# Part 1: Preprocessing the Data

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Introduction to data

- Data
  - `en_text` : A Python list of sentences, each sentence is a string of words separated by spaces.
  - `fr_text` : A Python list of sentences, each sentence is a string of words separated by spaces.
- 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 .
```

# Word tokenization

- Tokenization
  - Process of breaking a sentence/phrase to individual words/characters
  - E.g. `"I watched a movie last night, it was okay."` becomes,
  - `[I, watched, a, movie, last, night, it, was, okay]`
- Tokenization with Keras
  - Learns a mapping from word to a word ID using a given corpus.
  - Can be used to convert a given string to a sequence of IDs

```
from tensorflow.keras.preprocessing.text import Tokenizer
en_tok = Tokenizer()
```

# Fitting the Tokenizer

- Fitting the Tokenizer on data
  - Tokenizer needs to be fit on some data (i.e. sentences) to learn the word to word ID mapping.

```
en_tok = Tokenizer()
en_tok.fit_on_texts(en_text)
```

- Getting the word to ID mapping
  - Use the `Tokenizer` 's `word_index` attribute.

```
id = en_tok.word_index["january"] # => returns 51
```

- Getting the ID to word mapping

```
w = en_tok.index_word[51] # => returns 'january'
```

# Transforming sentences to sequences

```
seq = en_tok.texts_to_sequences(['she likes grapefruit , peaches , and lemons .'])
```

```
[[26, 70, 27, 73, 7, 74]]
```

# Limiting the size of the vocabulary

- You can limit the size of the vocabulary in a Keras `Tokenizer` .

```
tok = Tokenizer(num_words=50)
```

- Out-of-vocabulary (OOV) words
  - Rare words in the training corpus (i.e. collection of text).
  - Words that are not present in the training set.
- E.g.
  - `tok.fit_on_texts(["I drank milk"])`
  - `tok.texts_to_sequences(["I drank water"])`
  - The word `water` is a OOV word and will be ignored.

# Treating Out-of-Vocabulary words

- Defining a OOV token

```
tok = Tokenizer(num_words=50, oov_token='UNK')
```

- E.g.
  - `tok.fit_on_texts(["I drank milk"])`
  - `tok.texts_to_sequences(["I drank water"])`
  - The word `water` is a OOV word and will be replaced with `UNK` .
    - i.e. Keras will see "I drank UNK"

# Let's practice!

MACHINE TRANSLATION IN PYTHON

# Part 2: Preprocessing the text

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Adding special starting/ending tokens

The sentence:

```
'les états-unis est parfois occupé en janvier , et il est parfois chaud en novembre .'
```

becomes:

```
'sos les états-unis est parfois occupé en janvier , et il est parfois chaud en novembre . eos',
```

after adding special tokens

- `sos` - Start of a sentence/sequence
- `eos` - End of a sentence/sequence

# Padding the sentences

- Real world datasets never have the same number of words in all sentences
- Importing `pad_sequences`

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

- Converting sentences to sequences

```
sentences = [
  'new jersey is sometimes quiet during autumn .',
  'california is never rainy during july , but it is sometimes beautiful in february .'
]
seqs = en_tok.texts_to_sequences(sentences)
```

# Padding the sentences

```
preproc_text = pad_sequences(seqs, padding='post', truncating='post', maxlen=12)
for orig, padded in zip(seqs, preproc_text):
    print(orig, ' => ', padded)
```

First sentence gets five 0s padded to the end:

```
#  'new jersey is sometimes quiet during autumn .',
[18, 20, 2, 10, 32, 5, 46]  =>  [18 20  2 10 32  5 46  0  0  0  0  0]
```

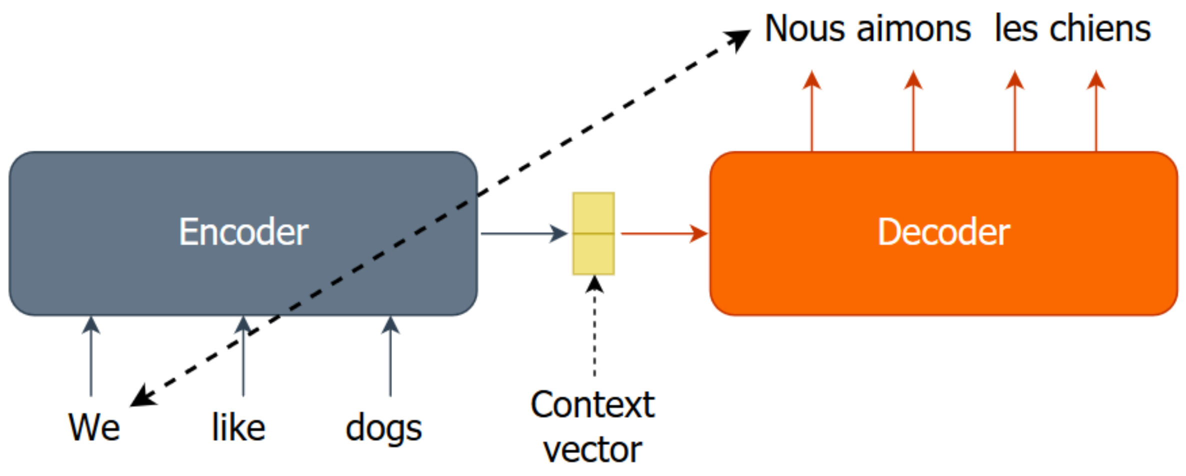
Second sentence gets one word truncated at the end:

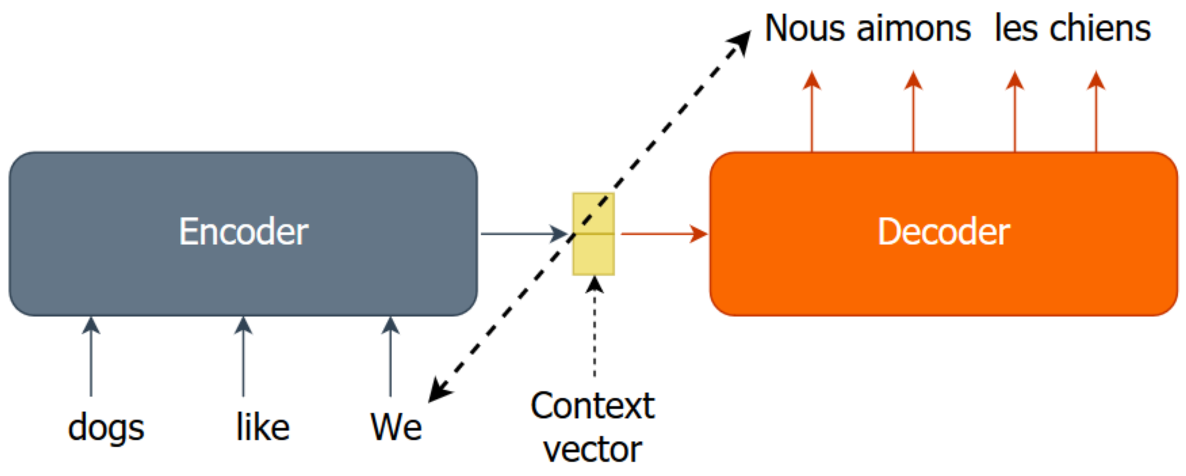
```
# 'california is never rainy during july , but it is sometimes beautiful in february .'
[21, 2, 11, 47, 5, 41, 7, 4, 2, 10, 30, 3, 38]  =>  [ 12 2 11 47  5 41  7  4  2 10 30  3]
```

- In Keras, 0 will never be allocated as a word ID

# Benefit of reversing sentences

- Helps to make a stronger initial connection between the encoder and the decoder



# Reversing the sentences

- Creating padded sequences and reversing the sequences on the time dimension

```
sentences = ["california is never rainy during july .",]
seqs = en_tok.texts_to_sequences(sentences)
pad_seq = preproc_text = pad_sequences(seqs, padding='post', truncating='post', maxlen=12)
```

```
[[21  2  9 25  5 27  0  0  0  0  0  0]]
```

# Reversing the sentences

```
pad_seq
```

```
[[21  2  9 25  5 27  0  0  0  0  0  0]]
```

```
pad_seq = pad_seq[:,::-1]
```

```
[[ 0  0  0  0  0  0 27  5 25  9  2 21]]
```

```
rev_sent = [en_tok.index_word[wid] for wid in pad_seq[0][-6:]]
print('Sentence: ', sentences[0])
print('\tReversed: ',' '.join(rev_sent))
```

```
Sentence:  california is never rainy during july .
    Reversed:  july during rainy never is california
```

# Let's practice!

MACHINE TRANSLATION IN PYTHON

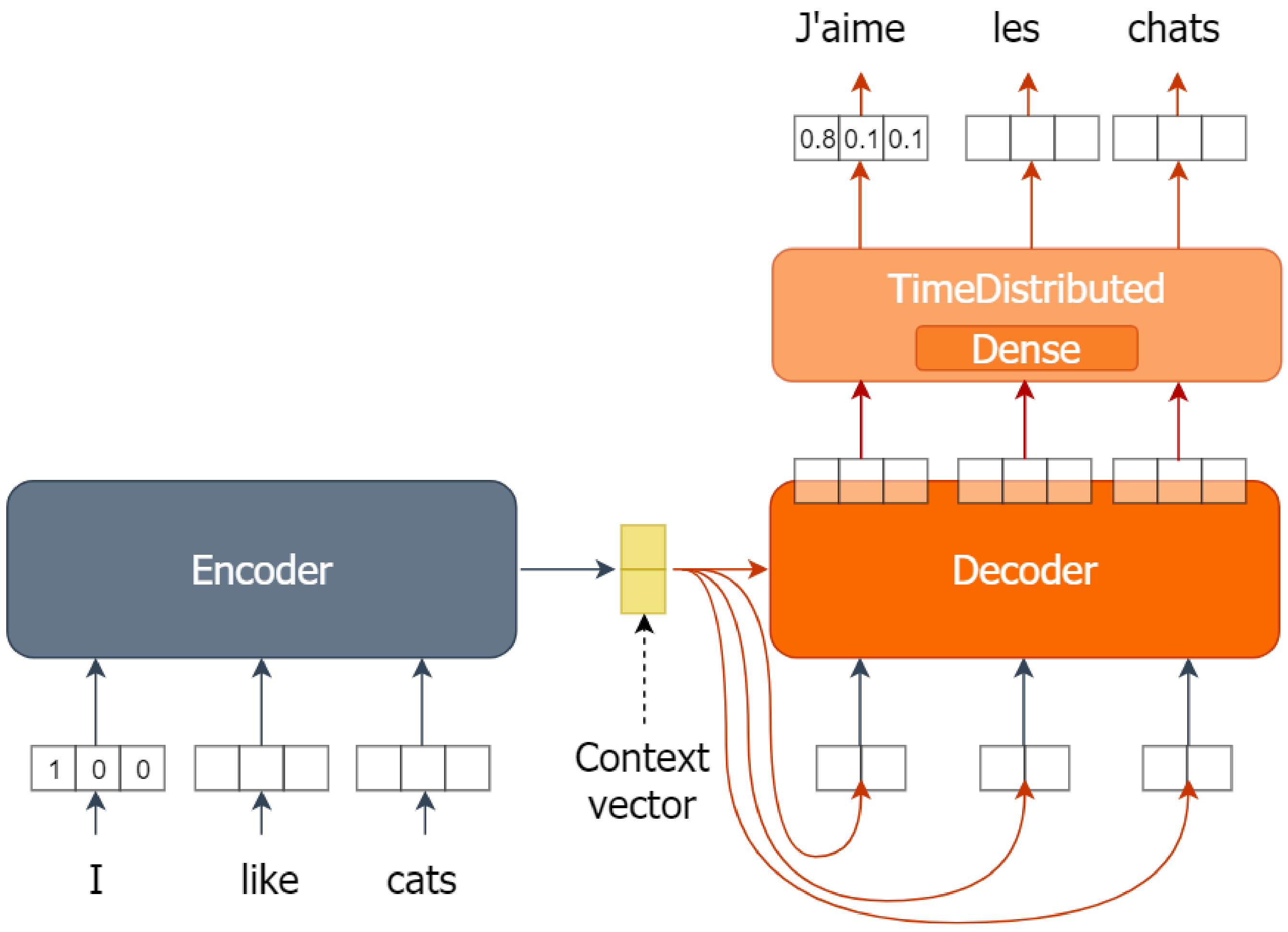
# Training the NMT model

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Revisiting the model

- Encoder GRU
  - Consumes English words
  - Outputs a context vector
- Decoder GRU
  - Consumes the context vector
  - Outputs a sequence of GRU outputs
- Decoder Prediction layer
  - Consumes the sequence of GRU outputs
  - Outputs prediction probabilities for French words



# Optimizing the parameters

- GRU layer and Dense layer have parameters
- Often represented by `W` (weights) and `b` (bias) (Initialized with random values)
- Responsible for transforming a given input to an useful output
- Changed over time to minimize a given loss using an optimizer
  - Loss: Computed as the difference between:
    - The predictions (i.e. French words generated with the model)
    - The actual outputs (i.e. actual French words).
- Informed the model during model compilation

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

# Training the model

- Training iterations

```
for ei in range(n_epochs): # Single traverse through the dataset
  for i in range(0,data_size,bsize): # Processing a single batch
```

- Obtaining a batch of training data

```
  en_x = sents2seqs('source', en_text[i:i+bsize], onehot=True, reverse=True)
  de_y = sents2seqs('target', en_text[i:i+bsize], onehot=True)
```

- Training on a single batch of data

```
  nmt.train_on_batch(en_x, de_y)
```

- Evaluating the model

```
  res = nmt.evaluate(en_x, de_y, batch_size=bsize, verbose=0)
```

# Training the model

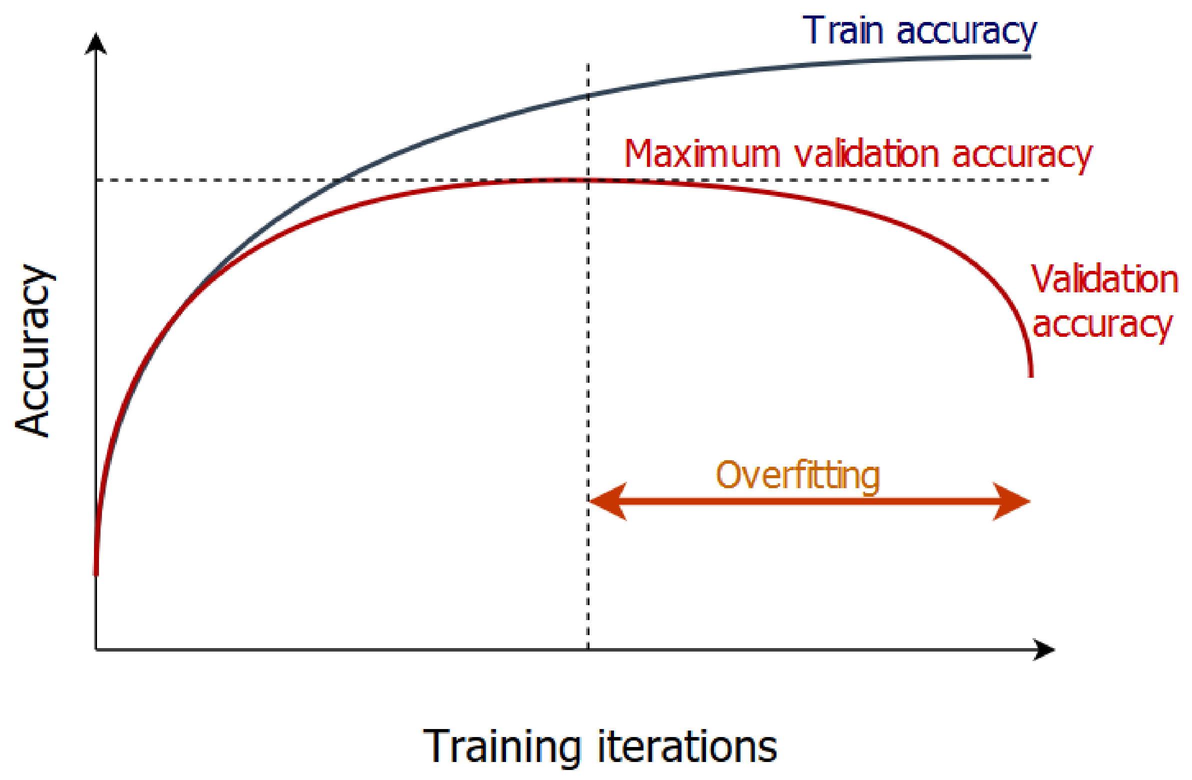
- Getting the training loss and the accuracy

```
res = nmt.evaluate(en_x, de_y, batch_size=bsize, verbose=0)
print("Epoch {} => Train Loss:{}, Train Acc: {}".format(
  ei+1,res[0], res[1]*100.0))
```

```
Epoch 1 => Train Loss:4.8036723136901855, Train Acc: 5.215999856591225
...
Epoch 1 => Train Loss:4.718592643737793, Train Acc: 47.0880001783371
...
Epoch 5 => Train Loss:2.8161656856536865, Train Acc: 56.40000104904175
Epoch 5 => Train Loss:2.527724266052246, Train Acc: 54.368001222610474
Epoch 5 => Train Loss:2.2689621448516846, Train Acc: 54.57599759101868
Epoch 5 => Train Loss:1.9934935569763184, Train Acc: 56.51199817657471
Epoch 5 => Train Loss:1.7581449747085571, Train Acc: 55.184000730514526
Epoch 5 => Train Loss:1.5613118410110474, Train Acc: 55.11999726295471
```

# Avoiding overfitting

- Break the dataset to two parts
  - Training set - The model will be trained on
  - Validation set - The model's accuracy will be monitored on
- When the validation accuracy stops increasing, stop the training.



# Splitting the dataset

- Define a train dataset size and validation dataset size

```
train_size, valid_size = 800, 200
```

- Shuffle the data indices randomly

```
inds = np.arange(len(en_text))
np.random.shuffle(inds)
```

- Get the train and valid indices

```
train_inds = inds[:train_size]
valid_inds = inds[train_size:train_size+valid_size]
```

# Splitting the dataset

- Split the dataset by separating,
  - Data having train indices to a train set
  - Data having valid indices to a valid set

```
tr_en = [en_text[ti] for ti in train_inds]
tr_fr = [fr_text[ti] for ti in train_inds]

v_en = [en_text[ti] for ti in valid_inds]
v_fr = [fr_text[ti] for ti in valid_inds]
```

# Training the model with validation

```
n_epochs, bsize = 5, 250
for ei in range(n_epochs):
  for i in range(0,train_size,bsize):
    en_x = sents2seqs('source', tr_en[i:i+bsize], onehot=True, pad_type='pre')
    de_y = sents2seqs('target', tr_fr[i:i+bsize], onehot=True)
    nmt.train_on_batch(en_x, de_y)
  v_en_x = sents2seqs('source', v_en, onehot=True, pad_type='pre')
  v_de_y = sents2seqs('target', v_fr, onehot=True)
  res = nmt.evaluate(v_en_x, v_de_y, batch_size=valid_size, verbose=0)
  print("Epoch: {} => Loss:{}, Val Acc: {}".format(ei+1,res[0], res[1]*100.0))
```

```
Epoch 1 => Train Loss:4.8036723136901855, Train Acc: 5.215999856591225
```

# Let's practice!

MACHINE TRANSLATION IN PYTHON

# Generating translations with the NMT

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Motivation

- You have a trained model
- Need to be able to assist humans on translation tasks
- Test the model on unseen data
- How?
  - Hold-out test set to evalute the model
  - You will test the model by asking it to predict translations for one sentence.

# Transforming the input

- English sentence

```
en_st = ['the united states is sometimes chilly during december , but it is sometimes freezing in june .']
```

- Transform the encoder sentence

```
en_seq = sents2seqs('source', en_st, onehot=True, reverse=True)
print(np.argmax(en_seq, axis=-1)
```

```
English: ['the united states is sometimes chilly during december ,
          but it is sometimes freezing in june .']
Reversed sentence: ['june in freezing sometimes is it ...']
Reversed sequence: [[34   3  54       10        2  4  7 45  5 69 10  2 23 22  6]]
```

# Generating the translation

- Generating a prediction

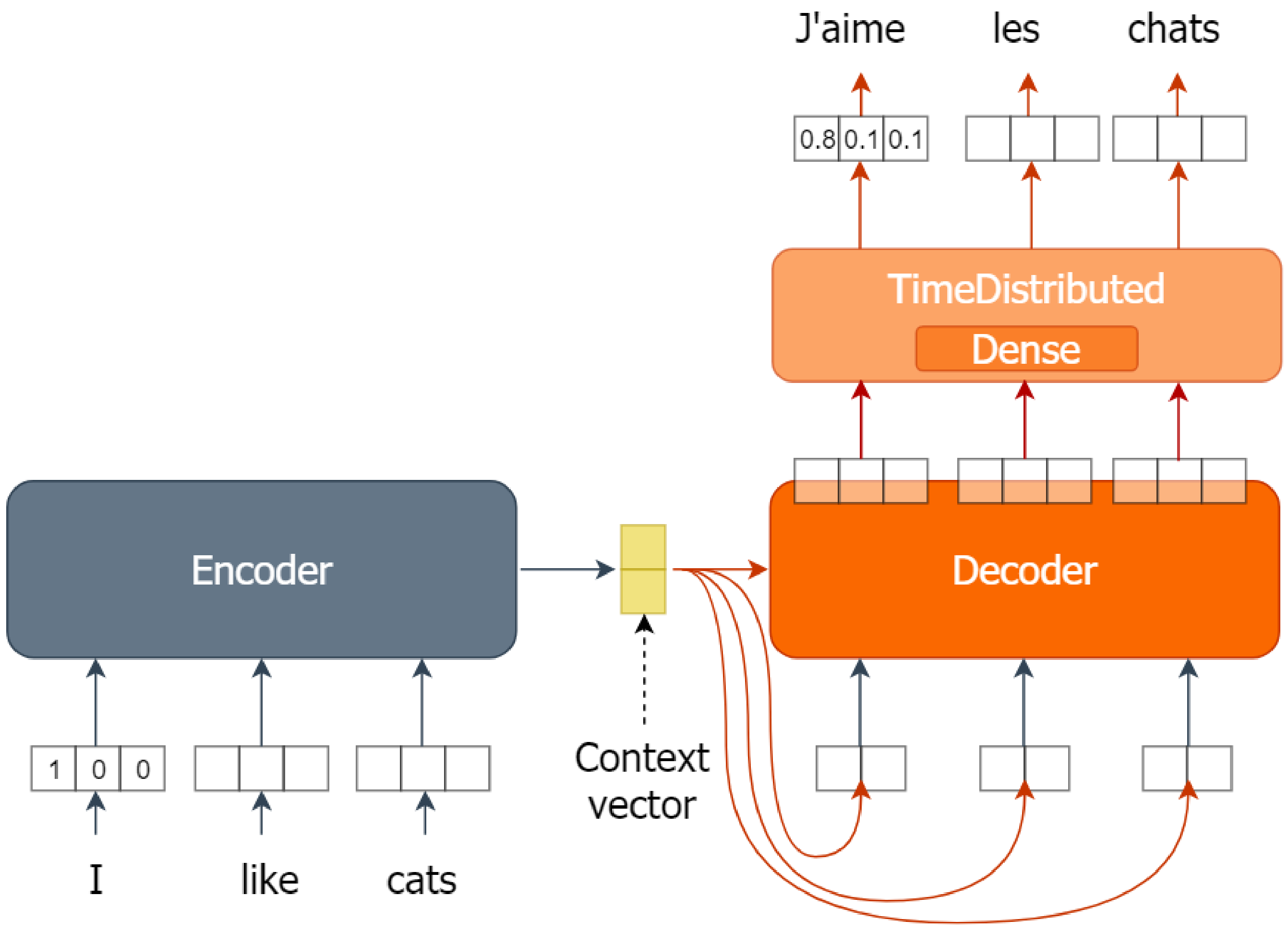
```
fr_pred = model.predict(en_seq)
```

- fr_pred.shape
  - `[sentences, seq len, vocab size]`
- Getting the predicted classes

```
fr_seq = np.argmax(fr_pred, axis=-1)[0]
```

```
[[ 3  7 35 34  2 ...  5  4  4  0  0]] # <= fr_seq
```

- fr_seq.shape
  - `[num sentences, sequence len]`



# Converting the prediction to a sentence

- Converting the produced word IDs to a sentence using list comprehension

```
fr_sentence = ' '.join([fr_id2word[i] for i in fr_seq if i != 0])
```

```
English: the united states is sometimes chilly during december , but it is sometimes freezing in june .
French:                    les états unis est parfois froid en décembre mais il est parfois le gel en
French (Google Translate): les etats-unis sont parfois froids en décembre, mais parfois gelés en juin
```

# List comprehension in more detail

- List comprehension

```
word_list = [fr_tok.index_word[i] for i in fr_seq if i != 0]
```

- For loop

```
word_list = []
for i in fr_seq:
  if i != 0:
    word_list.append(fr_tok.index_word[i])
```

# Let's practice!

MACHINE TRANSLATION IN PYTHON