Deep Learning for Sequential Data (II)

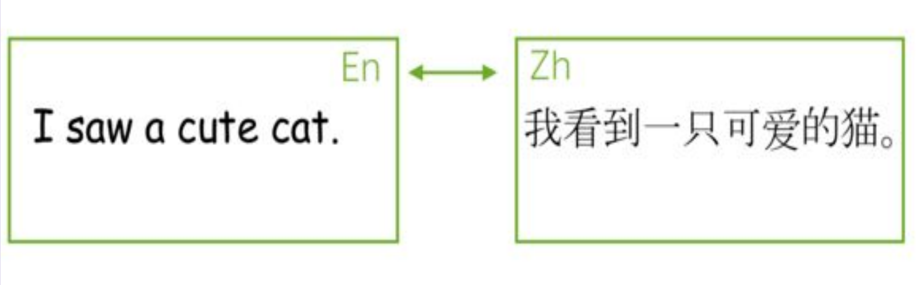
Seq2Seq and Transformers

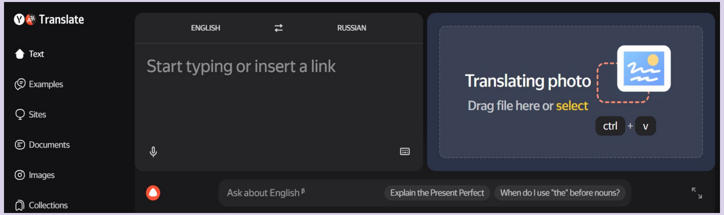
Learning Outcomes

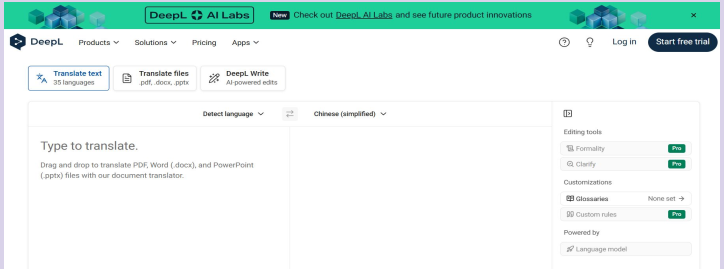
By the end of this week, you'll be able to:

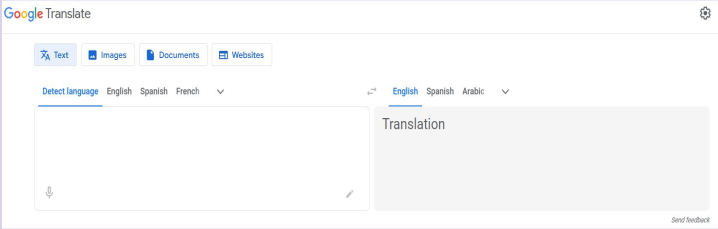
- Understand Seq2Seq and how they might be applied to solve problems such as machine translation, language modeling, and image captioning.
- Understand local and global attention mechanisms in deep models.
- Understand Transformer and BERT, two latest SOTA models for NLP.
- Understand the self-attention mechanism and how it is used in Transformer.

Use Case

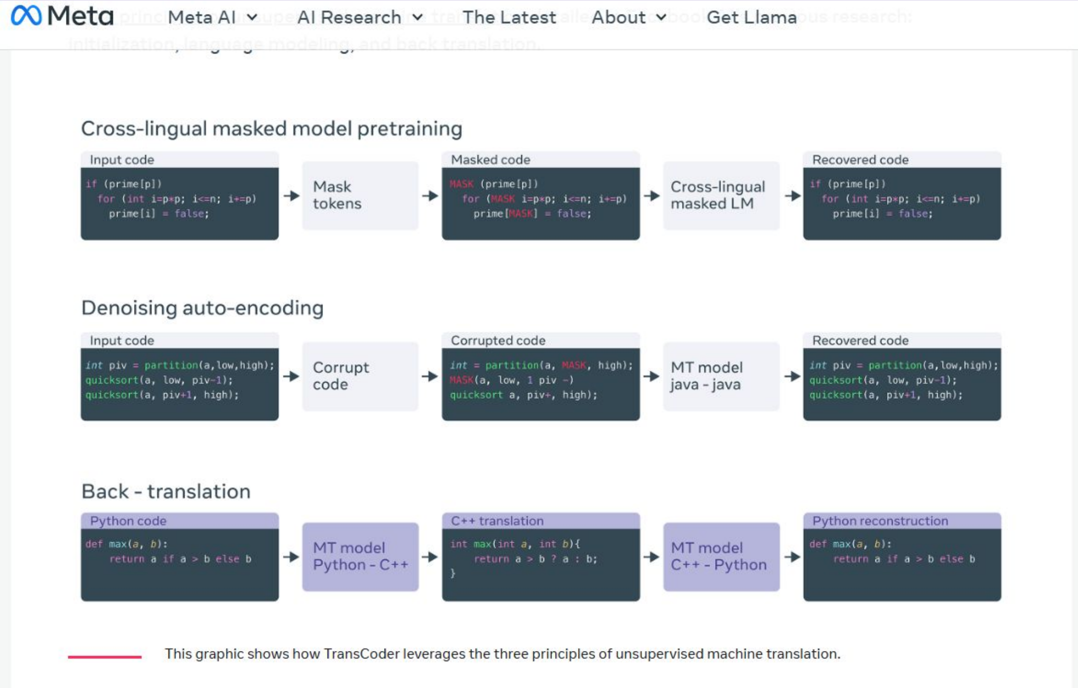

En
I saw a cute cat.
Zh
我看到一只可爱的猫。


Translate
Text
Examples
Sites
Documents
Images
Collections
ENGLISH
RUSSIAN
Start typing or insert a link
Translating photo
Drag file here or select
ctrl + v
Ask about English
Explain the Present Perfect
When do I use "the" before nouns?

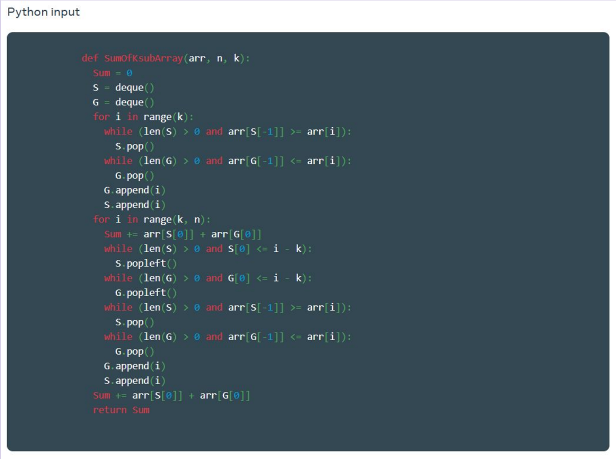

DeepL AI Labs
New Check out DeepL AI Labs and see future product innovations
DeepL
Products
Solutions
Pricing
Apps
Log in
Start free trial
Translate text
35 languages
Translate files
.pdf, .docx, .pptx
DeepL Write
AI-powered edits
Detect language
Chinese (simplified)
Type to translate.
Drag and drop to translate PDF, Word (.docx), and PowerPoint (.pptx) files with our document translator.
Editing tools
Formality
Clarify
Customizations
Glossaries
None set
Custom rules
Powered by
Language model


Google Translate
Text
Images
Documents
Websites
Detect language
English
Spanish
French
English
Spanish
Arabic
Translation
Send feedback

Use Case

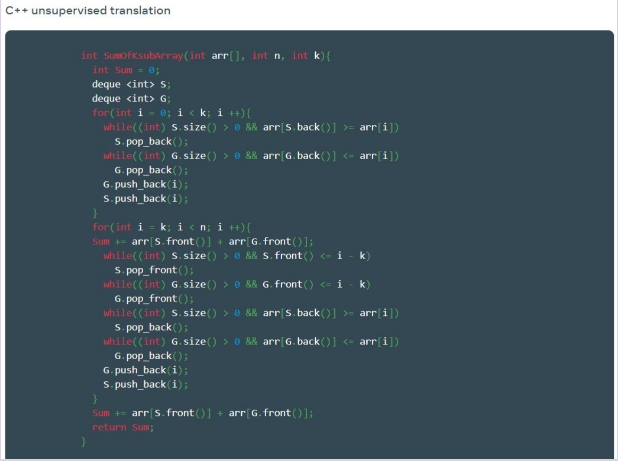


This graphic shows how TransCoder leverages the three principles of unsupervised machine translation.



Python input

```python
def SumOfKsubArray(arr, n, k):
  Sum = 0
  S = deque()
  G = deque()
  for i in range(k):
    while (len(S) > 0 and arr[S[-1]] >= arr[i]):
      S.pop()
    while (len(G) > 0 and arr[G[-1]] <= arr[i]):
      G.pop()
    G.append(i)
    S.append(i)
  for i in range(k, n):
    Sum += arr[S[0]] + arr[G[0]]
    while (len(S) > 0 and S[0] <= i - k):
      S.popleft()
    while (len(G) > 0 and G[0] <= i - k):
      G.popleft()
    while (len(S) > 0 and arr[S[-1]] >= arr[i]):
      S.pop()
    while (len(G) > 0 and arr[G[-1]] <= arr[i]):
      G.pop()
    G.append(i)
    S.append(i)
  Sum += arr[S[0]] + arr[G[0]]
  return Sum
```

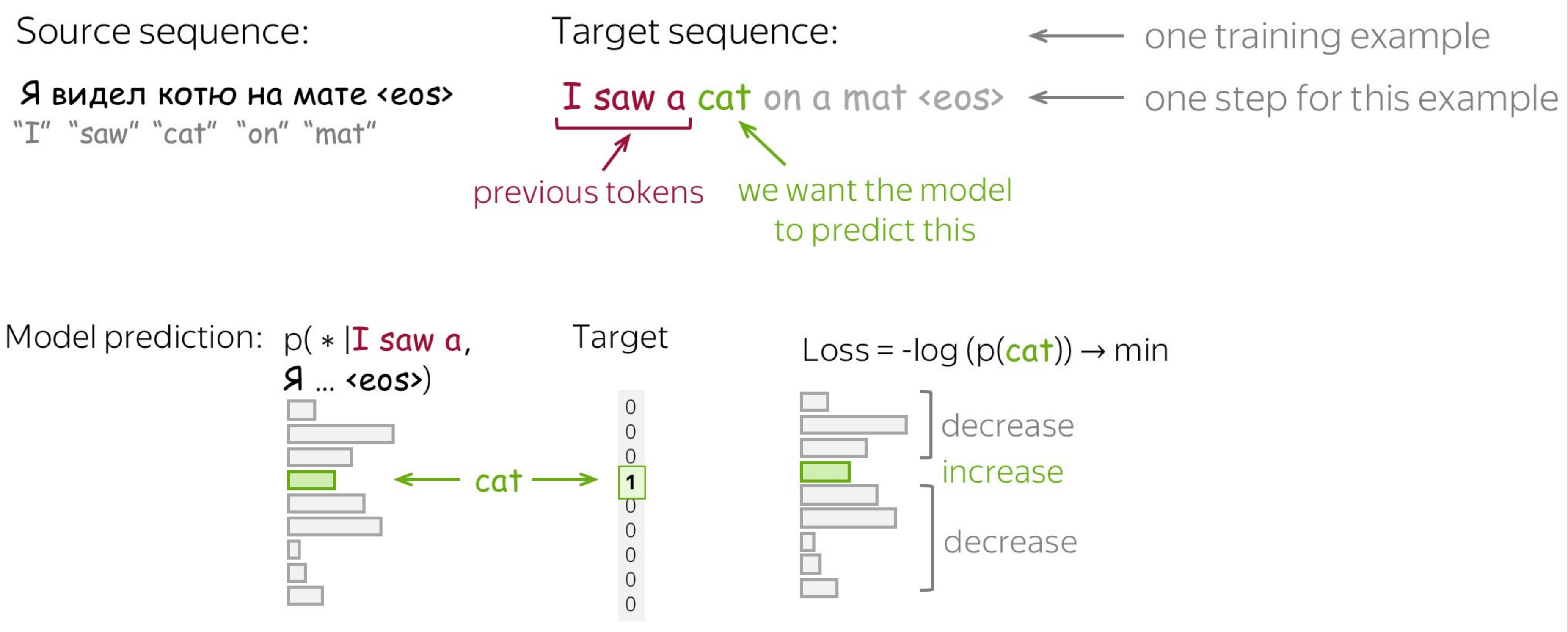


C++ unsupervised translation

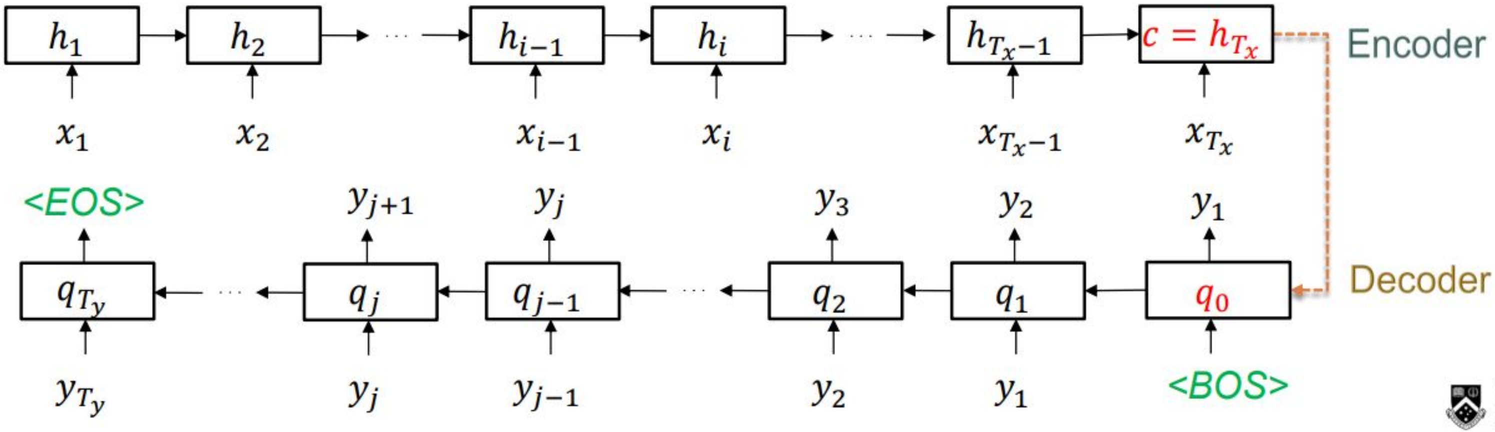
```cpp
int SumOfKsubArray(int arr[], int n, int k){
  int Sum = 0;
  deque <int> S;
  deque <int> G;
  for(int i = 0; i < k; i ++){
    while((int) S.size() > 0 && arr[S.back()] >= arr[i])
      S.pop_back();
    while((int) G.size() > 0 && arr[G.back()] <= arr[i])
      G.pop_back();
    G.push_back(i);
    S.push_back(i);
  }
  for(int i = k; i < n; i ++){
  Sum += arr[S.front()] + arr[G.front()];
    while((int) S.size() > 0 && S.front() <= i - k)
      S.pop_front();
    while((int) G.size() > 0 && G.front() <= i - k)
      G.pop_front();
    while((int) S.size() > 0 && arr[S.back()] >= arr[i])
      S.pop_back();
    while((int) G.size() > 0 && arr[G.back()] <= arr[i])
      G.pop_back();
    G.push_back(i);
    S.push_back(i);
  }
  Sum += arr[S.front()] + arr[G.front()];
  return Sum;
}
```

Sequence and Sequence

Sequence and Sequence

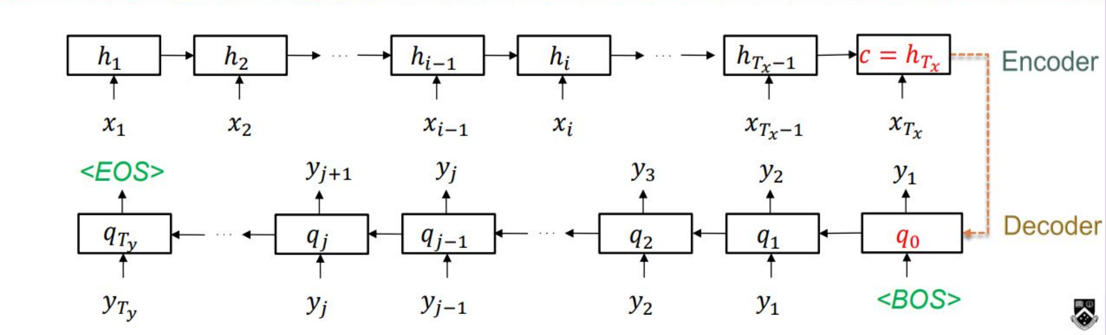


- Encoder
  - Produces the context vector $\boldsymbol{c} = \boldsymbol{h}_{T_x}$ of the input sequence
  - **Context vector** c summarizes input sequence $[\mathbf{x}_1, \ldots, \mathbf{x}_{T_x}]$.
- Decoder
  - Decodes the encode c to the output sequence
- Special symbols
  - <EOS> signifies the end of a sequence
  - <BOS> signifies the beginning of a sequence



MONASH University

Step 1: Dataset Preparation




```
dataset = datasets.load_dataset ("bentrevett/multi30k" )
```

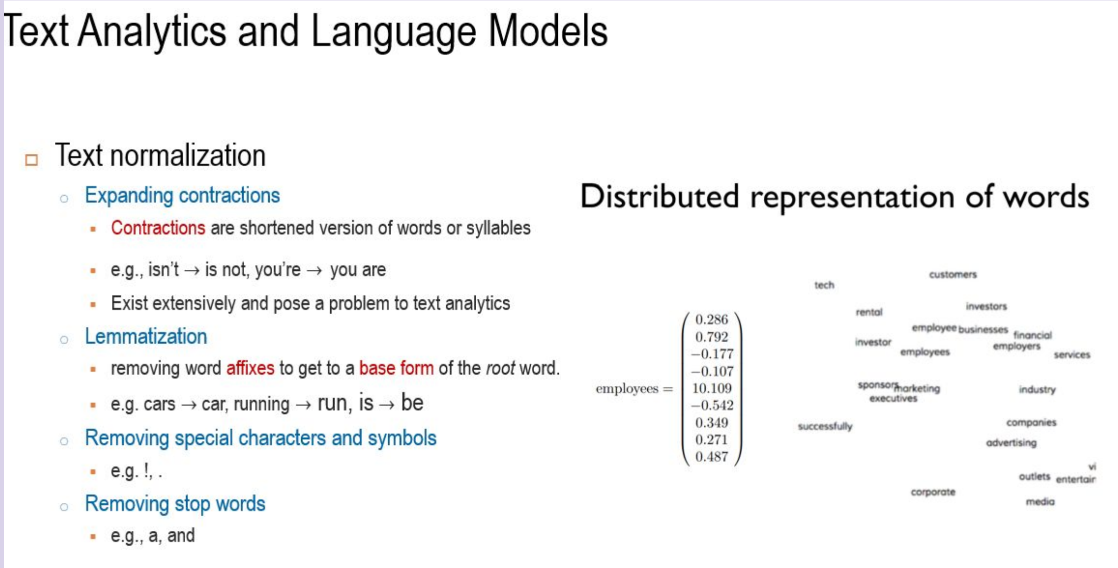
{'en': 'Two young, White males are outside near many bushes.',

'de': 'Zwei junge weiße Männer sind im Freien in der Nähe vieler Büsche.'}

Statistics

```
train
  (en) 29000 sentences, 377534 words, 13.0 words/sent
  (de) 29000 sentences, 360706 words, 12.4 words/sent
  (fr) 29000 sentences, 409845 words, 14.1 words/sent
  (cs) 29000 sentences, 297212 words, 10.2 words/sent
val
  (en) 1014 sentences, 13308 words, 13.1 words/sent
  (de) 1014 sentences, 12828 words, 12.7 words/sent
  (fr) 1014 sentences, 14381 words, 14.2 words/sent
  (cs) 1014 sentences, 10342 words, 10.2 words/sent
test_2016_flickr
  (en) 1000 sentences, 12968 words, 13.0 words/sent
  (de) 1000 sentences, 12103 words, 12.1 words/sent
  (fr) 1000 sentences, 13988 words, 14.0 words/sent
  (cs) 1000 sentences, 10497 words, 10.5 words/sent
test_2017_flickr
  (en) 1000 sentences, 11376 words, 11.4 words/sent
  (de) 1000 sentences, 10758 words, 10.8 words/sent
  (fr) 1000 sentences, 12596 words, 12.6 words/sent
test_2017_mscoco
  (en) 461 sentences, 5239 words, 11.4 words/sent
  (de) 461 sentences, 5158 words, 11.2 words/sent
  (fr) 461 sentences, 5710 words, 12.4 words/sent
```

Step 2: Tokenization

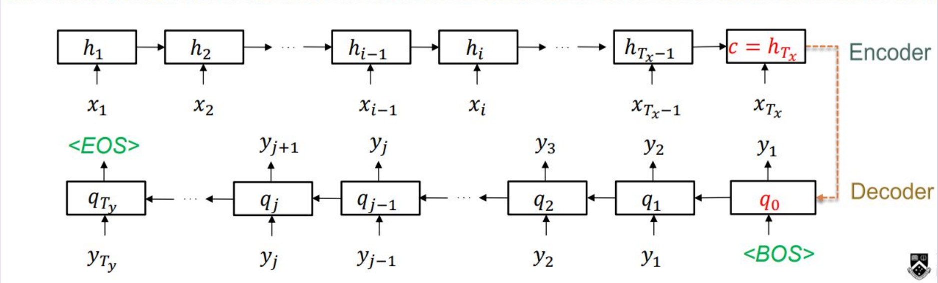



```
import spacy
```

1. A tokenizer is used to turn a string into a list of tokens

2. "good morning!" becomes ["good", "morning", "!"].

Step 2: Tokenization



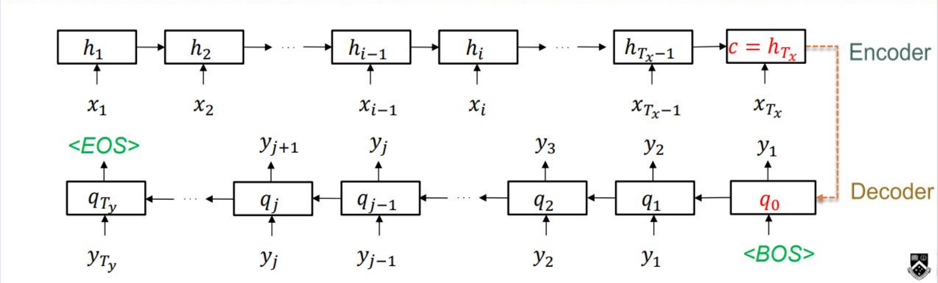

```
import spacy
```

```
en_nlp = spacy.load("en_core_web_sm")
de_nlp = spacy.load("de_core_news_sm")
```

```
string = "What a lovely day it is today!"

[token.text for token in en_nlp.tokenizer(string)]
```

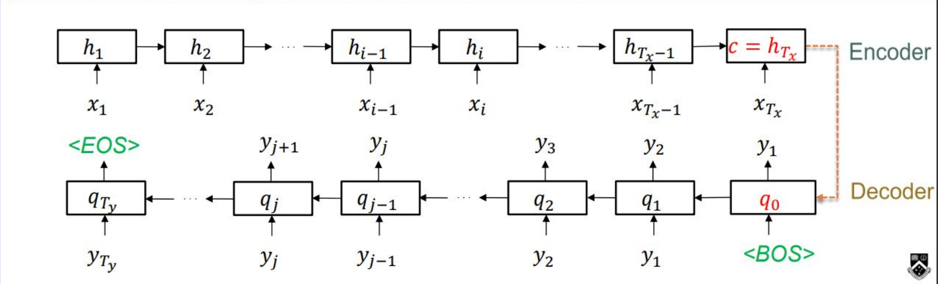
```
['What', 'a', 'lovely', 'day', 'it', 'is', 'today', '!']
```

Step 2: Tokenization




```python
max_length = 1_000
lower = True
sos_token = "<sos>"
eos_token = "<eos>"

fn_kwargs = {
    "en_nlp": en_nlp,
    "de_nlp": de_nlp,
    "max_length": max_length,
    "lower": lower,
    "sos_token": sos_token,
    "eos_token": eos_token,
}

train_data = train_data.map(tokenize_example, fn_kwargs=fn_kwargs)
valid_data = valid_data.map(tokenize_example, fn_kwargs=fn_kwargs)
test_data = test_data.map(tokenize_example, fn_kwargs=fn_kwargs)
```

Step 2: Tokenization




```
{'en': 'Two young, White males are outside near many bushes.',
 'de': 'Zwei junge weiße Männer sind im Freien in der Nähe vieler Büsche.',
 'en_tokens': ['<sos>',
  'two',
  'young',
  ',',
  'white',
  'males',
  'are',
  'outside',
  'near',
  'many',
  'bushes',
  '.',
  '<eos>'],
 'de_tokens': ['<sos>',
  'zwei',
  'junge',
  'weiße',
  'männer',
  'sind',
  'im',
  'freien',
  'in',
  'der',
  'nähe',
  'vieler',
  'büsche',
  '.',
  '<eos>']}
```

Step 2.1 : Vocabulary

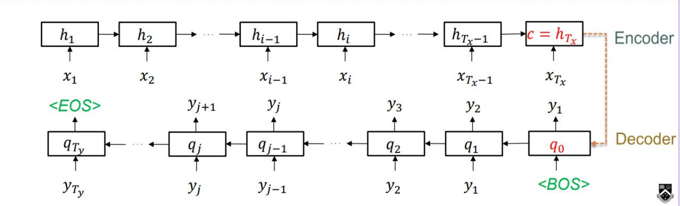
```
{'en': 'Two young, White males are outside near many bushes.',
 'de': 'Zwei junge weiße Männer sind im Freien in der Nähe vieler Büsche.',
 'en_tokens': ['<sos>',
  'two',
  'young',
  ',',
  'white',
  'males',
  'are',
  'outside',
  'near',
  'many',
  'bushes',
  '.',
  '<eos>'],
 'de_tokens': ['<sos>',
  'zwei',
  'junge',
  'weiße',
  'männer',
  'sind',
  'im',
  'freien',
  'in',
  'der',
  'nähe',
  'vieler',
  'büsche',
  '.',
  '<eos>']}
```

Vocabularies

Next, we'll build the *vocabulary* for the source and target languages. The vocabulary is used to associate each unique token in our dataset with an index (an integer), e.g. "hello" = 1, "world" = 2, "bye" = 3, "hates" = 4, etc

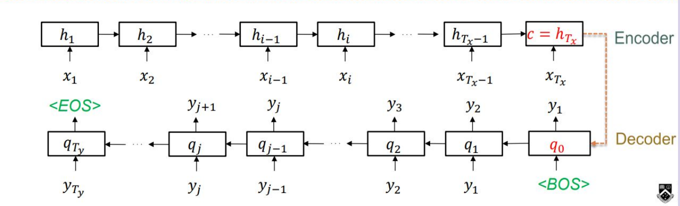
```python
min_freq = 2
unk_token = "<unk>"
pad_token = "<pad>"

special_tokens = [
    unk_token,
    pad_token,
    sos_token,
    eos_token,
]

en_vocab = torchtext.vocab.build_vocab_from_iterator(
    train_data["en_tokens"],
    min_freq=min_freq,
    specials=special_tokens,
)

de_vocab = torchtext.vocab.build_vocab_from_iterator(
    train_data["de_tokens"],
    min_freq=min_freq,
    specials=special_tokens,
)
```

Step 2.2: Encoder




```python
class Encoder(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, n_layers, dropout):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src):
        # src = [src length, batch size]
        embedded = self.dropout(self.embedding(src))
        # embedded = [src length, batch size, embedding dim]
        outputs, (hidden, cell) = self.rnn(embedded)
        # outputs = [src length, batch size, hidden dim * n directions]
        # hidden = [n layers * n directions, batch size, hidden dim]
        # cell = [n layers * n directions, batch size, hidden dim]
        # outputs are always from the top hidden layer
        return hidden, cell
```

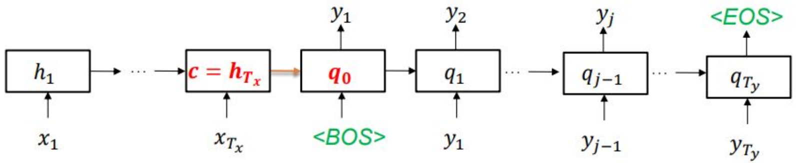
Step 2.4: Decoder




```python
class Decoder(nn.Module):
    def __init__(self, output_dim, embedding_dim, hidden_dim, n_layers, dropout):
        super().__init__()
        self.output_dim = output_dim
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(output_dim, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=dropout)
        self.fc_out = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell):
        # input = [batch size]
        # hidden = [n layers * n directions, batch size, hidden dim]
        # cell = [n layers * n directions, batch size, hidden dim]
        # n directions in the decoder will both always be 1, therefore:
        # hidden = [n layers, batch size, hidden dim]
        # context = [n layers, batch size, hidden dim]
        input = input.unsqueeze(0)
        # input = [1, batch size]
        embedded = self.dropout(self.embedding(input))
        # embedded = [1, batch size, embedding dim]
        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))
        # output = [seq length, batch size, hidden dim * n directions]
        # hidden = [n layers * n directions, batch size, hidden dim]
        # cell = [n layers * n directions, batch size, hidden dim]
        # seq length and n directions will always be 1 in this decoder, therefore:
        # output = [1, batch size, hidden dim]
        # hidden = [n layers, batch size, hidden dim]
        # cell = [n layers, batch size, hidden dim]
        prediction = self.fc_out(output.squeeze(0))
        # prediction = [batch size, output dim]
        return prediction, hidden, cell
```

Step 3: Training

Training of seq2seq



- We need to maximize **the log-likelihood**:

$$\max_{\theta}\left(\mathrm{J}(\theta) = \sum_{(x,y)\in\mathfrak{D}} \log P\,(\boldsymbol{y}|\boldsymbol{x},\theta\,)\right)$$

where $\theta = [\theta_e, \theta_d]$ and θ_e, θ_d are encoding and decoding parameters respectively.

- Product rule:

$$P(\boldsymbol{y}|\boldsymbol{x},\theta\,) = P\left(\boldsymbol{y}_{1:T_y}\;\middle|\;\boldsymbol{x}_{1:T_x},\theta\right) = P\left(\boldsymbol{y}_{1:T_y}\;\middle|\;\boldsymbol{c},\theta\right)$$

$$= P(\boldsymbol{y}_1 \mid \boldsymbol{c},\theta\,)P(\boldsymbol{y}_2 \mid \boldsymbol{y}_1,\boldsymbol{c},\theta\,)\ldots P(\boldsymbol{y}_j \mid \boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta\,)\ldots P\left(\boldsymbol{y}_{T_y}\;\middle|\;\boldsymbol{y}_{1:T_y-1},\boldsymbol{c},\theta\right) = \prod_{j=1}^{T_y} P(\boldsymbol{y}_j \mid \boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta\,)$$

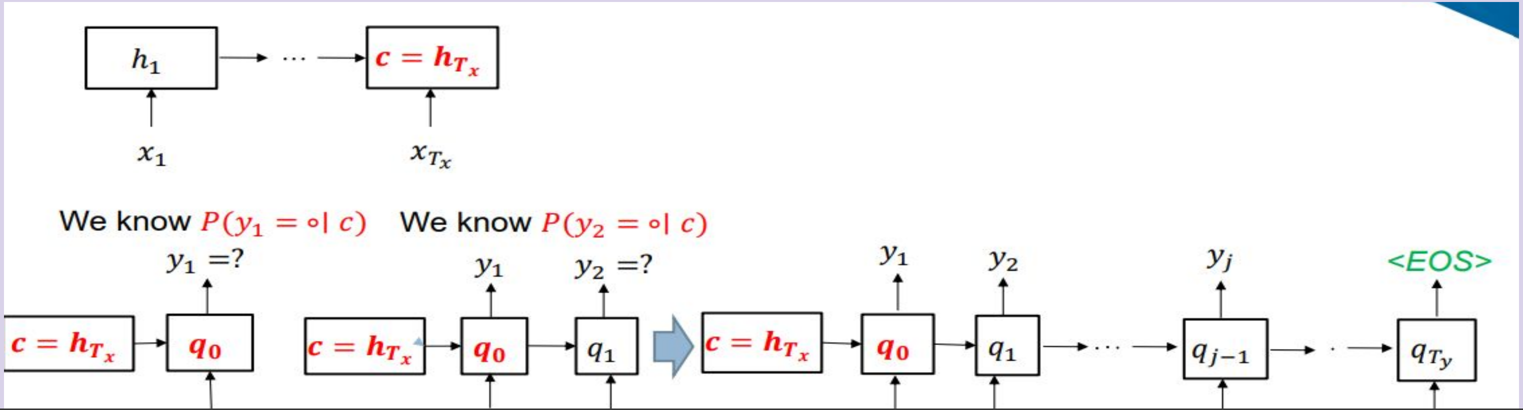
$$\log P\,(\boldsymbol{y}|\boldsymbol{x},\theta\,) = \log P\,(\boldsymbol{y}|\boldsymbol{c},\theta) = \sum_{j=1}^{T_y} \log P\,(\boldsymbol{y}_j|\boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta) = \sum_{j=1}^{T_y} \log P\,(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c},\theta)$$

  - We can compute $P(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c}) = g(\boldsymbol{y}_j,\boldsymbol{q}_{j-1},\boldsymbol{c})$ where g is a **nonlinear**, potentially **multi-layered NN** that outputs the probability of y_j.

- Pay attention on how c is used in every step during decoding

```python
input_dim = len(de_vocab)
output_dim = len(en_vocab)
encoder_embedding_dim = 256
decoder_embedding_dim = 256
hidden_dim = 512
n_layers = 2
encoder_dropout = 0.5
decoder_dropout = 0.5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

encoder = Encoder(
    input_dim,
    encoder_embedding_dim,
    hidden_dim,
    n_layers,
    encoder_dropout,
)

decoder = Decoder(
    output_dim,
    decoder_embedding_dim,
    hidden_dim,
    n_layers,
    decoder_dropout,
)

model = Seq2Seq(encoder, decoder, device).to(device)
```

Step 4: Inference



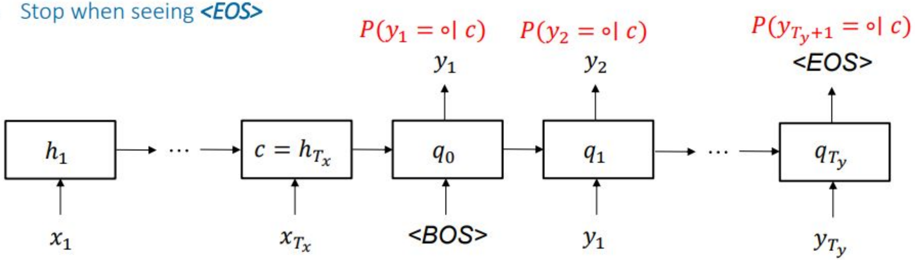

```python
model.load_state_dict(torch.load("tut1-model.pt"))

test_loss = evaluate_fn(model, test_data_loader, criterion, device)

print(f"| Test Loss: {test_loss:.3f} | Test PPL: {np.exp(test_loss):7.3f} |")
```

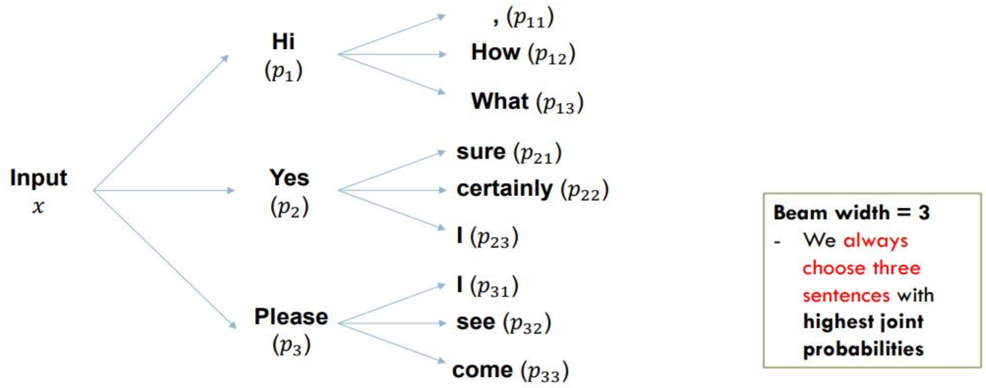
```
Test Loss: 3.780 | Test PPL:  43.833 |
```

Step 4.1: Inference; Greedy Vs Beam

Greedy Decoding

- Given **x**, find word $\boldsymbol{y_1}$ with highest probability
- Given $\boldsymbol{y_1}$ and **x**, find word $\boldsymbol{y_2}$ with highest probability
- ...
- Stop when seeing ***<EOS>***





Drawback of fixed context

- **Fixed** context vector c is **easily overwhelmed** by long inputs or long outputs.
- At a specific timestep j, some **words** or **items** in the **input sequence** might **possibly contribute more** to the **generation** of **next item** or **word** in the output sequence.
  - I want to see you every day → Je veux te voir chaque jour
  - I want to see **you** every day → Je veux te **?** (voir) ....
- How to **timely adapt** the **context vector** c_j?
  - $c_j = \alpha(h_1, .., h_{T_x}, q_{j-1})$
  - Computed using **attention mechanism**

Published as a conference paper at ICLR 2015

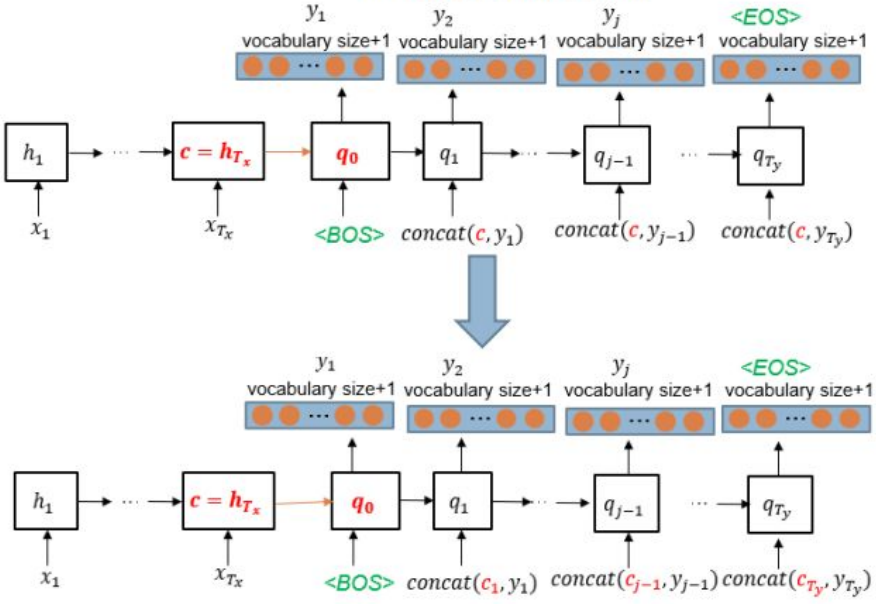
NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

Paper: paper link

Fixed context vector c



Timely varied context vectors c_j

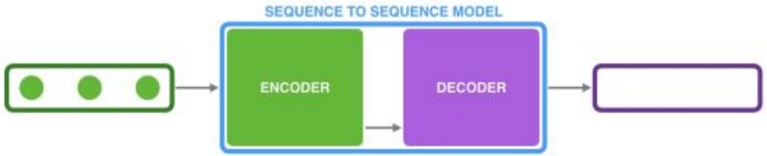
Attention is all you need

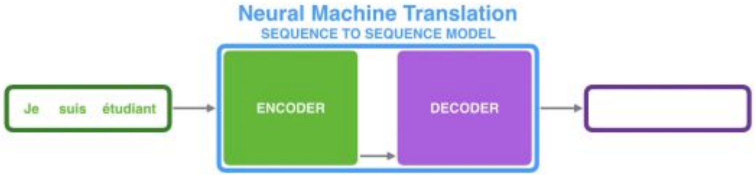
Attention mechanism

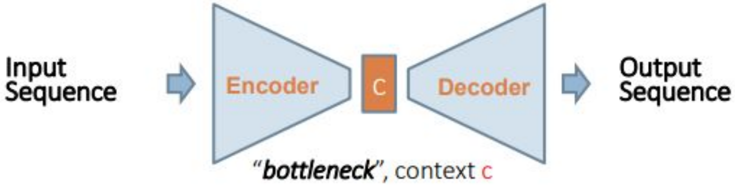
- So far, the input sequence is **summarised** by a **single** context c !

- **Fixed-length context** could be problematic as it is **easily overwhelmed** by long inputs or long outputs
  - The **fixed-length** context **c** might **not be powerful enough** to capture long input sequences

- Some **specific items** in an input sequence might be **more relevant** and **contributing in generating** a given item in output sequence.







- **Gratefully acknowledge the excellent visualizations used from Jay Alammar blog at:**
  - https://jalammar.github.io/
  - https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention mechanism

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

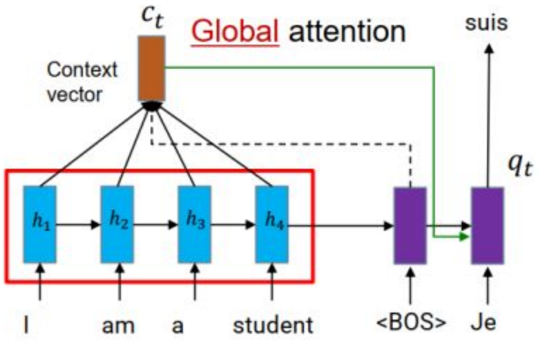
KyungHyun Cho Yoshua Bengio*
Université de Montréal

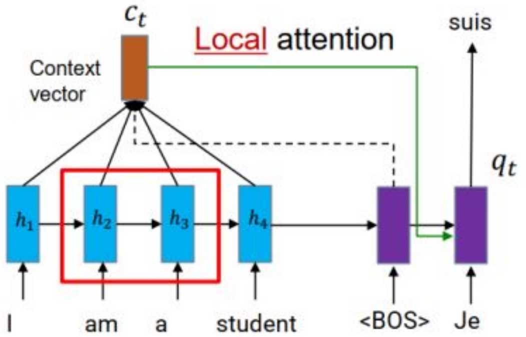
Effective Approaches to Attention-based Neural Machine Translation

Minh-Thang Luong Hieu Pham Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
{lmthang,hyhieu,manning}@stanford.edu

Bahdanau, Cho, Bengio, [Neural Machine Translation by Jointly Learning to Align and Translate](), ICLR 2015

Luong, Pham, Manning, [Effective Approach Attention-based Neural Machine Translation](), EMNLP, 2015





- **Attention mechanism allows the decoding network to refer to the input.**
  - **Global attention**
    - Use **all input hidden states** of the encoder when deriving the context c_t.
  - **Local attention**
    - Use a **selective window of input hidden states** of the encoder when deriving the context c_t.