Implementing the encoder

MACHINE TRANSLATION IN PYTHON

Thushan Ganegedara
Data Scientist and Author

Understanding the data

Printing some data in the dataset

```
for en_sent, fr_sent in zip(en_text[:3], fr_text[:3]):
  print("English: ", en_sent)
  print("\tFrench: ", fr_sent)
```

```
English:  new jersey is sometimes quiet during autumn , and it is snowy in april .
    French:  new jersey est parfois calme pendant l' automne , et il est neigeux en avril .

English:  the united states is usually chilly during july , and it is usually freezing in november .
    French:  les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .

English:  california is usually quiet during march , and it is usually hot in june .
    French:  california est généralement calme en mars , et il est généralement chaud en juin .
```

Tokenizing the sentences

Tokenization

- The process of breaking a sentence/phrase to individual tokens (e.g. words)

Tokenizing words in the sentences

```
first_sent = en_text[0]
print("First sentence: ", first_sent)
first_words = first_sent.split(" ")
print("\tWords: ", first_words)
```

```
First sentence:  new jersey is sometimes quiet during autumn , and it is snowy in april .
    Words:  ['new', 'jersey', 'is', 'sometimes', 'quiet', 'during', 'autumn', ',',
             'and', 'it', 'is', 'snowy', 'in', 'april', '.']
```

Computing the length of sentences

Computing average length of a sentence and the size of the vocabulary (English)

```
sent_lengths = [len(en_sent.split(" ")) for en_sent in en_text]
mean_length = np.mean(sent_lengths)
print('(English) Mean sentence length: ', mean_length)
```

```
(English) Mean sentence length:  13.20662
```

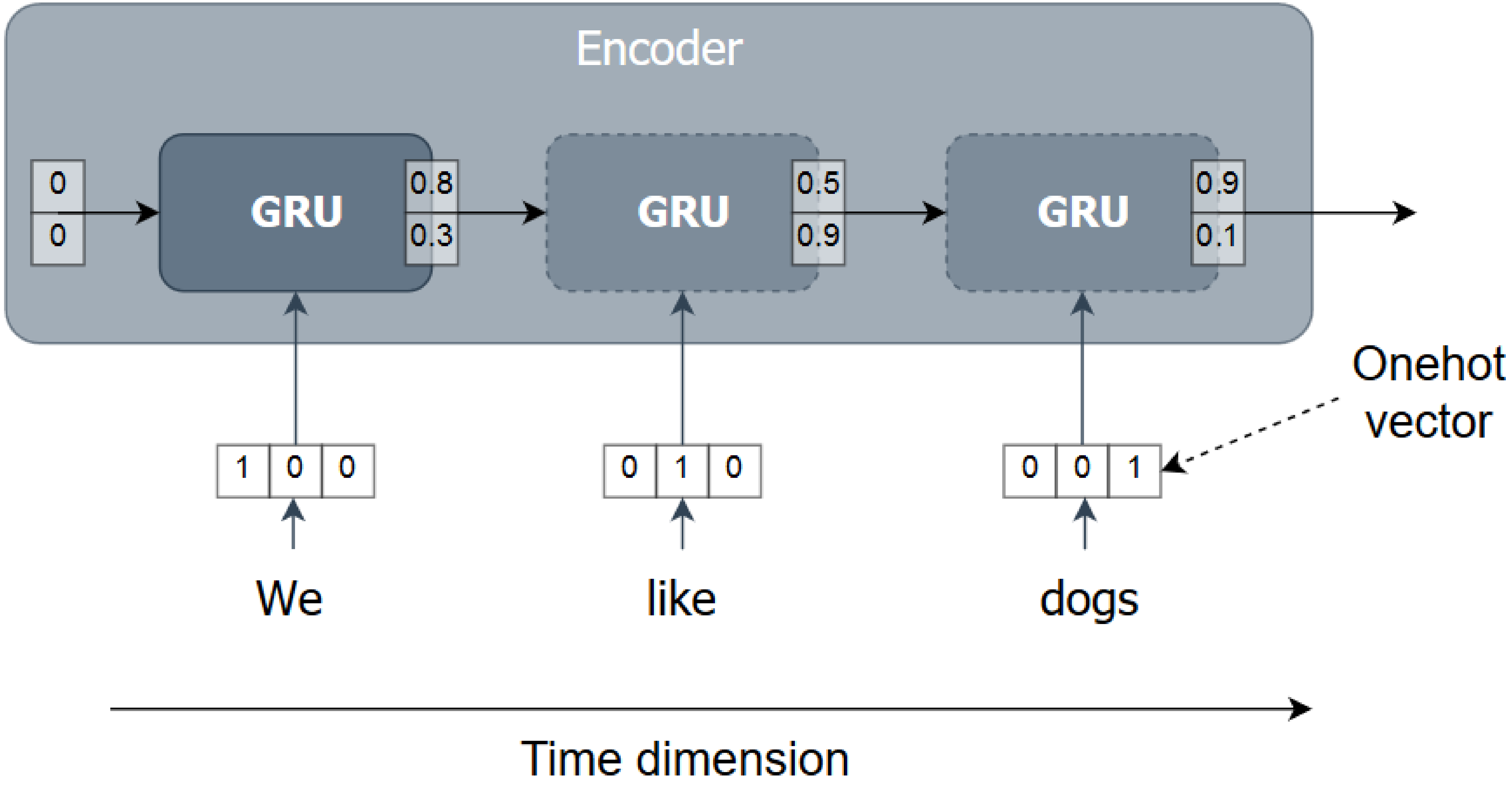
Computing the size of the vocabulary

```
all_words = []
for sent in en_text:
    all_words.extend(sent.split(" "))
vocab_size = len(set(all_words))
print("(English) Vocabulary size: ", vocab_size)
```

- A `set` object only contains unique items and no duplicates

```
(English) Vocabulary size:  228
```

The encoder


Encoder
0
0
GRU
0.8
0.3
GRU
0.5
0.9
GRU
0.9
0.1
Onehot vector
1 0 0
0 1 0
0 0 1
We
like
dogs
Time dimension

Implementing the encoder with Keras

- Input layer

```
en_inputs = Input(shape=(en_len, en_vocab))
```

- GRU layer

```
en_gru = GRU(hsize, return_state=True)
en_out, en_state = en_gru(en_inputs)
```

- Keras model

```
encoder = Model(inputs=en_inputs, outputs=en_state)
```

Understanding the Keras model summary

```
print(encoder.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 15, 150)           0
_________________________________________________________________
gru (GRU)                    [(None, 48), (None, 48)]  28656
=================================================================
Total params: 28,656
Trainable params: 28,656
Non-trainable params: 0
_________________________________________________________________
```

Let's practice!

MACHINE TRANSLATION IN PYTHON

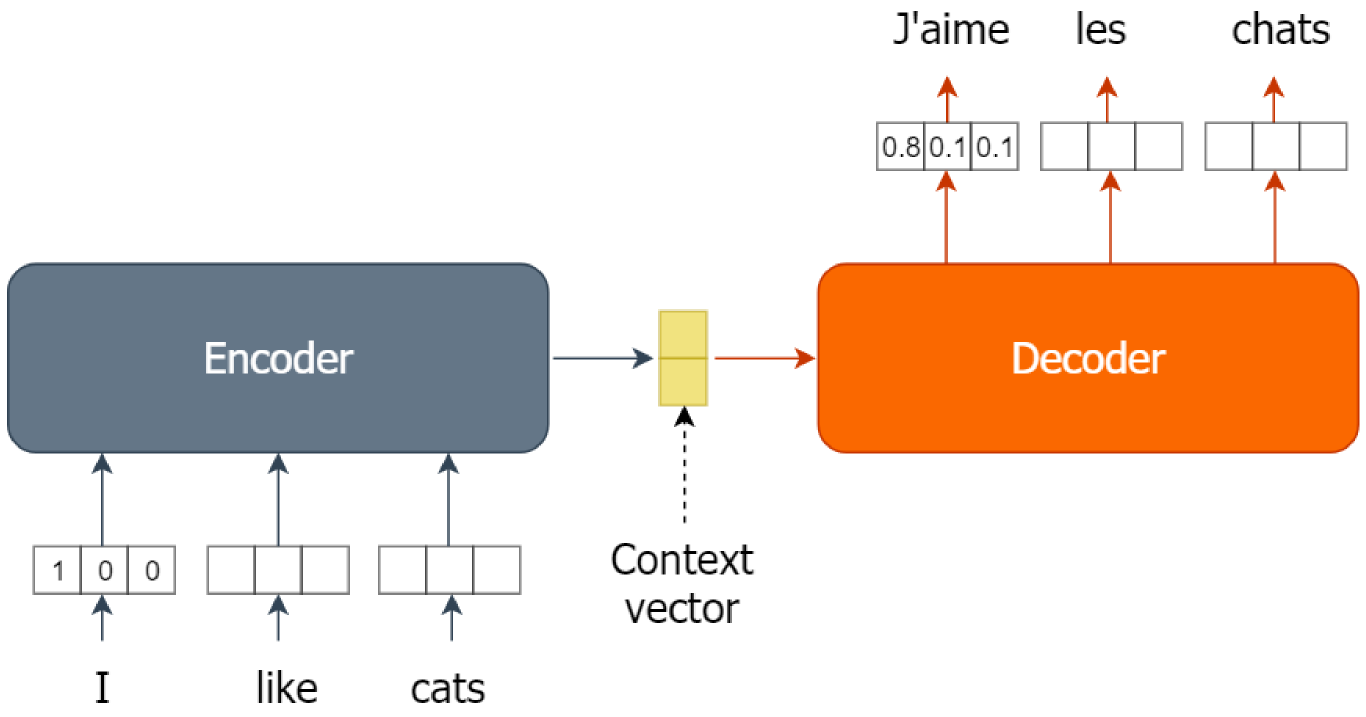
Defining the decoder

MACHINE TRANSLATION IN PYTHON

Thushan Ganegedara
Data Scientist and Author

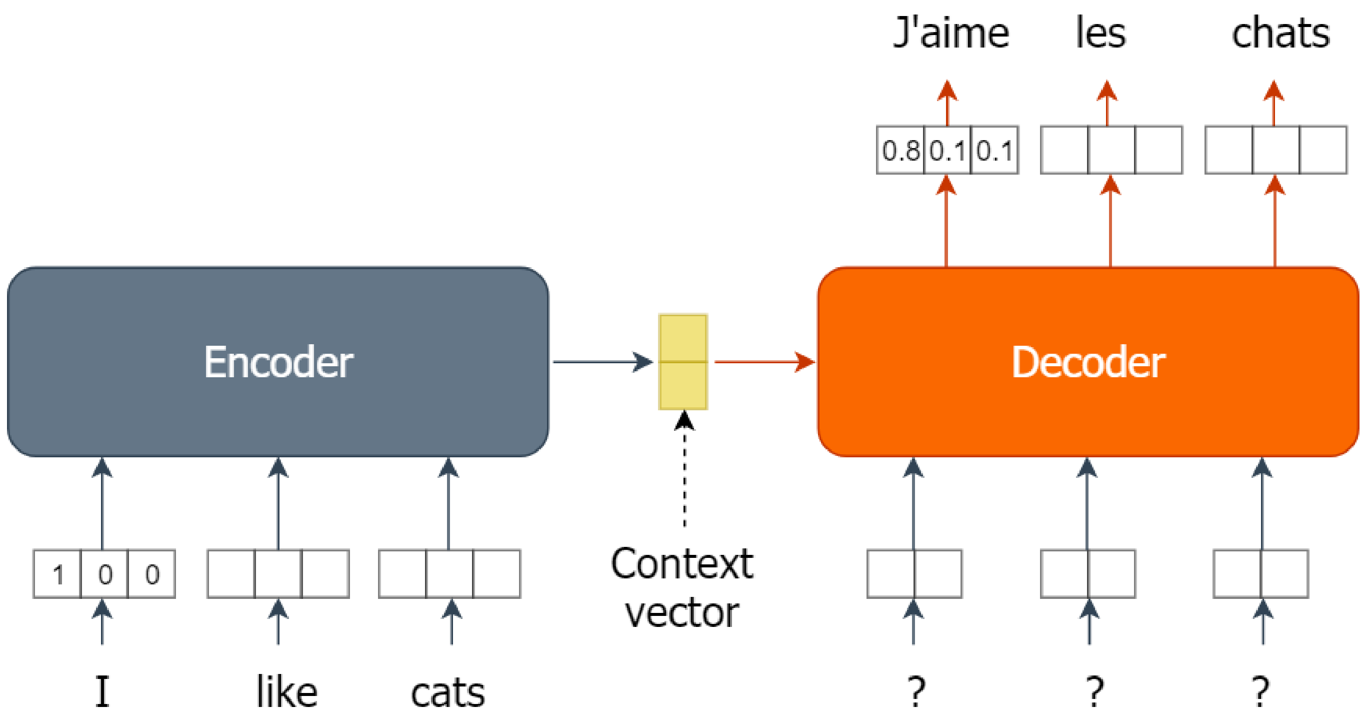
Encoder-decoder model

- Encoder consumes English words one-by-one
- Finally produces the context vector
- Decoder takes the context vector as the initial state
- Decoder produces French words one-by-one



Input of the decoder

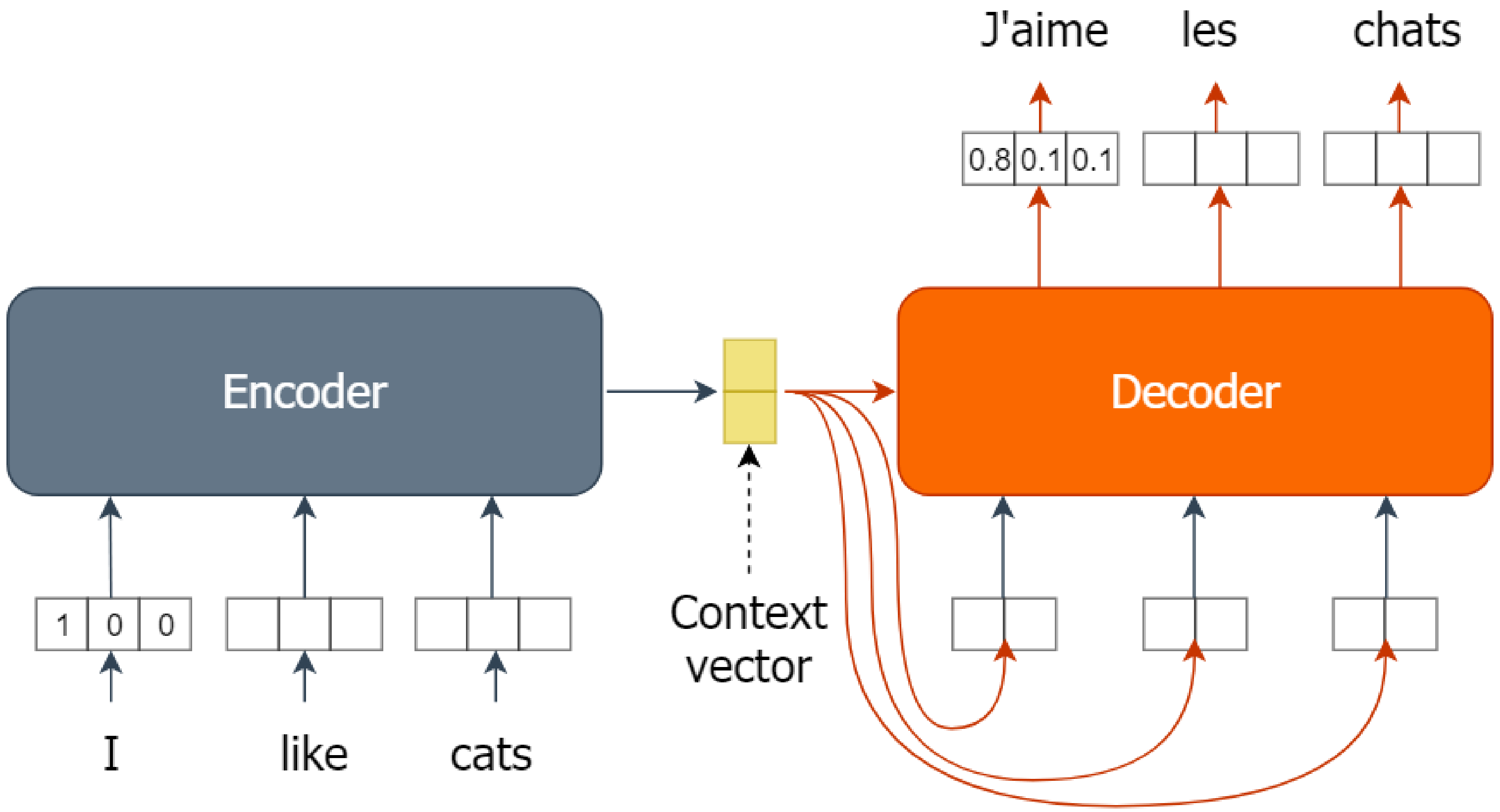
- Decoder is implemented using a Keras GRU layer
- GRU model require two inputs
  - A time-series input (???)
  - A hidden state



Input of the decoder

Repeat the context vector from the encoder N-many times

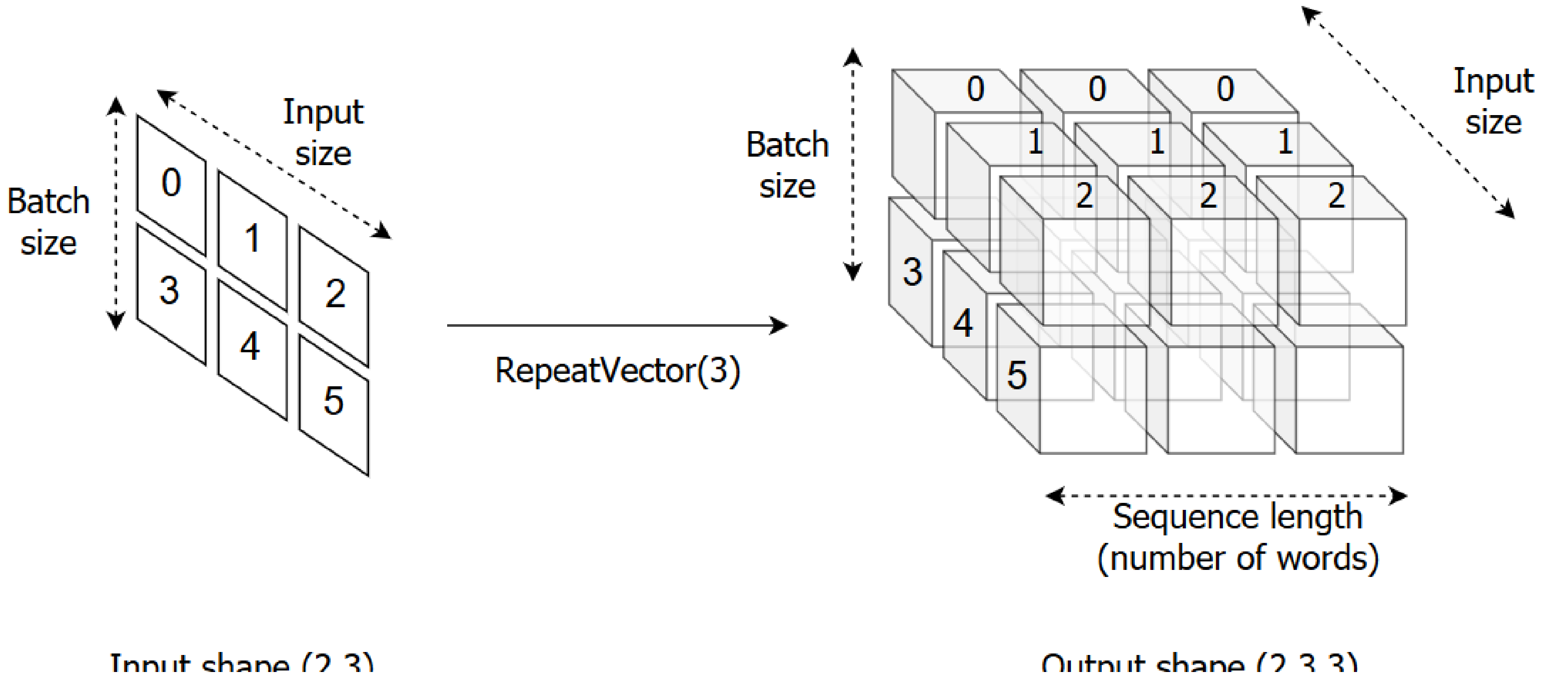
- To produce a french sentence of 10 words, you repeat the context vector 10 times.



Understanding the RepeatVector layer

RepeatVector layer:

- Takes one argument which defines the sequence length of the required output
- Takes in an input of (batch_size, input size) (e.g. Input of size `2 x 3`)
- Outputs data having shape (batch_size, sequence length, input size) (e.g. Output of size `2 x 3 x 3`)



Defining a RepeatVector layer

```
from tensorflow.keras.layers import RepeatVector
rep = RepeatVector(5)
```

```
r_inp = Input(shape=(3,))
r_out = rep(r_inp)
```

```
repeat_model = Model(inputs=r_inp, outputs=r_out)
```

- Note that the following two are equivalent

```
rep = RepeatVector(5)
r_out = rep(r_inp)
```

```
r_out = RepeatVector(5)(r_inp)
```

Predicting with the model

Predicting with the model

```
x = np.array([[0,1,2],[3,4,5]])
y = repeat_model.predict(x)
print('x.shape = ',x.shape,'\ny.shape = ',y.shape)
```

```
x.shape =  (2, 3)
y.shape =  (2, 5, 3)
```

Implementing the decoder

Defining the decoder

```
de_inputs = RepeatVector(fr_len)(en_state)
```

```
decoder_gru = GRU(hsize, return_sequences=True)
```

Fixing the initial state of the decoder

```
gru_outputs = decoder_gru(de_inputs, initial_state=en_state)
```

Defining the model

```
enc_dec = Model(inputs=en_inputs, outputs=gru_outputs)
```

Let's practice!

MACHINE TRANSLATION IN PYTHON

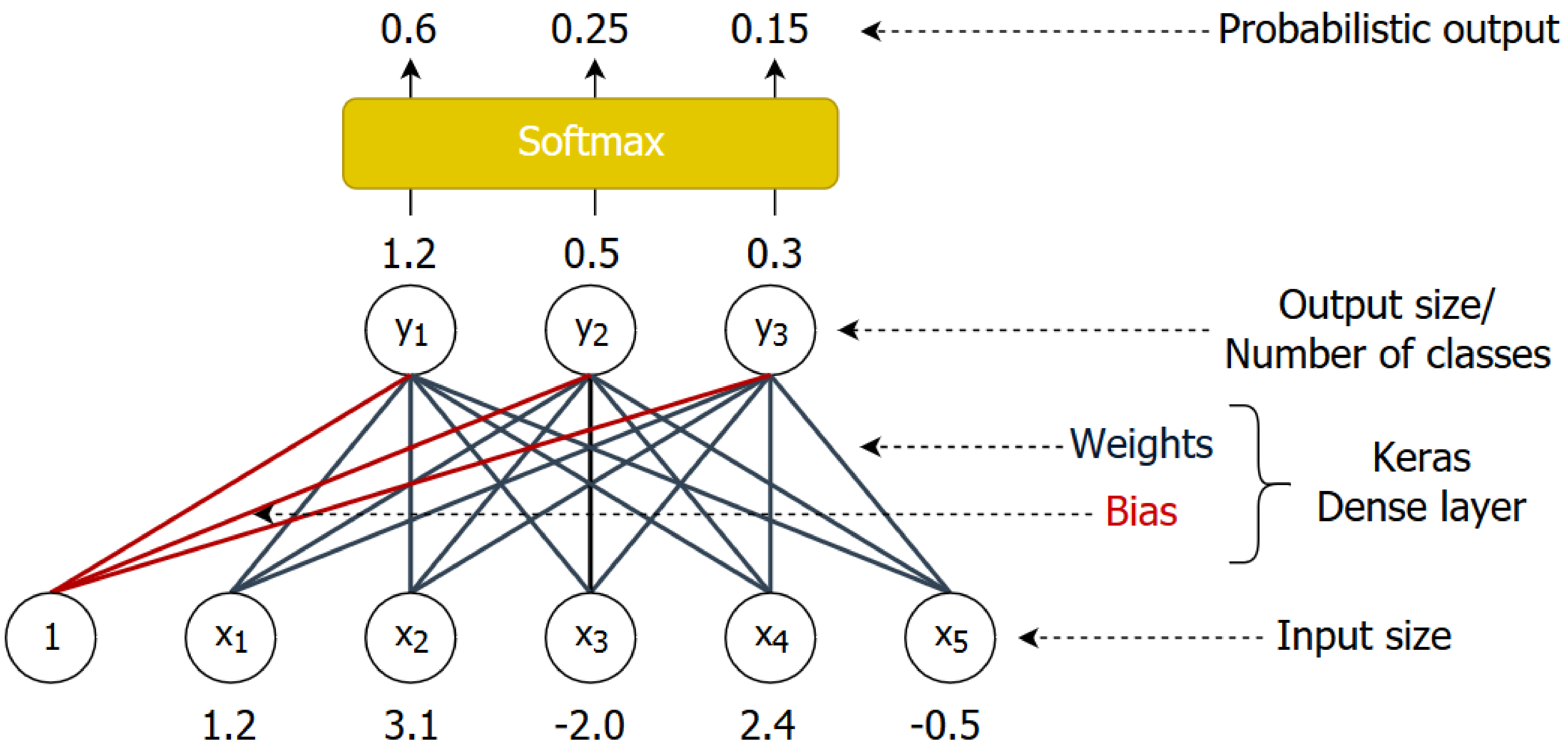
Dense and TimeDistributed layers

MACHINE TRANSLATION IN PYTHON

Thushan Ganegedara
Data Scientist and Author

Introduction to the Dense layer

- Takes an input vector and converts to a probabilistic prediction.
  - y = Weights.x + Bias



Understanding the Dense layer

Defining and using a Dense layer

```
dense = Dense(3, activation='softmax')
```

```
inp = Input(shape=(3,))
pred = dense(inp)
model = Model(inputs=inp, outputs=pred)
```

Defining a Dense layer with custom initialization

```
from tensorflow.keras.initializers import RandomNormal
init = RandomNormal(mean=0.0, stddev=0.05, seed=6000)
dense = Dense(3, activation='softmax',
              kernel_initializer=init, bias_initializer=init)
```

Inputs and outputs of the Dense layer

- Dense softmax layer
  - Takes a `(batch size, input size)` array
    - e.g. `x = [[1, 6, 8], [8, 9, 10]] # a 2x3 array`
  - Produces a `(batch size, num classes)` array
    - e.g. Number of classes = 4
    - e.g. `y = [[0.1, 0.3, 0.4, 0.2], [0.2, 0.5, 0.1, 0.2]] # a 2x4 array`
  - Output for each sample is a probability distribution over the classes
    - Sums to 1 along columns
  - Can get the class for each sample using `np.argmax(y, axis=-1)`
    - e.g. `np.argmax(y,axis=-1)` produces `[2,1]`

Understanding the TimeDistributed layer

- Allows `Dense` layers to process time-series inputs

```
dense_time = TimeDistributed(Dense(3, activation='softmax'))
```

```
inp = Input(shape=(2, 3))
pred = dense_time(inp)
model = Model(inputs=inp, outputs=pred)
```

Inputs and outputs of the TimeDistributed layer

- Takes a `(batch size, sequence length, input size)` array

```
x = [[[1, 6], [8, 2], [1, 2]],
    [[8, 9], [10, 8], [1, 0]]] # a 2x3x2 array
```

- Produces a `(batch size, sequence length, num classes)` array
  - e.g. Number of classes = 3

```
y = [[[0.1, 0.5, 0.4], [0.8, 0.1, 0.1], [0.6, 0.2, 0.2]],
     [[0.2, 0.5, 0.3], [0.2, 0.5, 0.3], [0.2, 0.8, 0.0]]] # a 2x3x3 array
```

- Output for each sample is a probability distribution over the classes
- Can get the class for each sample using `np.argmax(y, axis=-1)`

Slicing data on time dimension

```
y = [[[0.1, 0.5, 0.4], [0.8, 0.1, 0.1], [0.6, 0.2, 0.2]],
     [[0.2, 0.5, 0.3], [0.2, 0.5, 0.3], [0.2, 0.8, 0.0]]] # a 2x3x3 array
classes = np.argmax(y, axis=-1) # a 2 x 3 array
```

Iterating through time-distributed data

```
for t in range(3):
  # Get the t-th time-dimension slice of y and classes
  for prob, c in zip(y[:,t,:], classes[:,t]):
     print("Prob: ", prob, ", Class: ", c)
```

```
Prob:  [0.1 0.5 0.4] , Class:  1
Prob:  [0.2 0.5 0.3] , Class:  1
Prob:  [0.8 0.1 0.1] , Class:  0
...
```

Let's practice!

MACHINE TRANSLATION IN PYTHON

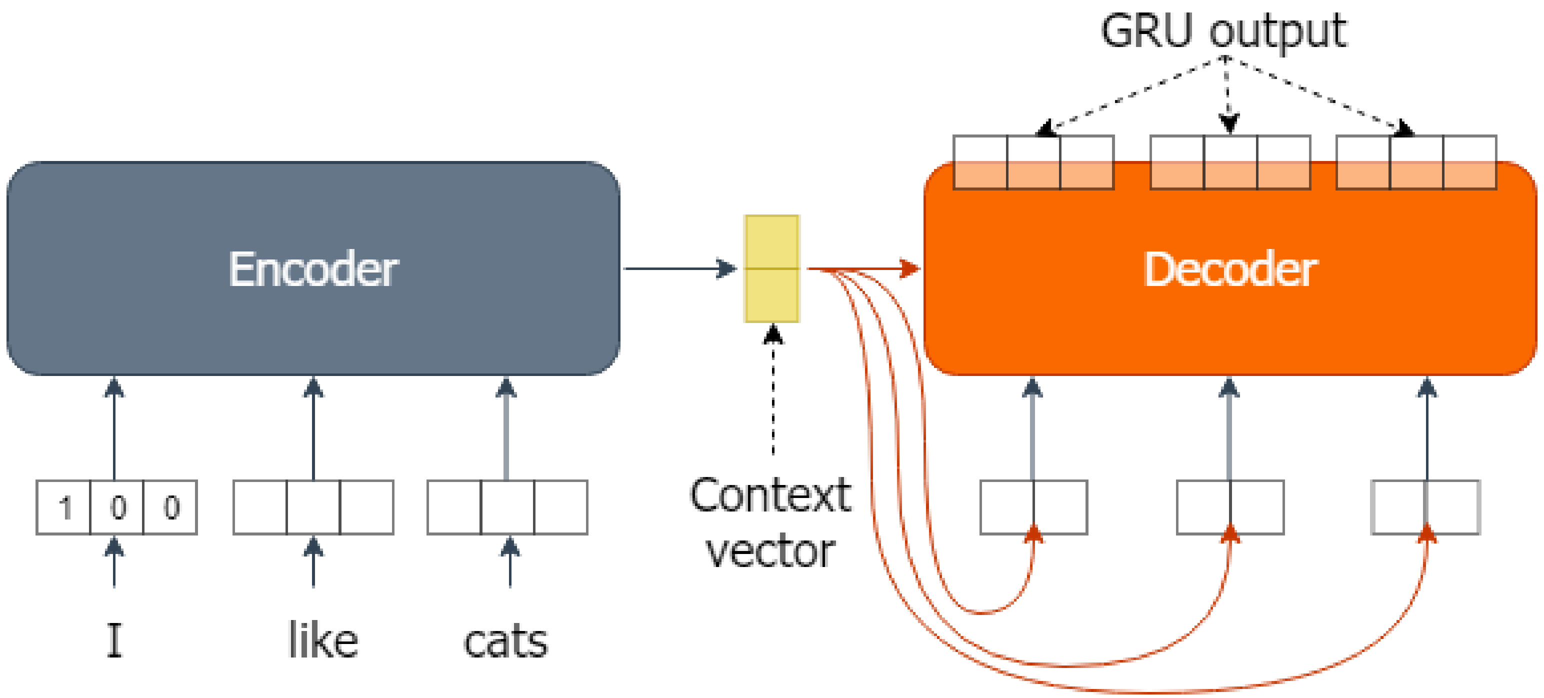
Implementing the full encoder decoder model

MACHINE TRANSLATION IN PYTHON

Thushan Ganegedara
Data Scientist and Author

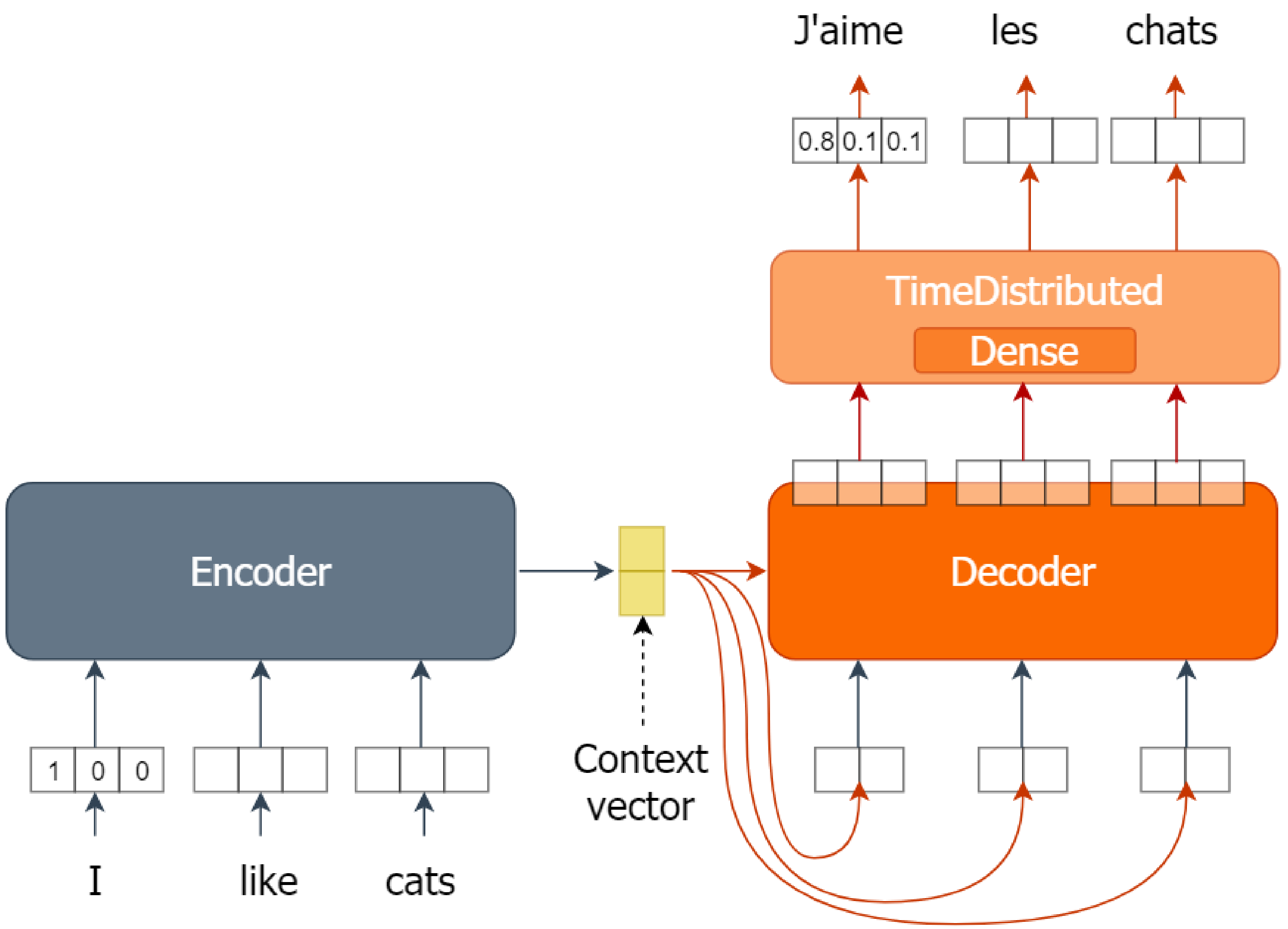
What you implemented so far

- The encoder consumes the English (i.e. source) input
- The encoder produces the context vector
- The decoder consumes a repeated set of context vectors
- The decoder outputs GRU output sequence



Top part of the decoder

- Implemented with `TimeDistributed` and `Dense` layers.



Implementing the full model

- Encoder

```
en_inputs = Input(shape=(en_len, en_vocab))
en_gru = GRU(hsize, return_state=True)
en_out, en_state = en_gru(en_inputs)
```

- Decoder

```
de_inputs = RepeatVector(fr_len)(en_state)
de_gru = GRU(hsize, return_sequences=True)
de_out = de_gru(de_inputs, initial_state=en_state)
```

Implementing the full model

- The softmax prediction layer

```
de_dense = keras.layers.Dense(fr_vocab, activation='softmax')
de_dense_time = keras.layers.TimeDistributed(de_dense)
de_pred = de_seq_dense(de_out)
```

Compiling the model

Defining the full model

```
nmt = keras.models.Model(inputs=en_inputs, outputs=de_pred)
```

Compiling the model

```
nmt.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

Let's practice!

MACHINE TRANSLATION IN PYTHON