Introduction to machine translation

MACHINE TRANSLATION IN PYTHON

Thushan Ganegedara
Data Scientist and Author

Machine translation

Привет

හෙලෝ

Hola

Hallo

Bonjour

你好

Olá

Machine translation

Привет

හෙලෝ

Hola

Hallo

Bonjour

你好

Olá

Hello

Course outline

- Chapter 1 - Introduction to machine translation
- Chapter 2 - Implement a machine translation model (encoder-decoder architecture)
- Chapter 3 - Training the model and generating translations
- Chapter 4 - Improving the translation model

Dataset (English-French sentence corpus)

- English corpus

```
new jersey is sometimes quiet during autumn , and it is snowy in april .
the united states is usually chilly during july , and it is usually freezing ...
california is usually quiet during march , and it is usually hot in june .
```

- French corpus

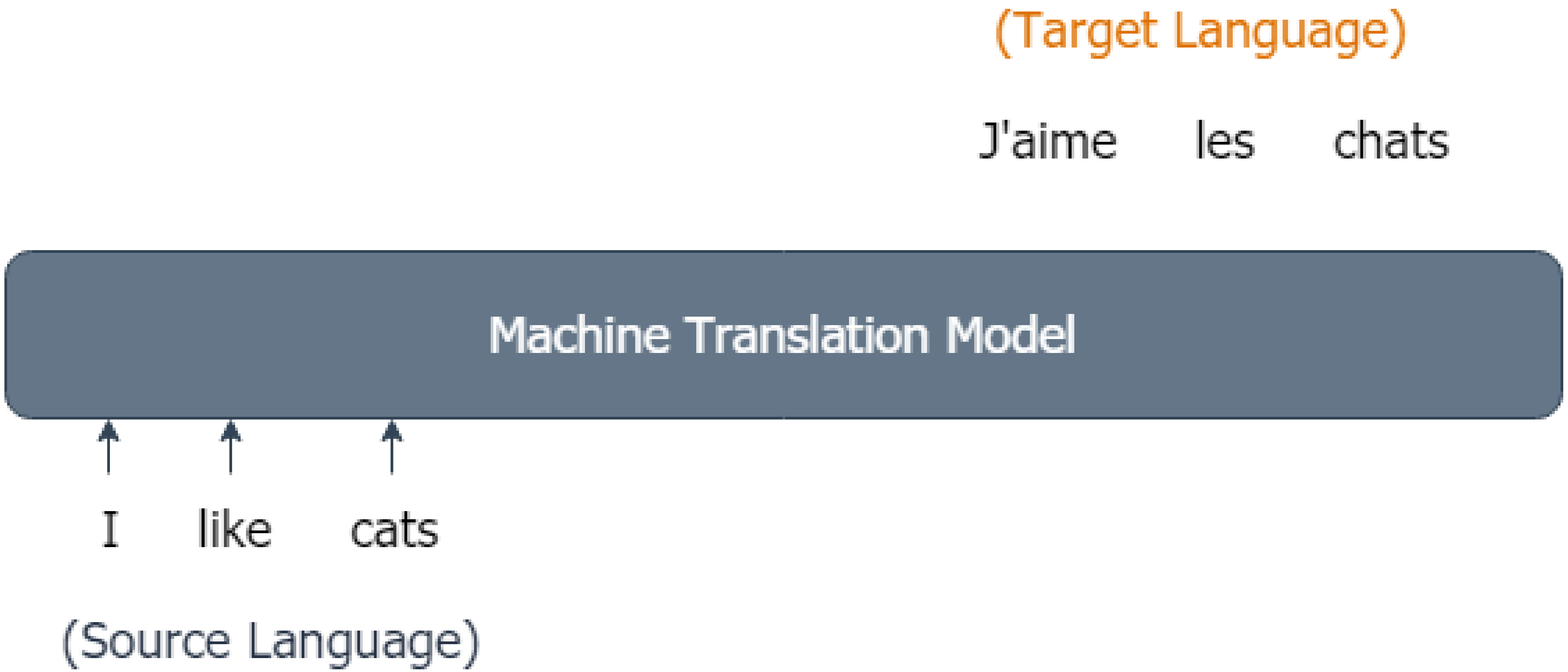
```
new jersey est parfois calme pendant l' automne , et il est neigeux en avril .
les états-unis est généralement froid en juillet , et il gèle habituellement ...
california est généralement calme en mars , et il est généralement chaud en juin .
```

[1] https://github.com/udacity/deep [2] learning/tree/master/language [3] translation/data

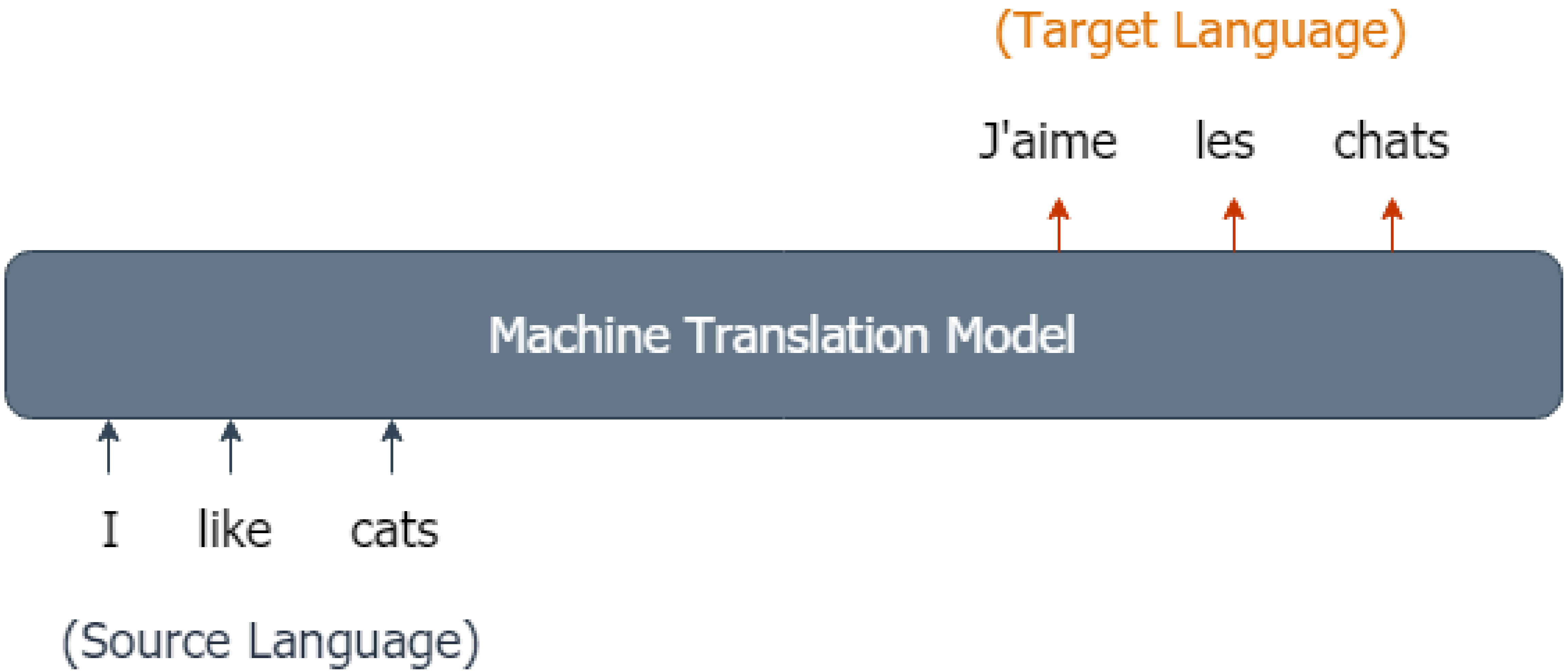
Machine translation - Overview

Machine translation - Overview

Machine translation - Overview



Machine translation - Overview



One-hot encoded vectors

- A vector of ones and zeros
- Vector length is determined by the size of the vocabulary
- Vocabulary - the collection of unique words in the dataset

I	1	0	0	0	0
like	0	1	0	0	0
cats	0	0	1	0	0

One-hot encoded vectors

A mapping containing words and their corresponding indices

```
word2index = {"I":0, "like": 1, "cats": 2}
```

Converting words to IDs or indices

```
words = ["I", "like", "cats"]
word_ids = [word2index[w] for w in words]
print(word_ids)
```

```
[0, 1, 2]
```

One-hot encoded vectors

One-hot encoding **without** specifying output vector length

```
onehot_1 = to_categorical(word_ids)
print([(w,ohe.tolist()) for w,ohe in zip(words, onehot_1)])
```

```
[('I', [1.0, 0.0, 0.0]), ('like', [0.0, 1.0, 0.0]), ('cats', [0.0, 0.0, 1.0])]
```

One-hot encoding **with** specifying output vector length

```
onehot_2 = to_categorical(word_ids, num_classes=5)
print([(w,ohe.tolist()) for w,ohe in zip(words, onehot_2)])
```

```
[('I', [1.0, 0.0, 0.0, 0.0, 0.0]), ('like', [0.0, 1.0, 0.0, 0.0, 0.0]),
('cats', [0.0, 0.0, 1.0, 0.0, 0.0])]
```

Let's practice!

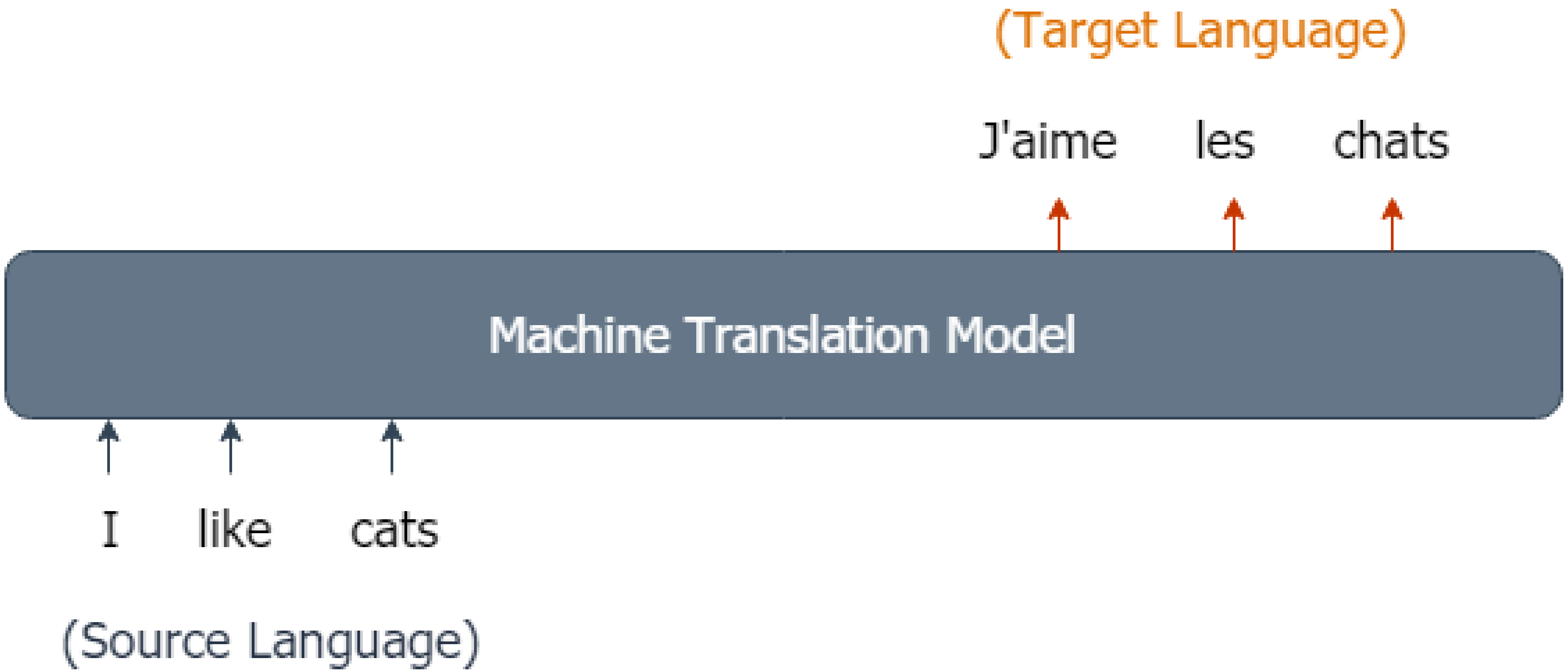
MACHINE TRANSLATION IN PYTHON

Encoder decoder architecture

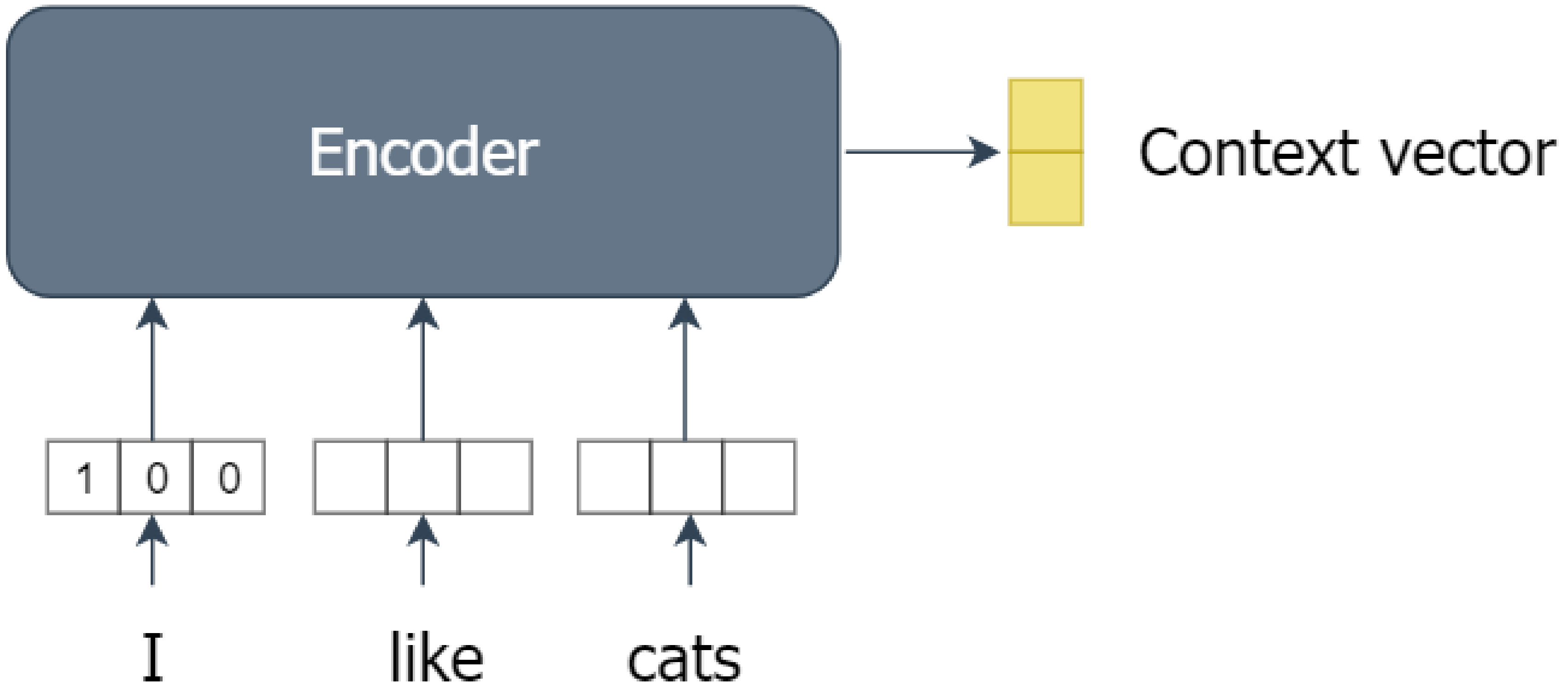
MACHINE TRANSLATION IN PYTHON

Thushan Ganegedara
Data Scientist and Author

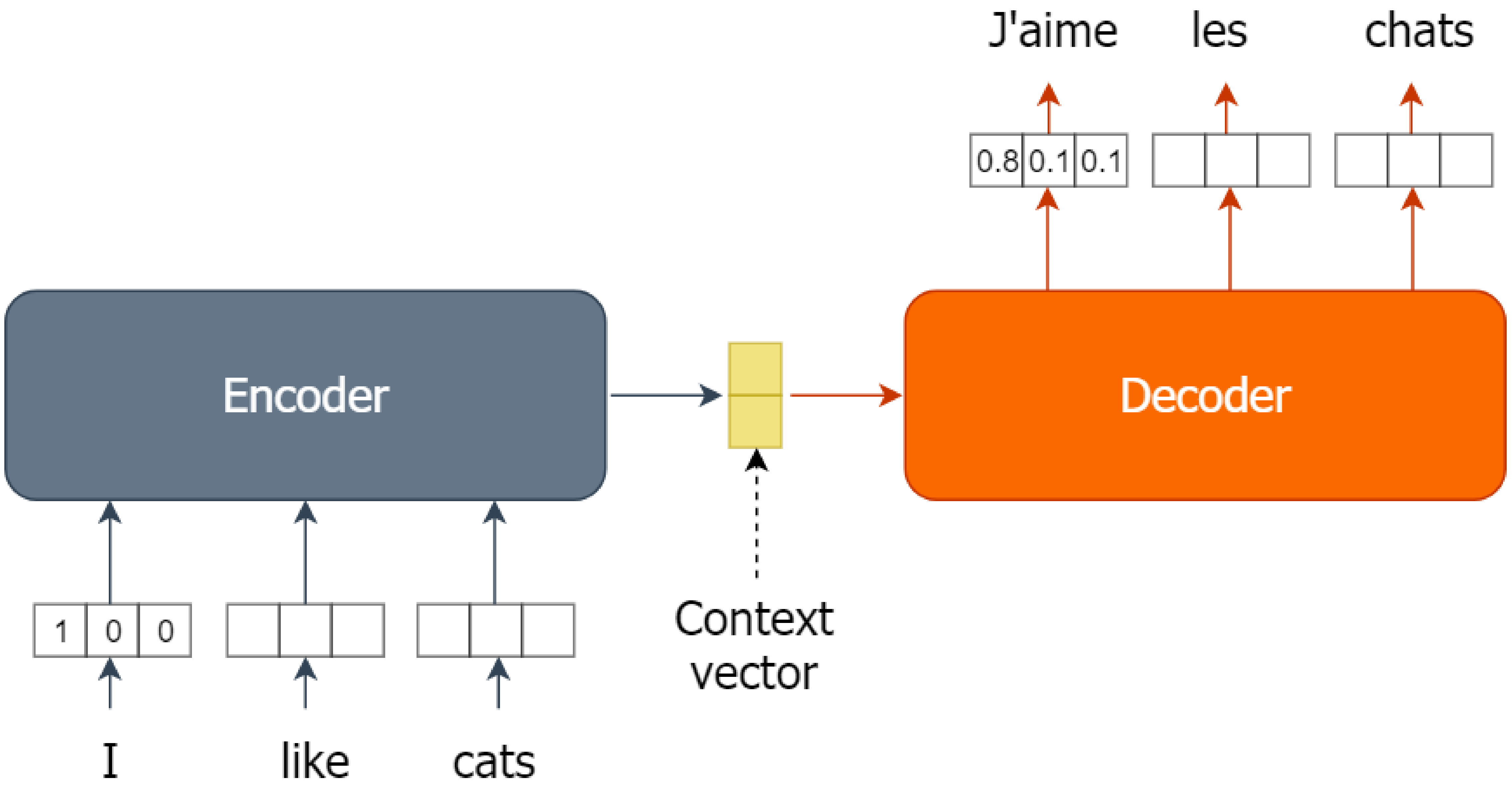
Encoder decoder model



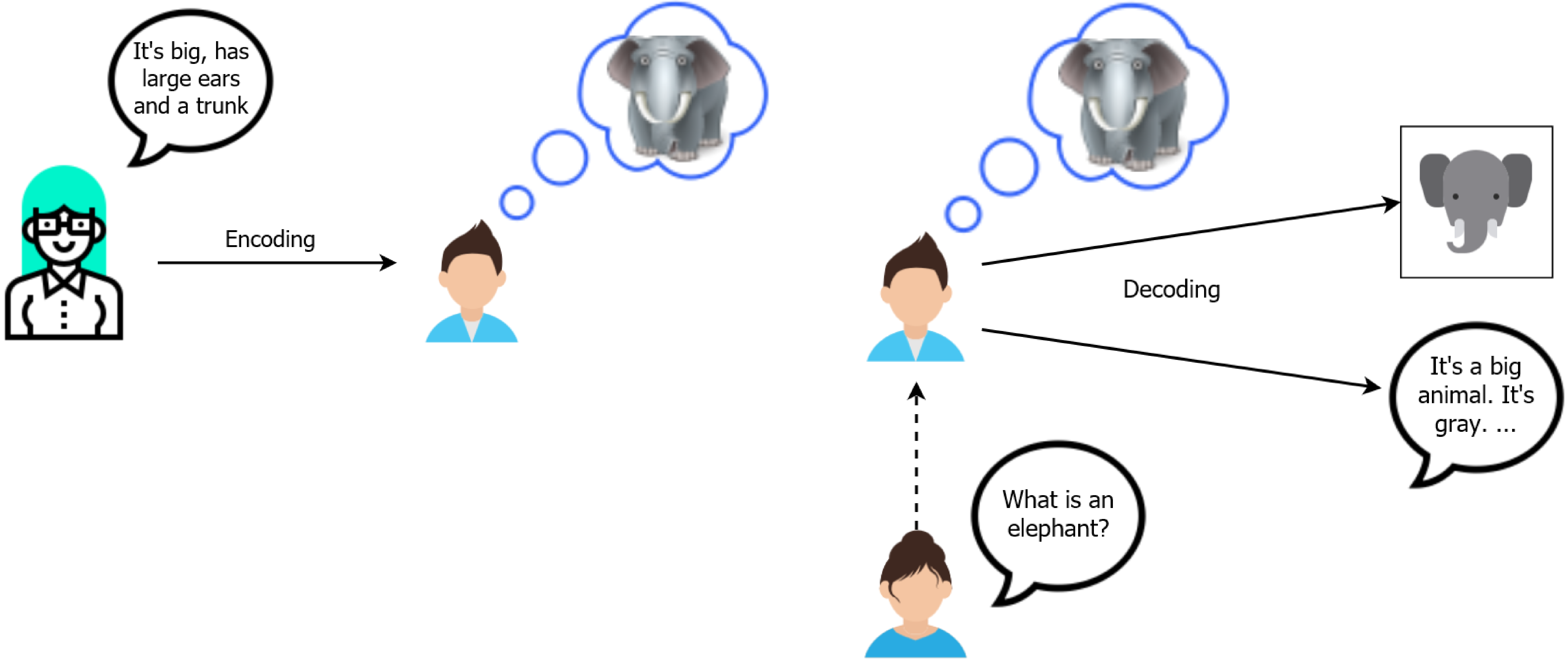
Encoder



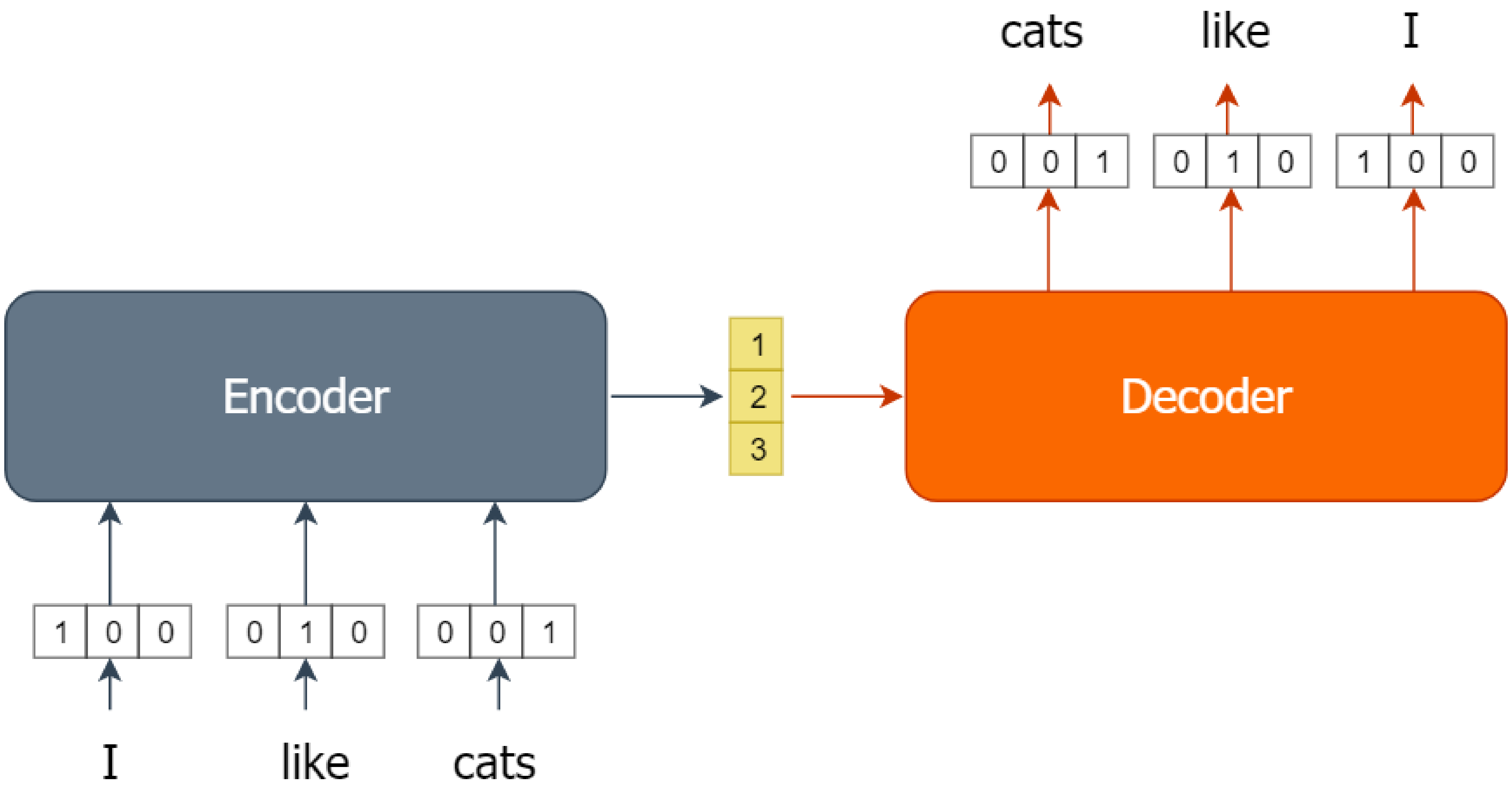
Encoder and Decoder



Analogy: Encoder decoder architecture



Reversing sentences - encoder decoder model



Writing the encoder

```
def words2onehot(word_list, word2index):
  word_ids = [word2index[w] for w in word_list]
  onehot = to_categorical(word_ids, 3)
  return onehot
```

```
def encoder(onehot):
  word_ids = np.argmax(onehot, axis=1)
  return word_ids
```

Writing the encoder

```
onehot = words2onehot(["I", "like", "cats"], word2index)
context = encoder(onehot)
print(context)
```

```
[0, 1, 2]
```

Writing the decoder

- Decoder: Word IDs ? Reverse the IDs ? one-hot vectors

```
def decoder(context_vector):
  word_ids_rev = context_vector[::-1]
  onehot_rev = to_categorical(word_ids_rev, 3)
  return onehot_rev
```

  - Helper function: convert one-hot vectors to human readable words

```
def onehot2words(onehot, index2word):
  ids = np.argmax(onehot, axis=1)
  return [index2word[id] for id in ids]
```

Writing the decoder

```
onehot_rev = decoder(context)
reversed_words = onehot2words(onehot_rev, index2word)

print(reversed_words)
```

```
['cats', 'like', 'I']
```

Let's practice!

MACHINE TRANSLATION IN PYTHON

Understanding sequential models

MACHINE TRANSLATION IN PYTHON

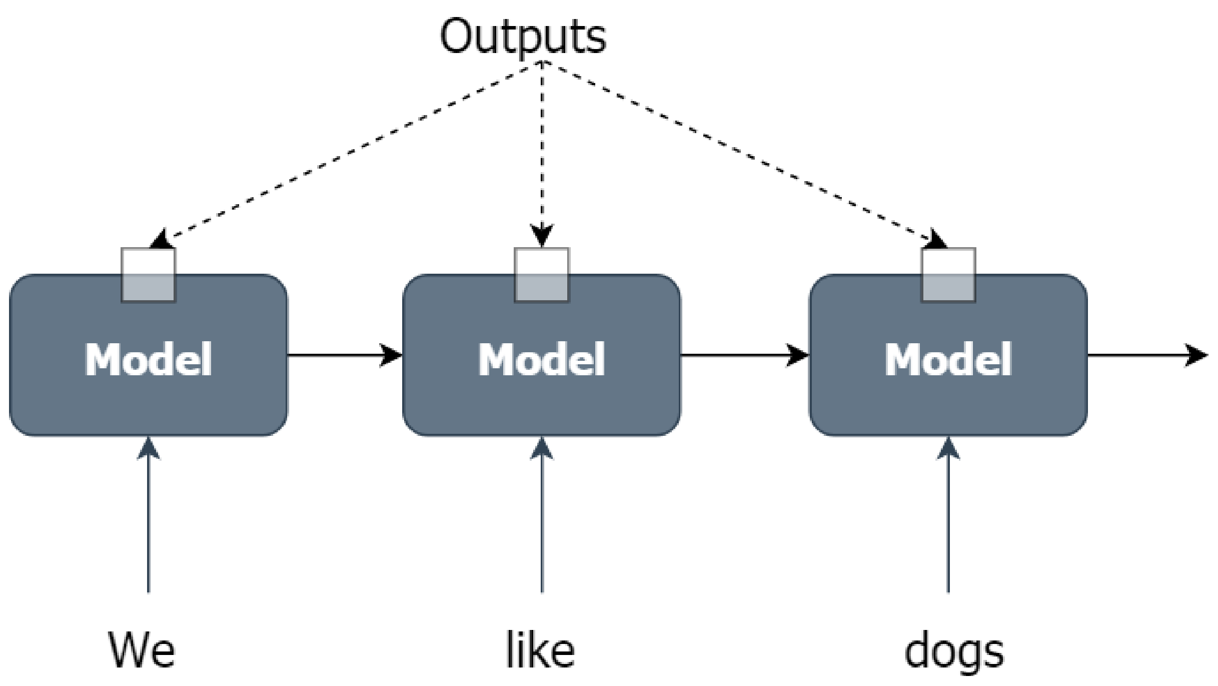
Thushan Ganegedara
Data Scientist and Author

Time series inputs and sequential models

- A sentence is a time series input
  - Current word is affected by previous words
  - E.g. He went to the pool for a ....
- The encoder/decoder uses a machine learning model
  - Models that can learn from time-series inputs
  - Models are called **sequential models**

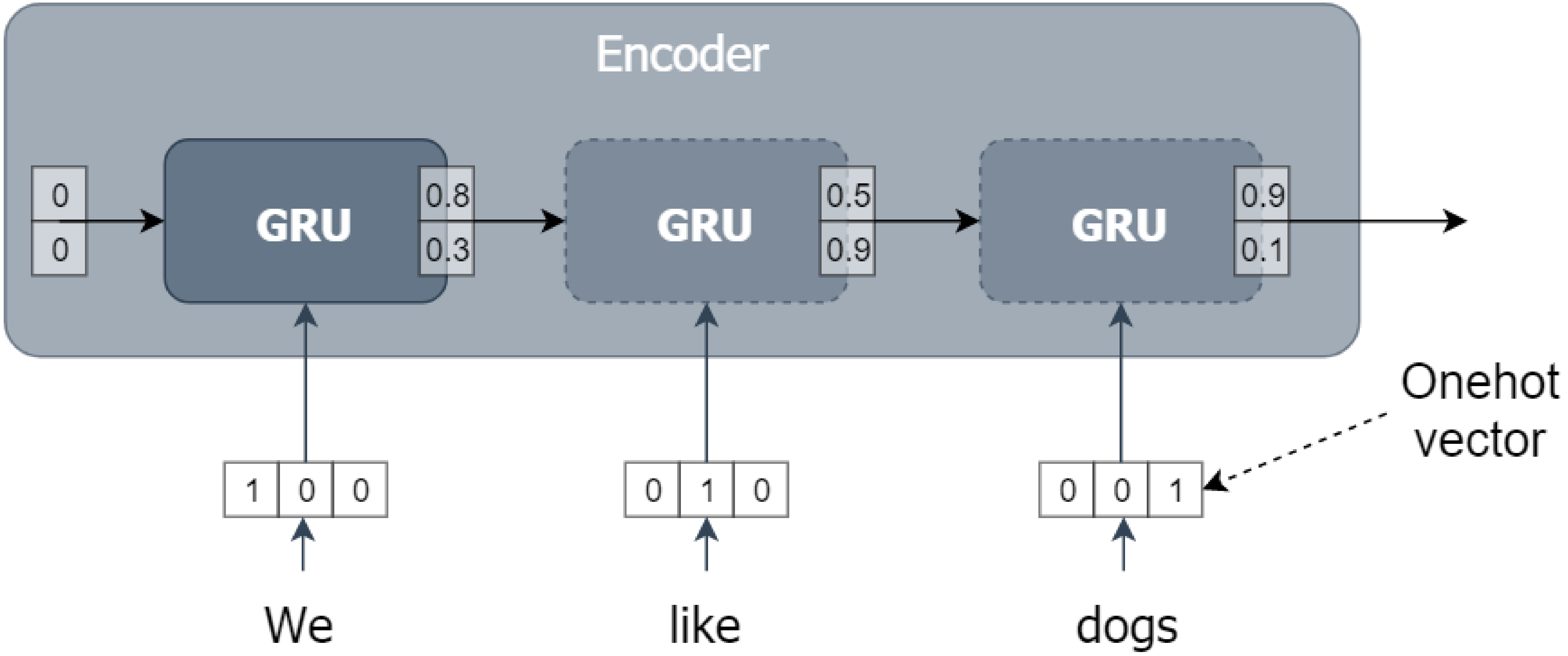
Sequential models

- Sequential models
  - Moves through the input while producing an output at each time step



Encoder as a sequential model

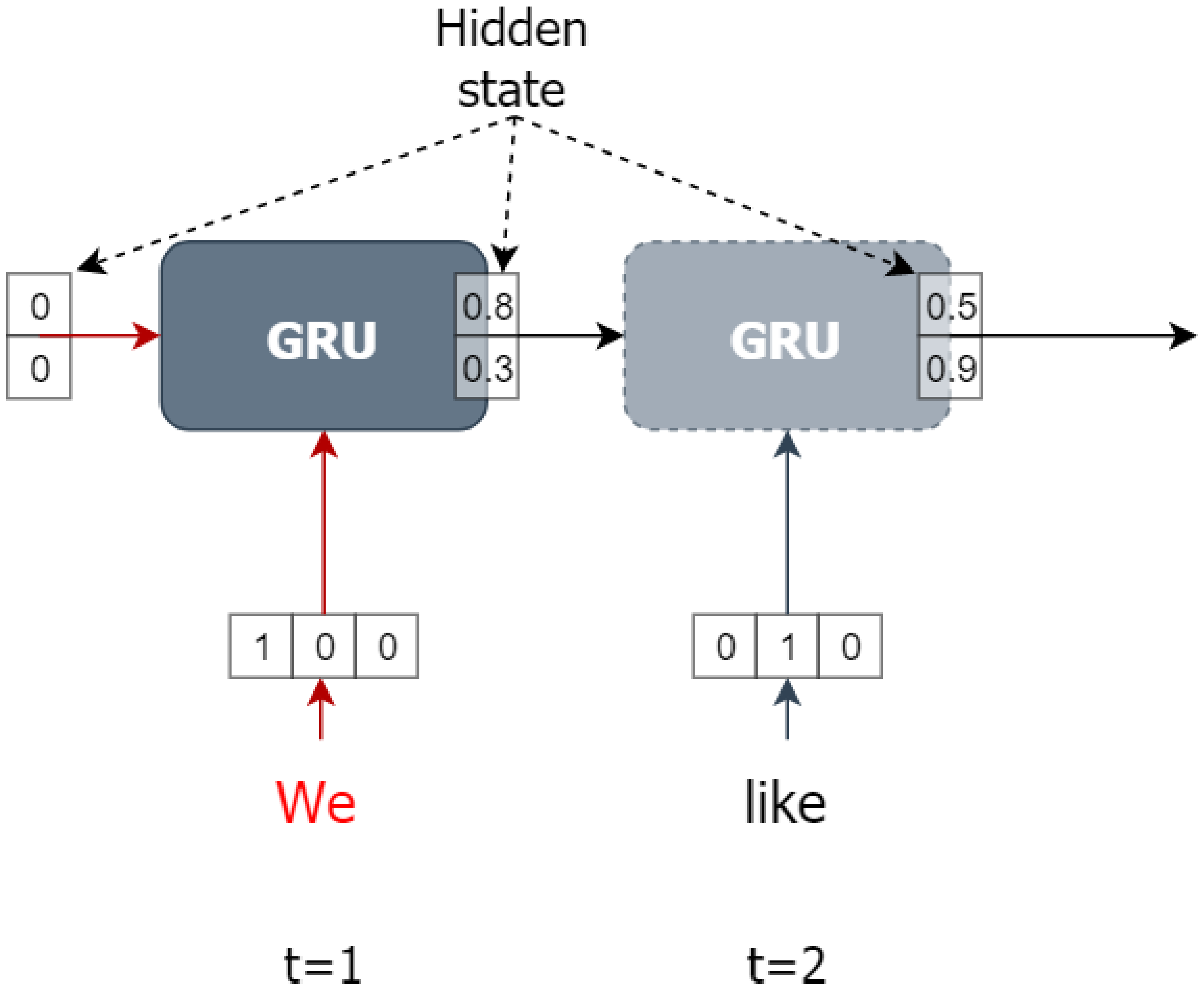
- GRU - Gated Recurrent Unit



Introduction to the GRU layer

At time step 1, the GRU layer,

- Consumes the input "We"
- Consumes the initial state (0,0)
- Outputs the new state (0.8, 0.3)

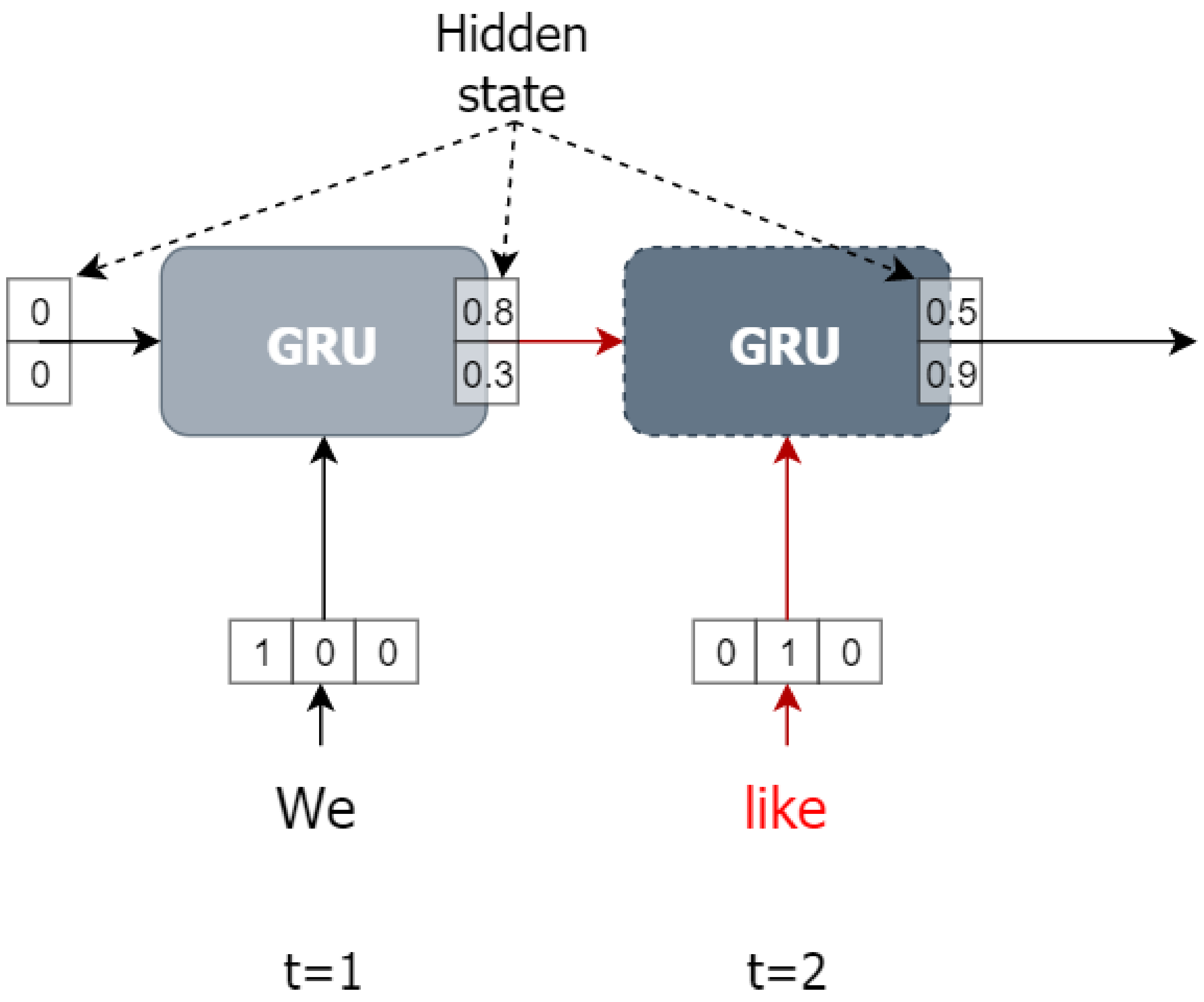


Introduction to GRU layer

At time step 2, the GRU layer,

- Consumes the input "like"
- Consumes the initial state (0.8,0.3)
- Outputs the new state (0.5, 0.9)

The hidden state represents "memory" of what the model has seen

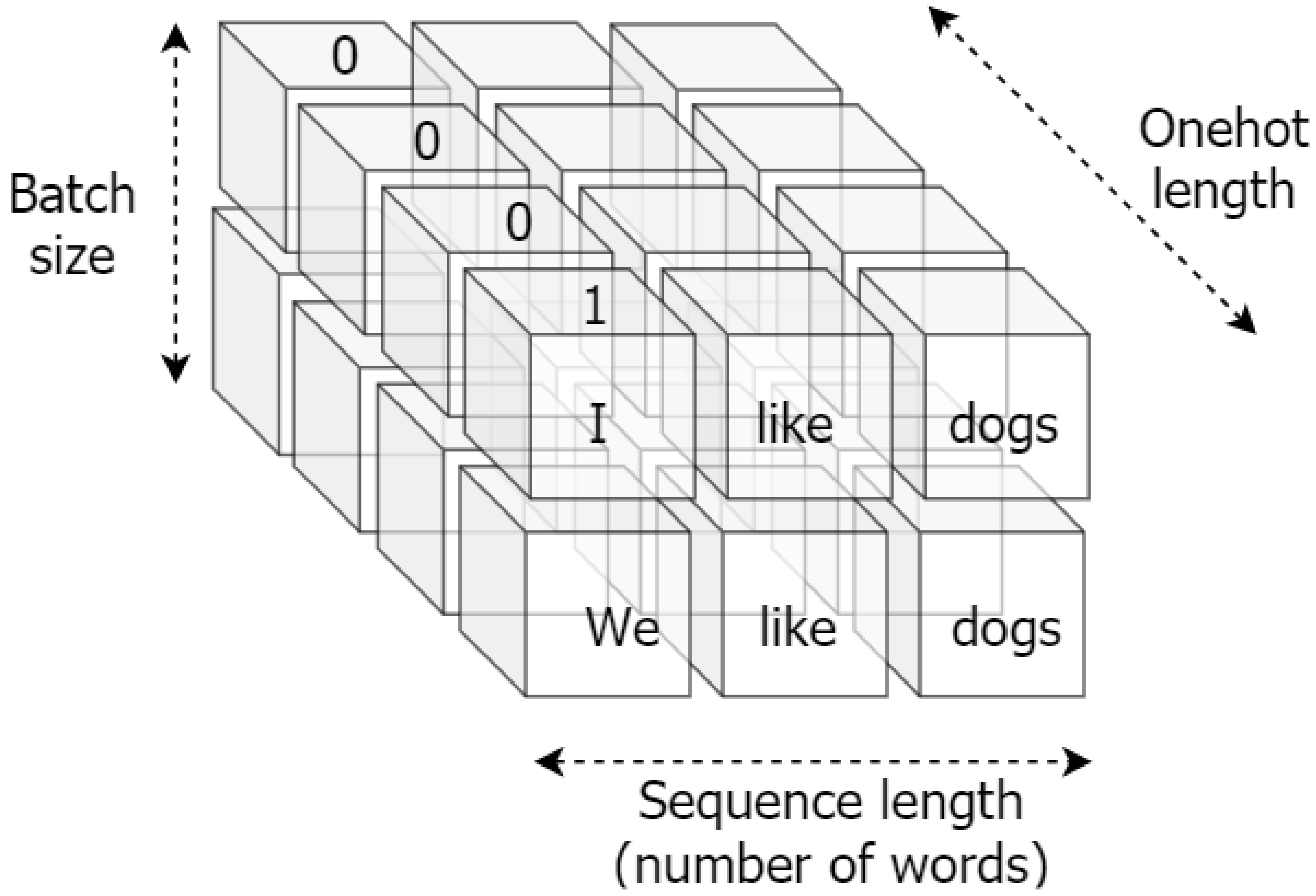


Keras (Functional API) refresher

- Keras has two important objects: `Layer` and `Model` objects.
- Input layer
  - `inp = keras.layers.Input(shape=(...))`
- Hidden layer
  - `layer = keras.layers.GRU(...)`
- Output
  - `out = layer(inp)`
- Model
  - `model = Model(inputs=inp, outputs=out)`

Understanding the shape of the data

- Sequential data is 3-dimensional
  - Batch dimension (e.g. batch = groups of sentences)
  - Time dimension - sequence length
  - Input dimension (e.g. onehot vector length)
- GRU model input shape
  - (Batch, Time, Input)
  - (batch size, sequence length, onehot length)



Implementing GRUs with Keras

Defining Keras layers

```
inp = keras.layers.Input(batch_shape=(2,3,4))
gru_out = keras.layers.GRU(10)(inp)
```

Defining a Keras model

```
model = keras.models.Model(inputs=inp, outputs=gru_out)
```

Implementing GRUs with Keras

Predicting with the Keras model

```
x = np.random.normal(size=(2,3,4))
y = model.predict(x)
print("shape (y) =", y.shape, "\ny = \n", y)
```

```
shape (y) = (2, 10)
y =
[[ 0.2576233   0.01215531  ... -0.32517594  0.4483121 ],
 [ 0.54189587 -0.63834655  ... -0.4339783   0.4043917 ]]
```

Implementing GRUs with Keras

A GRU that takes arbitrary number of samples in a batch

```
inp = keras.layers.Input(shape=(3,4))
gru_out = keras.layers.GRU(10)(inp)
model = keras.models.Model(inputs=inp, outputs=gru_out)
```

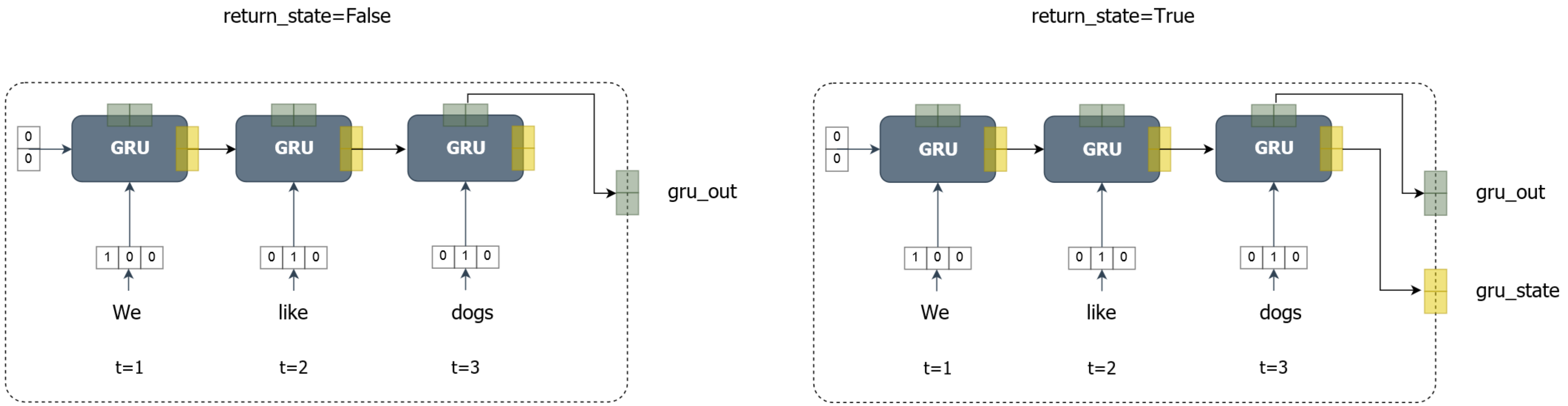
```
x = np.random.normal(size=(5,3,4))
y = model.predict(x)
print("y = \n", y)
```

```
y =
 [[-1.3941444e-02 -3.3123985e-02 ... 6.5081201e-02  1.1245312e-01]
 [ 1.1409521e-03  3.6983326e-01 ... -3.4610277e-01 -3.4792548e-01]
 [ 2.5911796e-01 -3.9517123e-01 ... 5.8505309e-01  3.6908010e-01]
 [-2.8727052e-01 -5.1150680e-02 ... -1.9637148e-01 -1.5587148e-01]
 [ 3.1303680e-01  2.3338445e-01 ... 9.1499090e-04 -2.0590121e-01]]
```

GRU layer's return_state argument

```
inp = keras.layers.Input(batch_shape=(2,3,4))
gru_out2, gru_state = keras.layers.GRU(10, return_state=True)(inp)
print("gru_out2.shape = ", gru_out2.shape)
print("gru_state.shape = ", gru_state.shape)
```

```
gru_out2.shape =  (2, 10)
gru_state.shape =  (2, 10)
```

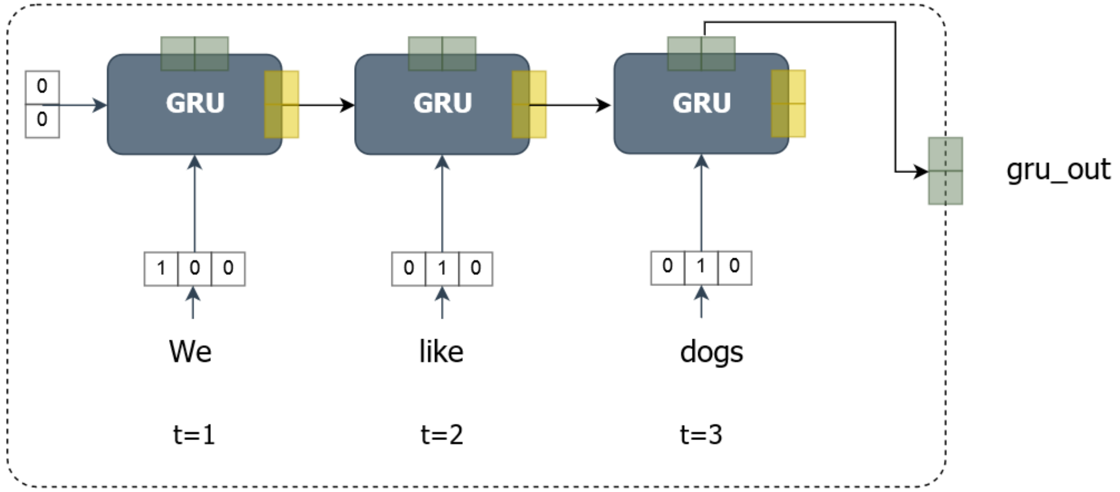


GRU layer's return_sequences argument

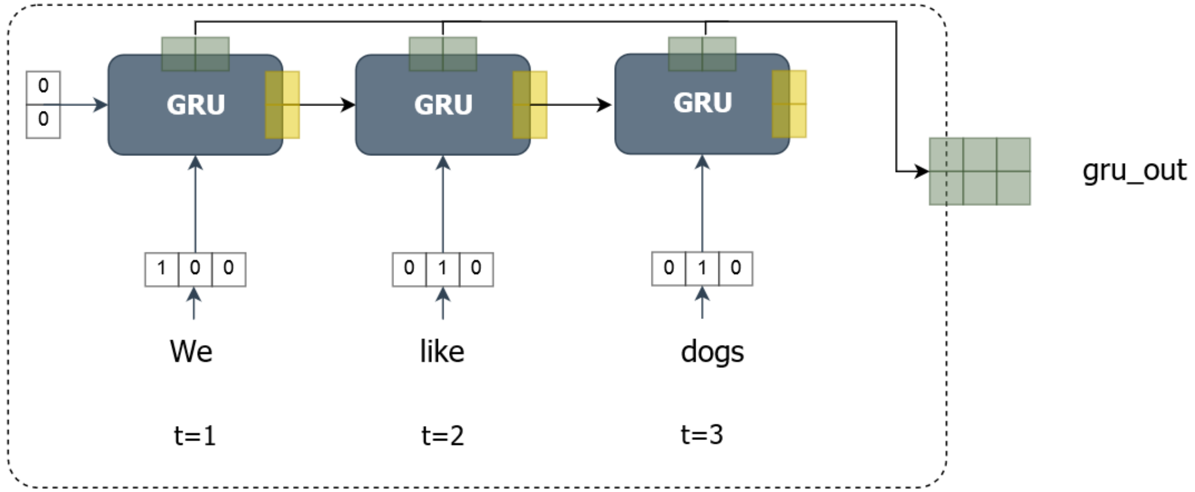
```
inp = keras.layers.Input(batch_shape=(2,3,4))
gru_out3 = keras.layers.GRU(10, return_sequences=True)(inp)
print("gru_out3.shape = ", gru_out2.shape)
```

```
gru_out3.shape =  (2, 3, 10)
```

return_sequences=False



return_sequences=True



Let's practice!

MACHINE TRANSLATION IN PYTHON