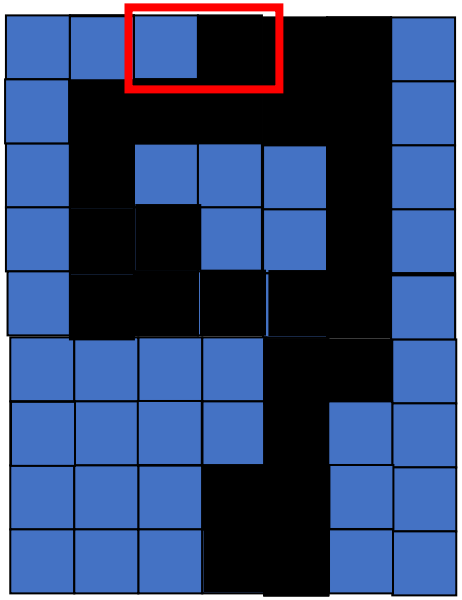
Position invariance

Fully Connected vs Conv1D

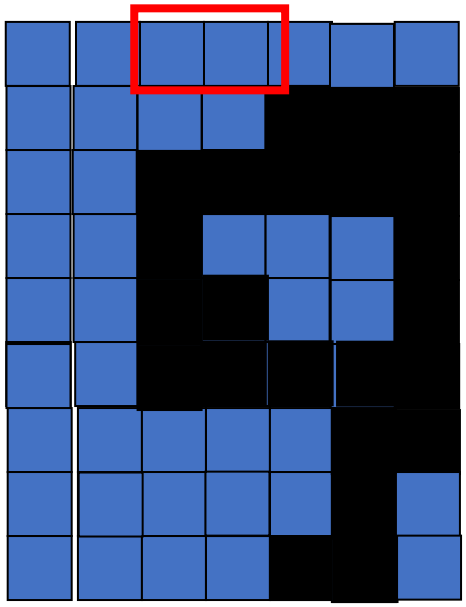


http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

Neural Network cont'd



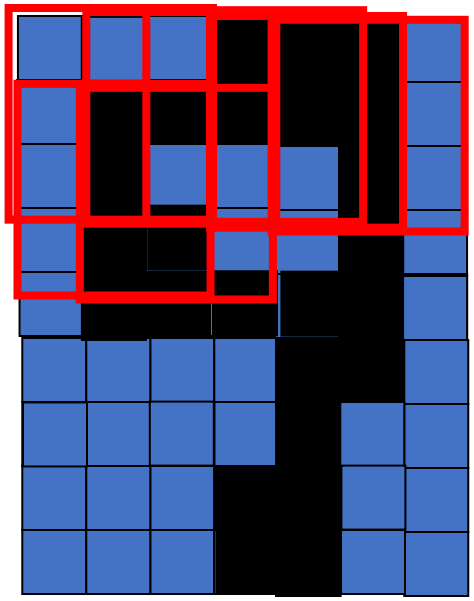
A training sample.



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

Position invariance

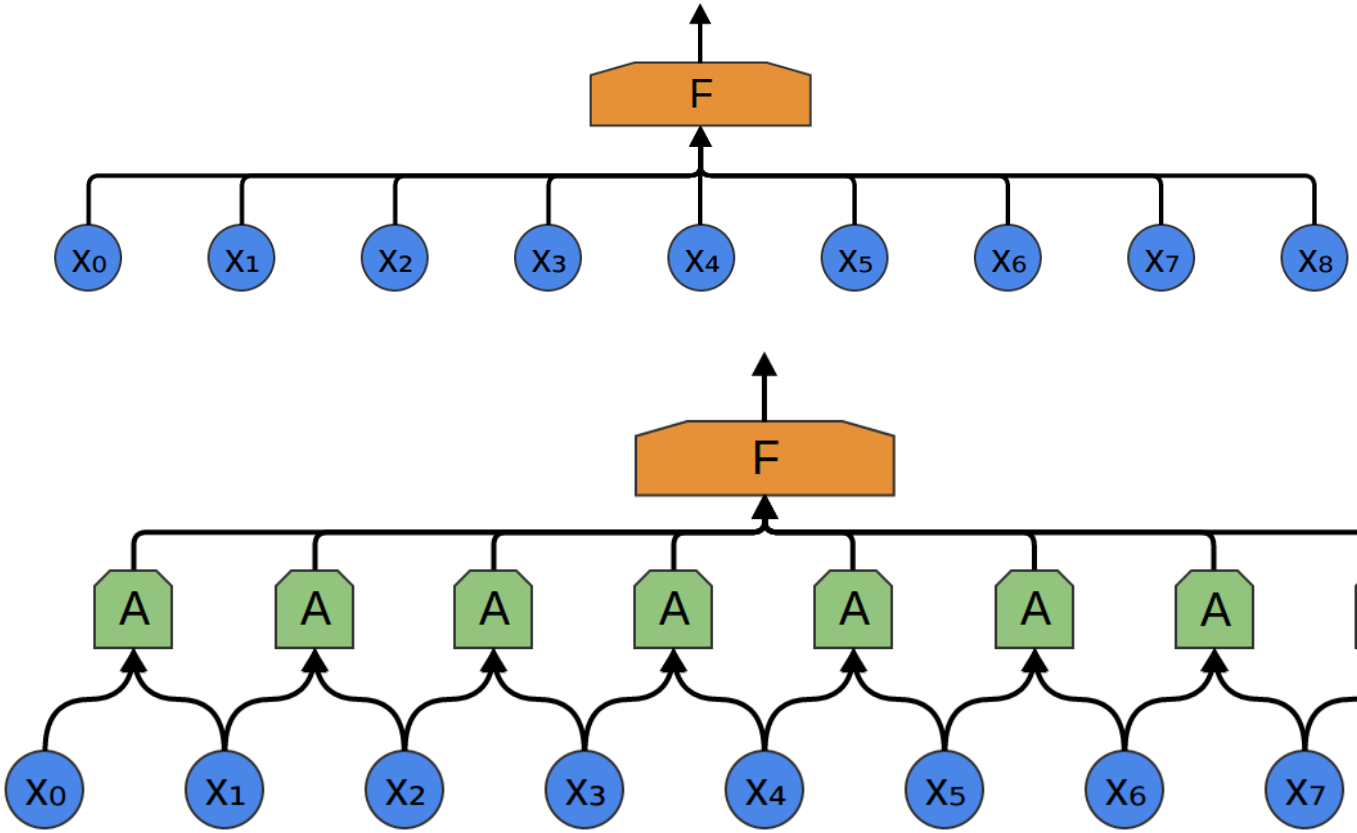
Conv2D

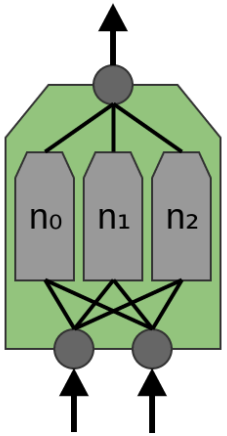


A training sample.

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

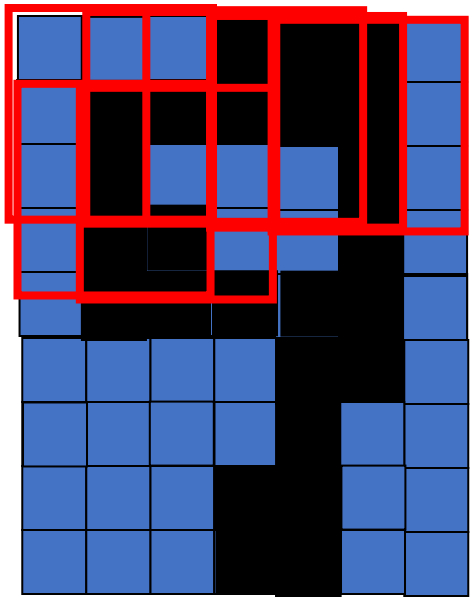
Fully Connected vs Conv1D





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

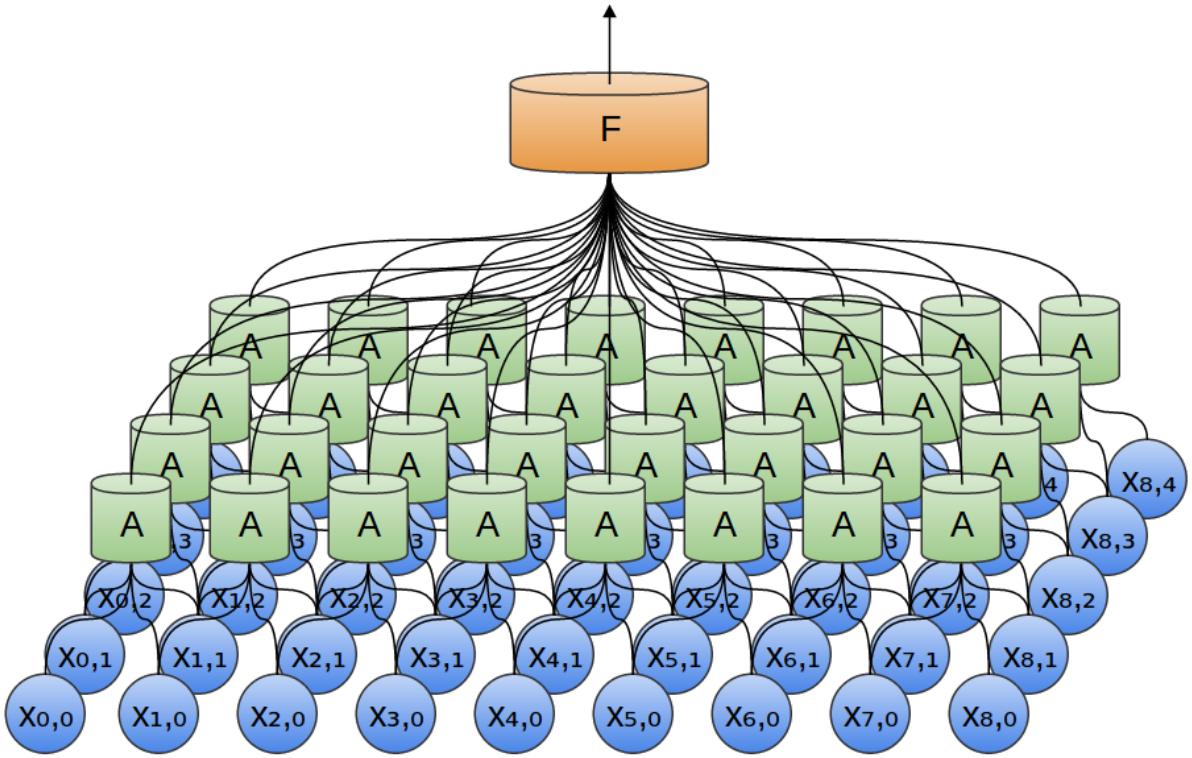
Convolutional Neural Network



A training sample.

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

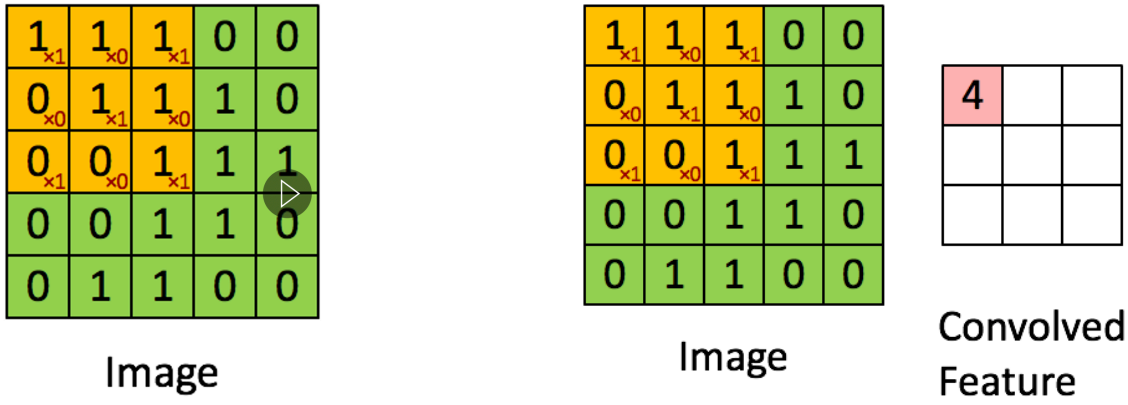
Conv2D



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

Conv2D – Single Filter

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



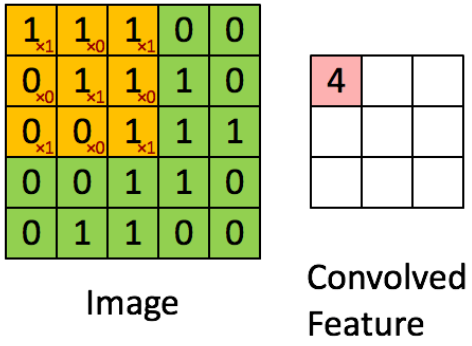
http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
http://deeplearning.stanford.edu/tutorial/supervised/Pooling/

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

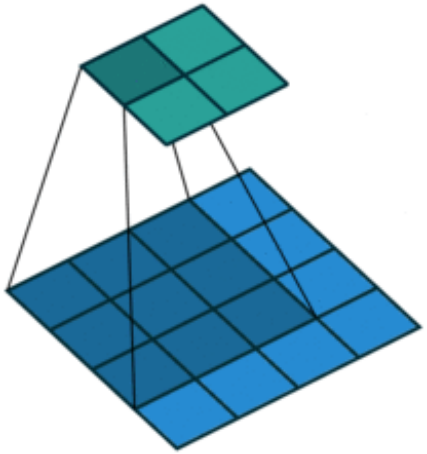
- Input:
  - [None, 5, 5, 1]
- Output:
  - [None, 3, 3, 1]

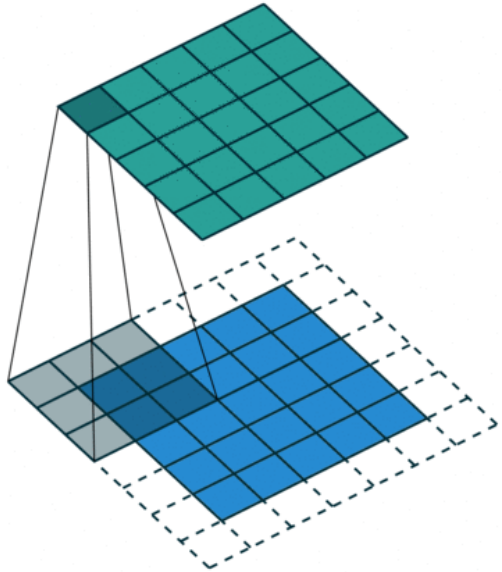


Padding

```
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]



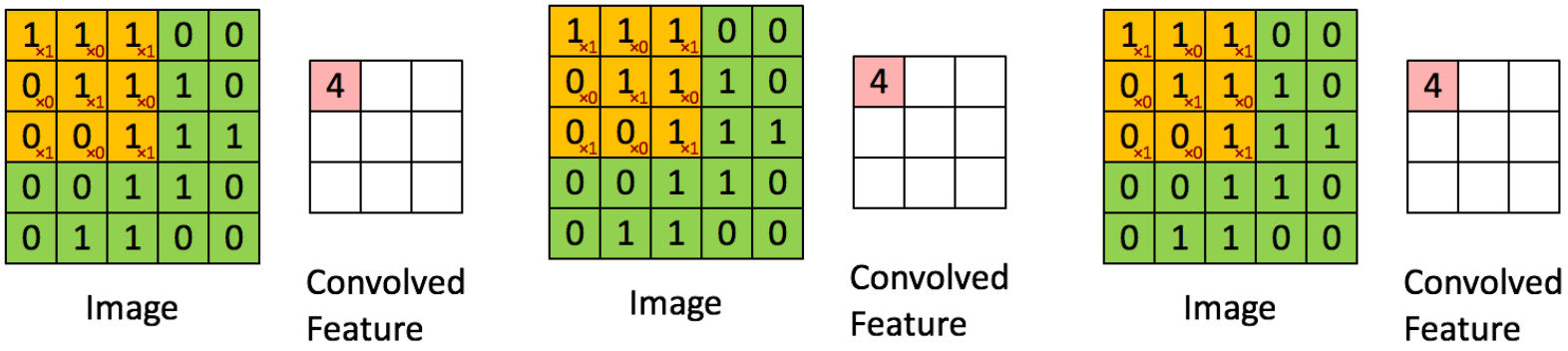


https://github.com/vdumoulin/conv_arithmetic

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
)
```

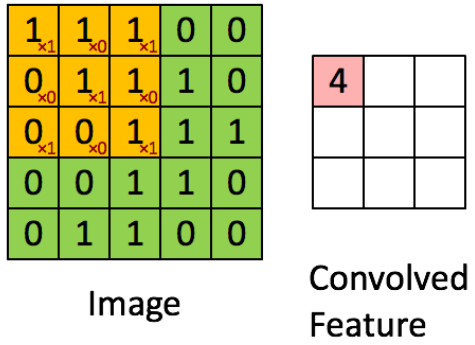
- filters
  - Number of filters to be applied



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
)
```

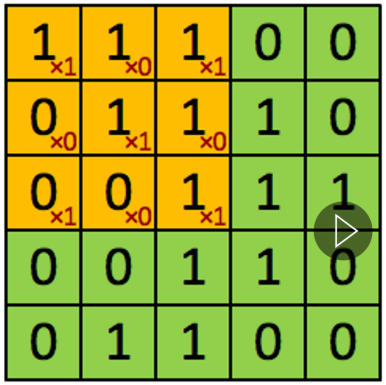
- filters
  - Number of filters to be applied
  - E.g. 6
- Input:
  - [None, 5, 5, 1]
- Output:
  - [None, 3, 3, 6]



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]

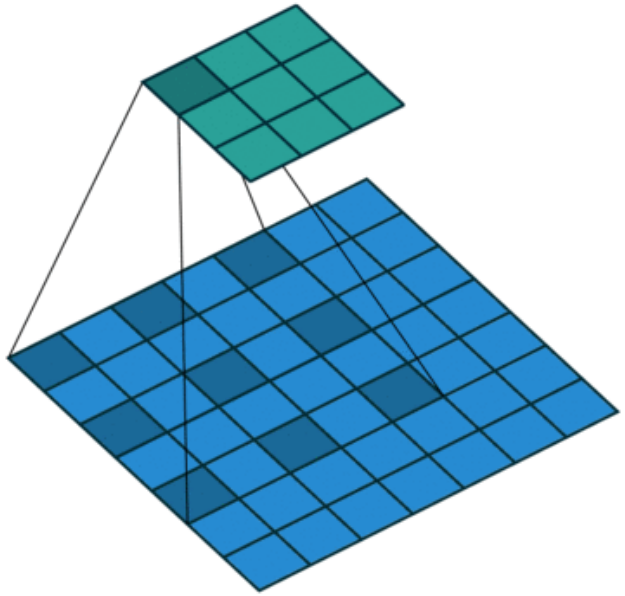


Image

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
)
```

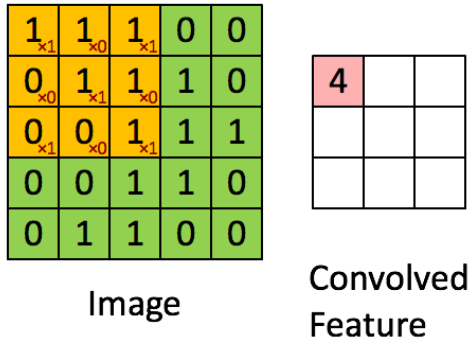
- Expanding the scanner



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
)
```

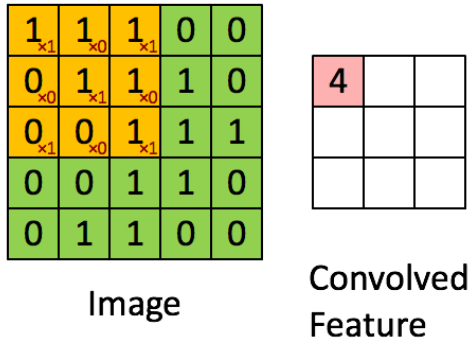
- filters
  - Number of filters to be applied
  - E.g. 6
- Input:
  - [None, 5, 5, 1]
- Output:
  - [None, 3, 3, 6]



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]



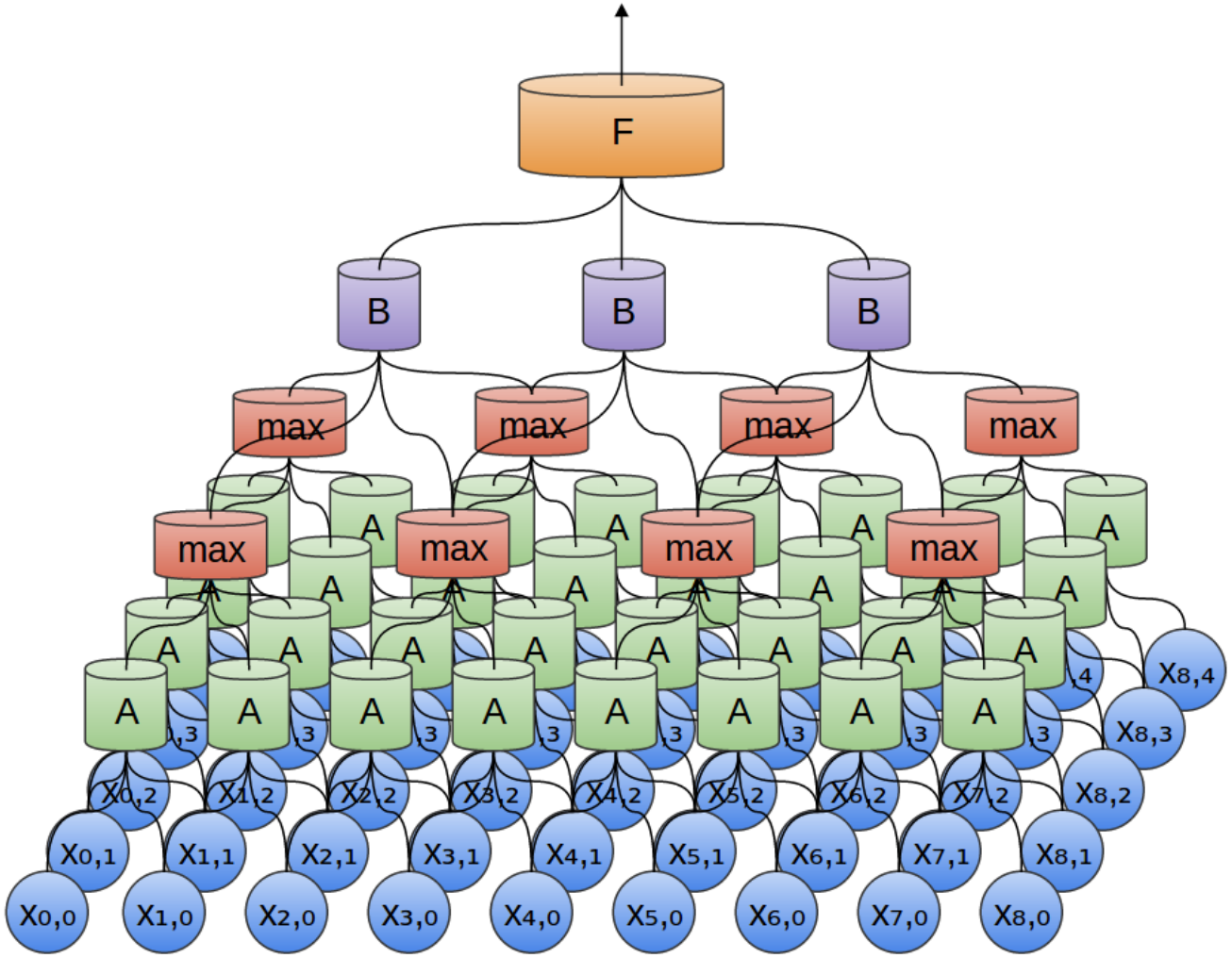
Convolutional Neural Network

- Features:
  - Position Invariance
  - Receptive field of different granularity

- Try it out:

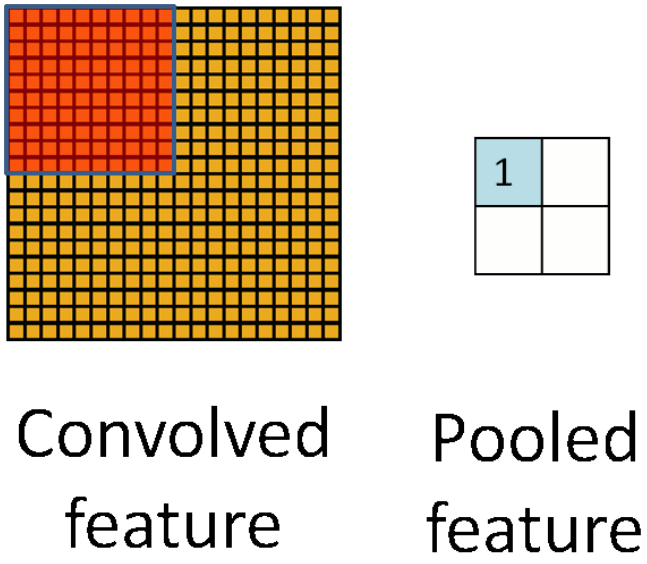
```
4 import tensorflow as tf
5 x = tf.random.normal(shape=(4, 5, 5, 1))
6
7 y = tf.keras.layers.Conv2D(2, 3, activation='relu', strides=2, padding='valid')(x)
8 print(y.shape)
```

Pooling Layer



Convolutional Neural Network

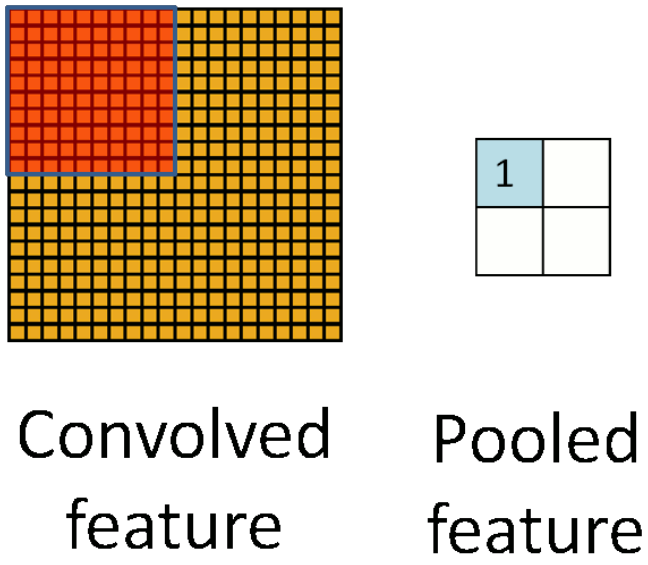
- Pooling
  - Position Invariance Feature Extraction
  - Reduce the window
  - Average, Max, etc.



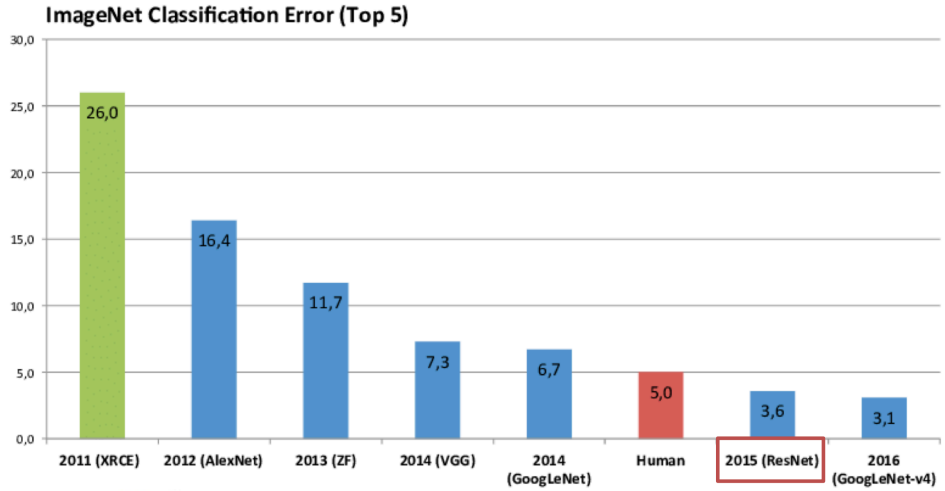
http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
http://deeplearning.stanford.edu/tutorial/supervised/Pooling/

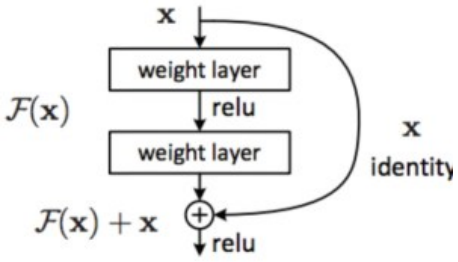
Pooling Layer

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

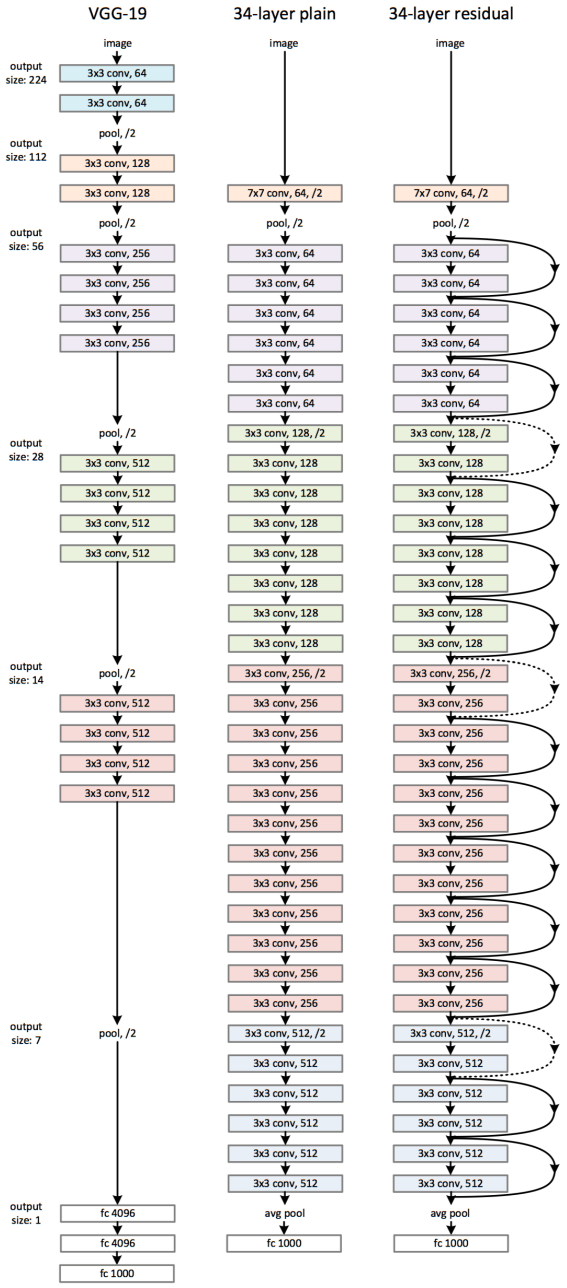


Application





$H(x) = x + F(x)$



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 This film was just brilliant"

  -> {"this": 2, "film":1, "was": 1, "just", 1, "brilliant":1}

  -> [1,1,1,1,2]

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” -> sentiment 0
- doc2: "The man bit the dog” -> sentiment 1

	the-dog	dog-bit	bit-the	the-man	man-bit	sentiment
doc1	1	1	1	1	0	0
doc2	1	0	1	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, …]

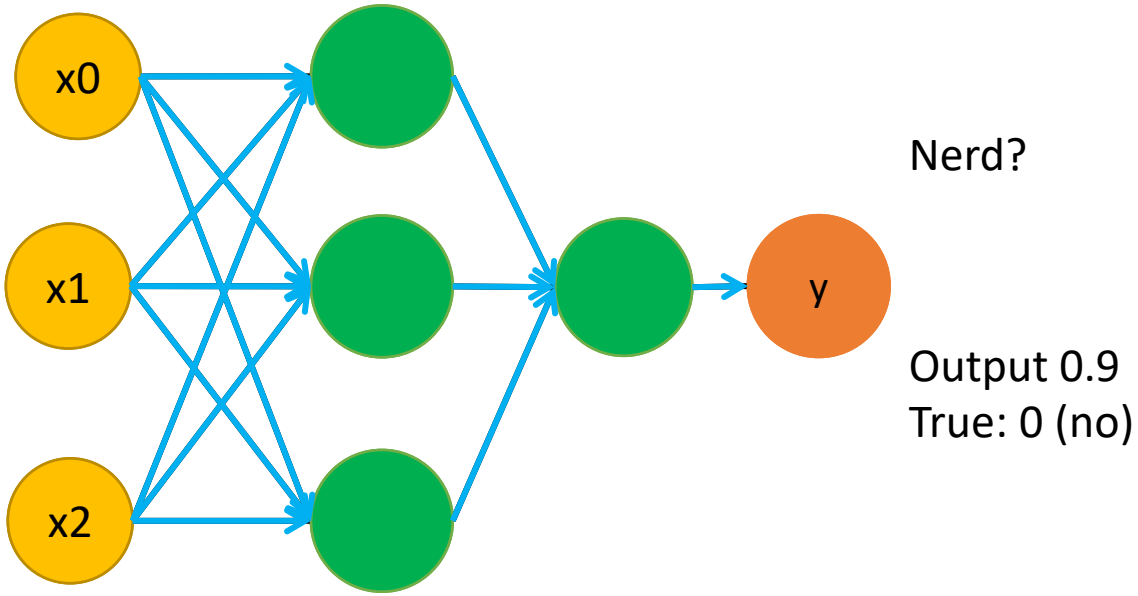
Term weighting

- 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
- …

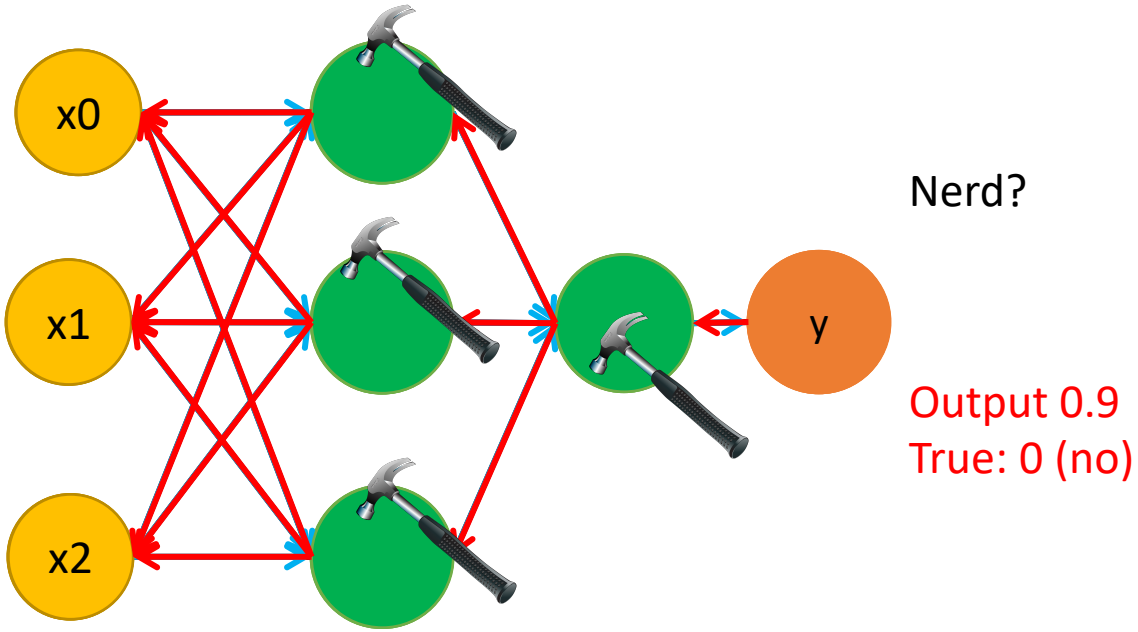
Recurrent Neural Network

- 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.

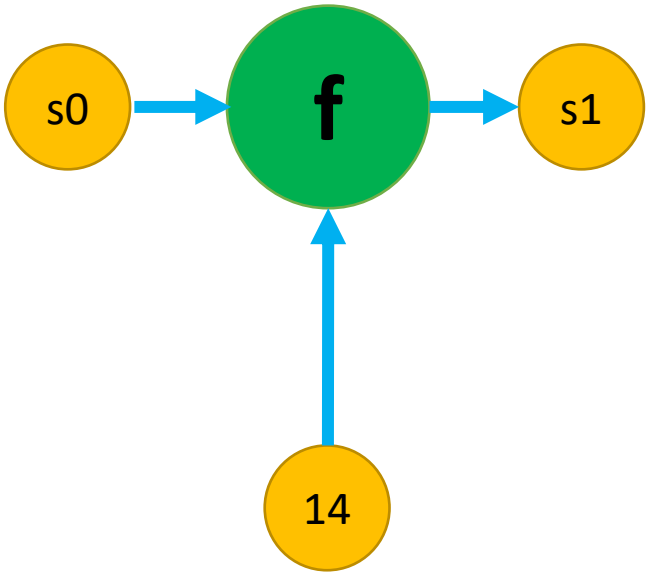
Building vocabulary

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

this film was just brilliant casting location scenery story

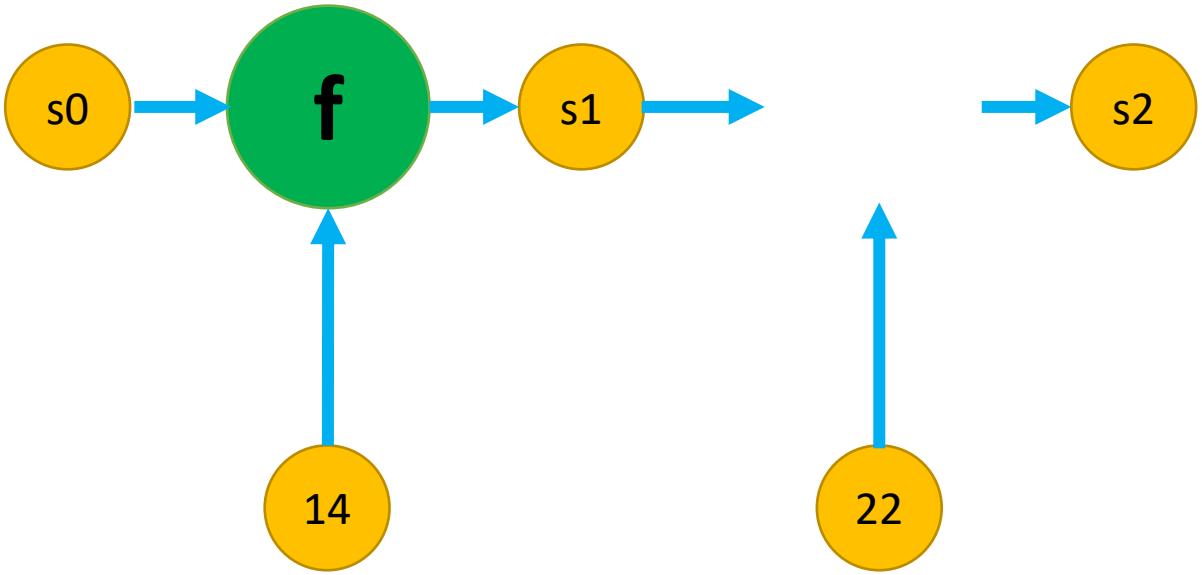
14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 39

Recurrent layer



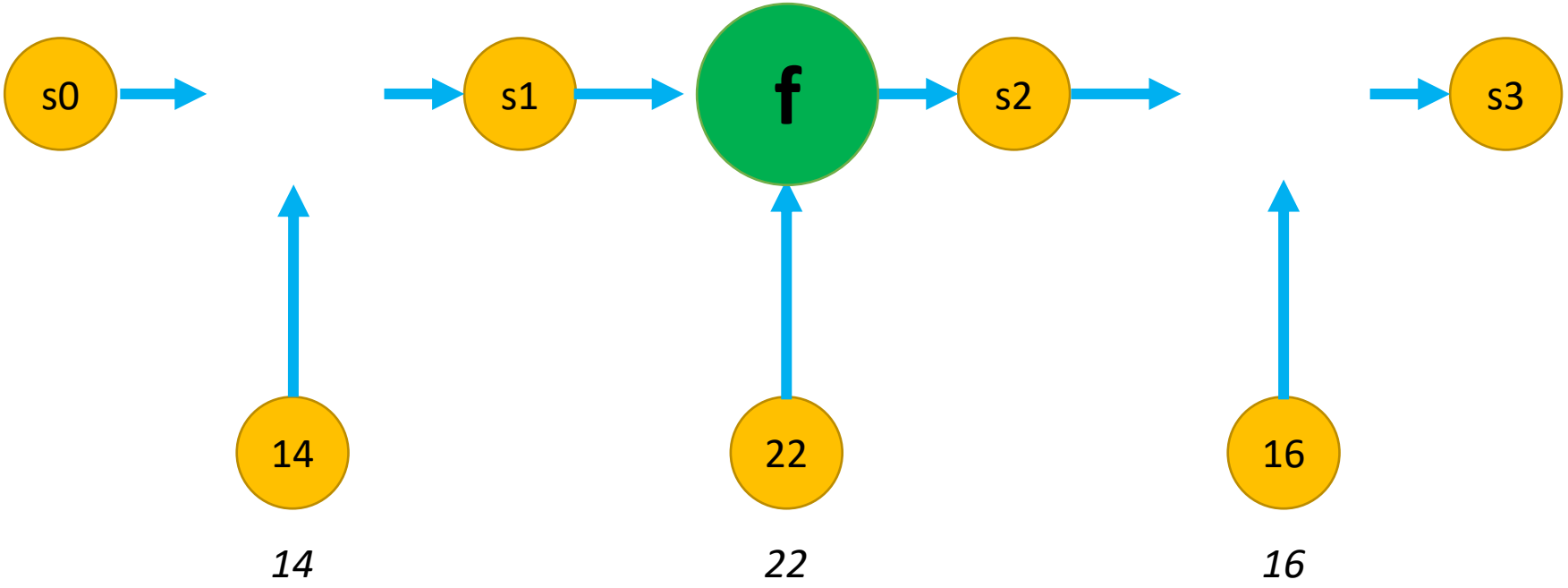
14 *22* *16*

Recurrent layer

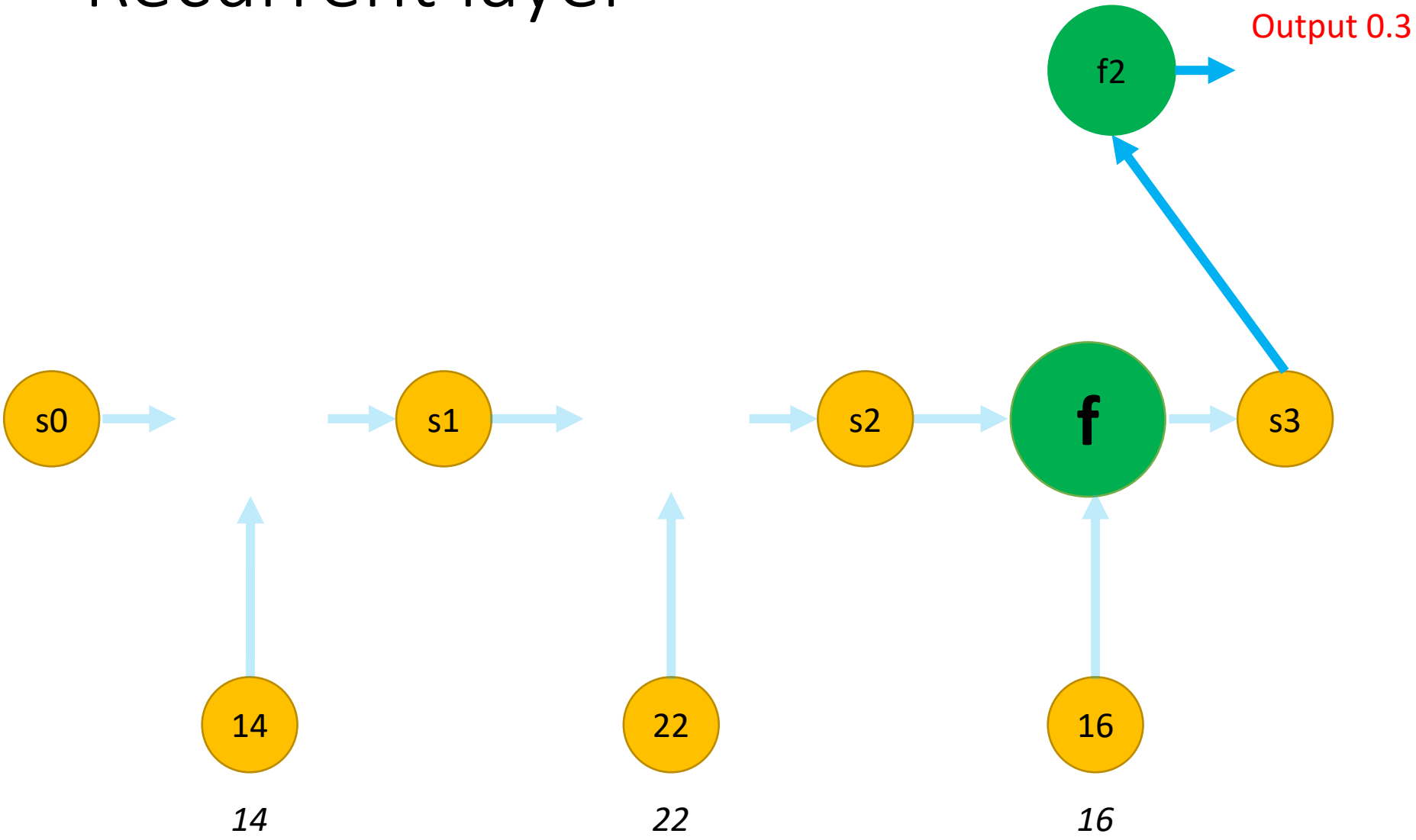


14 *22* *16*

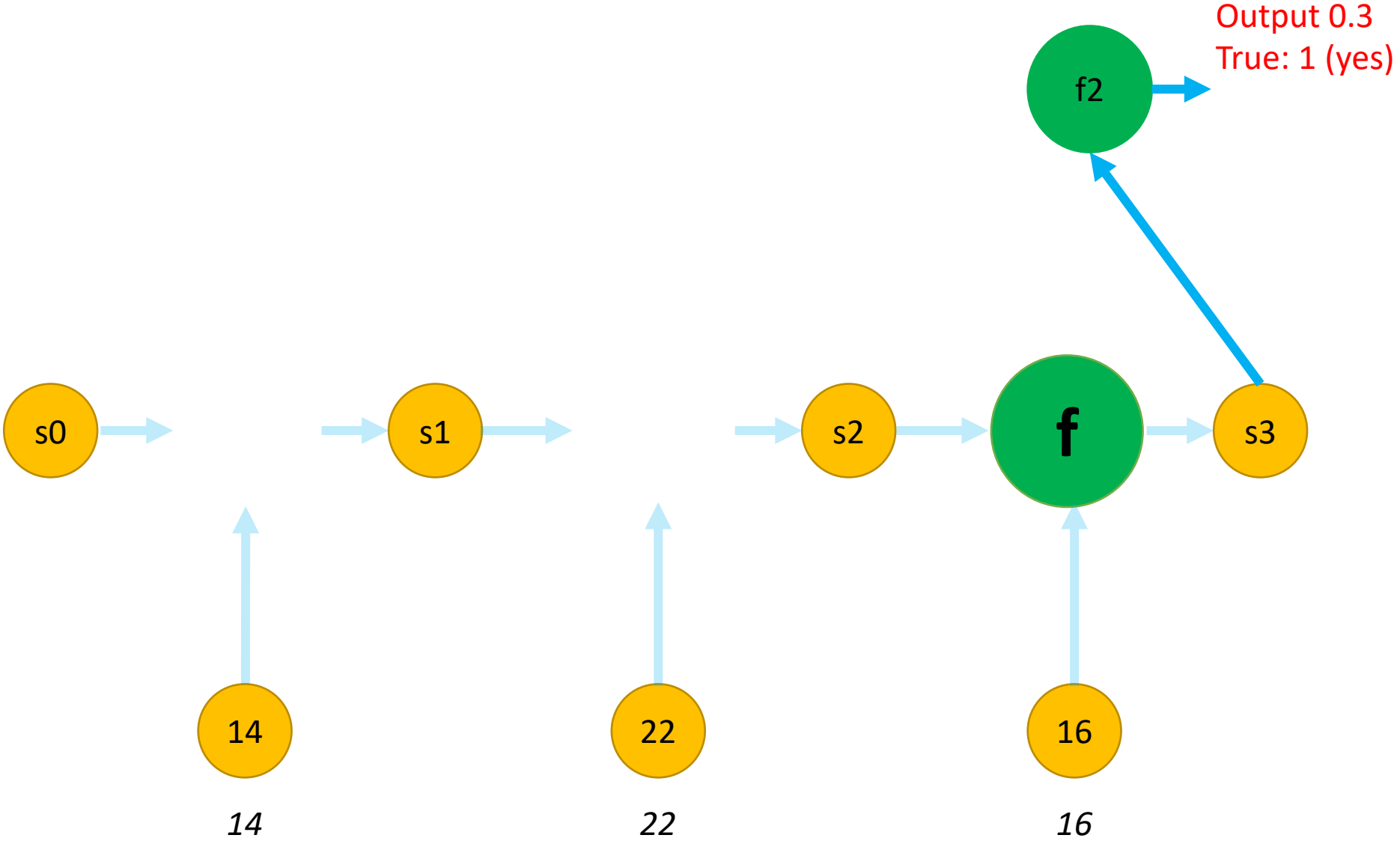
Recurrent layer



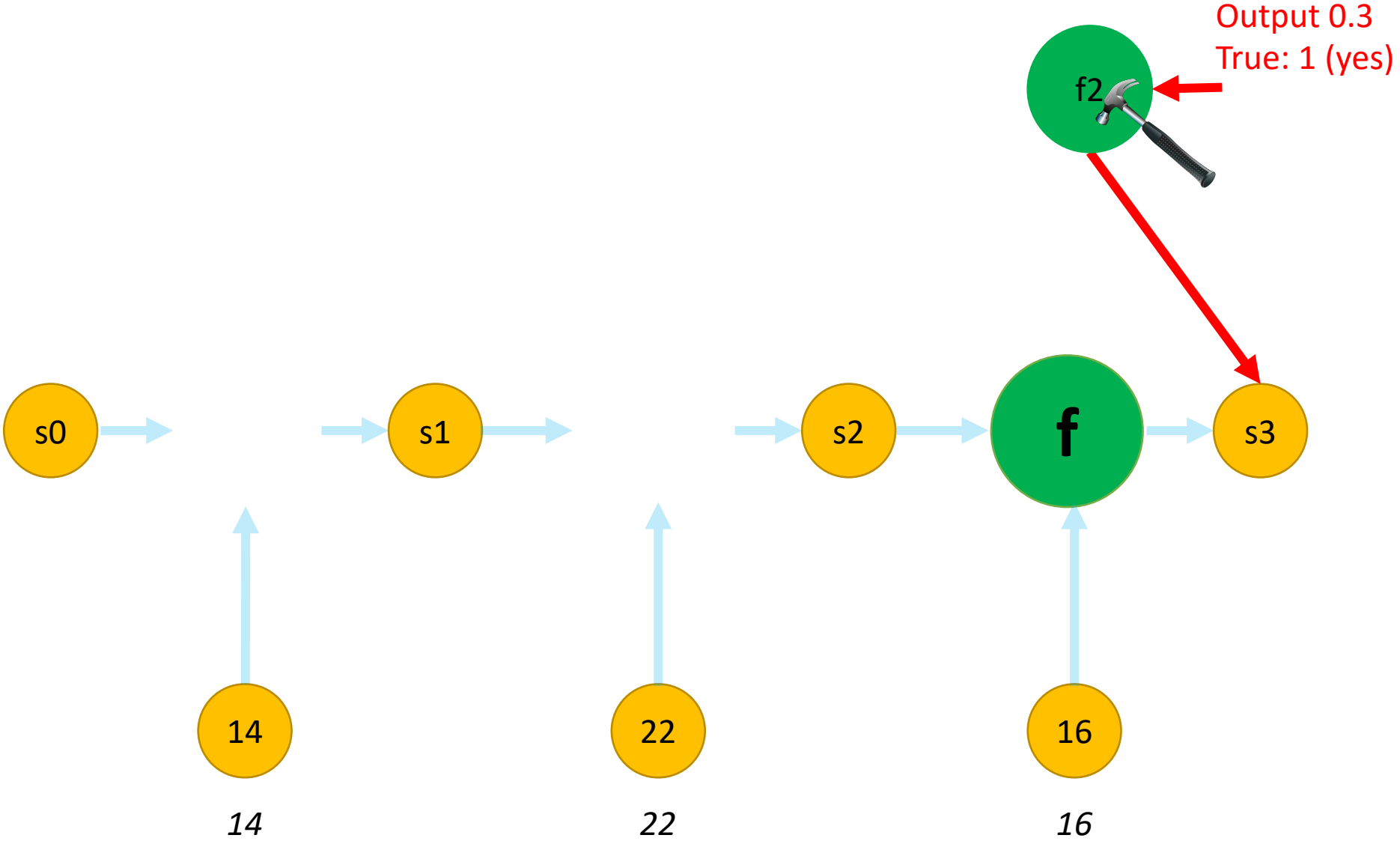
Recurrent layer



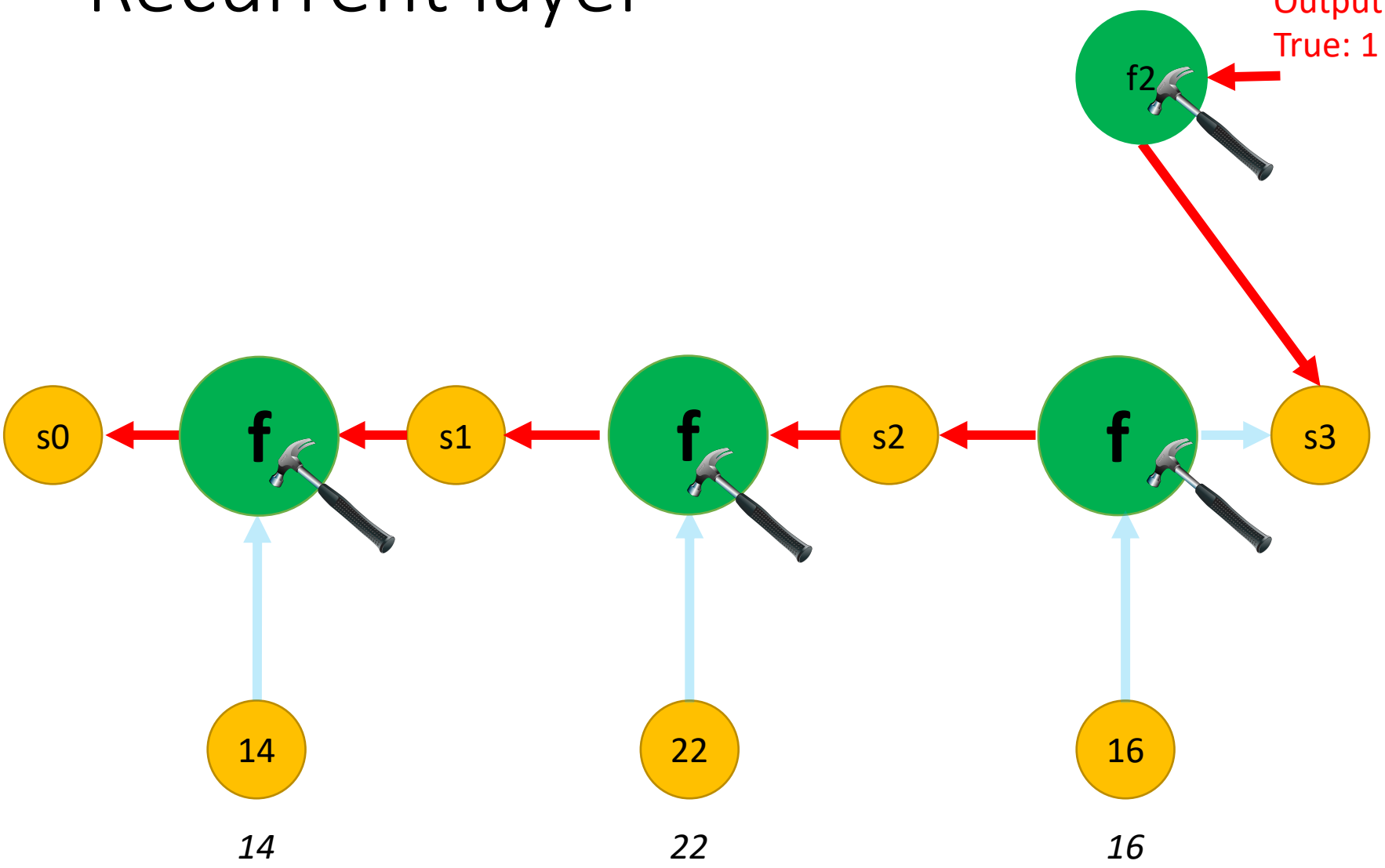
Recurrent layer



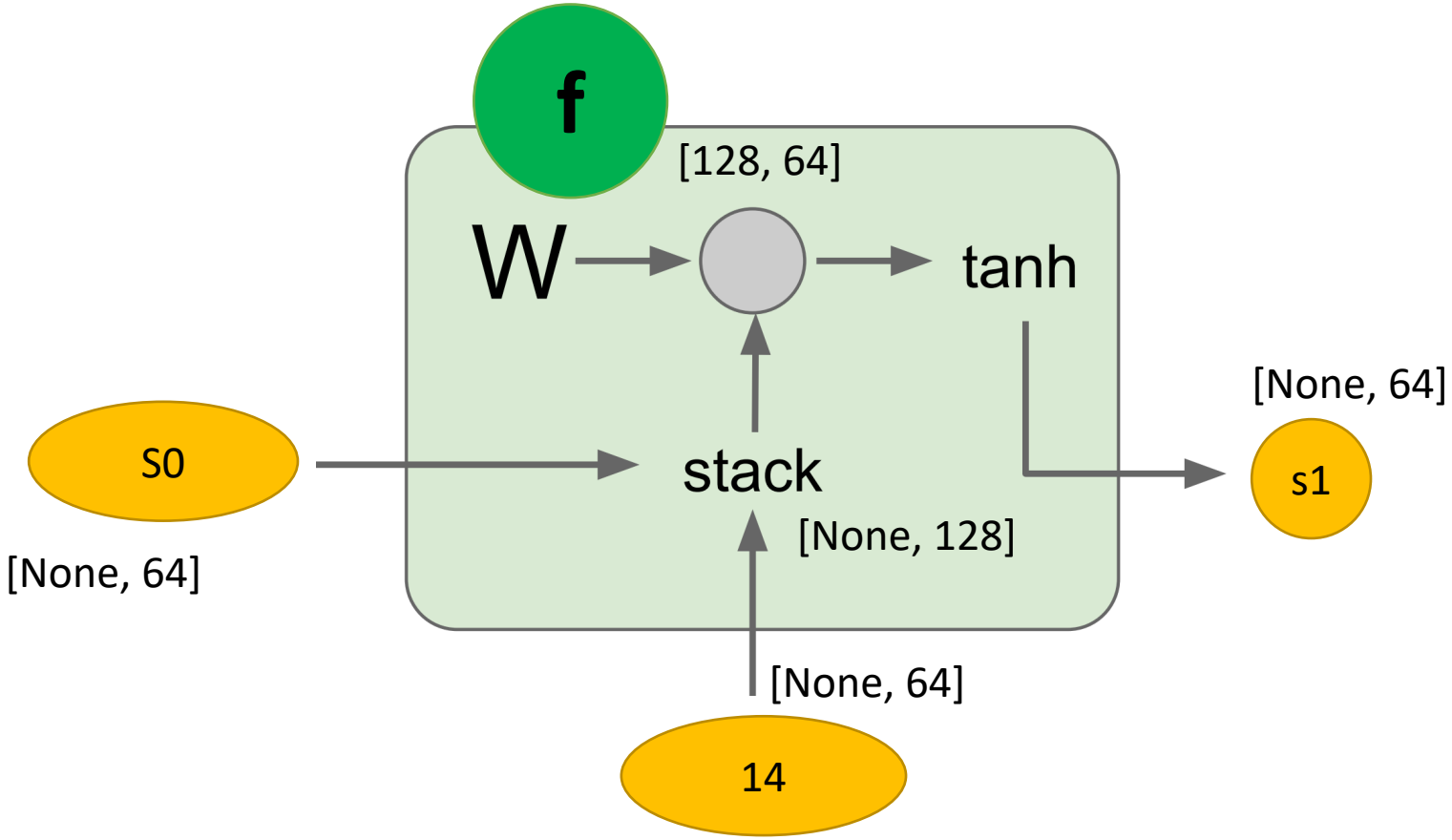
Recurrent layer



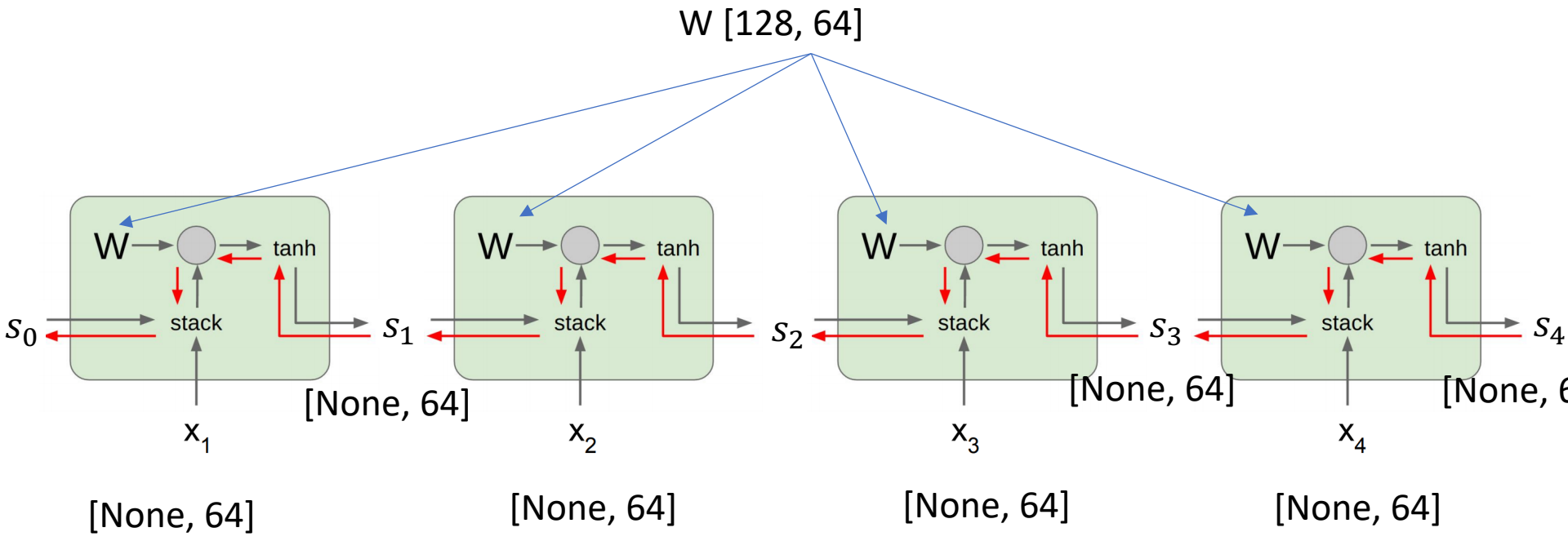
Recurrent layer



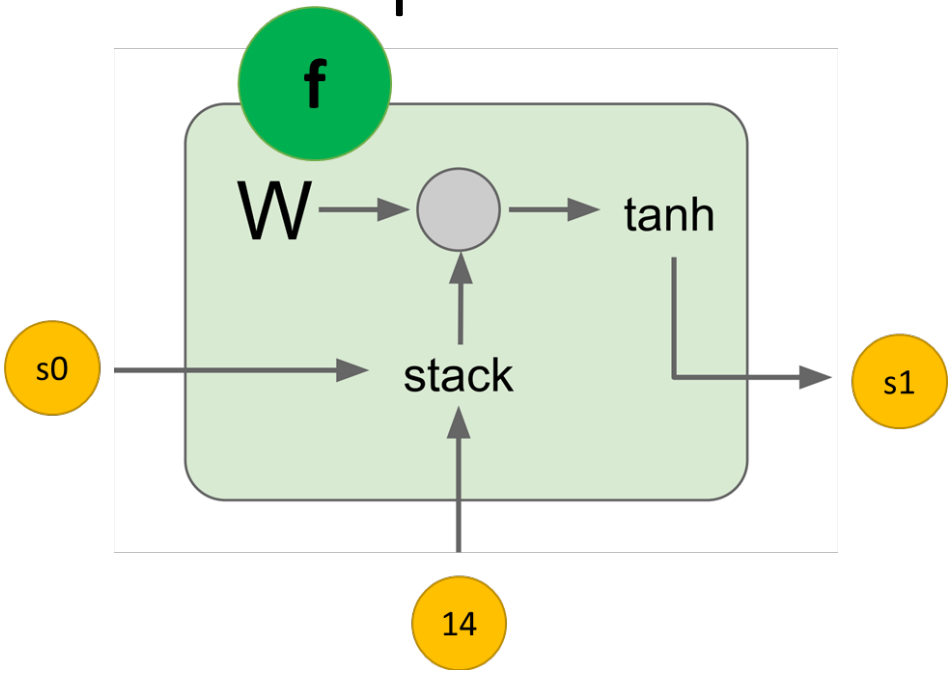
Cell implementation – Vanilla RNN



Cell implementation – Vanilla RNN



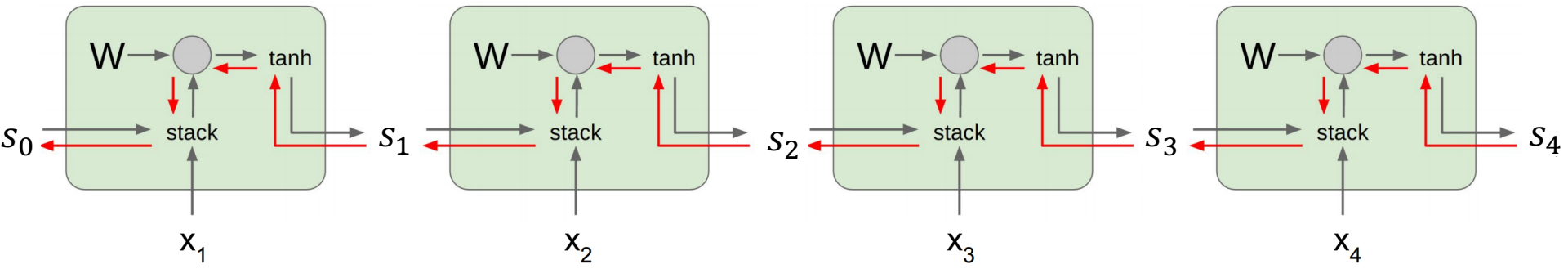
Cell implementation – Vanilla RNN




```
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
```

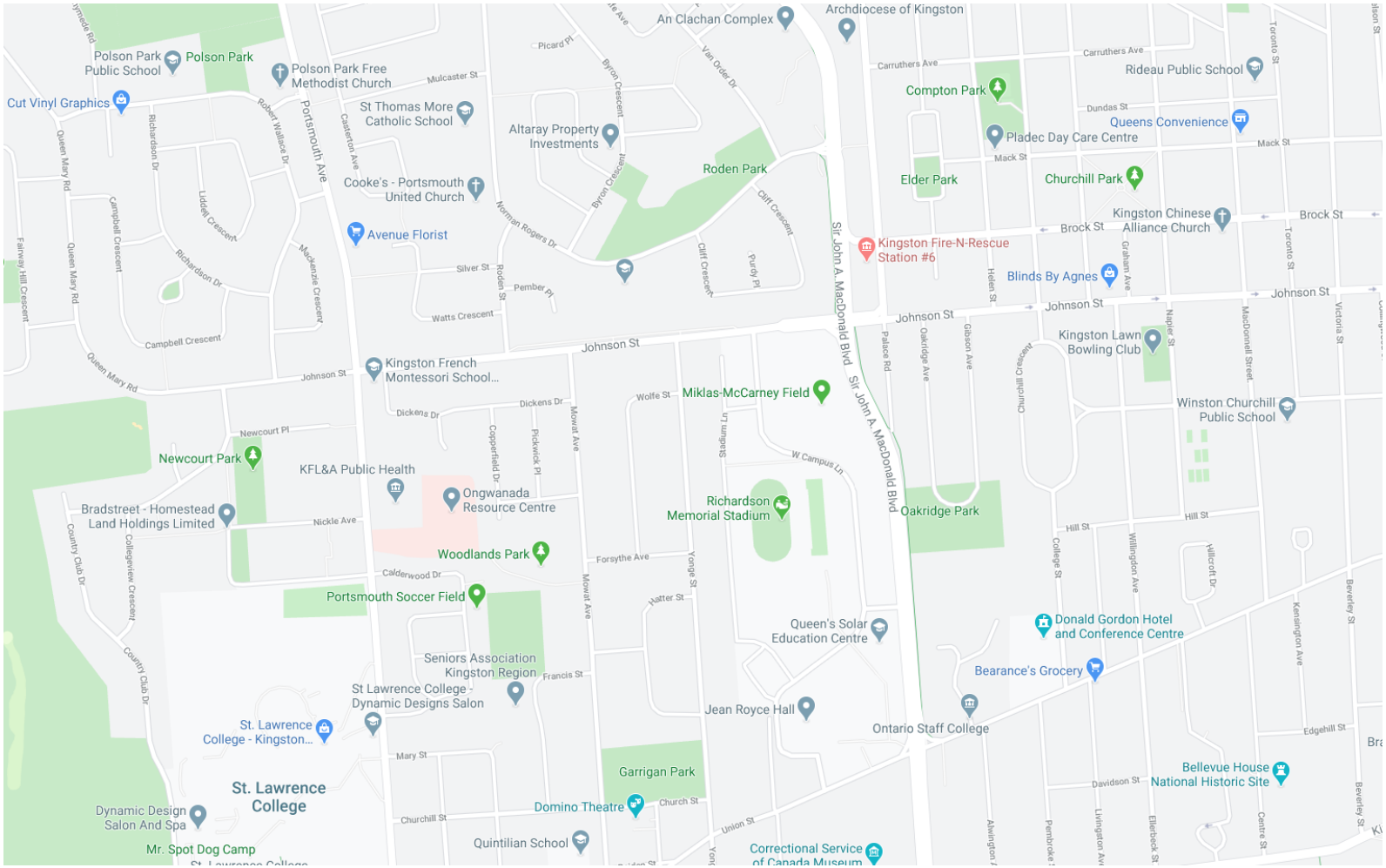
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Cell implementation – Vanilla RNN



Cell implementation – Vanilla RNN

- Problem: like driving in a driveway/roadway

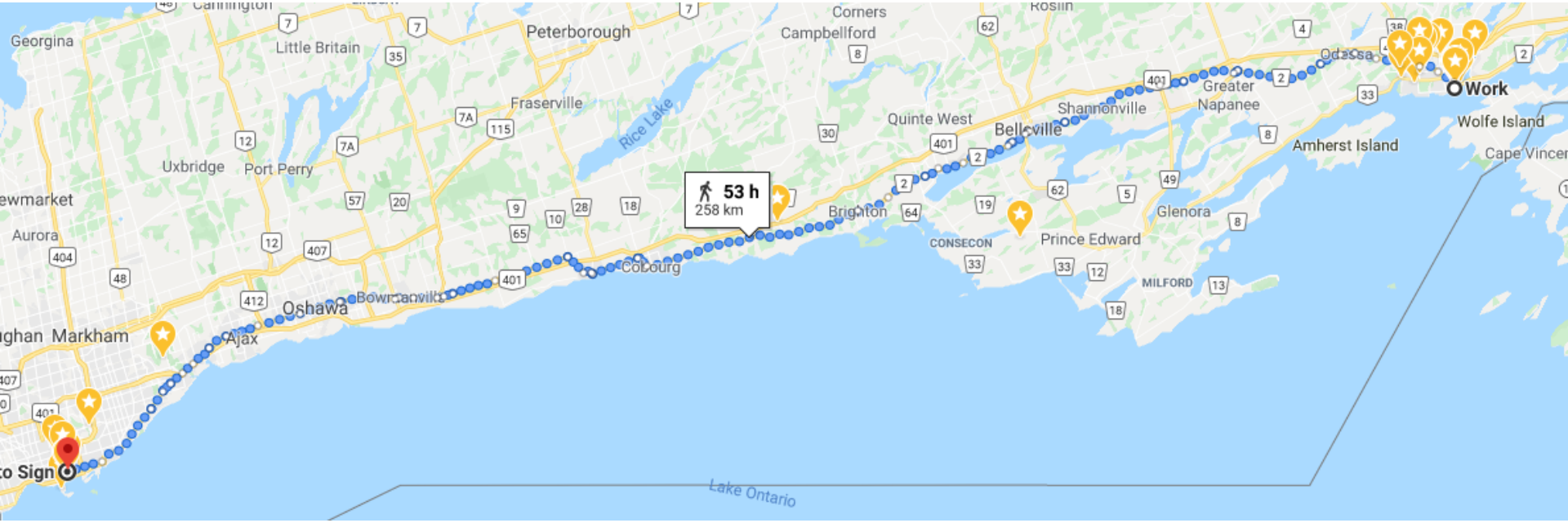


Cell implementation – Vanilla RNN

- 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)

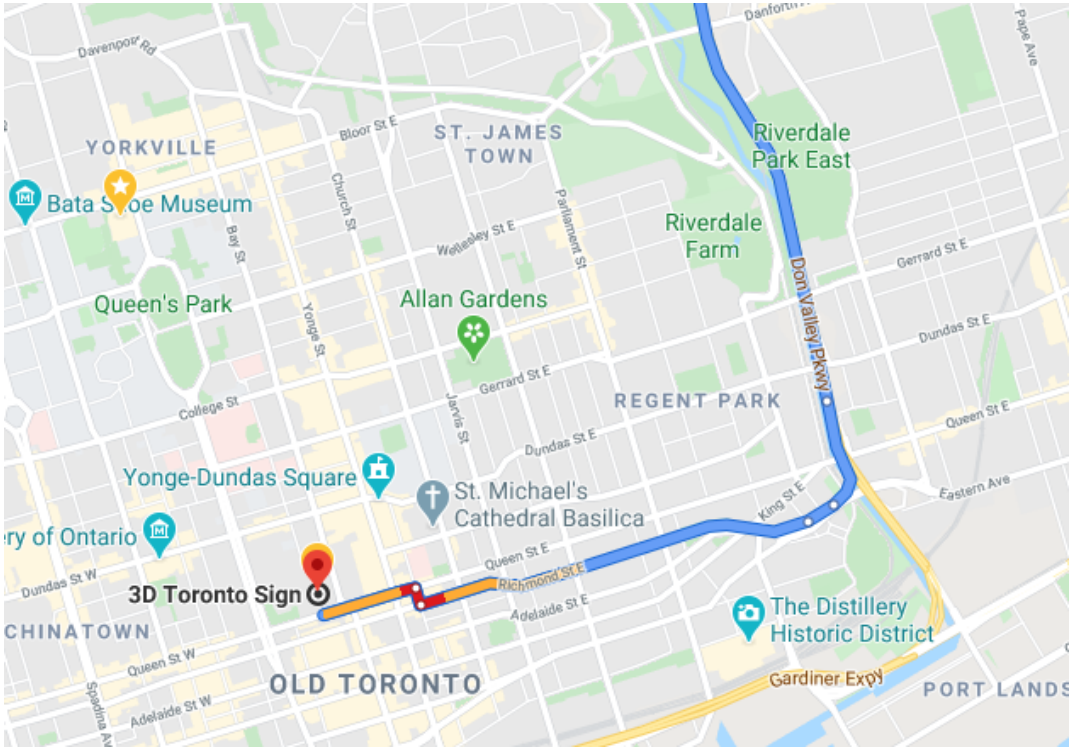
Cell implementation – Vanilla RNN

- 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

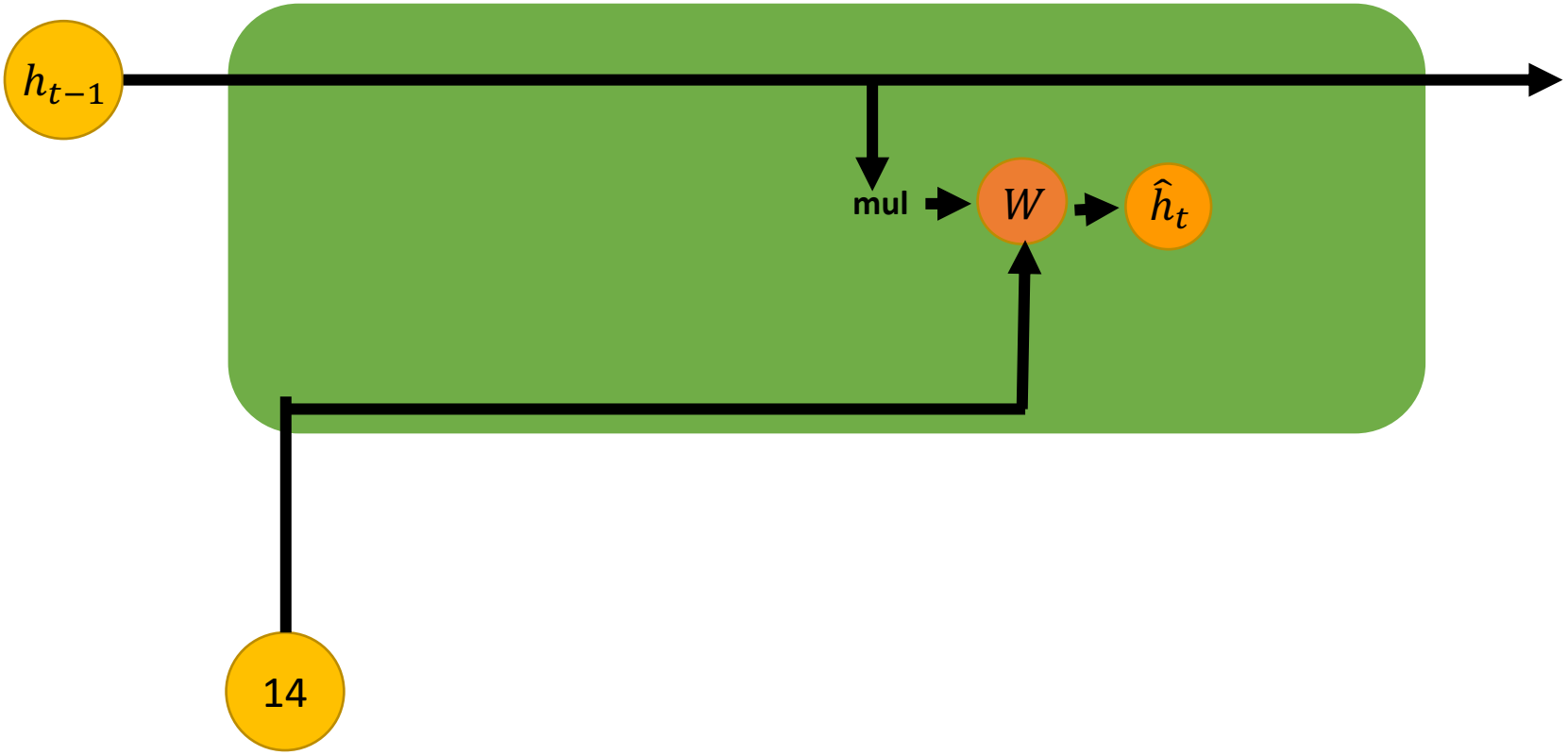
- GRU (similar idea to LSTM)
  - Long-short term memory
  - Long -> high way memory
    - Where to get off the high way
  - Short -> driveway/roadway memory
    - Where to pick up people (signal) of your final destination

Cell implementation – GRU

Cell implementation – GRU (high way)

14

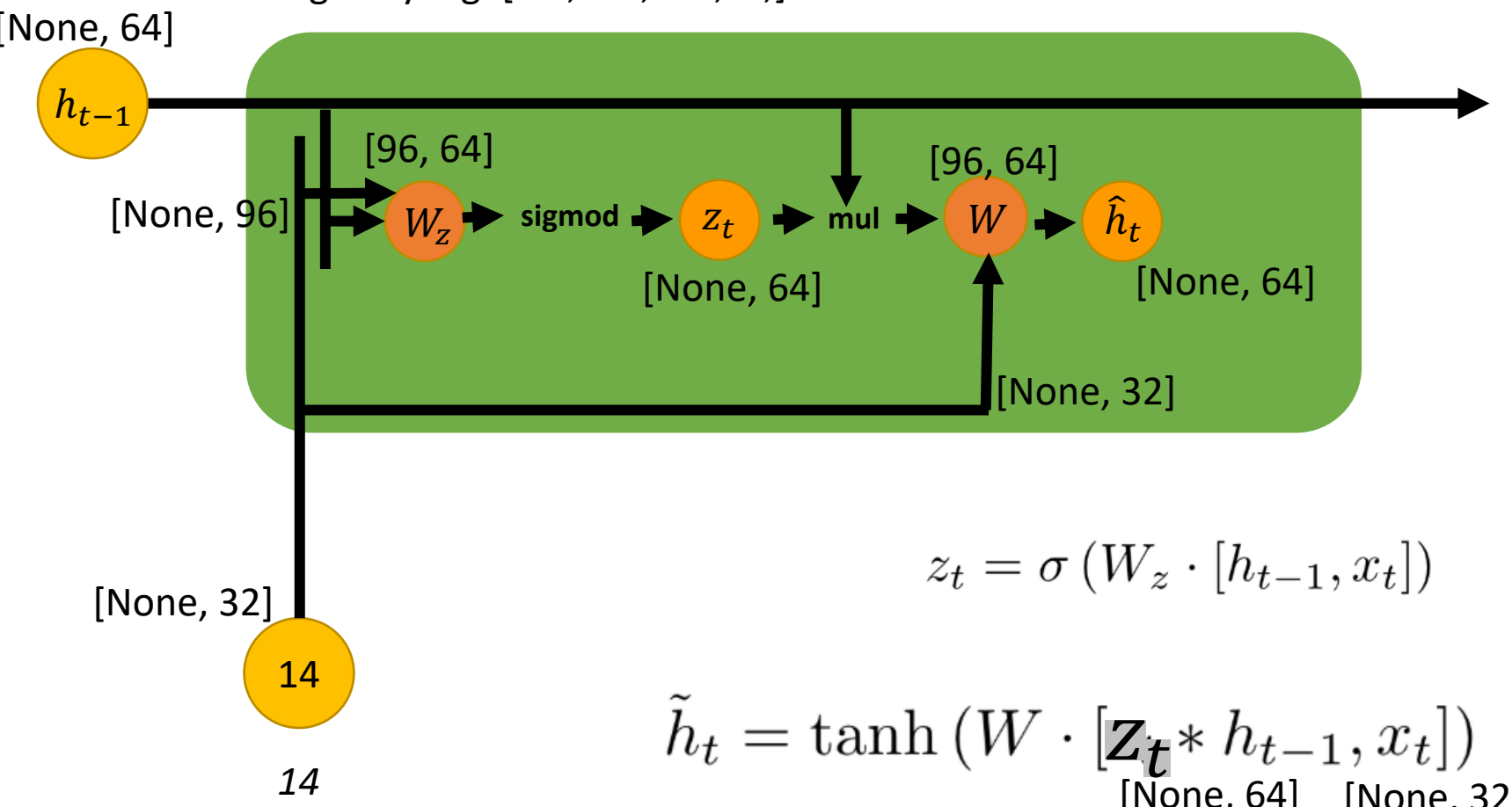
Cell implementation – GRU (roadway)



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 e.g. [0.1, 0.3, 0.2, …,]



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

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

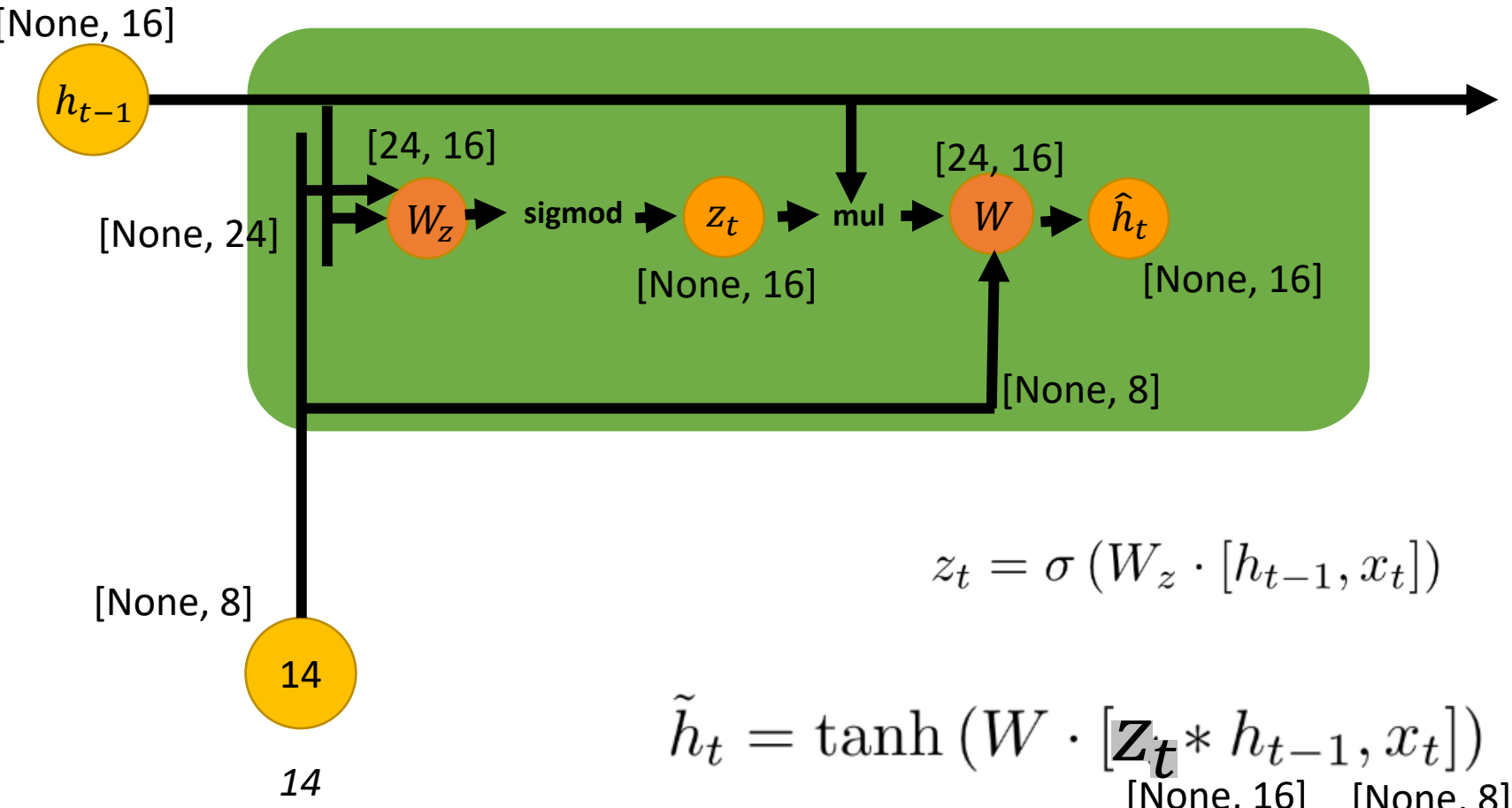
[None, 64] [None, 32]

[None, 96]

Cell implementation – GRU (roadway - gated) – (with a different shapes)

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 e.g. [0.1, 0.3, 0.2, …,]



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

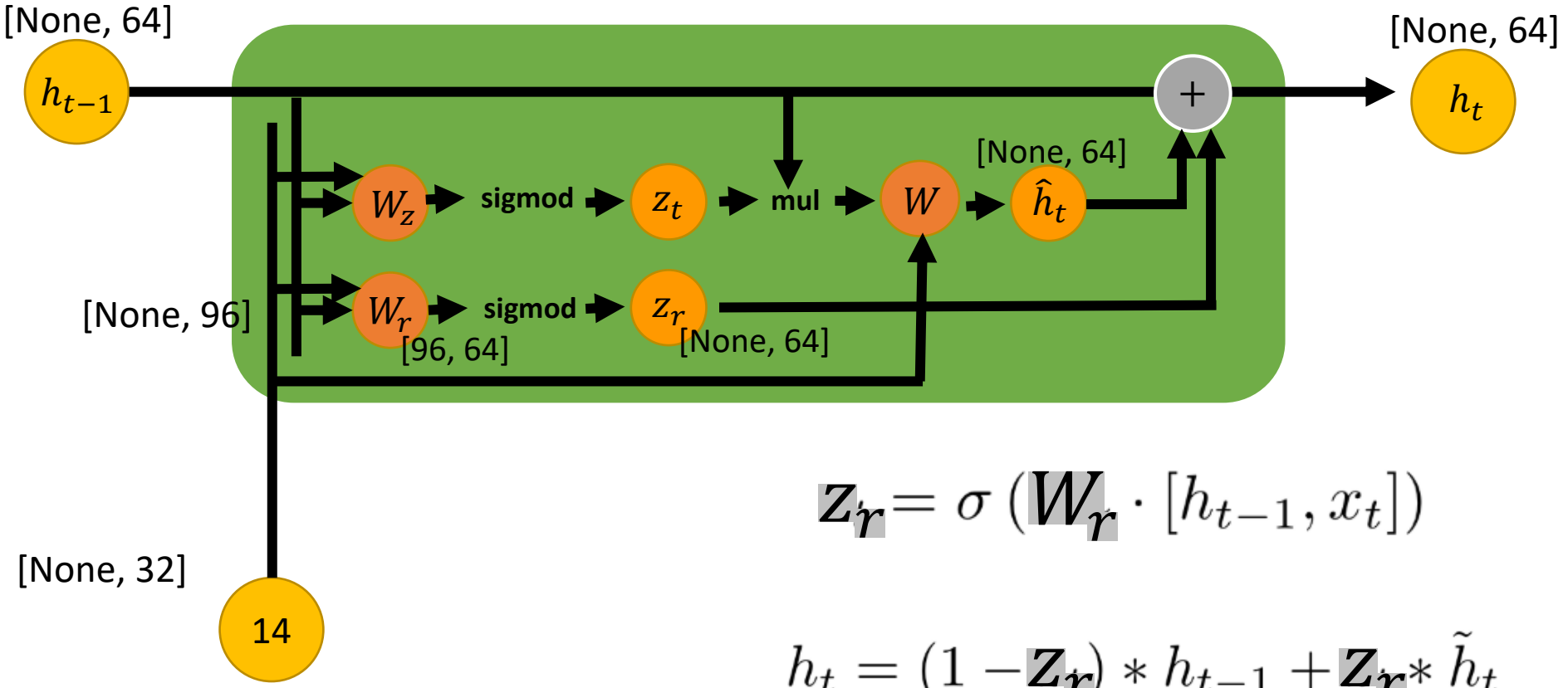
$$\tilde{h}_t = \tanh\left(W \cdot [z_t * h_{t-1}, x_t]\right)$$

[None, 16] [None, 8]

[None, 24]

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 e.g. [0.1, 0.8, ….,]

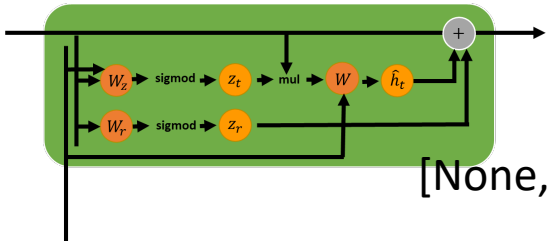


$$z_r = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

[None, 64] [None, 64] [None, 64]

Cell implementation – GRU (merged)



[None, 64]

[None, 64]

[None, 64]

[None, 64]

[None, 64]

[None, 64]

[None, 64]

14

[None, 32]

22

[None, 32]

16

[None, 32]

14

[None, 1]

22

[None, 1]

16

[None, 1]

[None, 1]

[No

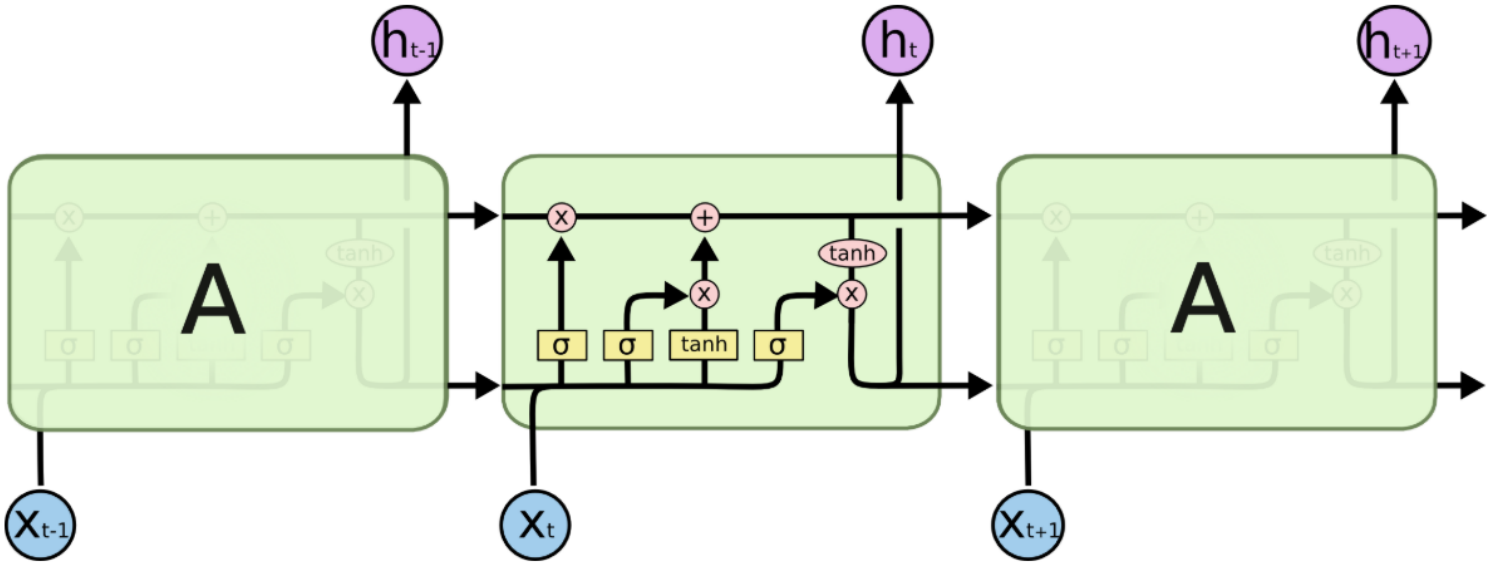
[None, 5]

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



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

Recap – Layer API

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Dropout, Input

def build():
    img_in = Input(shape=(256, 256, 2))
    flattened = Flatten()(img_in)
    fc1 = Dense(64)(flattened)
    #fc1 = Dropout(0.3)(fc1)
    fc2 = Dense(32)(fc1)
    #fc2 = Dropout(0.3)(fc2)
    output = Dense(1, activation = 'sigmoid')(fc2)
    model = tf.keras.Model(inputs=img_in, outputs=output)
    return model


model = build()
model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss='binary_crossentropy',
        metrics=['BinaryAccuracy', 'AUC']
        )
```

API Embedding

☆ ☆ ☆ ☆ ☆

tf.keras.layers.Embedding

TensorFlow 1 version | View source on GitHub

Turns positive integers (indexes) into dense vectors of fixed size.

Inherits From: `Layer`

View aliases

```
tf.keras.layers.Embedding(
    input_dim, output_dim, embeddings_initializer='uniform',
    embeddings_regularizer=None, activity_regularizer=None,
    embeddings_constraint=None, mask_zero=False, input_length=None, **kwargs
)
```

API Embedding

Arguments	
`input_dim`	Integer. Size of the vocabulary, i.e. maximum integer index + 1.
`output_dim`	Integer. Dimension of the dense embedding.
`embeddings_initializer`	Initializer for the `embeddings` matrix (see `keras.initializers`).
`embeddings_regularizer`	Regularizer function applied to the `embeddings` matrix (see `keras.regularizers`).
`embeddings_constraint`	Constraint function applied to the `embeddings` matrix (see `keras.constraints`).
`mask_zero`	Boolean, whether or not the input value 0 is a special "padding" value that should be masked out. This is useful when using recurrent layers which may take variable length input. If this is `True`, then all subsequent layers in the model need to support masking or an exception will be raised. If mask_zero is set to True, as a consequence, index 0 cannot be used in the vocabulary (input_dim should equal size of vocabulary + 1).
`input_length`	Length of input sequences, when it is constant. This argument is required if you are going to connect `Flatten` then `Dense` layers upstream (without it, the shape of the dense outputs cannot be computed).

Input shape:

2D tensor with shape: `(batch_size, input_length)`.

Output shape:

3D tensor with shape: `(batch_size, input_length, output_dim)`.

API RNN

```
tf.keras.layers.GRU(
    units, activation='tanh', recurrent_activation='sigmoid', use_bias=True,
    kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
    bias_initializer='zeros', kernel_regularizer=None, recurrent_regularizer=None,
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
    recurrent_constraint=None, bias_constraint=None, dropout=0.0,
    recurrent_dropout=0.0, implementation=2, return_sequences=False,
    return_state=False, go_backwards=False, stateful=False, unroll=False,
    time_major=False, reset_after=True, **kwargs
)
```

Suppose x # [None, 500, 1] (same as [None, 500])

dtype=tf.int64 (input sequence with word index) (500 is seq_len)

embedded =Embedding(10000, 32)(x) # [None, 500, 32]

output = GRU(75)(embedded) # [None, 75]

output = GRU(75, return_sequences=True)(embedded) # [None, 500, 75]

output2 = GRU(75, return_sequences=True)(output) # [None, 500, 75]

GRU(75, return_sequences=True, return_state=True)(embedded) # ([None, 100 500 75], [None, 75])

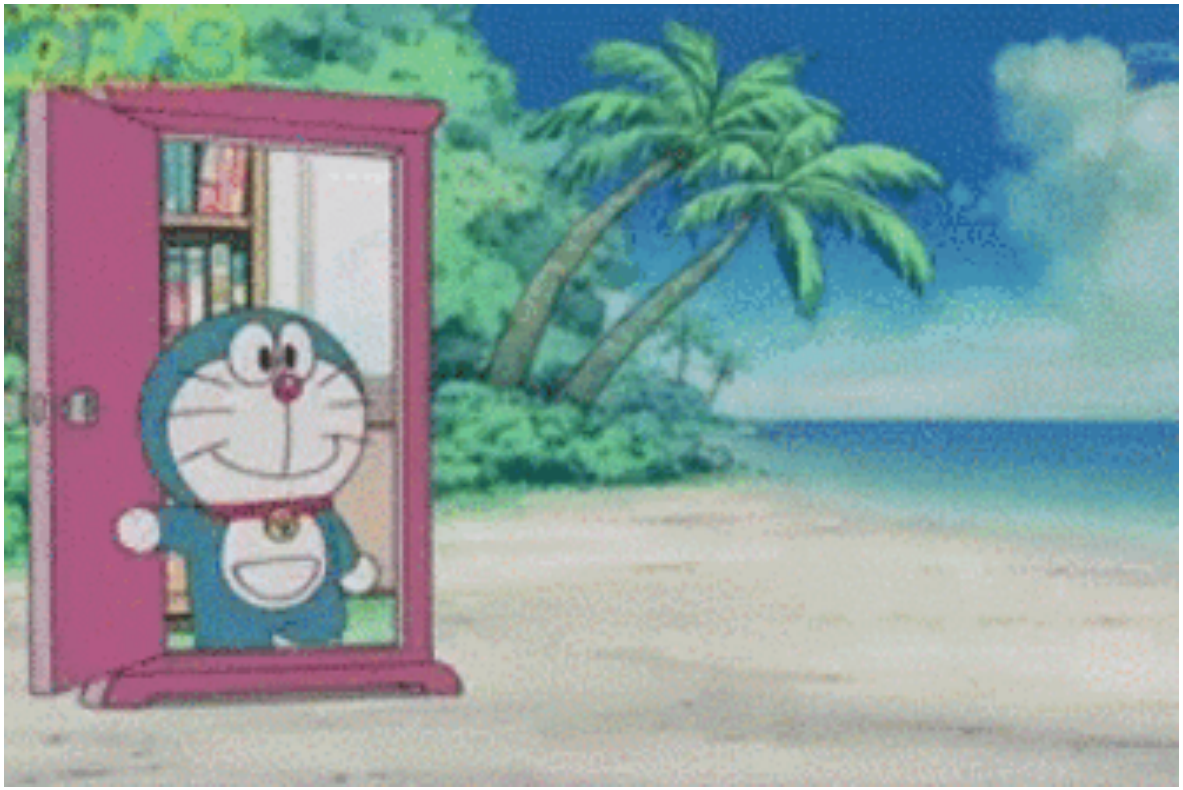
Attention Mechanism

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

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

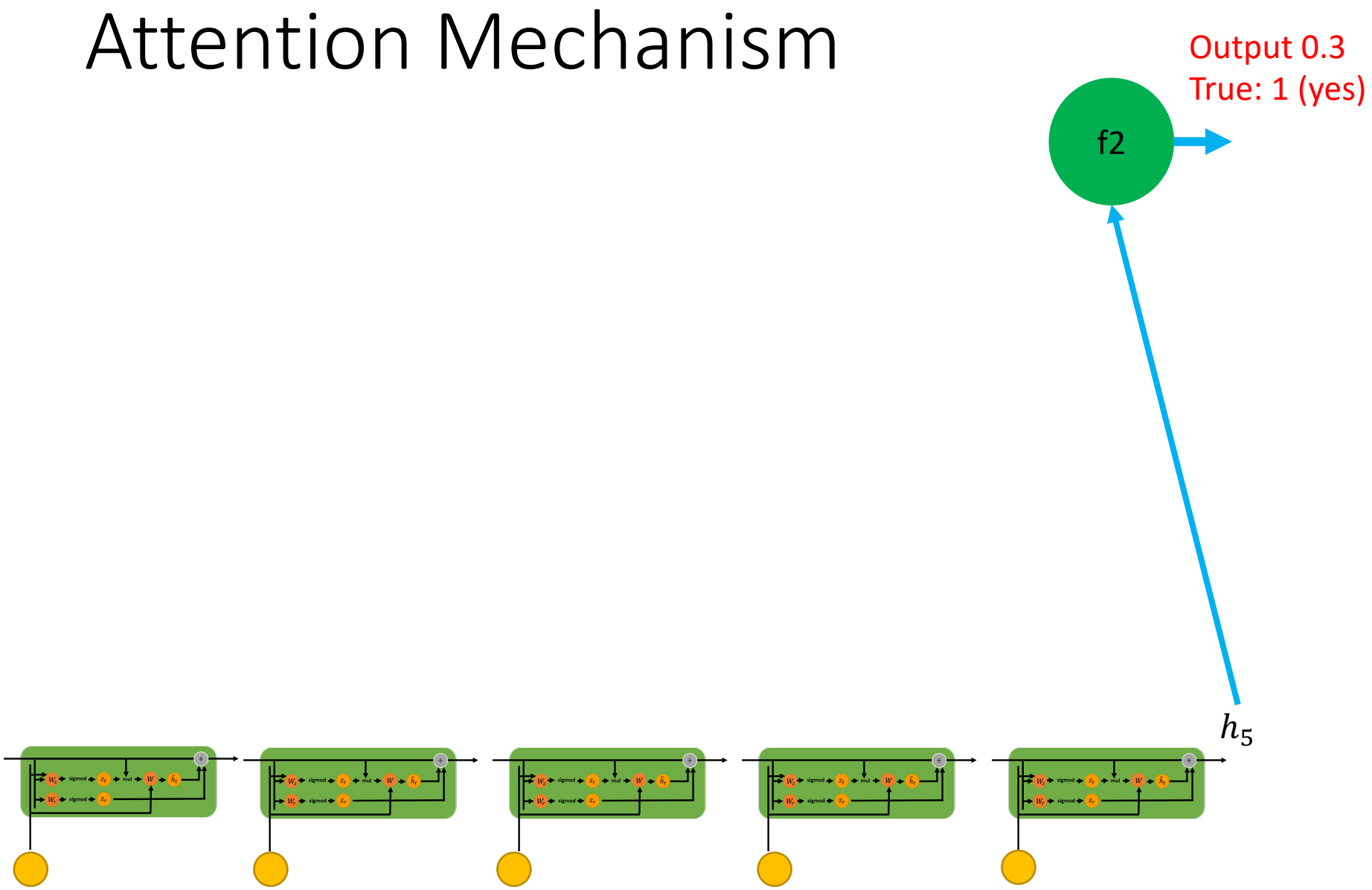
Attention Mechanism

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

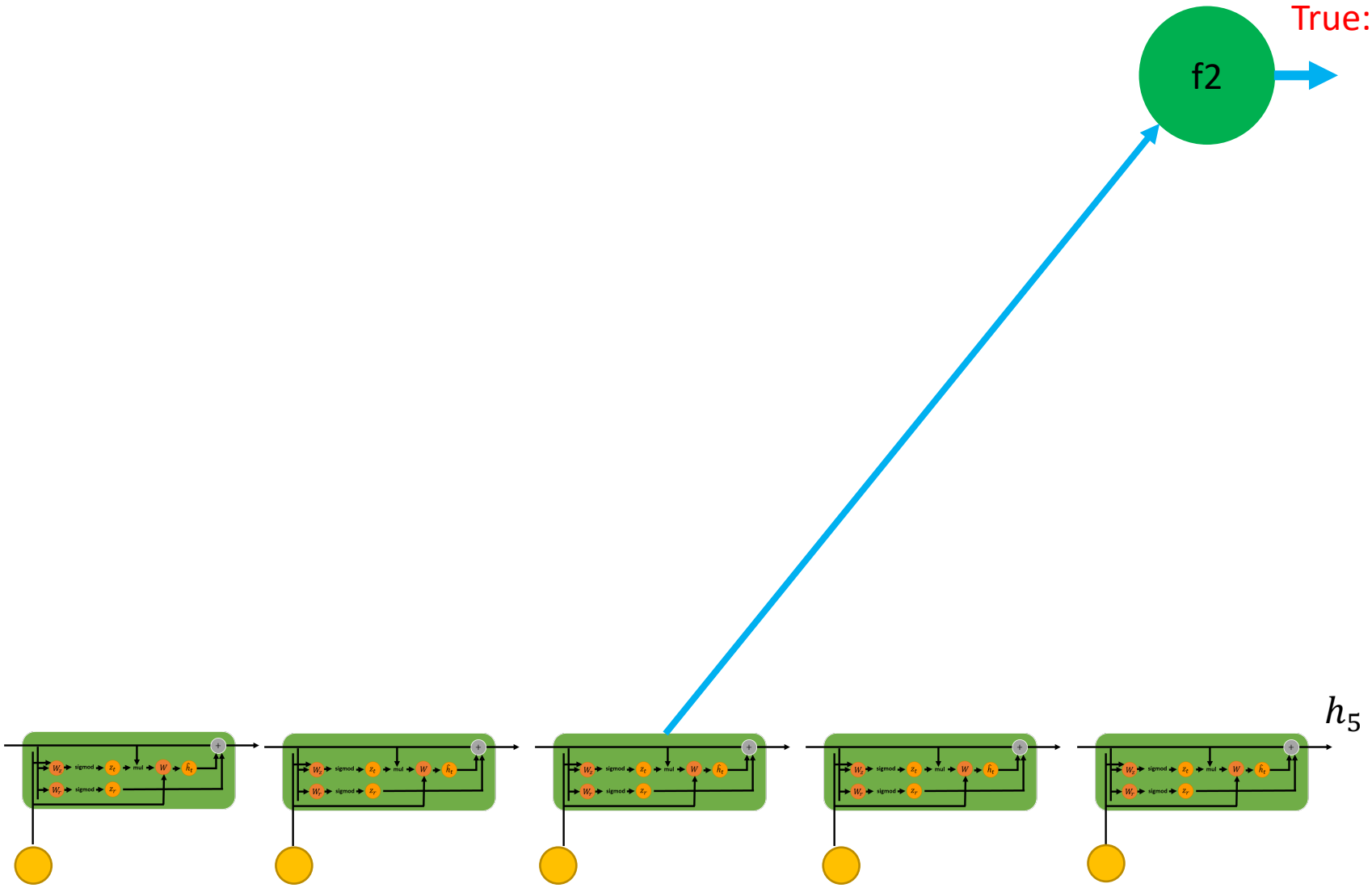


https://www.pinterest.ca/pin/726979564827879912/?lp=true

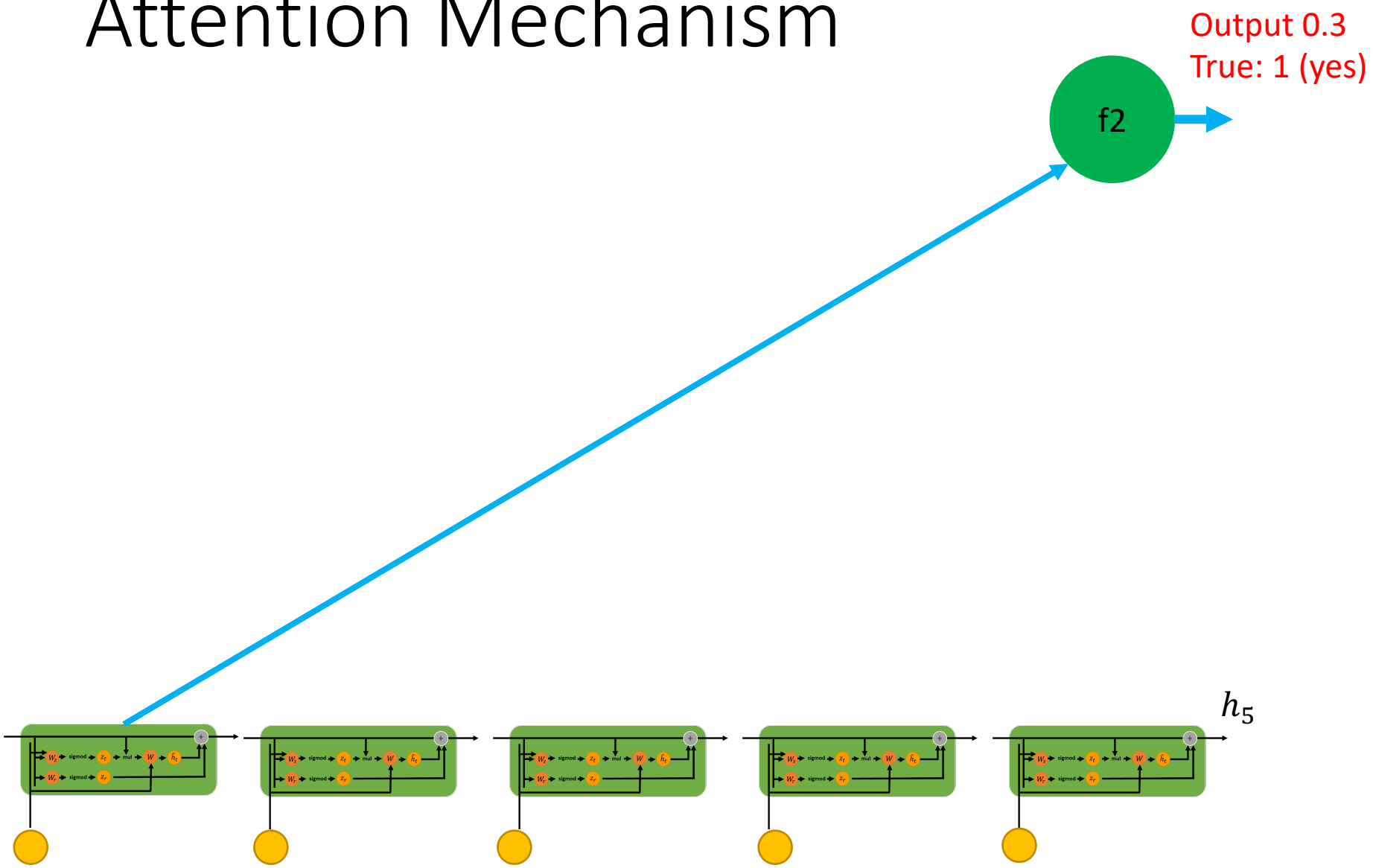
Attention Mechanism



Attention Mechanism



Attention Mechanism



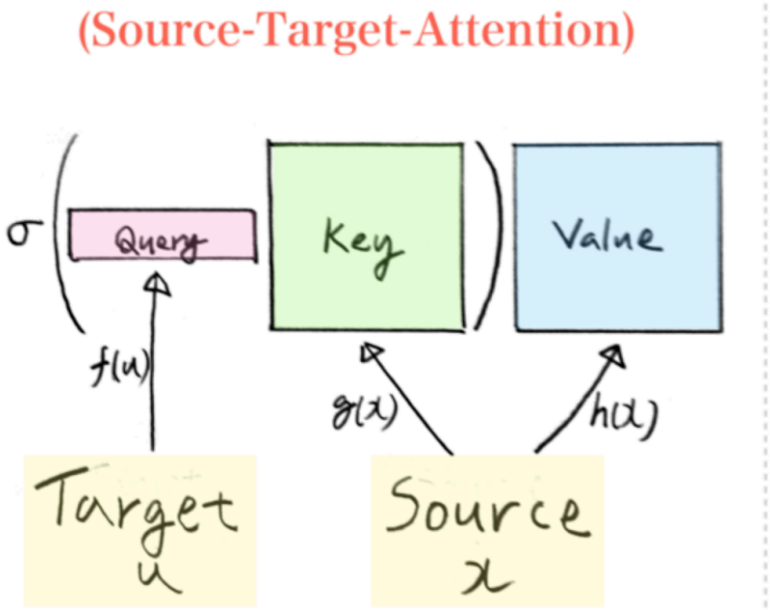
Output 0.3
True: 1 (yes)

f2

h_0 h_1 h_2 h_3 h_4 h_5

ATTENTION

- As mapping a query and a set of key-value pairs to an output



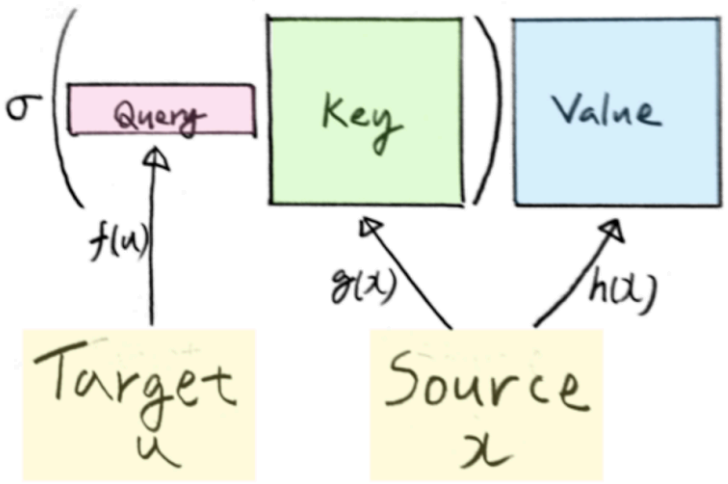
- Query: based on what we are attending to
- Key: value used for weight calculation
- Value: values used for aggregation
- Final output: weighted averaged of values where weights are based on query and key

https://arxiv.org/abs/1706.03762

ATTENTION

- As mapping a query and a set of key-value pairs to an

(Source-Target-Attention)



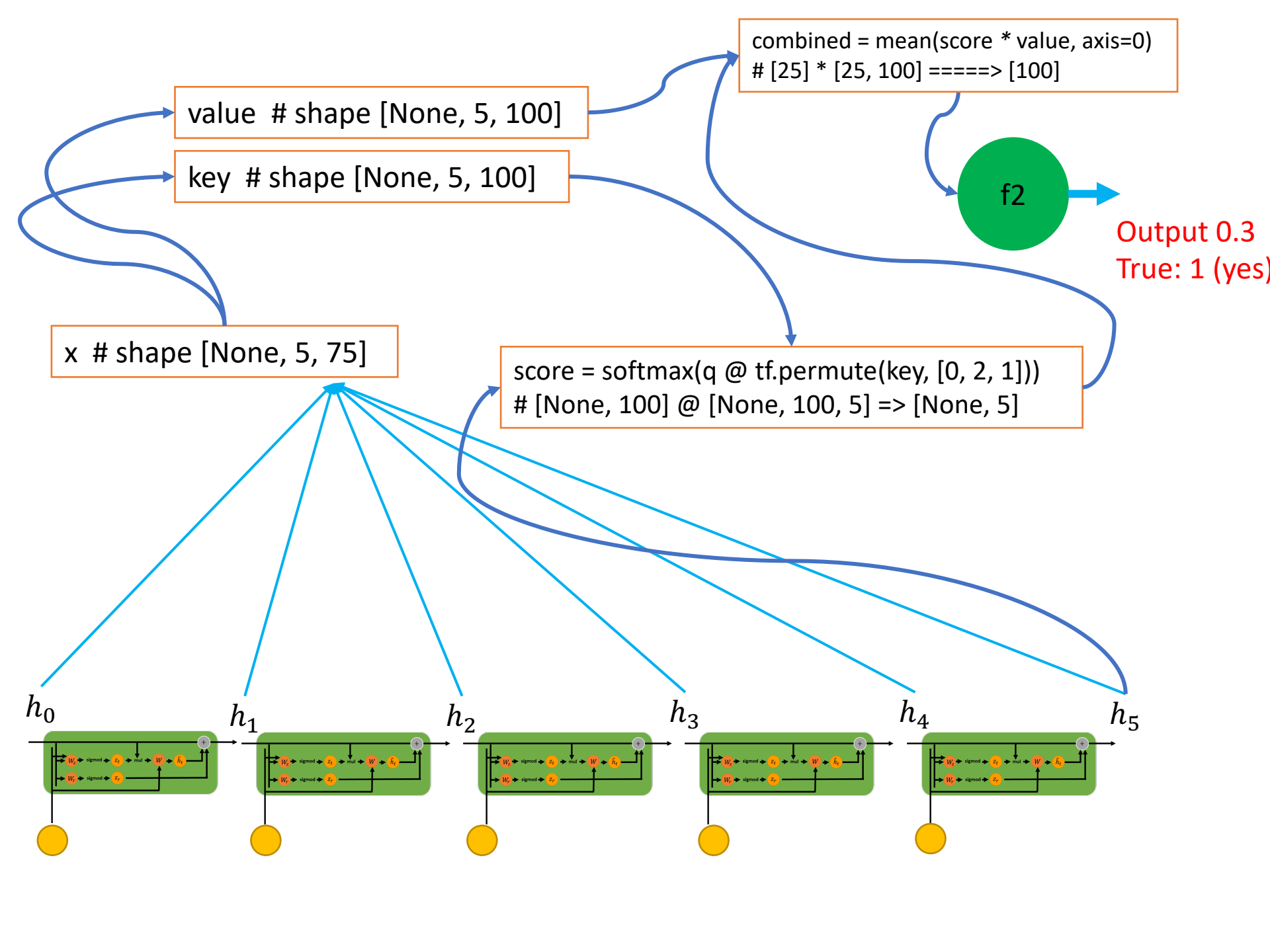

```
u  # shape [100]
x  # shape [25, 100]

key = g(x)
# [25, 100]  25 items for lookup

value = h(x)
# [25, 100] values to be combined

score = softmax(u @ Transpose(key))
 # [25]

combined = mean(score * value, axis=0)
# [25] * [25, 100] =====> [100]
```

https://arxiv.org/abs/1706.03762

Output 0.3
True: 1 (yes)

f2

h_0 h_1 h_2 h_3 h_4 h_5


combined = mean(score * value, axis=0)
# [25] * [25, 100] =====> [100]
value # shape [None, 5, 100]
key # shape [None, 5, 100]
f2
Output 0.3
True: 1 (yes)
x # shape [None, 5, 75]
score = softmax(q @ tf.permute(key, [0, 2, 1]))
# [None, 100] @ [None, 100, 5] => [None, 5]
h_0
h_1
h_2
h_3
h_4
h_5

API Attention (multiple query)

```
tf.keras.layers.Attention(
    use_scale=False, **kwargs
)
```

Inputs are `query` tensor of shape `[batch_size, Tq, dim]`, `value` tensor of shape `[batch_size, Tv, dim]` and `key` tensor of shape `[batch_size, Tv, dim]`. The calculation follows the steps:

1. Calculate scores with shape `[batch_size, Tq, Tv]` as a `query` - `key` dot product: `scores = tf.matmul(query, key, transpose_b=True)`.
2. Use scores to calculate a distribution with shape `[batch_size, Tq, Tv]`: `distribution = tf.nn.softmax(scores)`.
3. Use `distribution` to create a linear combination of `value` with shape `[batch_size, Tq, dim]`: `return tf.matmul(distribution, value)`.