NATURAL LANGUAGE PROCESSING

Jeff Prosise

Natural Language Processing (NLP)

- Using deep-learning models to process human language

  - **Text and Document Classification**
  - **Named Entity Recognition**
  - **Keyword Extraction**
  - **Question Answering**
  - **Neural Machine Translation**

- Capabilities have grown exponentially in recent years thanks to new neural architectures, including transformer encoder-decoders
- Key concepts: Word embeddings, neural attention, and self-attention

Tokenizing Text

```
lines = ['Quick brown fox', # Stop words removed
         'Jumps over lazy lazy brown dog']

tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
vectors = tokenizer.texts_to_matrix(lines)
```

	brown	lazy	quick	fox	jumps	over	dog
0	1	0	1	1	0	0	0
0	1	1	0	0	1	1	1

Vocabulary

brown	1
lazy	2
quick	3
fox	4
jumps	5
over	6
dog	7

Turning Text into Sequences

```
lines = ['Quick brown fox', # Stop words removed
         'Jumps over lazy lazy brown dog']

tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
sequences = tokenizer.texts_to_sequences(lines)
padded_sequences = pad_sequences(sequences, 5)
```

Vocabulary

brown	1
lazy	2
quick	3
fox	4
jumps	5
over	6
dog	7

5

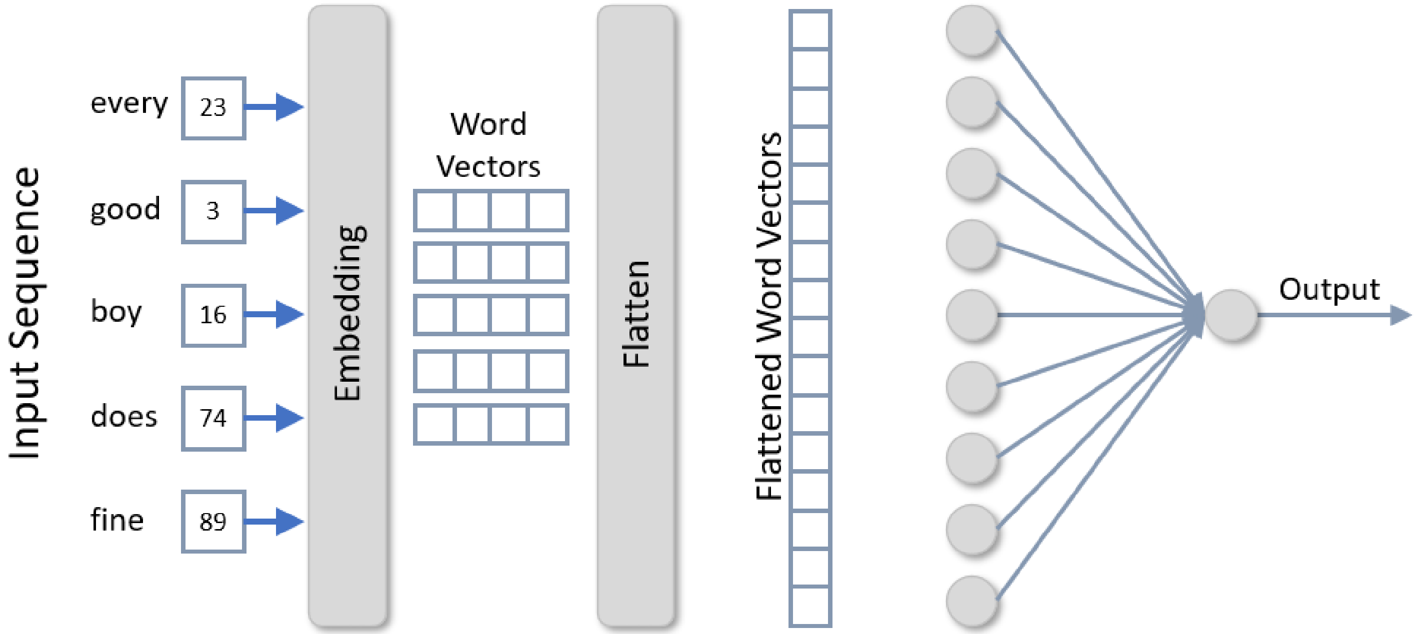
0	0	3	1	4
6	2	2	1	7

Padded Sequences

Word Embeddings

- Embedding layers turn sequences of word indexes into arrays of *word vectors*, which encode information about relationships between words
- Keras makes this easy with its **Emdedding** class

Lines of text are input as **sequences**, which are arrays of integers representing individual words (e.g., indices into a dictionary). An **embedding layer** transforms integers representing words into **word vectors**, or arrays of floating-point numbers. Word vectors encode information about **relationships between words**, such as the fact that both "excellent" and "amazing" express positive sentiment.



Using an Embedding Layer

```
model = Sequential()
model.add(Embedding(10000, 32, input_length=500))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=50)
```

Vocabulary length = 10,000
Output dimension = 32
Length of each sequence = 500

Making Predictions

```
sequence = tokenizer.texts_to_sequences(['Can you attend a code review on Tuesday?'])
padded_sequence = pad_sequences(sequence, maxlen=500)
model.predict(padded_sequence)
```

Demo

Spam Filtering

atmosera®

Gold Microsoft Partner Microsoft | Azure Expert MSP

Automating Text Vectorization

```
import tensorflow as tf

model = Sequential()
model.add(InputLayer(input_shape=(1,), dtype=tf.string))
model.add(TextVectorization(max_tokens=10000, output_sequence_length=500))
model.add(Embedding(10000, 32, input_length=500))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.layers[0].adapt(x)
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=50)
```

Using *n*-Grams

```
import tensorflow as tf

model = Sequential()
model.add(InputLayer(input_shape=(1,), dtype=tf.string))
model.add(TextVectorization(max_tokens=10000, output_sequence_length=500, ngrams=2))
model.add(Embedding(10000, 32, input_length=500))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.layers[0].adapt(x)
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=50)
```

Making Predictions

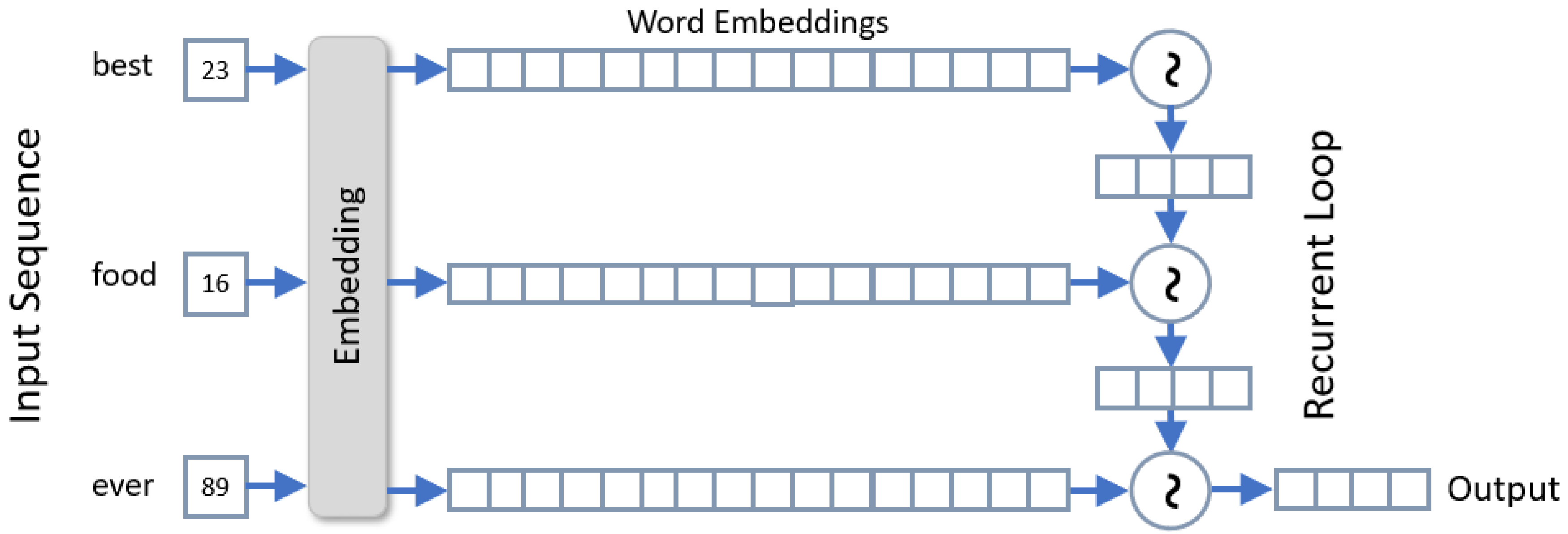
```
model.predict(['Can you attend a code review on Tuesday?'])
```

Demo

Sentiment Analysis

Recurrent Neural Networks

- Carry information (context) forward as a sequence is processed



Tokenized phrase is input to the embedding layer as a **sequence of word indexes**

Each token is converted into a **word embedding**, which is an array (vector) of floating-point values modeling each word's **relationship to other words**

A recurrent layer **loops** through the word embeddings in the sequence, computing a value for each that **factors in the output from the previous iteration**

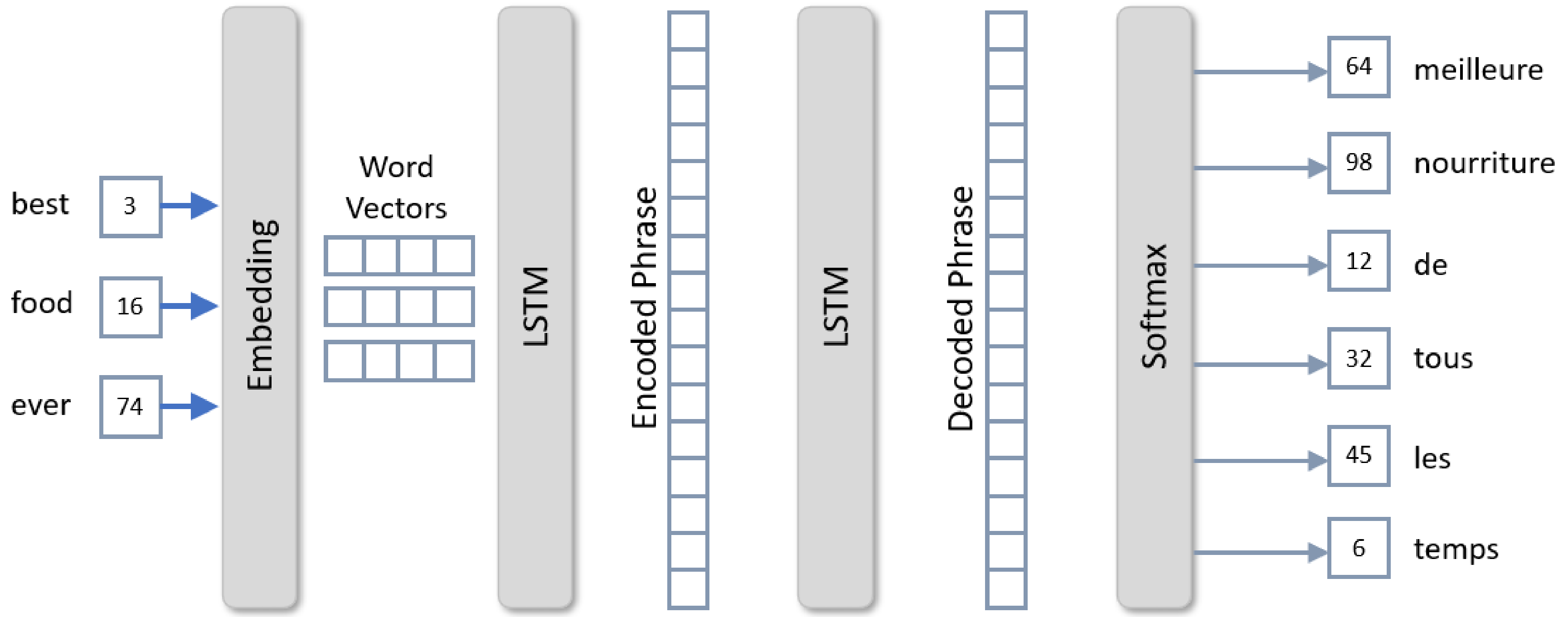
Using LSTM

```
model = Sequential()
model.add(InputLayer(input_shape=(1,), dtype=tf.string))
model.add(TextVectorization(max_tokens=10000, output_sequence_length=500))
model.add(Embedding(10000, 32, input_length=500))
model.add(LSTM(32, return_sequences=True)) # Generate output for next LSTM layer
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.layers[0].adapt(x)
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=50)
```

Neural Machine Translation (NMT)

- NMT models translate text from one language to another
  - Superior to rules-based machine translation (RBMT)
  - Superior to statistical machine translation (SMT)
- Accomplished today with either (or a hybrid) of two architectures
  - LSTM encoder-decoders (prevalent 2012 to 2017)
  - Transformer encoder-decoders (2017 to present)
- Both are <u>sequence-to-sequence</u> models that accept a tokenized sequence as input and generate a tokenized sequence as output
  - For example, tokenized English sentence -> tokenized French sentence

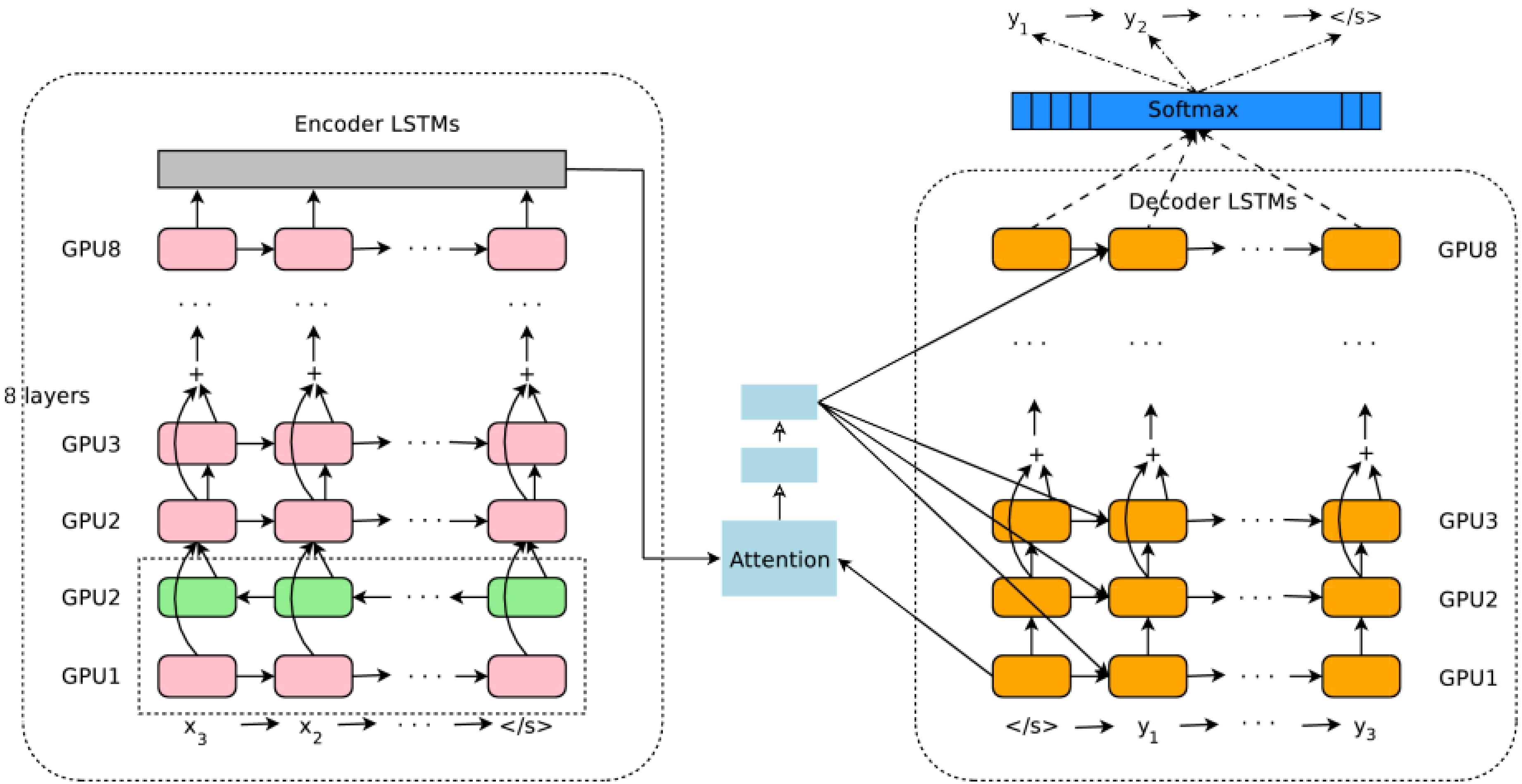
LSTM Encoder-Decoder Architecture



Tokenized input is **transformed to word vectors** by an embedding layer. Word vectors are input to an **LSTM encoder** that produces a dense vector representation of the input phrase.

An **LSTM decoder** transforms the vector generated by the encoder into a dense vector representing the translated phrase. A **softmax output layer** translates the vector into a **set of probabilities**. The word assigned to each position in the output sequence is the word in the vocabulary **assigned the highest probability**.

Google Translate circa 2016



"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation" (https://arxiv.org/abs/1609.08144) atmosera®

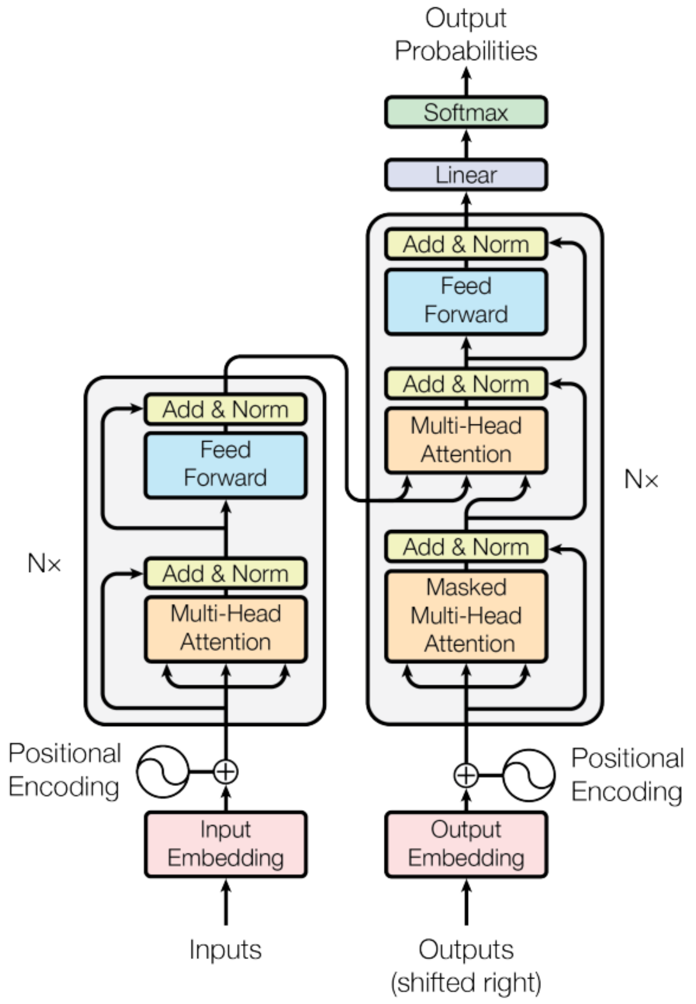
Gold Microsoft Partner Microsoft | Azure Expert MSP

Demo

NMT with LSTM Encoder-Decoders

Transformers

- Introduced in landmark 2017 paper "Attention is All You Need"
- Replaced **LSTM** layers with self-attention (multi-head attention) layers
  - Focuses on words that are most important
  - Discerns between different meanings of the same word (polysemy)
  - Connects pronouns to subjects
- The basis for virtually all state-of-the-art NLP models today



KerasNLP

- Contains classes for building transformer-based deep-learning models
  - **TokenAndPositionEmbedding** class implements positional embedding layers
  - **TransformerEncoder** class implements transformer encoders that include multi-head attention modules for self-attention
  - **TransformerDecoder** class implements transformer decoders that include multi-head attention modules for self-attention
  - **WordPieceTokenizer** class tokenized input for BERT models
- Free, open-source, and by the same team that brought you Keras

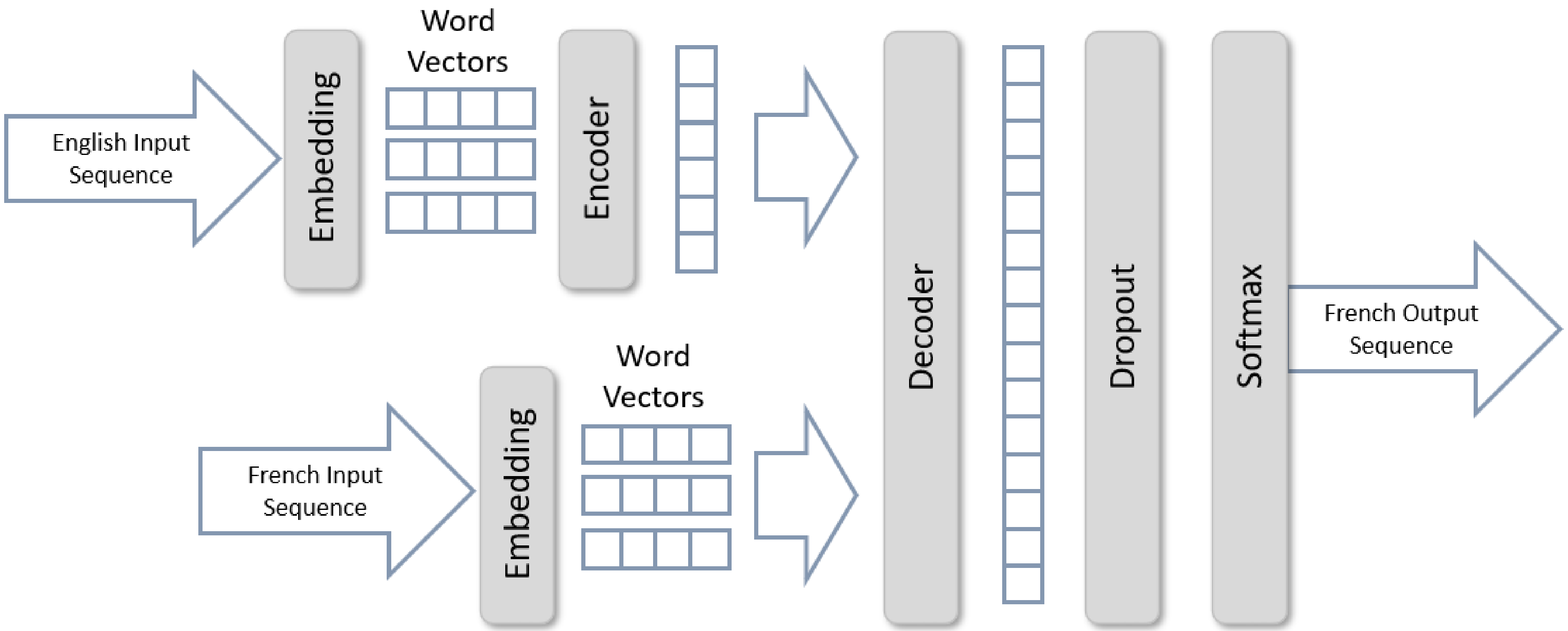
https://keras.io/api/keras_nlp/

Sentiment Analysis with KerasNLP

```
from keras_nlp.layers import TokenAndPositionEmbedding, TransformerEncoder

model = Sequential()
model.add(InputLayer(input_shape=(1,), dtype=tf.string))
model.add(TextVectorization(max_tokens=10000, output_sequence_length=500))
model.add(TokenAndPositionEmbedding(vocabulary_size=10000, sequence_length=500,
                                    embedding_dim=128))
model.add(TransformerEncoder(intermediate_dim=128, num_heads=3))
model.add(GlobalAveragePooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.layers[0].adapt(x)
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=50)
```

Transformer Encoder-Decoder Architecture



The model has **two inputs**: one for **tokenized English input**, and another for **tokenized French input**. Embedding layers translate each into word vectors.

The decoder translates the inputs into an output sequence, and a softmax output layer predicts the **next word in the French sequence**. The next word is appended to the text predicted thus far and **fed back into the model** to predict the **next word**. The cycle repeats until the **entire English phrase** has been translated.

Translating Text

hello world

[start]

salut (65.05%)
bonjour (18.31%)
change (3.88%)
faites (0.76%)
bravo (0.73%)

Translating Text, Cont.

hello	**world**
[start]	**salut**

le (83.57%)
les (5.76%)
des (3.77%)
du (1.69%)
[end] (1.61%)

Translating Text, Cont.

Translating Text, Cont.

hello	world			
[start]	salut	le	monde	

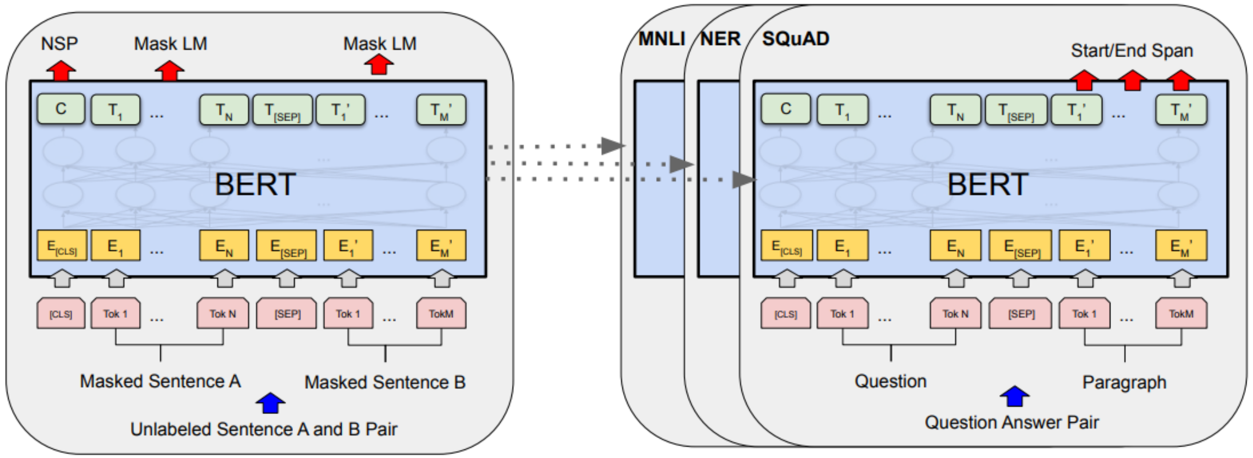
[end] (99.18%)
est (0.53%)
se (0.08%)
a (0.04%)
le (0.03%)

Demo

NMT with Transformer Encoder-Decoders

BERT

- Bidirectional Encoder Representations from Transformers (BERT)
- Built by Google and instilled with language understanding by pretraining with billions of words and phrases
- Can be fine-tuned to perform a variety of NLP tasks



Pre-training

Fine-Tuning

Masked Language Modeling (MLM)

- Turns unlabeled text into training ground for language structure
- During training, a specified percentage (usually 15%) of the tokens are randomly masked (dropped) from the training sequences, and the model is trained to predict the missing words

?	and	seven	years	**?**	our
fathers	**?**	forth	on	this	continent
a	new	**?**	conceived	in	liberty
and	dedicated	**?**	the	proposition	**?**

Hugging Face Transformers

- Thousands of pretrained transformer models for image classification, object detection, neural machine translation, question answering, text classification, document summarization, and more
  - Models are free and many can be used as-is (without additional training)
  - Also includes several pretrained BERT models that can be used as-is or fine-tuned to perform domain-specific tasks
- Install Python **transformers** package, which also requires TensorFlow or PyTorch

https://huggingface.co/

Demo

Fine-Tuning BERT

Analyzing Sentiment

```
from transformers import pipeline

model = pipeline('sentiment-analysis')
result = model('Great food and excellent service!')

# Output: [{'label': 'POSITIVE', 'score': 0.9998843669891357}]
```

Translating Text

```
from transformers import AutoTokenizer, TFAutoModelForSeq2SeqLM

# Initialize a tokenizer and model for translating Dutch to English
tokenizer = AutoTokenizer.from_pretrained('Helsinki-NLP/opus-mt-nl-en')
model = TFAutoModelForSeq2SeqLM.from_pretrained('Helsinki-NLP/opus-mt-nl-en',
                                                from_pt=True)
# Tokenize the input text
text = 'Hallo vrienden, hoe gaat het vandaag?'
tokenized_text = tokenizer.prepare_seq2seq_batch([text], return_tensors='tf')

# Translate the text and decode the output
translation = model.generate(**tokenized_text)
translated_text = tokenizer.batch_decode(translation, skip_special_tokens=True)[0]
# Output: 'Hello, friends. How are you today?'
```

Captioning an Image

```python
import torch
from transformers import ViTFeatureExtractor, AutoTokenizer, VisionEncoderDecoderModel

def predict(image, extractor, tokenizer, model):
    pixels = extractor(images=image, return_tensors='pt').pixel_values

    with torch.no_grad():
        ids = model.generate(pixels, max_length=16, num_beams=4,
                             return_dict_in_generate=True).sequences

    preds = tokenizer.batch_decode(ids, skip_special_tokens=True)
    preds = [pred.strip() for pred in preds]
    return preds

loc = 'ydshieh/vit-gpt2-coco-en'
feature_extractor = ViTFeatureExtractor.from_pretrained(loc)
tokenizer = AutoTokenizer.from_pretrained(loc)
model = VisionEncoderDecoderModel.from_pretrained(loc)
predict(image, feature_extractor, tokenizer, model)
```

a train traveling over a bridge over a river

Optical Character Recognition


```python
from PIL import Image
from transformers import TrOCRProcessor, VisionEncoderDecoderModel

image = Image.open('license-plate.png').convert('RGB')
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-large-printed')
model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-large-printed')

pixel_values = processor(images=image, return_tensors='pt').pixel_values
generated_ids = model.generate(pixel_values)
generated_text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]

# 6TRJ244
```

Generating a Text Embedding

```
from transformers import AutoTokenizer, TFAutoModel

bert_id = 'sebastian-hofstaetter/distilbert-dot-margin_mse-T2-msmarco'
bert_tokenizer = AutoTokenizer.from_pretrained(bert_id)
bert_model = TFAutoModel.from_pretrained(bert_id, from_pt=True)

def get_embedding(text):
    tokenized_text = bert_tokenizer(text, return_tensors='tf')
    embedding = bert_model(tokenized_text)[0][:, 0, :][0]
    return embedding
```

Comparing Embedding Vectors

```python
import numpy as np

v1 = get_embedding('My name is Jeff')
v2 = get_embedding('My wife\'s name is Lori')
np.dot(v1, v2) # 104.78082

v1 = get_embedding('My name is Jeff')
v2 = get_embedding('Where is the bathroom?')
np.dot(v1, v2) # 93.94092
```

Demo

Question Answering

atmosera®

Gold Microsoft Partner Microsoft

Azure Expert MSP