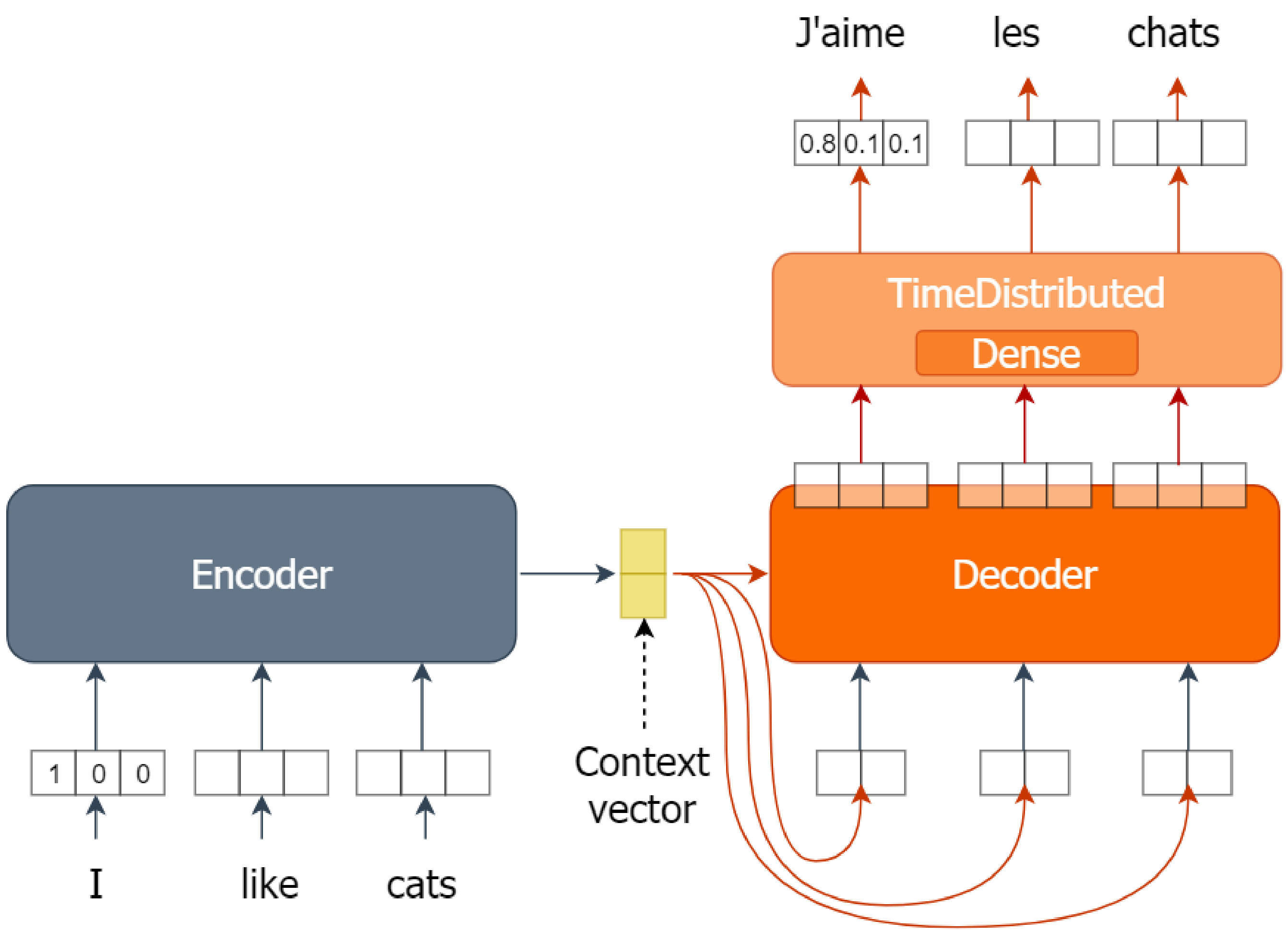
# Introduction to Teacher Forcing

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

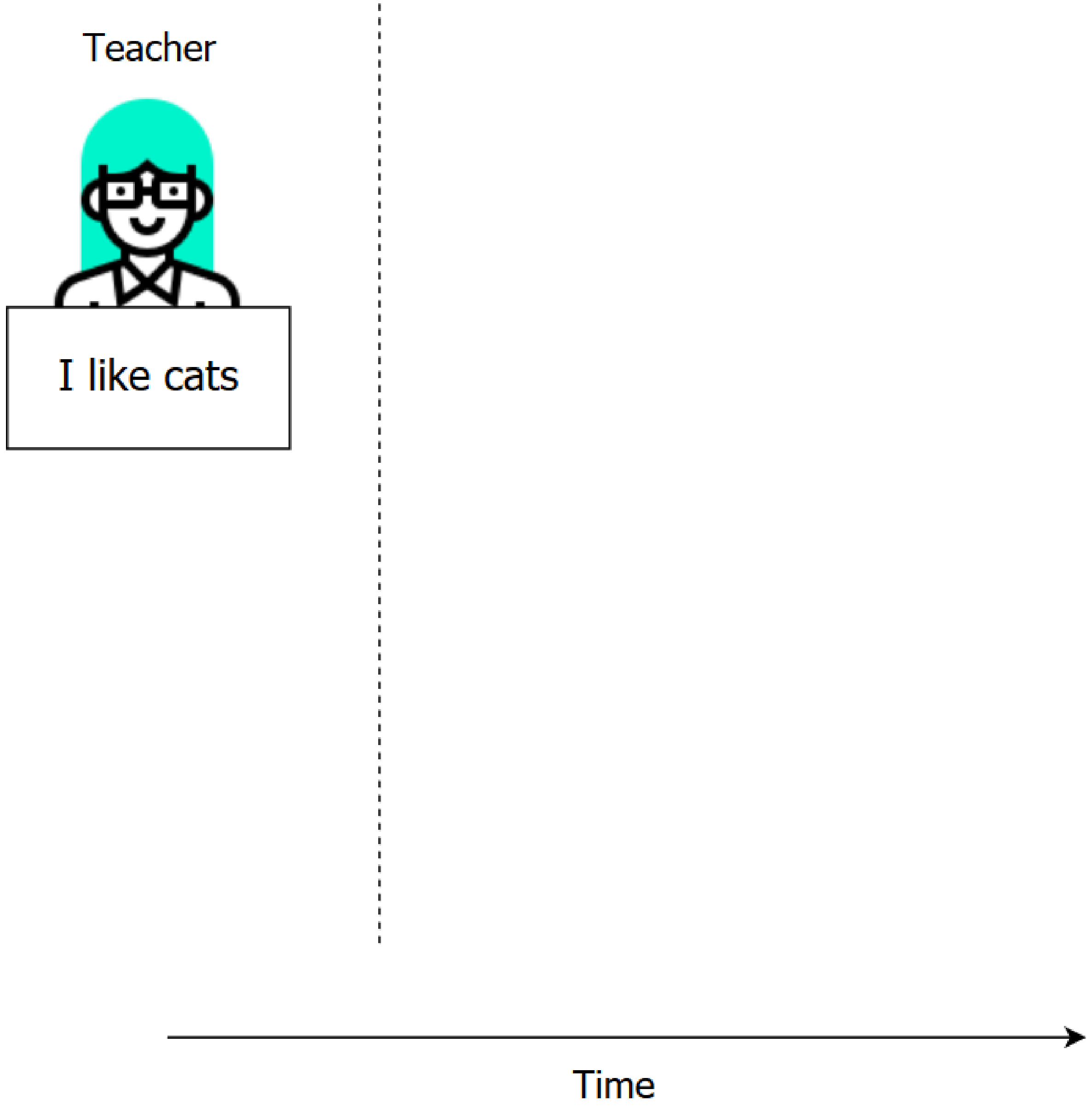
# The previous machine translator model

- The previous model

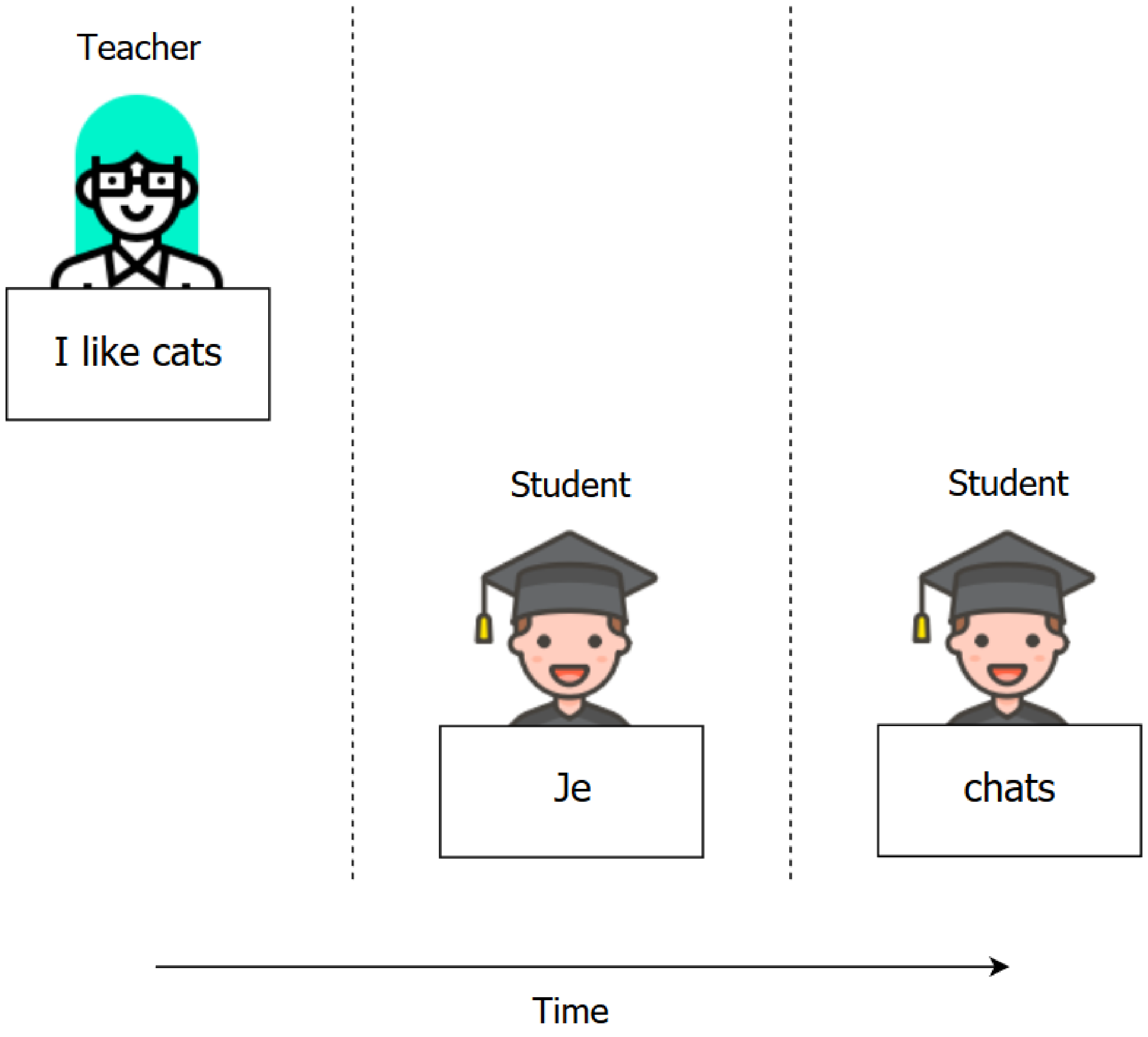



- Encoder GRU
  - Consumes English words
  - Outputs a context vector
- Decoder GRU
  - Consumes the context vector
  - Outputs a sequence of GRU outputs
- Decoder Prediction layer
  - Consumes the sequence of GRU outputs
  - Outputs prediction probabilities for French words

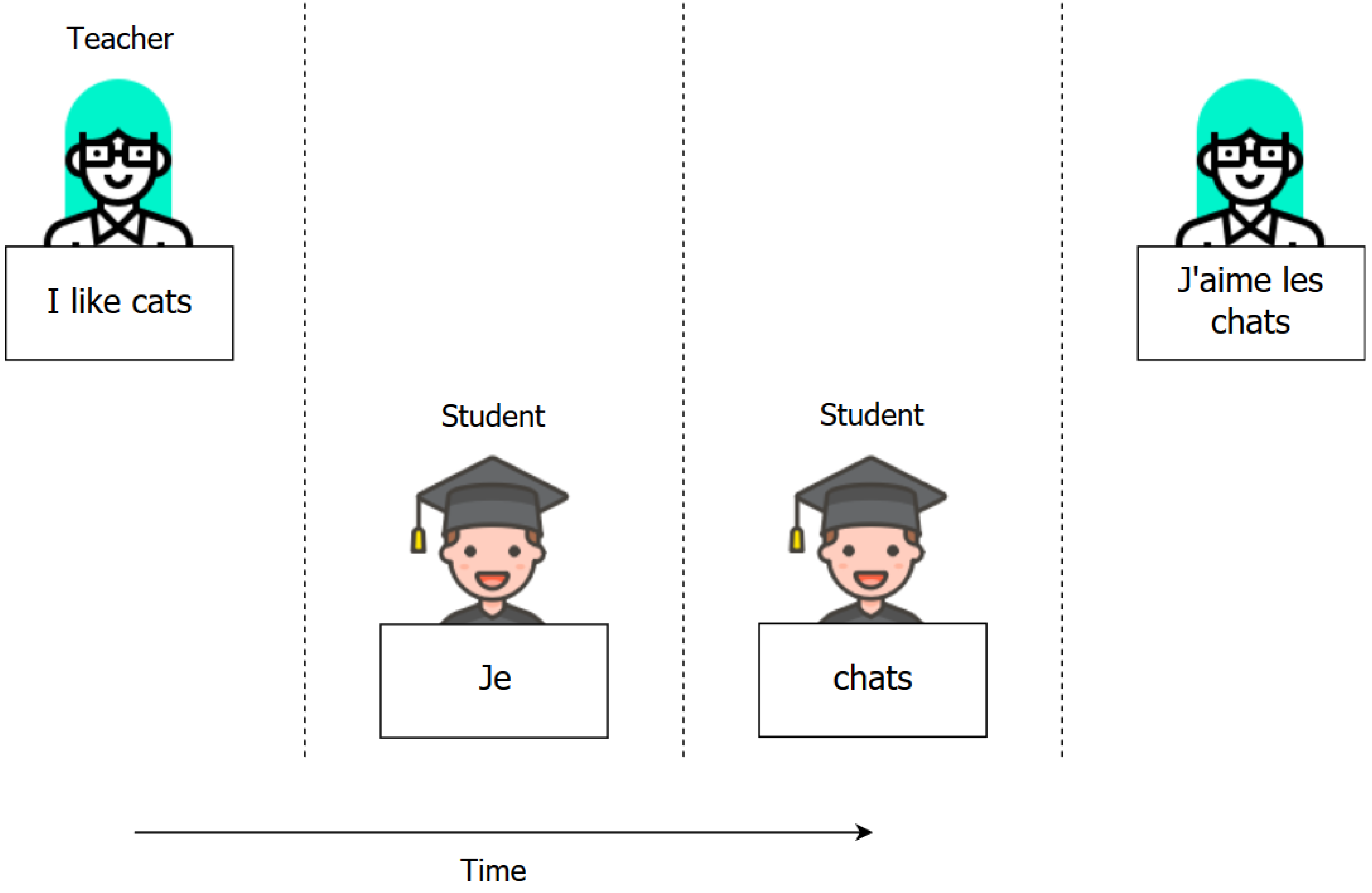
# Analogy: Training without Teacher Forcing



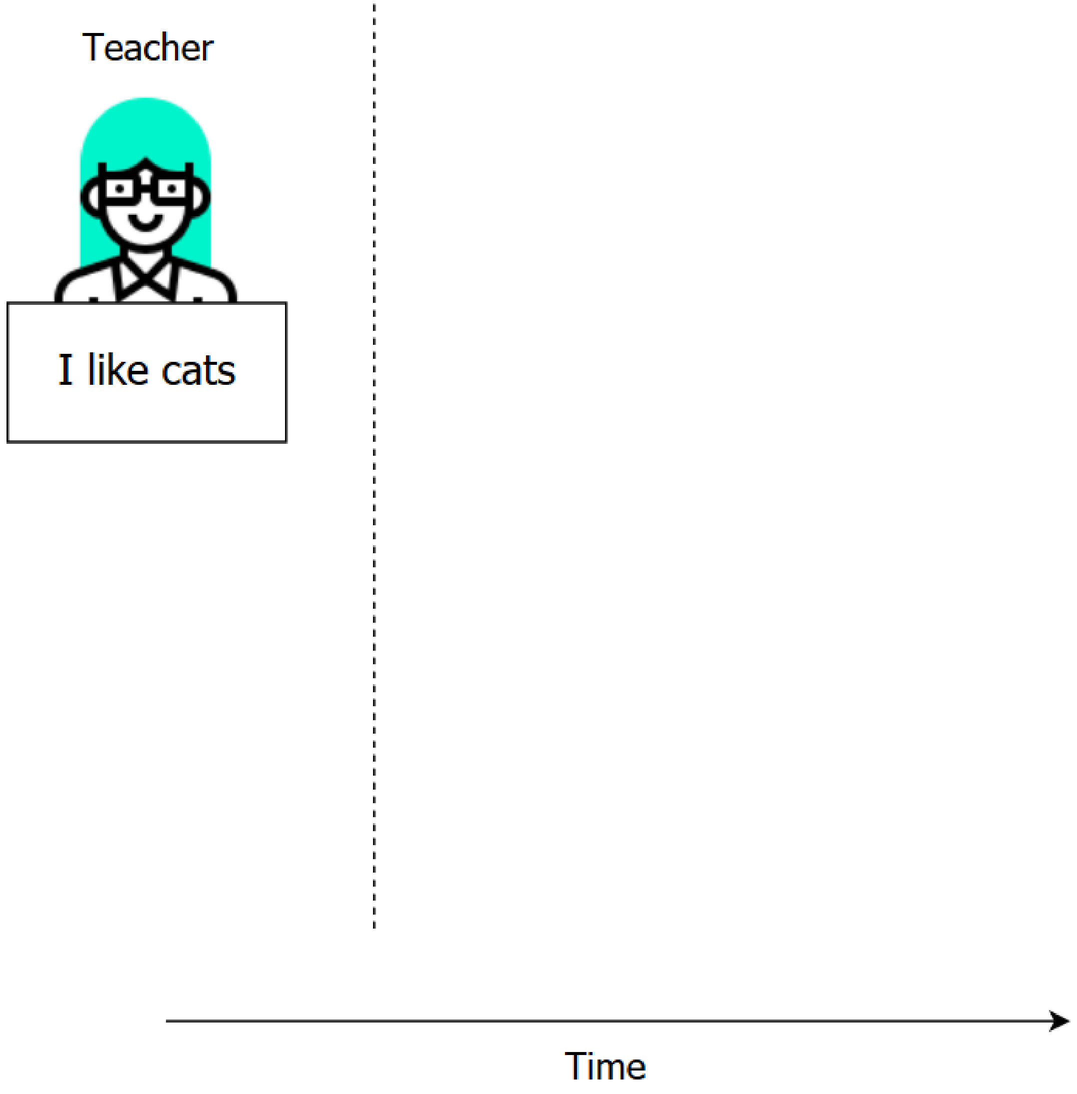
# Analogy: Training without Teacher Forcing



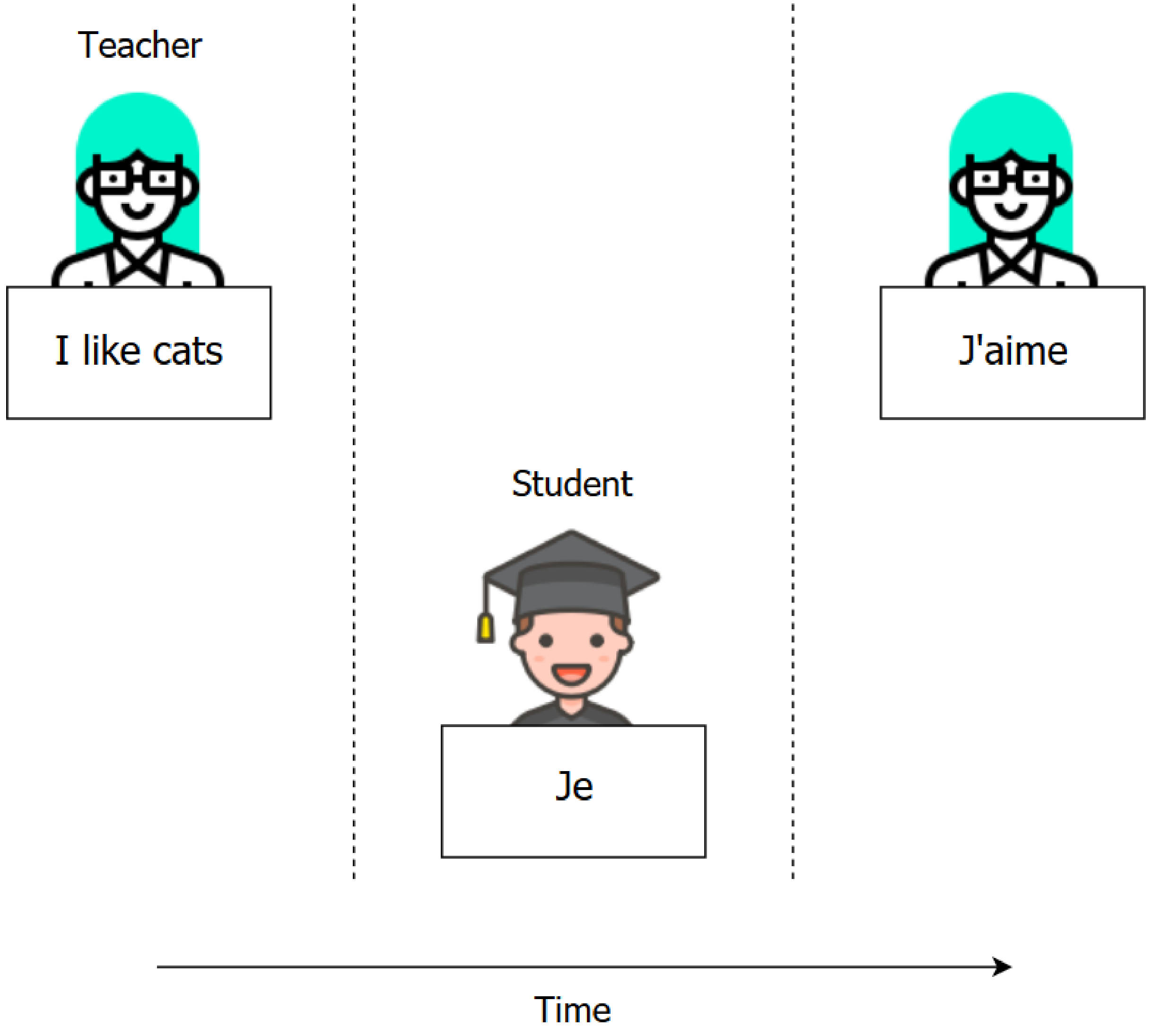
# Analogy: Training without Teacher Forcing



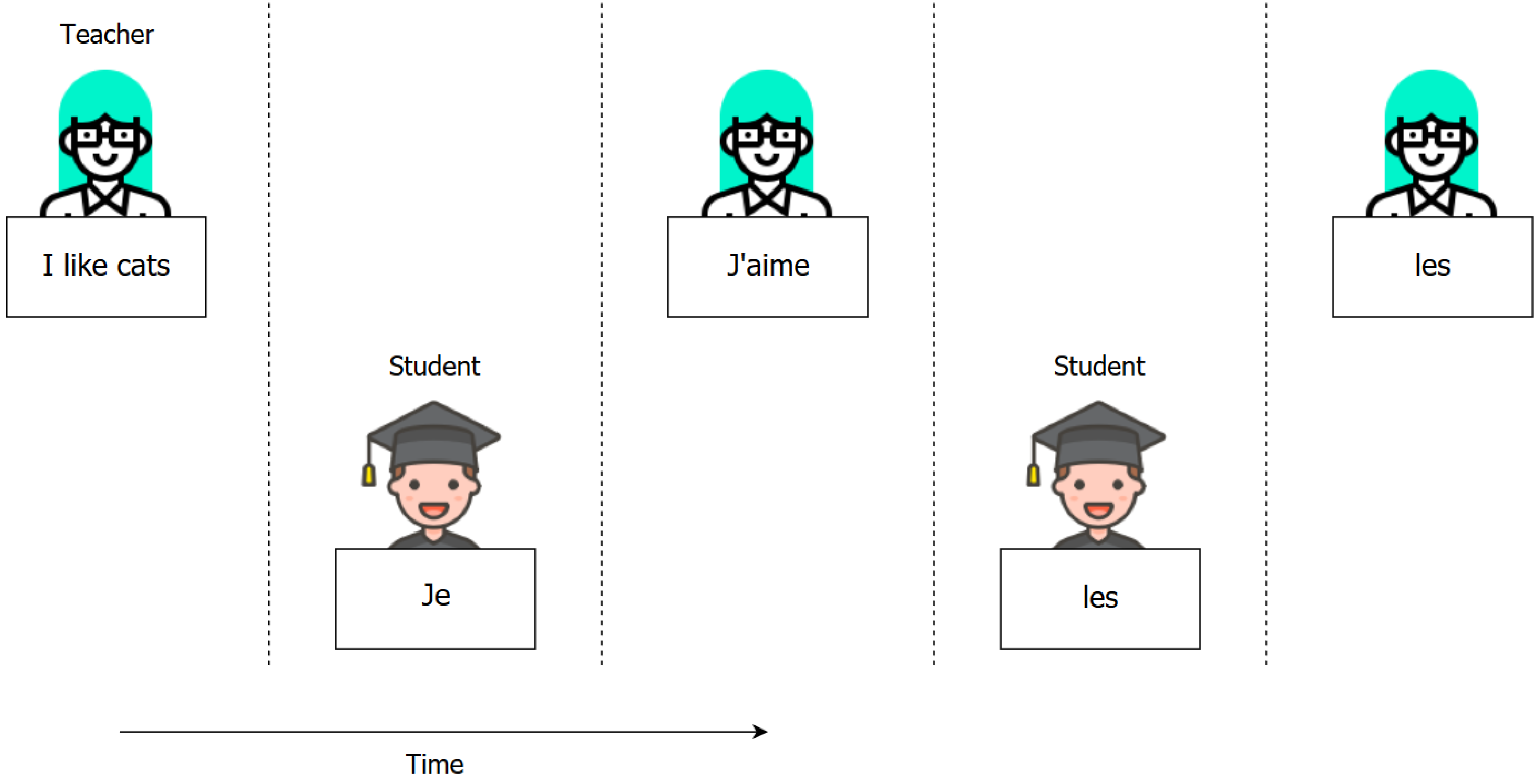
# Analogy: Training with Teacher Forcing



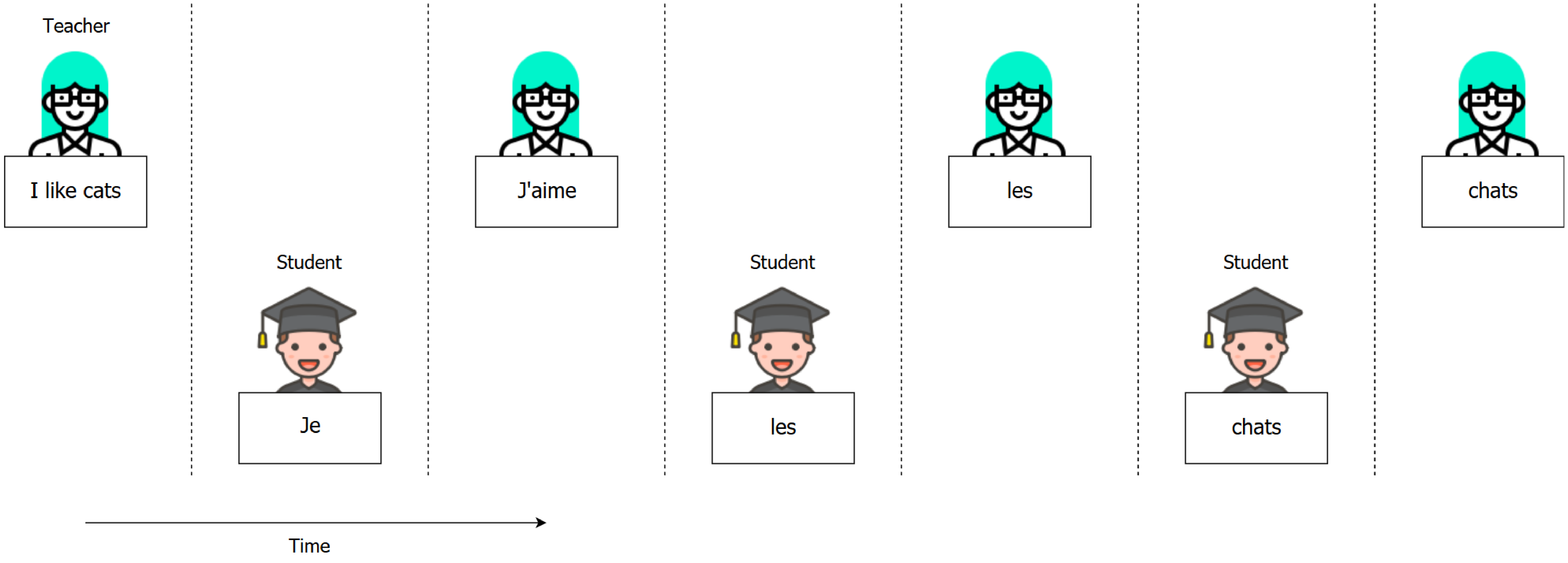
# Analogy: Training with Teacher Forcing



# Analogy: Training with Teacher Forcing

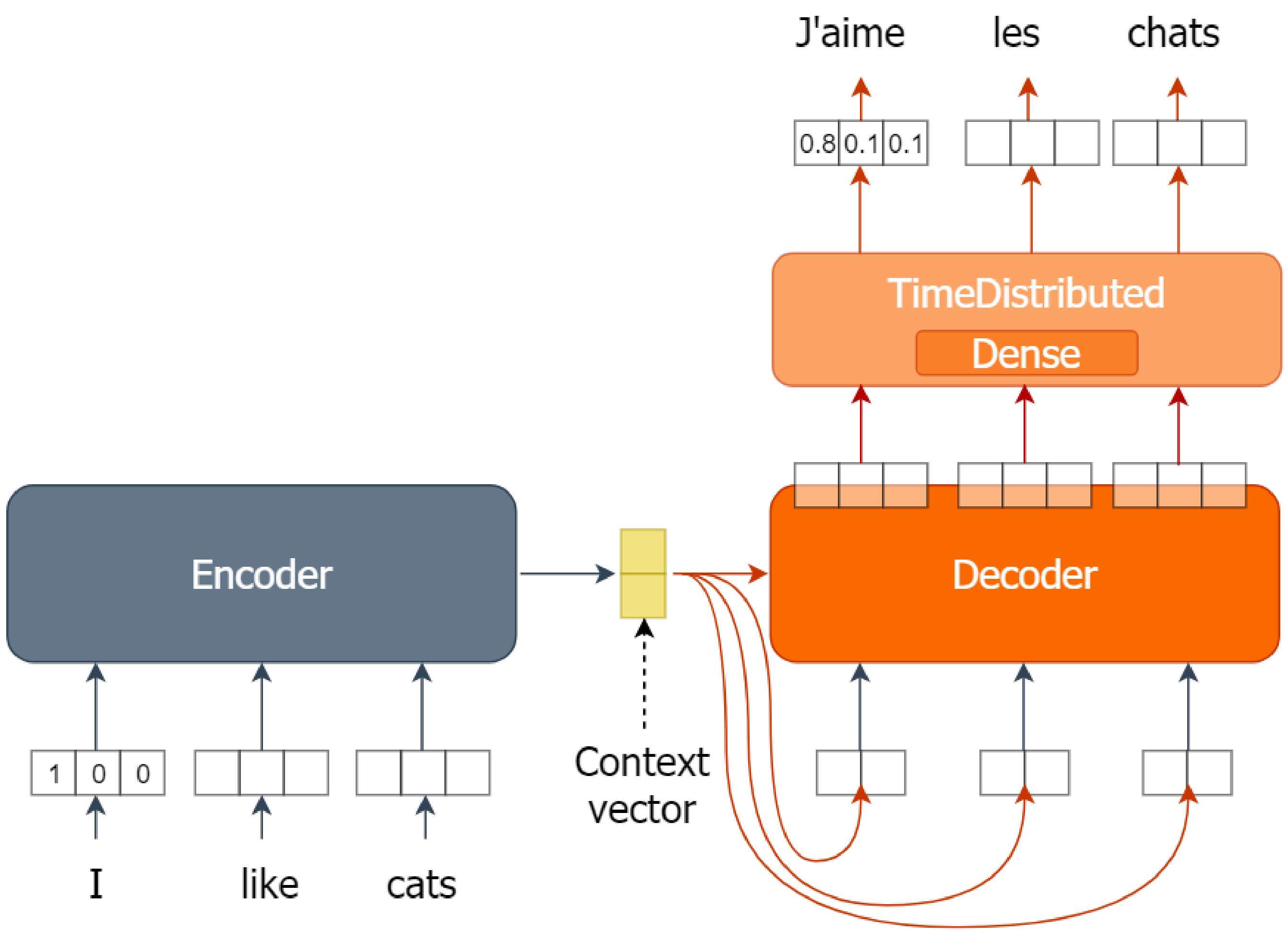


# Analogy: Training with Teacher Forcing

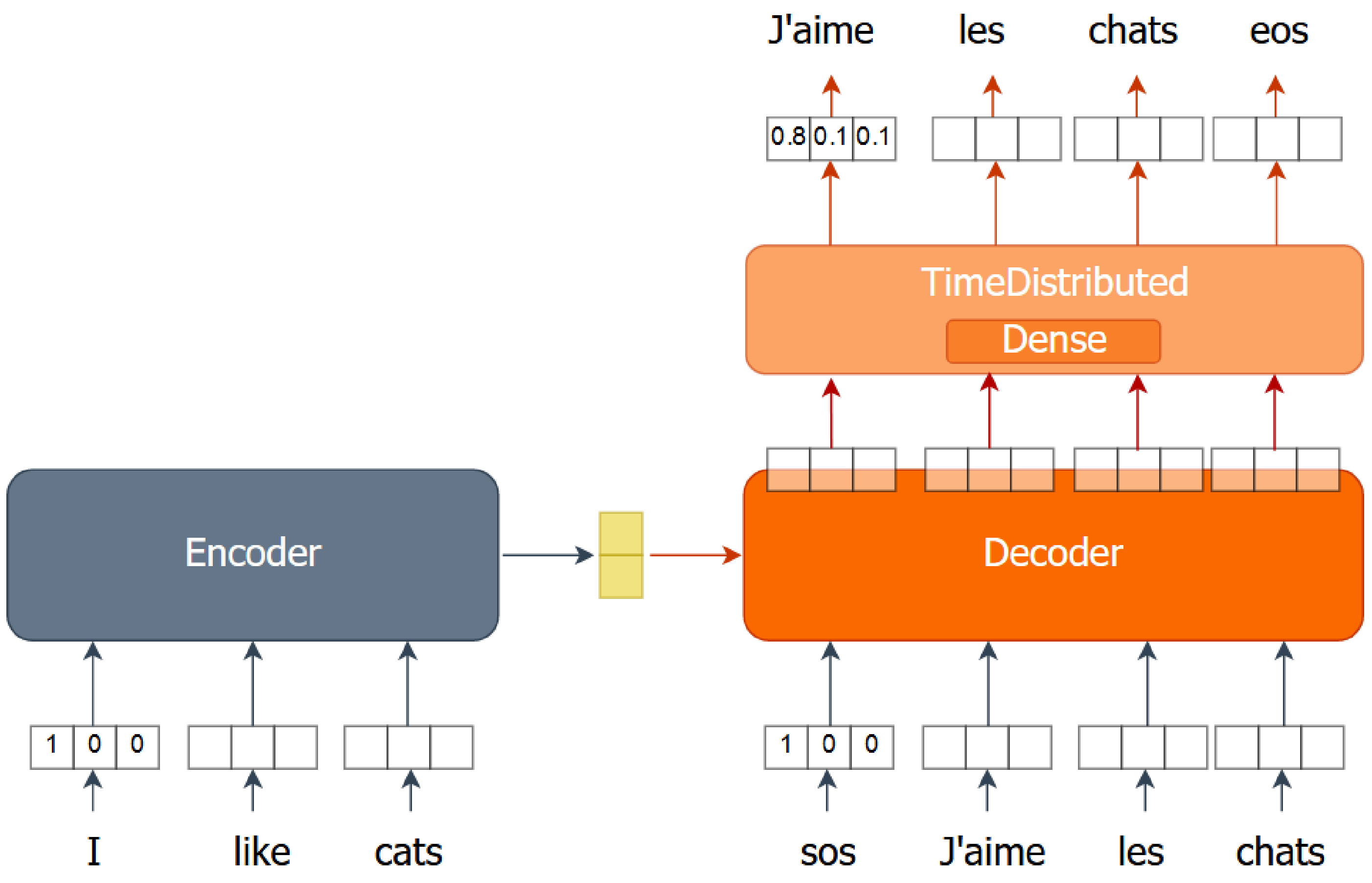


# The previous machine translator model

- The previous model




- Teacher-forced model



# Implementing the model with Teacher Forcing

- Encoder

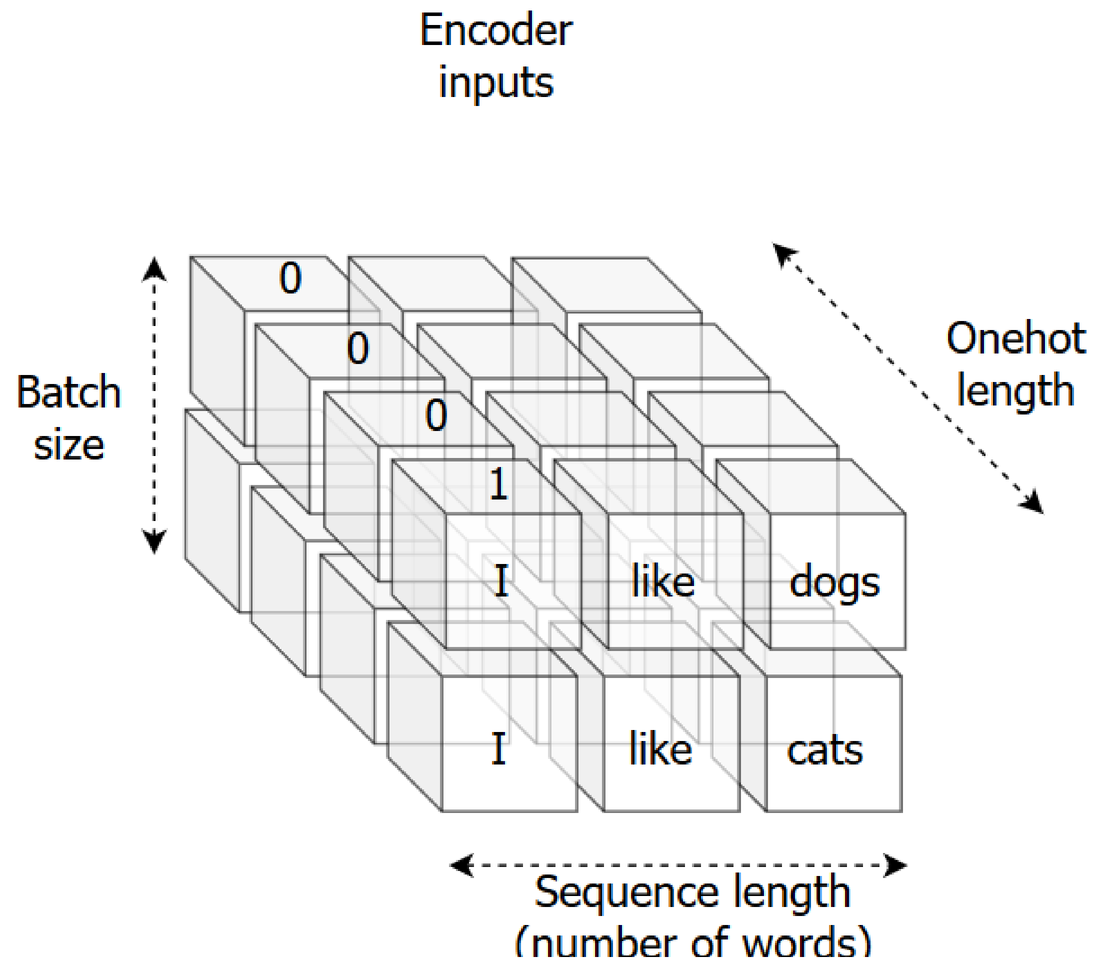
```
en_inputs = layers.Input(shape=(en_len, en_vocab))
en_gru = layers.GRU(hsize, return_state=True)
en_out, en_state = en_gru(en_inputs)
```

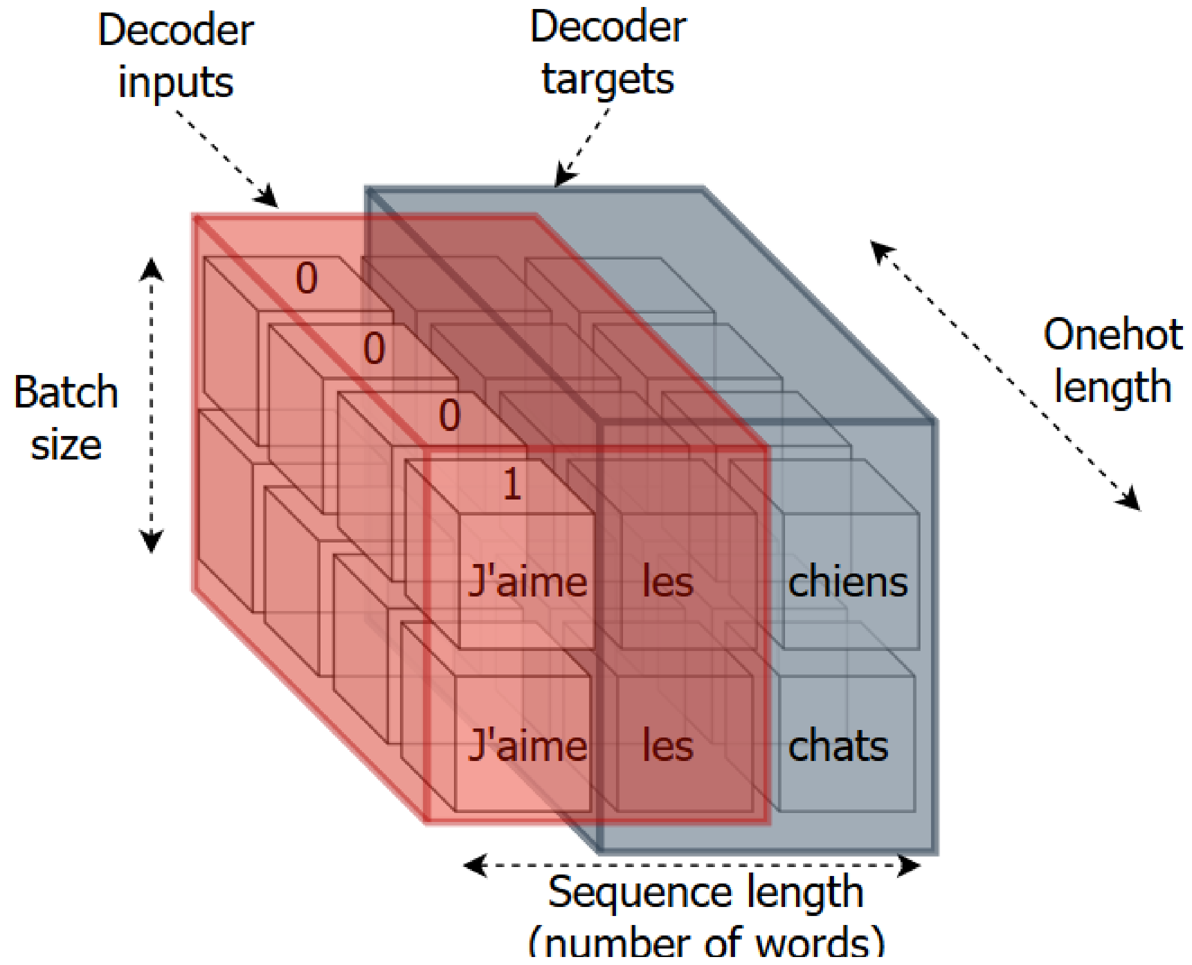
- Decoder GRU

```
de_inputs = layers.Input(shape=(fr_len-1, fr_vocab))
de_gru = layers.GRU(hsize, return_sequences=True)
de_out = de_gru(de_inputs, initial_state=en_state)
```

# Inputs and outputs

- Encoder input - e.g. `I` , `like` , `dogs`
- Decoder input - e.g. `J'aime` , `les`
- Decoder output - e.g. `les` , `chiens`



# Implementing the model with Teacher Forcing

- Encoder

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

- Decoder GRU

```
de_inputs = layers.Input(shape=(fr_len-1, fr_vocab))
de_gru = layers.GRU(hsize, return_sequences=True)
de_out = de_gru(de_inputs, initial_state=en_state)
```

- Decoder Prediction

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

# Compiling the model

```
nmt_tf = Model(inputs=[en_inputs, de_inputs], outputs=de_pred)
nmt_tf.compile(optimizer='adam', loss="categorical_crossentropy", metrics=["acc"])
```

# Preprocessing data

- Encoder
  - Inputs - All English words (onehot encoded)
    - `en_x = sents2seqs('source', en_text, onehot=True, reverse=True)`
- Decoder

```
de_xy = sents2seqs('target', fr_text, onehot=True)
```

  - Inputs - All French words except the last word (onehot encoded)
    - `de_x = de_xy[:,:-1,:]`
  - Outputs/Targets - All French words except the first word (onehot encoded)
    - `de_y = de_xy[:,1:,:]`

# Let's practice!

MACHINE TRANSLATION IN PYTHON

# Training the model with Teacher Forcing

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Model training in detail

- Model training requires:
  - A loss function (e.g. categorical crossentropy)
  - An optimizer (e.g. Adam)

# Model training in detail

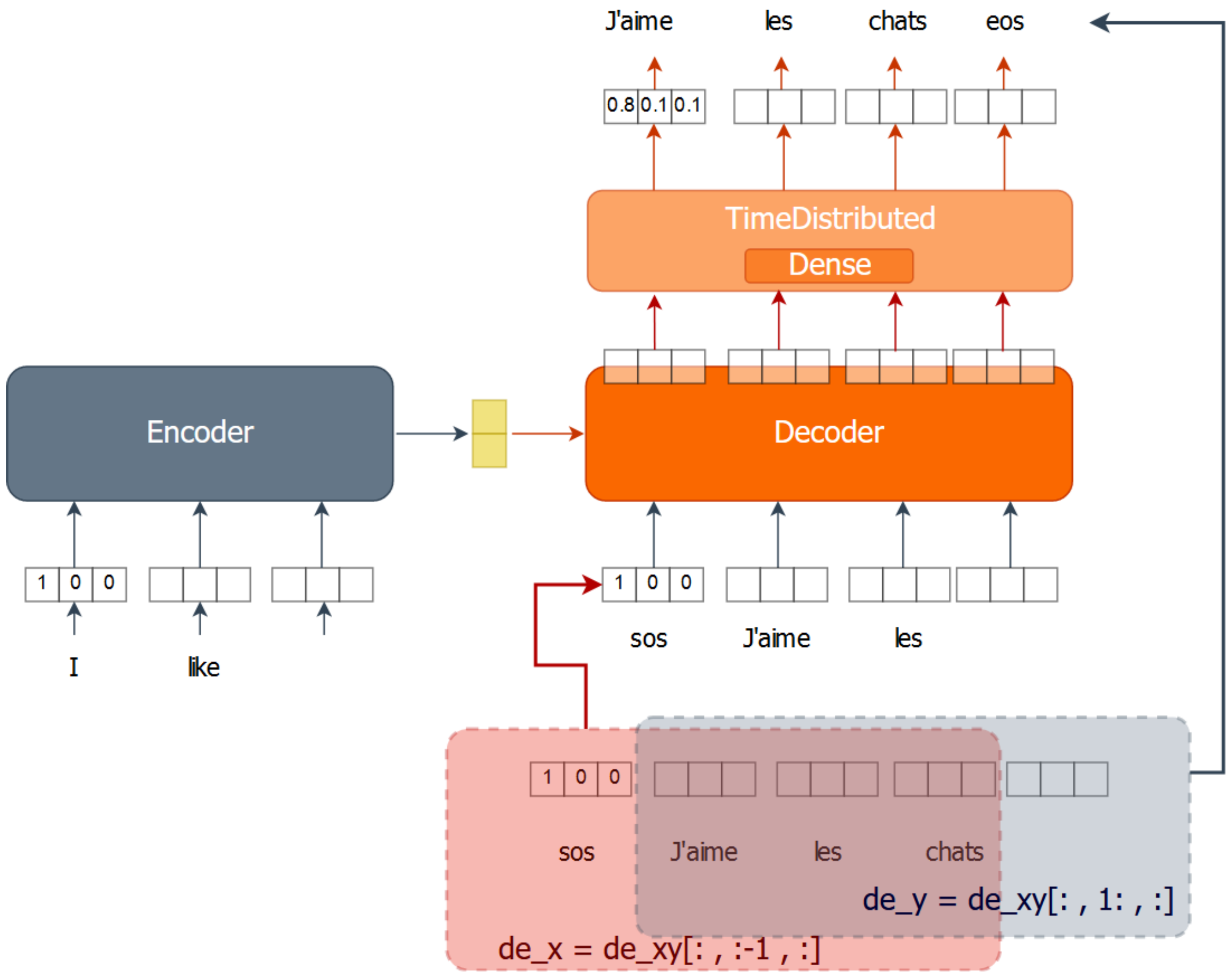
- To compute loss, following items are required:
  - Probabilistic predictions generated using inputs ( `[batch_size, seq_len, vocab_size]` )
    - e.g. `[[0.11,...,0.81,0.04], [0.05,...,0.01, 0.93], ..., [0.78,..., 0.03,0.01]]`
  - Actual onehot encoded French targets ( `[batch_size, seq_len, vocab_size]` )
    - e.g. `[[0, ..., 1, 0], [0, ..., 0, 1],..., [0, ..., 1, 0]]`
  - Crossentropy: difference between the targets and predicted words
- The loss is passed to an optimizer which will change the model parameters to minimize the loss

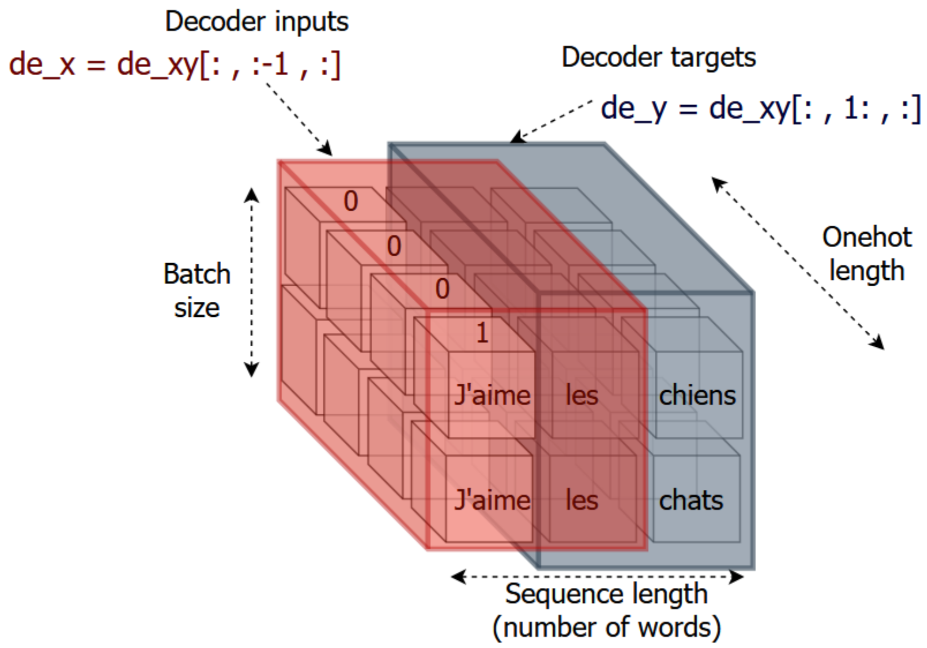
# Training the model with Teacher Forcing

```
n_epochs, bsize = 3, 250
for ei in range(n_epochs):
  for i in range(0,data_size,bsize):
    # Encoder inputs, decoder inputs and outputs
    en_x = sents2seqs('source', en_text[i:i+bsize], onehot=True, reverse=True)
    de_xy = sents2seqs('target', fr_text[i:i+bsize], onehot=True)
    # Separating decoder inputs and outputs
    de_x = de_xy[:,:-1,:]
    de_y = de_xy[:,1:,:]
    # Training and evaulating on a single batch
    nmt_tf.train_on_batch([en_x,de_x], de_y)
    res = nmt_tf.evaluate([en_x,de_x], de_y, batch_size=bsize, verbose=0)
    print("{} => Train Loss:{}, Train Acc: {}".format(ei+1,res[0], res[1]*100.0))
```

# Array slicing in detail

```
de_x = de_xy[:,:-1,:]
de_y = de_xy[:,1:,:]
```



# Creating training and validation data

```
train_size, valid_size = 800, 200
# Creating data indices
inds = np.arange(len(en_text))
np.random.shuffle(inds)

# Separating train and valid indices
train_inds = inds[:train_size]
valid_inds = inds[train_size:train_size+valid_size]

# Extracting train and valid data
tr_en = [en_text[ti] for ti in train_inds]
tr_fr = [fr_text[ti] for ti in train_inds]

v_en = [en_text[vi] for vi in valid_inds]
v_fr = [fr_text[vi] for vi in valid_inds]
print('Training (EN):\n', tr_en[:2], '\nTraining (FR):\n', tr_fr[:2])
print('\nValid (EN):\n', tr_en[:2], '\nValid (FR):\n', tr_fr[:2])
```

# Training with validation

```
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, reverse=True)
    de_xy = sents2seqs('target', tr_fr[i:i+bsize], onehot=True)
    de_x, de_y = de_xy[:,:-1,:], de_xy[:,1:,:]
    nmt_tf.train_on_batch([en_x, de_x], de_y)
  v_en_x = sents2seqs('source', v_en, onehot=True, reverse=True)
  v_de_xy = sents2seqs('target', v_fr, onehot=True)
  v_de_x, v_de_y = v_de_xy[:,:-1,:], v_de_xy[:,1:,:]
  res = nmt_tf.evaluate([v_en_x, v_de_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 => Loss:4.784221172332764, Val Acc: 1.4999999664723873
Epoch 2 => Loss:4.716882228851318, Val Acc: 44.458332657814026
Epoch 3 => Loss:4.63267183303833, Val Acc: 47.333332896232605
```

# Let's train!

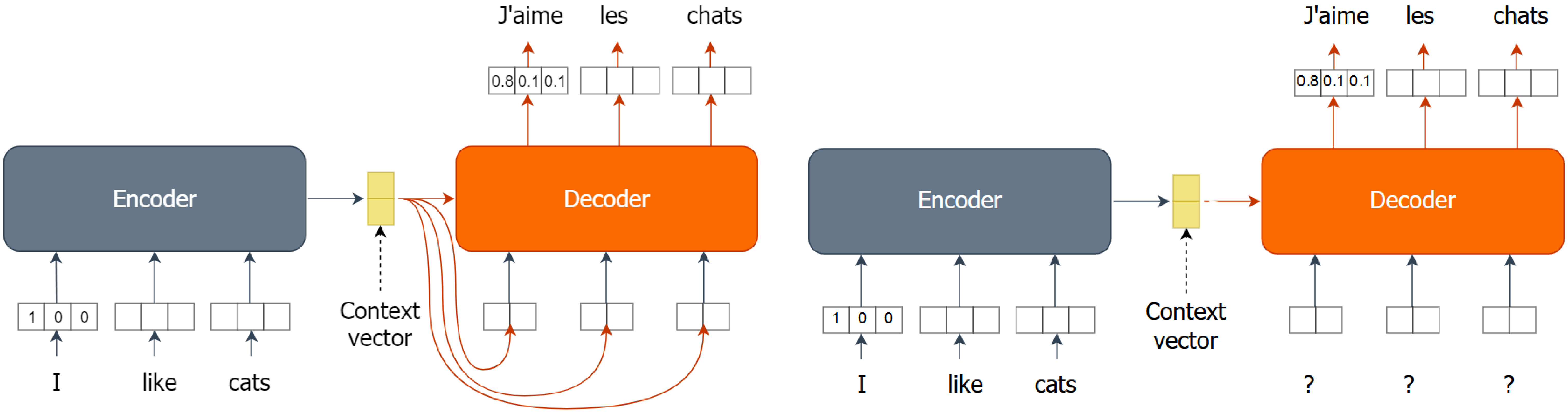
MACHINE TRANSLATION IN PYTHON

# Generating translations from the model

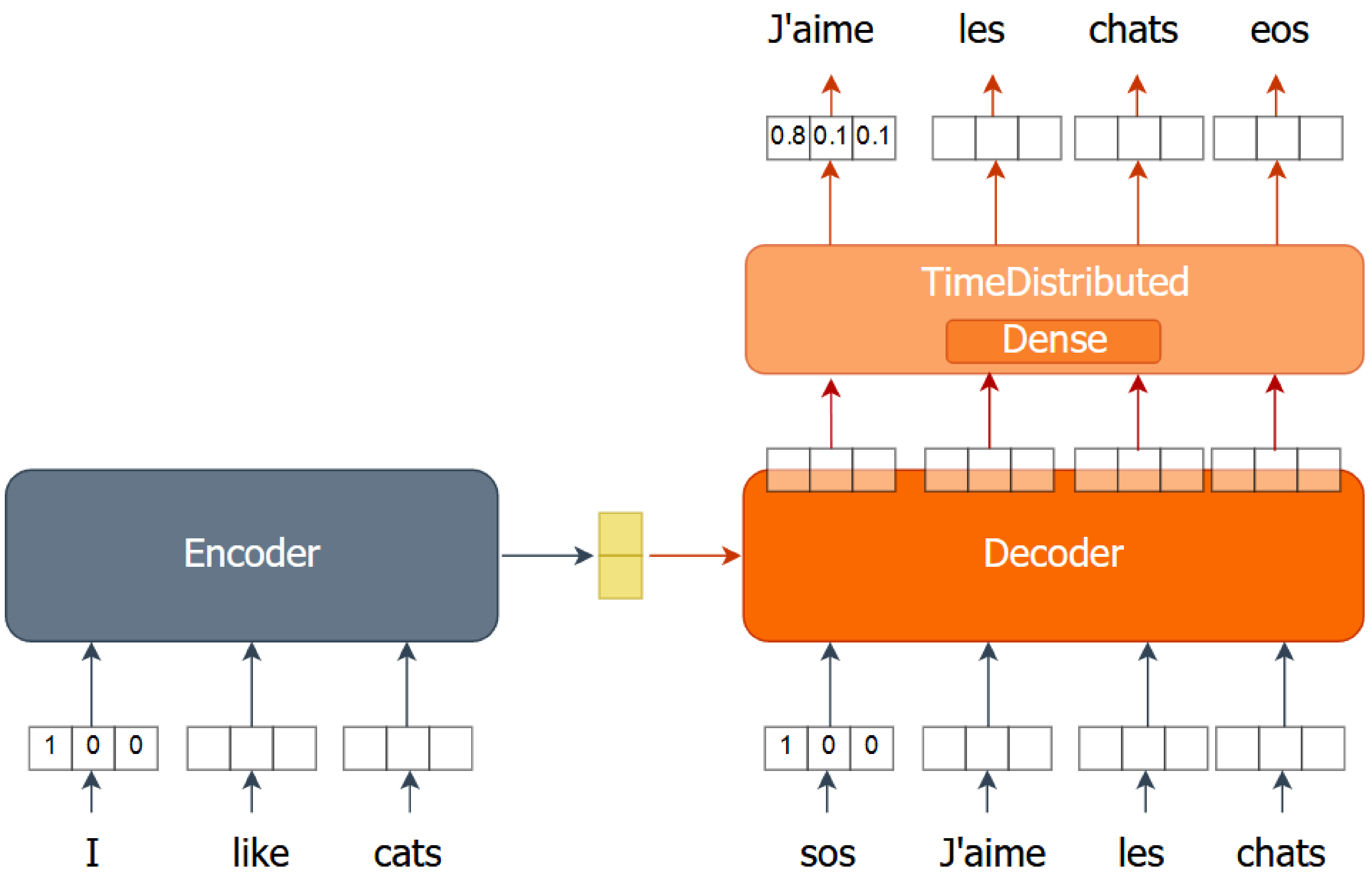
MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Previous model vs new model

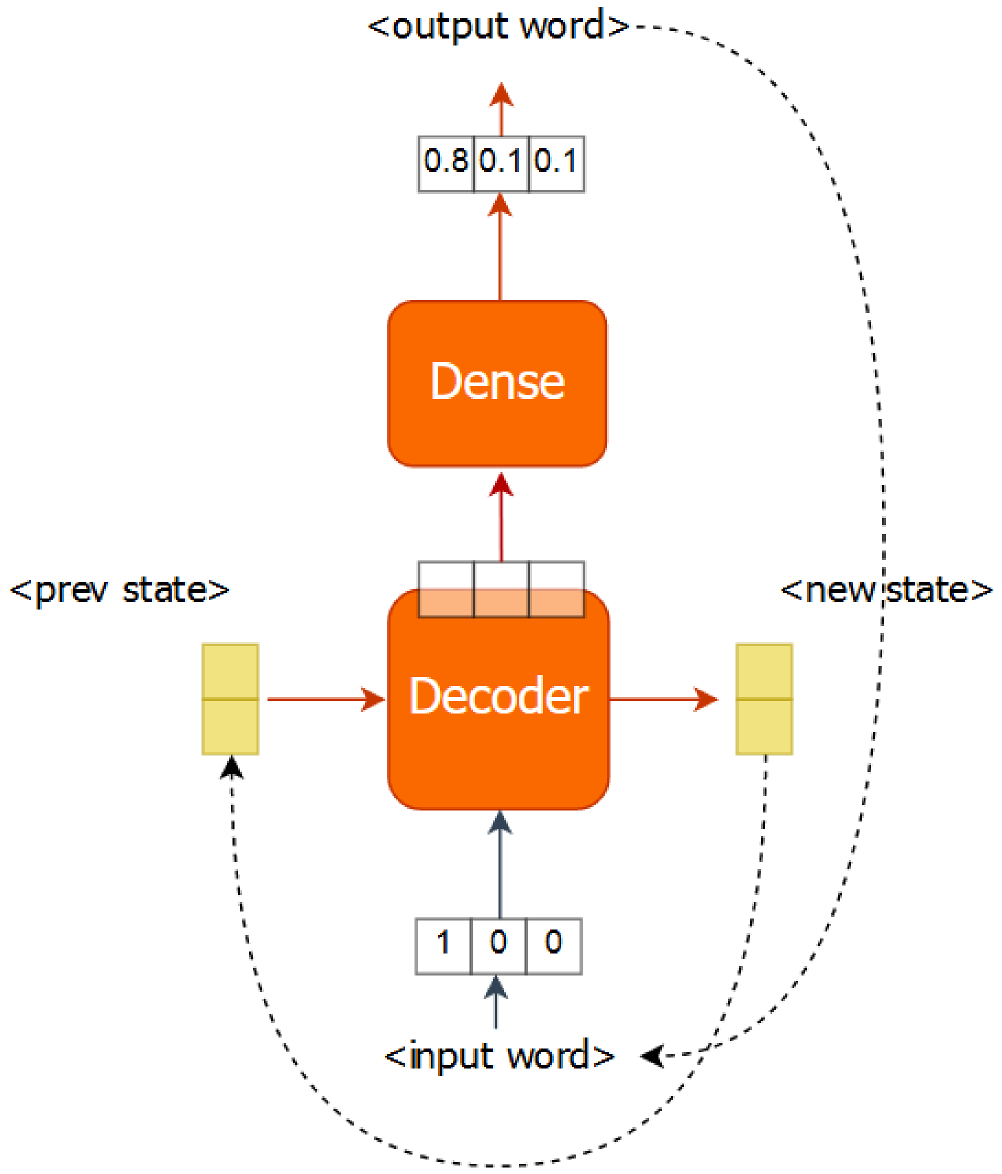


# Trained model



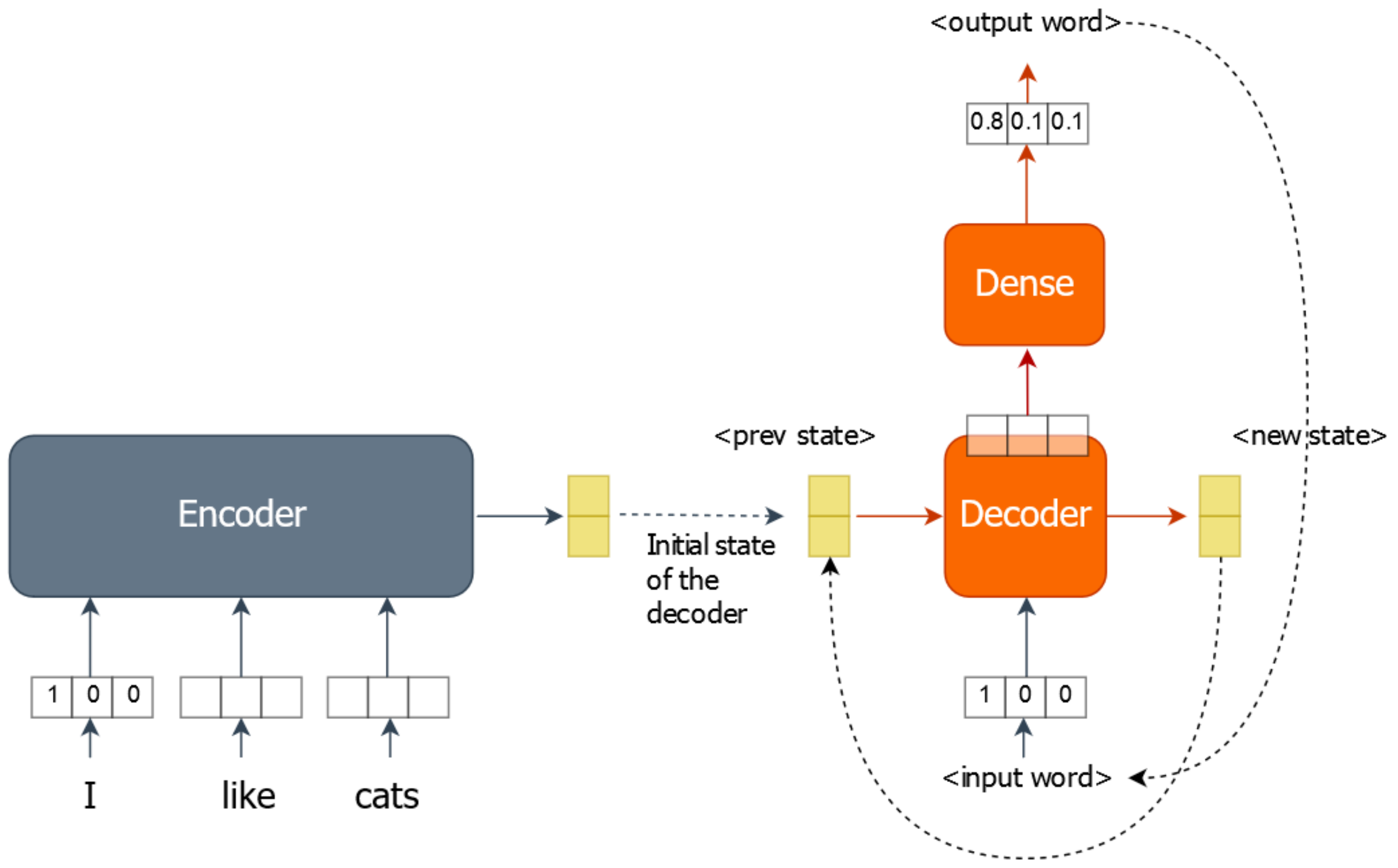
# Decoder of the inference model

- Takes in
  - A onehot encoded word
  - A state input (gets the state from previous timestep)
- Produces
  - A new state
  - A prediction (i.e. a word)
- Recursively feed the predicted word and the state back to the model as inputs

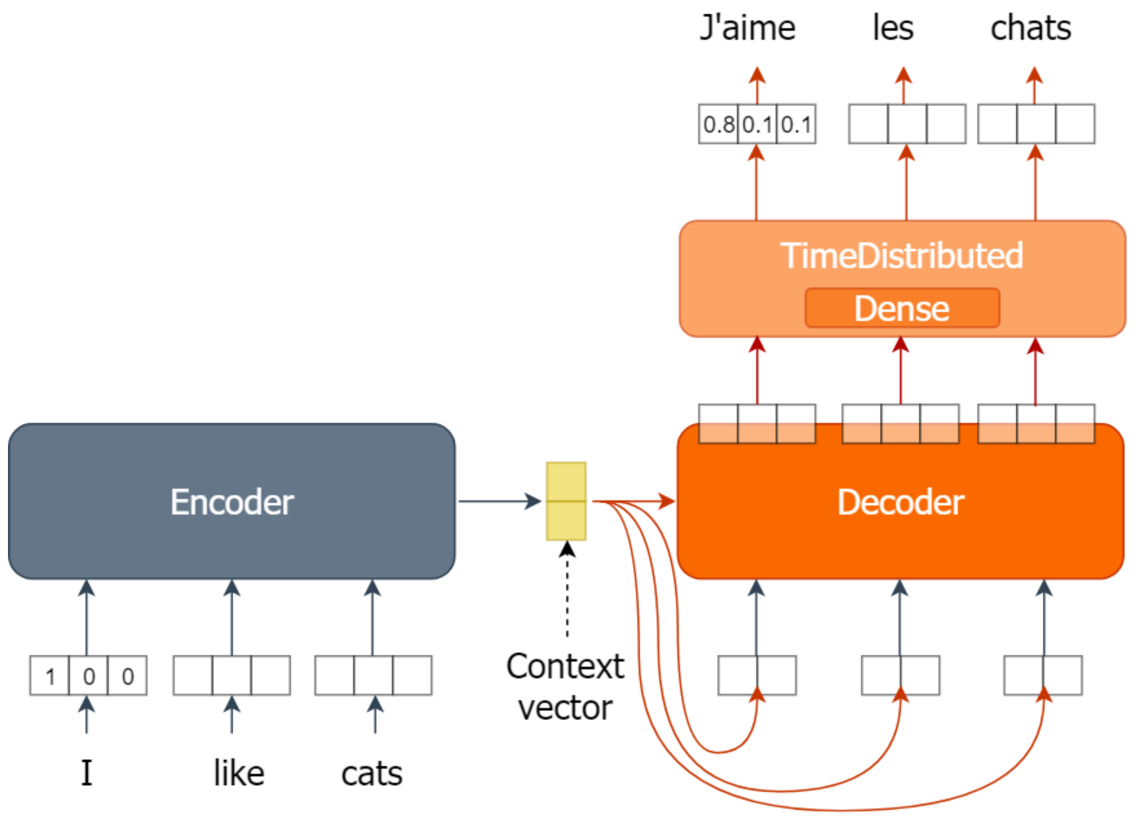


# Full inference model

- Inference model with the recursive decoder




- Inference model from the previous chapter



# Value of sos and eos tokens

- `sos` marks beginning of a translation (i.e. a French sentence).
  - Feed in `sos` as the first word to the decoder and keep predicting
- `eos` marks the end of a translation.
  - Predictions stop when the word predicted by the model is `eos`
- As a safety measure use a maximum length the model can predict for

# Defining the generator encoder

- Importing layers and Model

```
# Import Keras layers
import tensorflow.keras.layers as layers
from tensorflow.keras.models import Model
```

- Defining model layers

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

- Defining `Model` object

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

# Defining the generator decoder

- Defining the decoder `Input` layers

```
de_inputs = layers.Input(shape=(1, fr_vocab))
de_state_in = layers.Input(shape=(hsize,))
```

- Defining the decoder's interim `layers`

```
de_gru = layers.GRU(hsize, return_state=True)
de_out, de_state_out = de_gru(de_inputs, initial_state=de_state_in)
de_dense = layers.Dense(fr_vocab, activation='softmax')
de_pred = de_dense(de_out)
```

- Defining the decoder `Model`

```
decoder = Model(inputs=[de_inputs, de_state_in], outputs=[de_pred, de_state_out])
```

# Copying the weights

- Get weights of the layer `l1`
  - `w = l1.get_weights()`
- Set the weights of the layer `l2` with `w`
  - `l2.set_weights(w)`
- In our model, there are three layers with weights
  - Encoder `GRU`, Decoder `GRU` and Decoder `Dense`

```
en_gru_w = tr_en_gru.get_weights()
en_gru.set_weights(en_gru_w)
```

Which can also be written as,

```
en_gru.set_weights(tr_en_gru.get_weights())
```

# Generating translations

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

- Converting the English sentence to a sequence

```
en_seq = sents2seqs('source', en_st, onehot=True, reverse=True)
```

- Getting the context vector

```
de_s_t = encoder.predict(en_seq)
```

- Converting "sos" (initial word to the decoder) to a sequence

```
de_seq = word2onehot(fr_tok, 'sos', fr_vocab)
```

# Generating translations

```
fr_sent = ''
for _ in range(fr_len):
  de_prob, de_s_t = decoder.predict([de_seq,de_s_t])
  de_w = probs2word(de_prob, fr_tok)
  de_seq = word2onehot(fr_tok, de_w, fr_vocab)
  if de_w == 'eos': break
  fr_sent += de_w + ' '
```

# Time to translate!

MACHINE TRANSLATION IN PYTHON

# Using word embedding for machine translation

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# Introduction to word embeddings

- One hot encoded vectors

```
cat_vector = np.array([[1,0,0,0...,0]])
dog_vector = np.array([[0,1,0,0...,0]])
window_vector = np.array([[0,0,1,0...,0]])
```

- Word vectors

```
cat_vector = np.array([[0.393,-0.263,0.086,0.011,-0.322,...,0.388]])
dog_vector = np.array([[0.399,-0.300,0.047,-0.059,-0.111,...,0.037]])
window_vector = np.array([[0.133,0.149,-0.307,0.090,-0.143,...,0.526]])
```

# Similarity between word vectors

```
from sklearn.metrics.pairwise import cosine_similarity
cat_vector = np.array([[0.393,-0.263,0.086,0.011,-0.322,...,0.388]])
dog_vector = np.array([[0.399,-0.300,0.047,-0.059,-0.111,...,0.037]])
window_vector = np.array([[0.133,0.149,-0.307,0.090,-0.143,...,0.526]])
```

```
cosine_similarity(cat_vector, dog_vector)
```

```
0.601
```

```
cosine_similarity(cat_vector, window_vector)
```

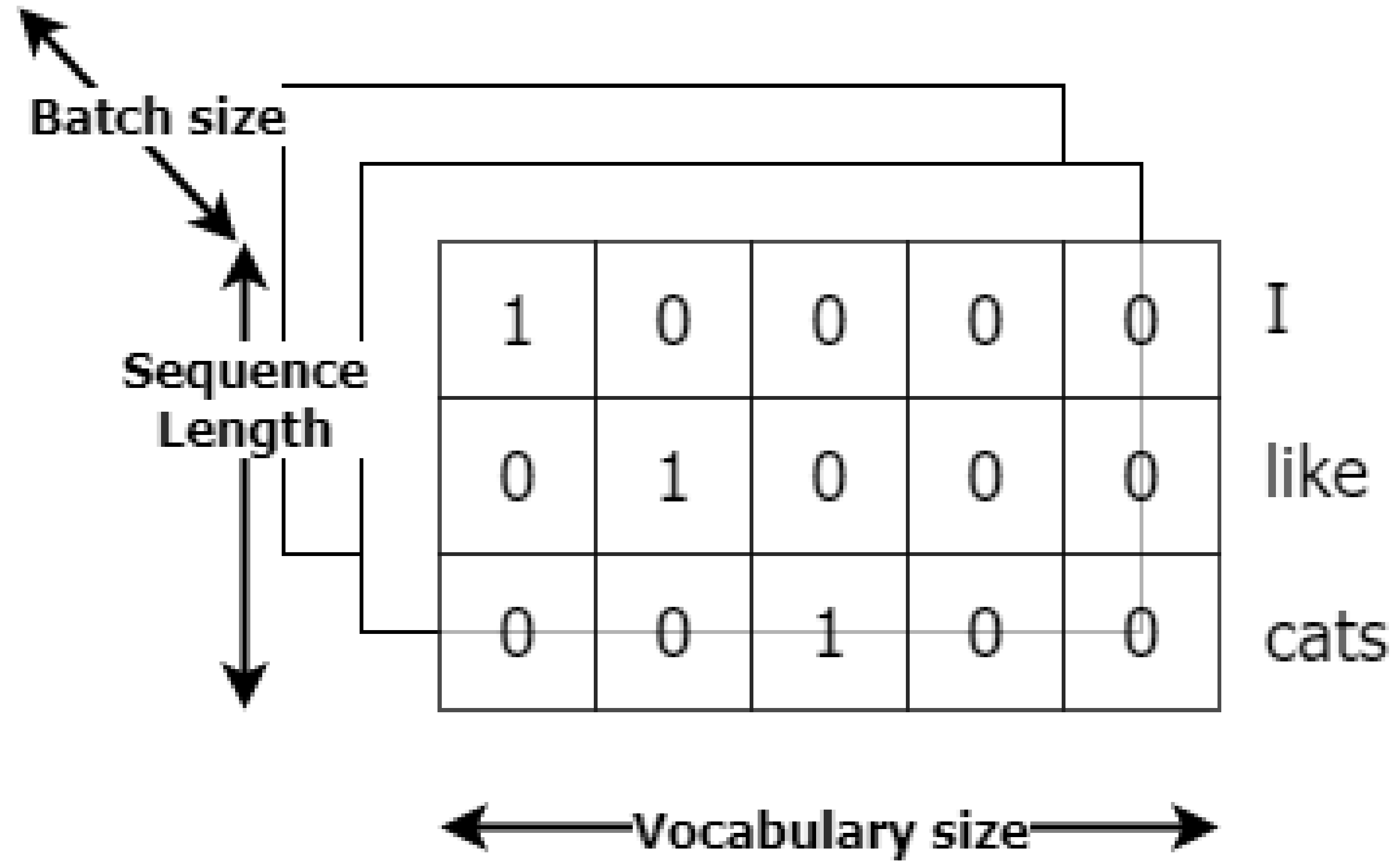
```
0.323
```

[1] https://nlp.stanford.edu/projects/glove/

# Implementing embeddings for the encoder

- Without an embedding layer

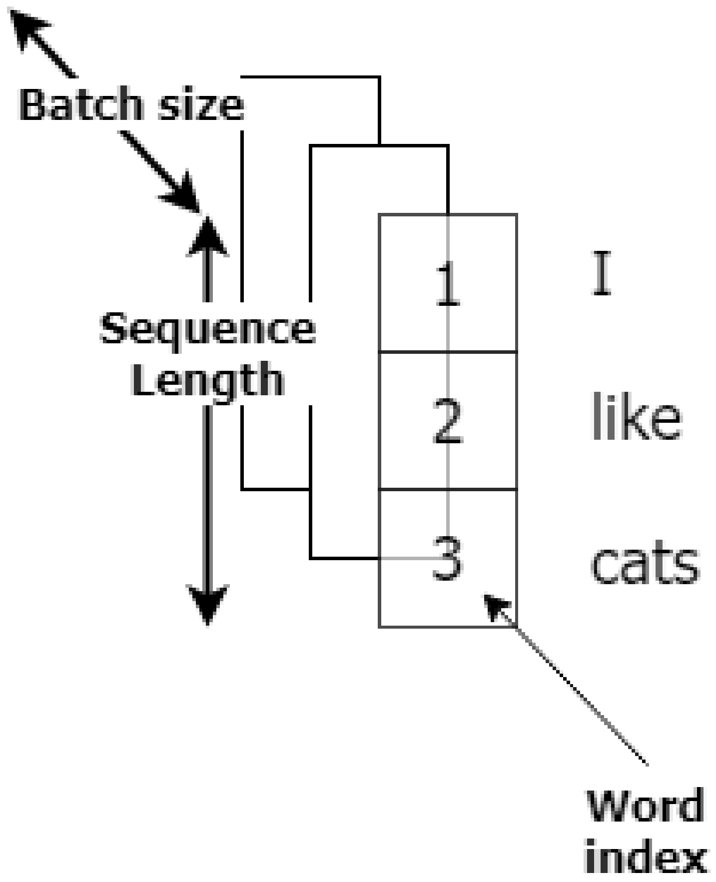
```
en_inputs = Input(shape=(en_len, en_vocab))
```



# Implementing embeddings for the encoder

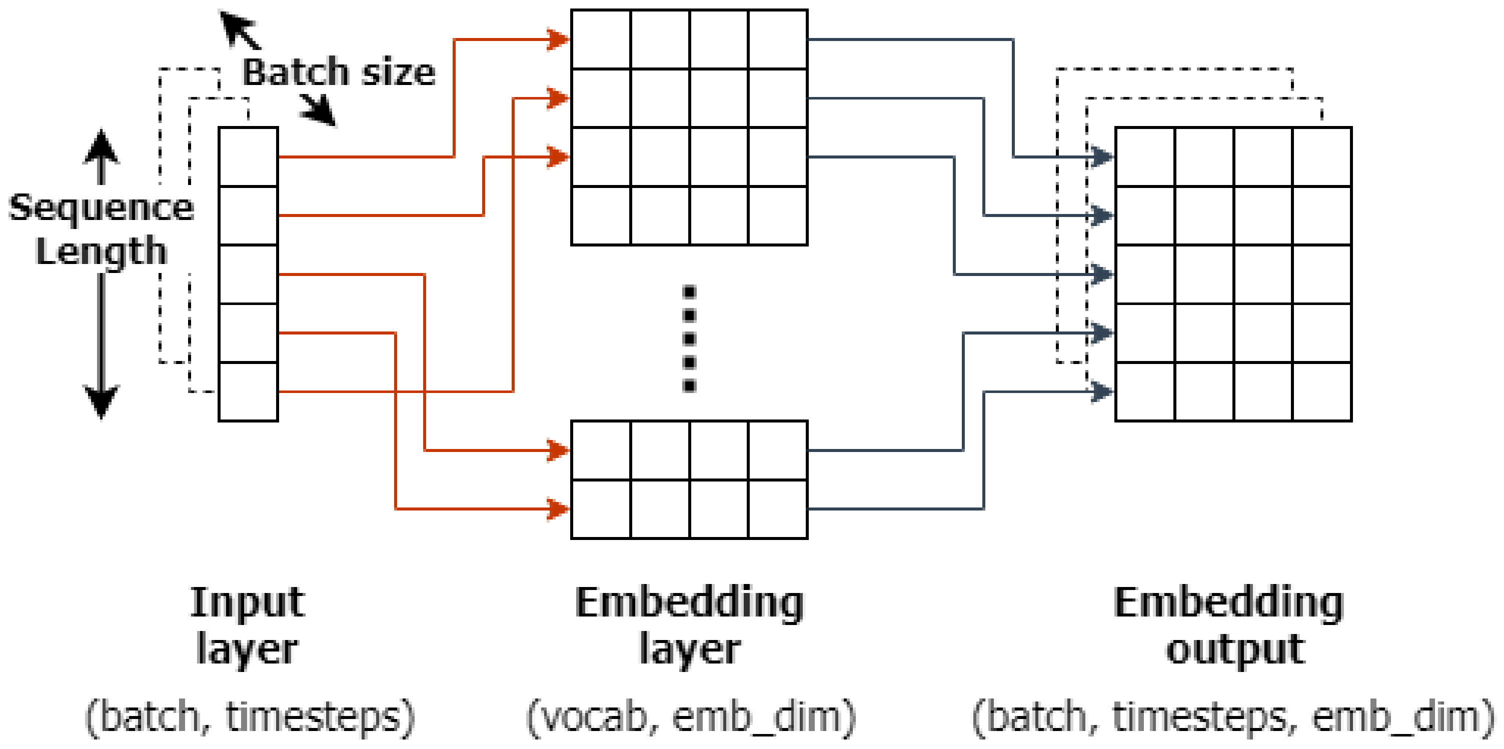
- With an embedding layer

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



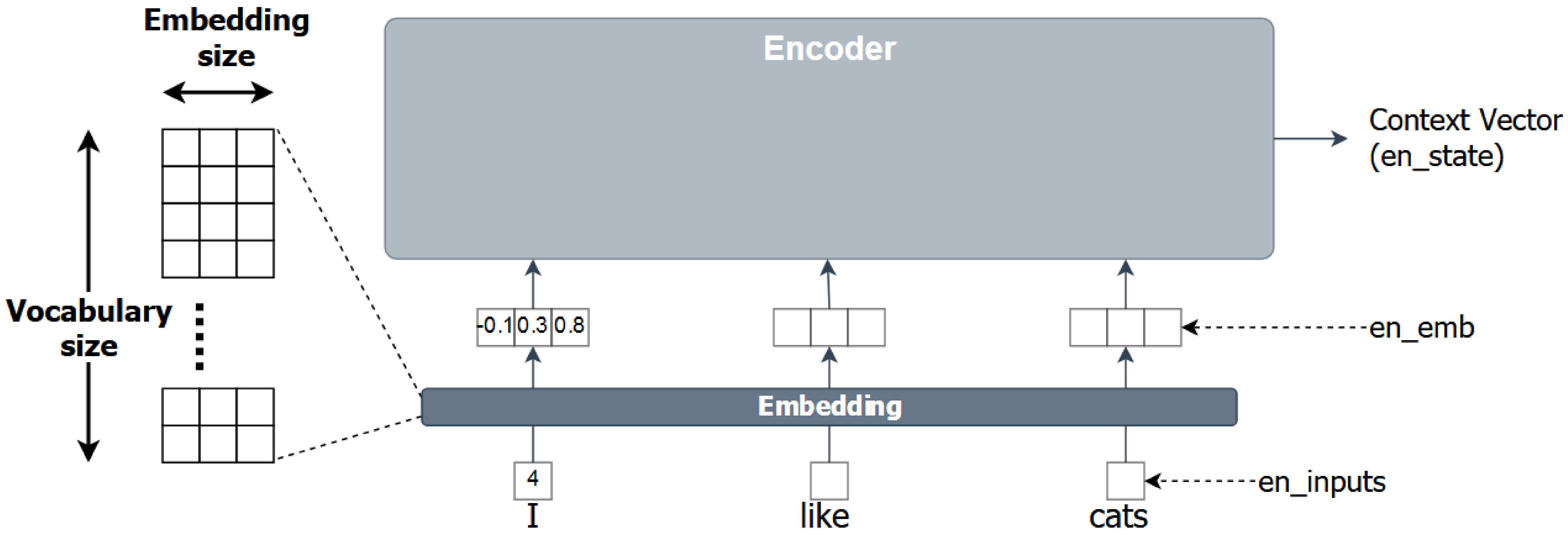
# Implementing embeddings for the encoder

```
en_inputs = Input(shape=(en_len,))
en_emb = Embedding(en_vocab, 96, input_length=en_len)(en_inputs)
```



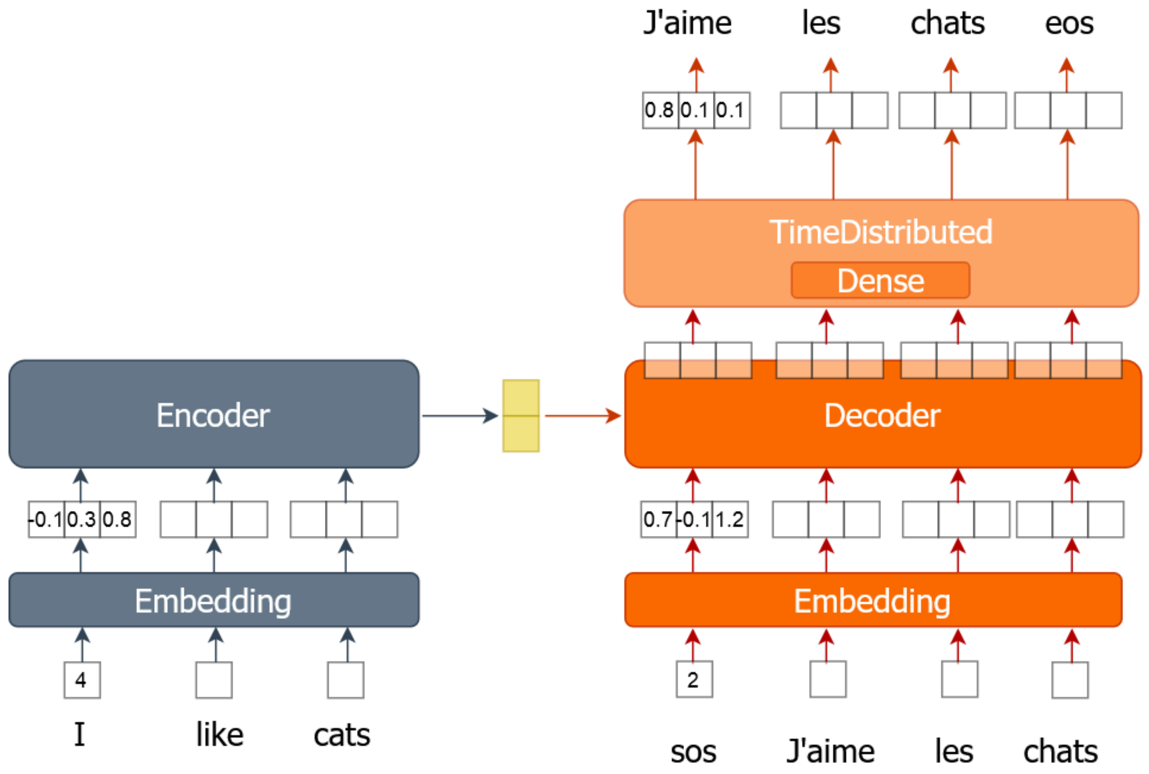
# Implementing the encoder with embedding

```
en_inputs = Input(shape=(en_len,))
en_emb = Embedding(en_vocab, 96, input_length=en_len)(en_inputs)
en_out, en_state = GRU(hsize, return_state=True)(en_emb)
```



# Implementing the decoder with embedding

```
de_inputs = Input(shape=(fr_len-1,))
de_emb = Embedding(fr_vocab, 96, input_length=fr_len-1)(de_inputs)
de_out, _ = GRU(hsize, return_sequences=True, return_state=True(
  de_emb, initial_state=en_state)
```



# Training the model

```
for ei in range(3):
  for i in range(0, train_size, bsize):
    en_x = sents2seqs('source', tr_en[i:i+bsize], onehot=False, reverse=True)
    de_xy = sents2seqs('target', tr_fr[i:i+bsize], onehot=False)
    de_x = de_xy[:,:-1]
    de_xy_oh = sents2seqs('target', tr_fr[i:i+bsize], onehot=True)
    de_y = de_xy_oh[:,1:,:]
    nmt_emb.train_on_batch([en_x, de_x], de_y)
    res = nmt_emb.evaluate([en_x, de_x], de_y, batch_size=bsize, verbose=0)
    print("{} => Loss:{}, Train Acc: {}".format(ei+1,res[0], res[1]*100.0))
```

# Let's practice!

MACHINE TRANSLATION IN PYTHON

# Wrap-up and the final showdown

MACHINE TRANSLATION IN PYTHON

**Thushan Ganegedara**
Data Scientist and Author

# What you've done so far

- Chapter 1
  - Introduction to encoder-decoder architecture
  - Understanding GRU layer
- Chapter 2
  - Implementing the encoder
  - Implementing the decoder
  - Implementing the decoder prediction layer

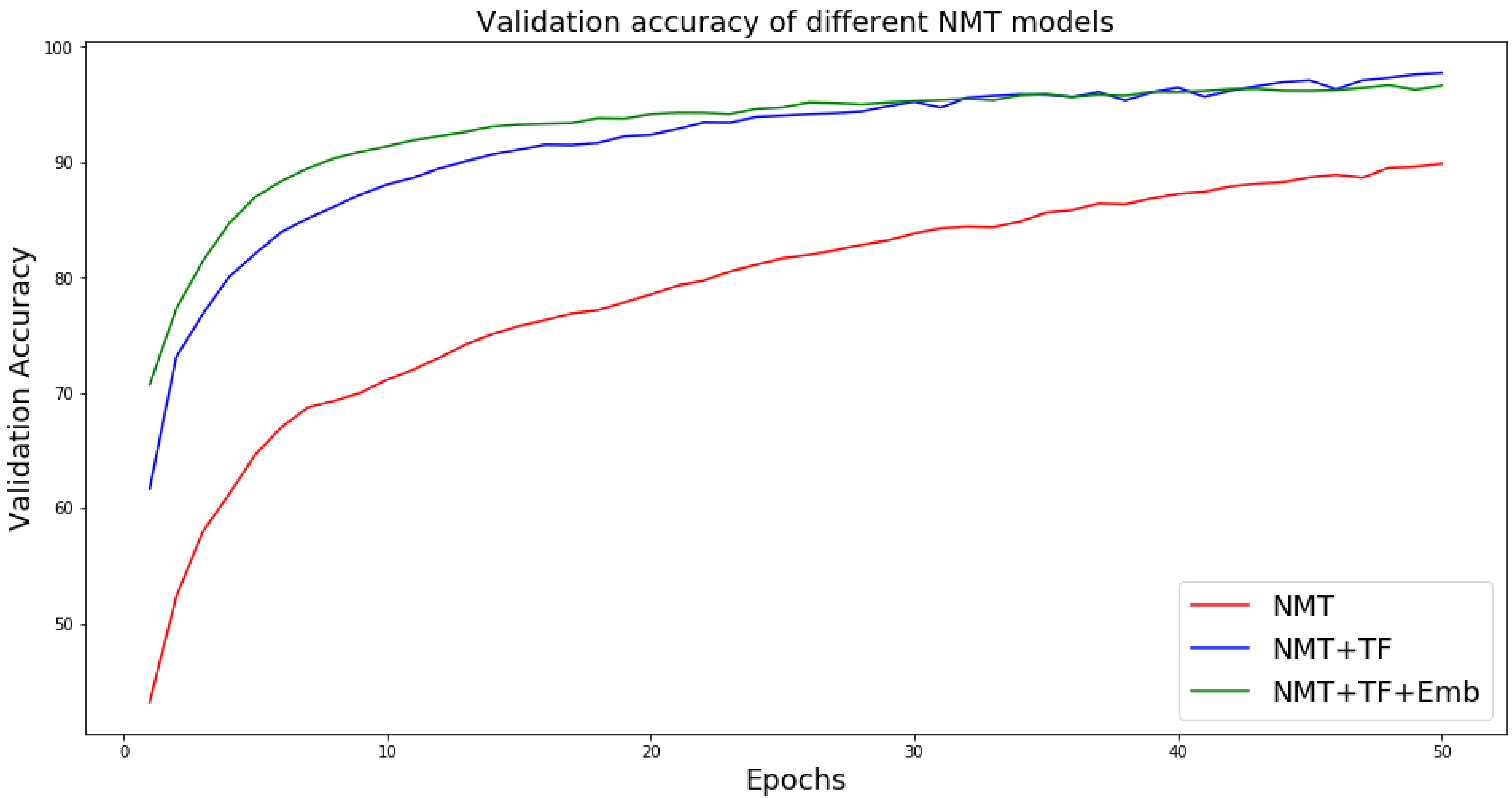
# What you've done so far

- Chapter 3
  - Preprocessing data
  - Training the machine translation model
  - Generating translations
- Chapter 4
  - Introduction to teacher forcing
  - Training a model with teacher forcing
  - Generating translations
  - Using word embeddings for machine translation

# Machine transation models

- Model 1
  - The encoder consumes English words (onehot encoded) and outputs a context vector
  - The decoder consumes the context vector and outputs the translation
- Model 2
  - The encoder consumes English words (onehot encoded) and outputs a context vector
  - The decoder consumes a given word (onehot encoded) of the translation and predicts the next word
- Model 3
  - Instead of onehot encoding, uses word vectors
  - Word vectors capture the semantic relationship between words

# Performance of different models



# Latest developments and further reading

- Evaluating machine translation models
  - BLEU score (**Papineni et al., BLEU: a Method for Automatic Evaluation of Machine Translation.**)
- Word piece models
  - Enables the model to avoid out of vocabulary words (**Sennrich et al., Neural Machine Translation of Rare Words with Subword Units.**)
- Transformer models (**Vaswani et al., Attention Is All You Need**)
  - State-of-the-art performance on many NLP tasks including machine translation
  - Has an encoder-decoder architecture, but does not use sequential models
  - The latest Google machine translator is a Transformer model

# All the best!

MACHINE TRANSLATION IN PYTHON