Machine Learning: Algorithms and Applications CSCI933/433

Philip O. Ogunbona

Advanced Multimedia Research Lab
University of Wollongong

Transformer

Outline

Language modelling

Why model language?

- It allows us to represent and understand language in a computational way.
- Word model captures meaning, context and relationships between words in a given sentence.
- It enables machines to understand, generate, and analyze human language.

- Example applications of word modelling in natural language processing (NLP):
  - Semantic understanding: Words carry meaning and to comprehend the meaning of a sentence or document we need understand individual words - model the words
  - Language generation: (includes machine translation, text summarization, and machine-human dialogue/conversation) word modelling allows the relationship between words to be exploited in generating coherent and meaningful sentences.

Language modelling

- Example applications of word modelling in natural language processing (NLP):
  - Text classification and sentiment analysis: Word representations (modelling) become features for training machine learning models that perform sentiment analysis or text classification tasks.
  - Language understanding: To understand human language, NLP models need to recognize and interpret the different aspects of text, such as named entities, part-of-speech tags, syntactic structures, and sentiment. Word modeling helps in these tasks by providing representations that capture relevant linguistic features.

Text encoding (modelling)

Characteristics of word modelling

- Standardize text; turn to lower case & removing punctuation
- Words, characters, n-grams are tokens
- Process of segmenting text into tokens is tokenization
- Tokens are transformed into numeric tensors (vectors) (Vectorizing)
- Packed tensors (vectors) form input to deep networks
- Token-Vector transformation methods (e.g.):
  - one-hot encoding
  - token embedding (word embedding)

Text standardization

- Text standardization is a form of feature engineering that aims to remove encoding differences
  - Remove punctuations
  - Convert to lower case
- Standardization techniques allow the use of less training data and models tend to generalize better

Text splitting - tokenization

- Words tokenization: tokens are units made up of group of characters.
- N-gram tokenization: tokens are groups of N consecutive words.
- Character-level tokenization: each character is its own token (rarely used).

Text encoding

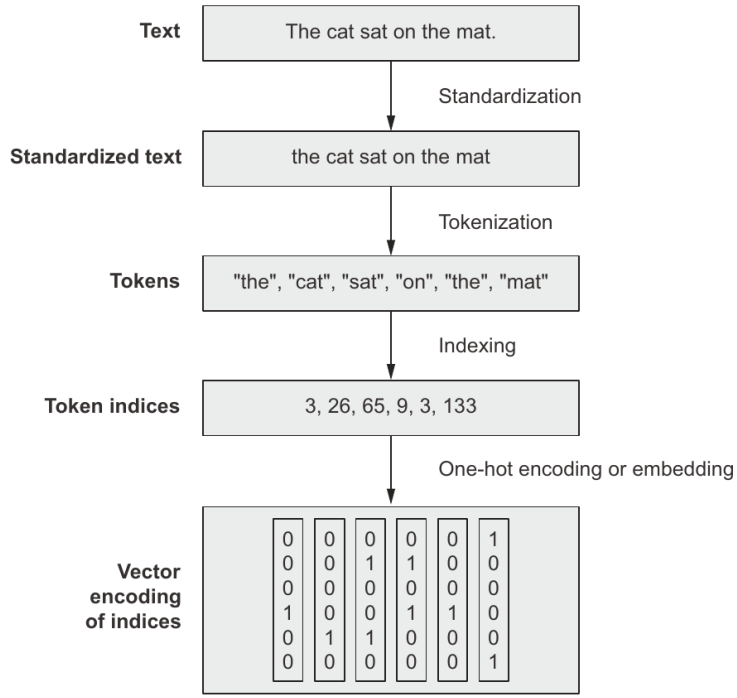
One-hot Encoding

- Given a vocabulary of N tokens

  Unique integer index i, is associated with every token

  Every integer index i, is turned into a binary vector of size N

  Binary vector has all zeros but a 1 in the i-th entry



Text encoding

One-hot Encoding

```
import tensorflow
from tensorflow.keras.preprocessing.text
import Tokenizer
samples = ['The cat sat on the mat.',
           'The dog ate my homework.']
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)
sequences = tokenizer.texts_to_sequences(samples)
#Turns strings into lists of integer
one_hot_results = tokenizer.texts_to_matrix(samples,
                                            mode='binary')
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Text encoding

Word embedding

- Word embedding (supposedly) maps human language into a geometric space reflecting semantic relationship

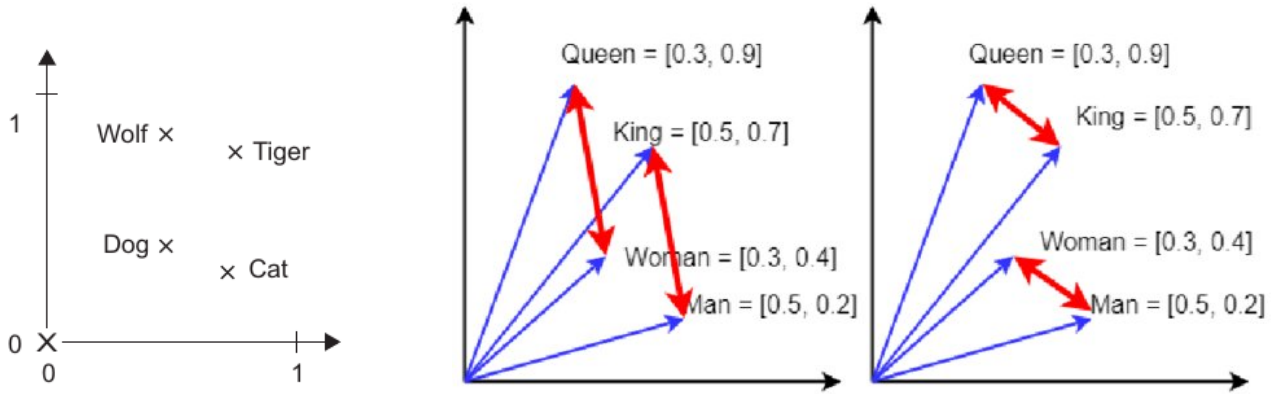


Figure 1: Word embedding example (Word2Vec)
Similar vectors allow moves from `Cat` to `Tiger` and from `Dog` to `Wolf`; akin to mapping from "pet" to "wild"; notice `Cat` to `Dog` vector and `Tiger` to `Wolf` vector; there is also the classical "King + Woman - Man = Queen"

Text encoding

Word embedding

- How to get word embedding?:
  - Learn embeddings jointly with the task of interest (e.g. document classification or sentiment prediction)
    Start with random word vectors and learn word vectors in the same way you learn the weights of a neural network
  - Use pre-computed (also called pre-trained) word embeddings that were learned on a related task to your task of interest.

- Some pre-trained word embeddings:
  - Word2vec: `https://code.google.com/archive/p/word2vec`
  - Global Vectors for Word Representation (GloVe): `https://nlp.stanford.edu/project/glove`
  - BERT - Bidirectional Encoder Representations from Transformers (from Google Research)
  - fastText (from Cornell, Caltech and Amazon)
  - GPT-2, GPT-3 [Generative Pre-trained Transformer] (from OpenAI)

Word embedding generation

Tensorflow/keras API for embedding

tensorflow API

```
tf.keras.layers.Embedding(
input_dim,
output_dim,
embeddings_initializer='uniform',
embeddings_regularizer=None,
activity_regularizer=None,
embeddings_constraint=None,
mask_zero=False,
input_length=None,
sparse=False,
**kwargs
)
```

Example usage

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(1000, 64,
# The model will take as input an integer
#matrix of size (batch,
# input_length), and the largest
#integer (i.e. word index) in the input
# should be no larger than 999 (vocabulary si
# Now model.output_shape is (None, 10, 64),
#where `None` is the batch
# dimension.
input_array = np.random.randint(1000,
                size=(32, 10))
model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
print(output_array.shape)
# prints (32, 10, 64)
```

Neural machine translation

Neural machine translation (NMT)

- Given a text in one language, the task of NMT is to generate the translation into a different language (e.g. English to French).
- Simple RNN-based NMT model is shown in Figure 2

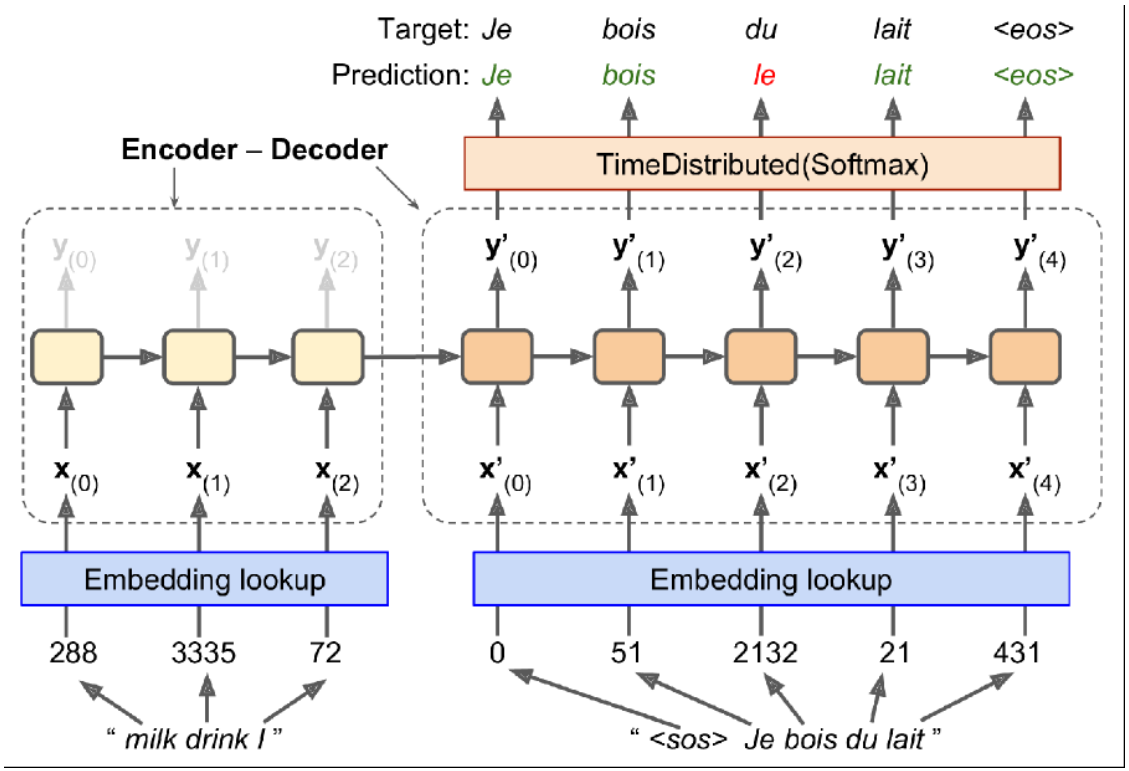


Figure 2: Simple Encoder-Decoder NMT model (Géron, 2019)

Neural machine translation

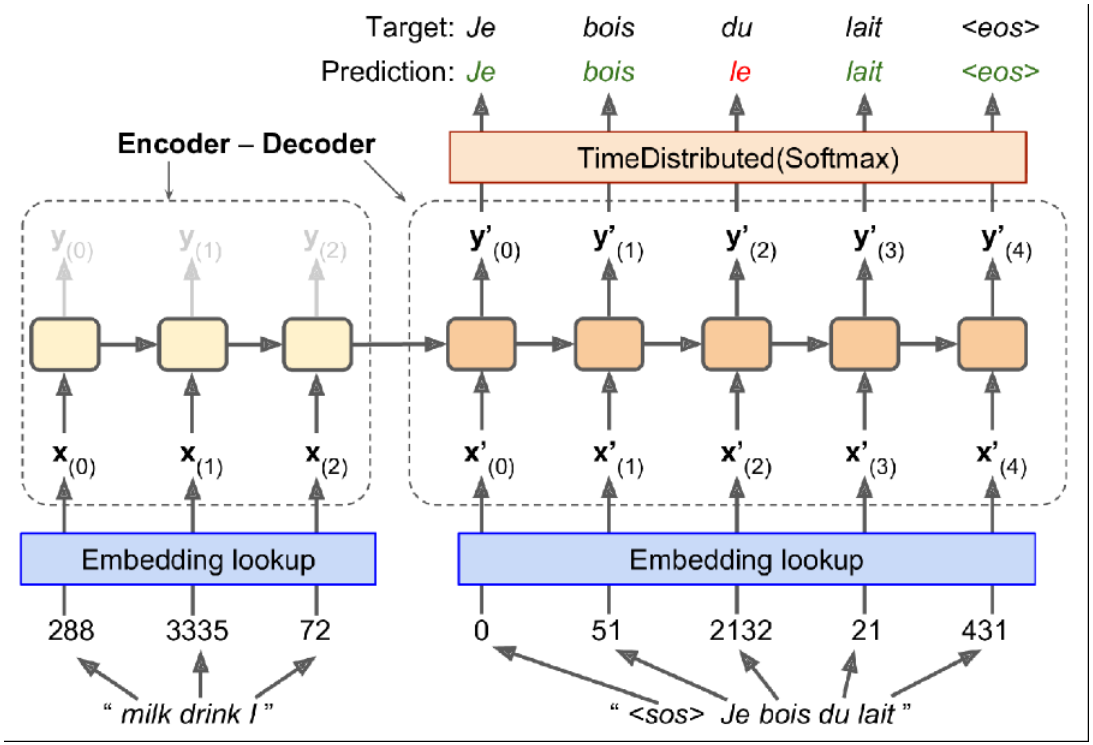


Figure 3: Simple Encoder-Decoder NMT model (Géron, 2019)

- Input: "I drink milk"; Target output: "; Je bois du lait" ; Note how input is reversed in Figure 3 to ensure correct order of translation
- Notice how correct output is also fed into the decoder, but shifted by one time step, viz. $<$ SOS $>$ token

Neural machine translation

Algorithm flow

1. Each word is represented by its ID (from the vocabulary)
2. Embedding layer takes ID and forms word embeddings (which are fed into the encoder & decoder)
3. Encoder last state is fed to the decoder along with the correct translation (but delayed by one time step)
4. At each time step, decoder outputs a score for each word from the output vocabulary (i.e. French)
5. Softmax layer turns the scores into probabilities and the word with highest probability is output

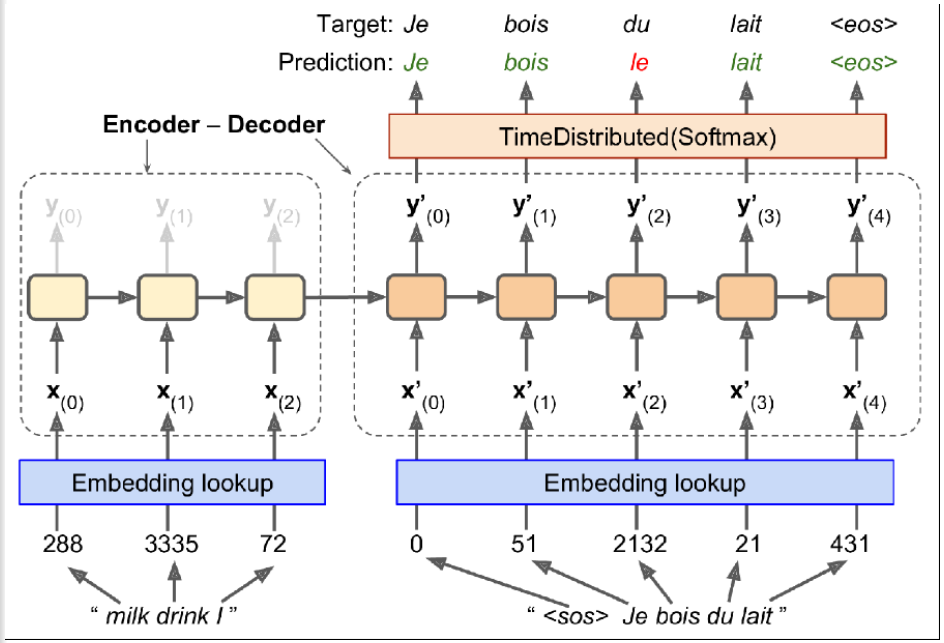


Figure 4: NMT Encoder-Decoder model

Neural machine translation

- At inference, the configuration of the decoder is different as shown in Figure 5.
- Input is the previous output

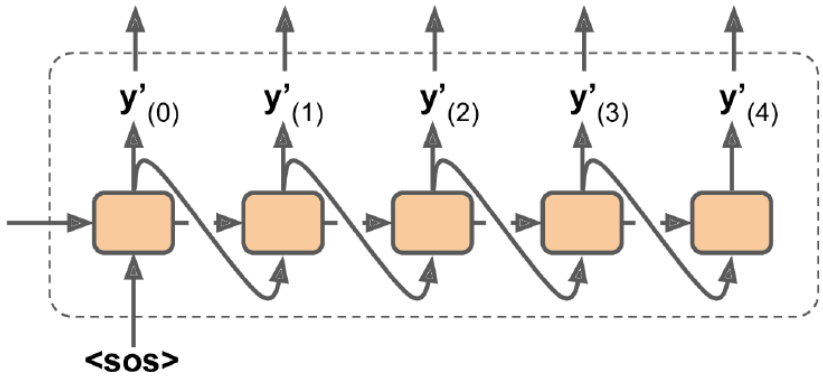


Figure 5: Input of decoder at inference

Neural machine translation

Taking care of reality

Variable sentence length

- Assumption of constant-length sentence does not hold in practice
- Use padding to create constant-length sentences
- Group sentence into buckets of (e.g.) $1-6, 7-12$ word sentences and use pads to make up constant length
- "I drink milk" becomes "$<$ pad $>$ $<$ pad $><$ pad $>$ milk drink I"

Ignore end-of-sentence token $<$ eos $>$

- $<$ eos $>$ should not contribute to loss calculation
- Use mask to suppress usage in computation

Neural machine translation

Taking care of reality

Large output vocabulary

- Encoder-decoder model requires computation of output probability by Softmax (Huge computational burden for large vocabulary)
- Use Sample Softmax ⇓
- Consider only logits output by model for correct word and randomly sample any of the incorrect words; compute approximate loss based on the two.

Good to "know" the future - Bidirectional RNN

- A conventional RNN treats input text as "causal signal".
- There is advantage in looking ahead before encoding a word.
- For example the word queen in "Queen of the United Kingdom", "the quuen of hearts" and "the queen bee" require looking ahead to generate the appropriate encoding.
- Use Bidirectional recurrent layer and concatenate the output at each time step.

Beam search

Beam search - key idea

- How to give the model a chance to fix mistakes it made earlier?
- Keep track of the k most promising sentences and at each decoder step, try to extend them by one word, keeping only k most likely sentences.
- Parameter k is called the beam width.
- Beam search is computationally expensive because it maintains k copies of the model and generates conditional probabilities on the order of the size of the vocabulary for each copy.
- A better solution is provided by Attention mechanism.

Attention

Attention

- Attention is the process of focusing on what is important and fading out what is not important (Chollet, 2021, Chp. 11.4).
- Importance scoring is the starting point of all attention mechanisms.
- Attention can make features Context-aware.

NMT with Attention

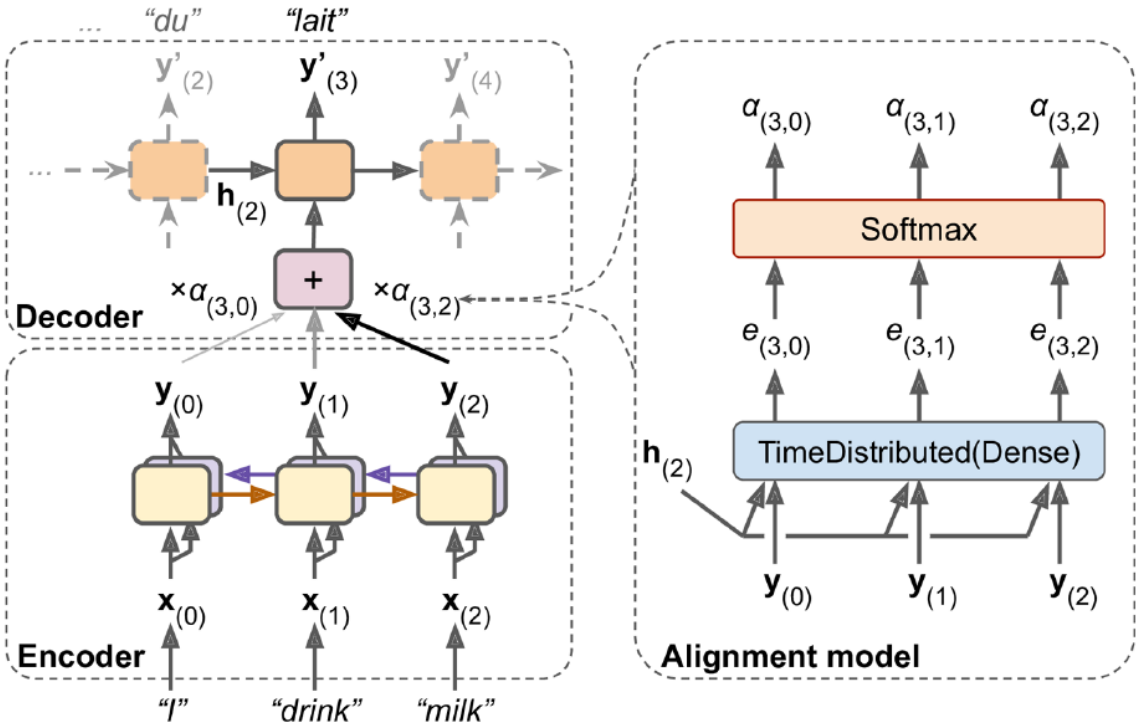


Figure 6: Neural machine translation with attention mechanism (shown on the right)

NMT with Attention

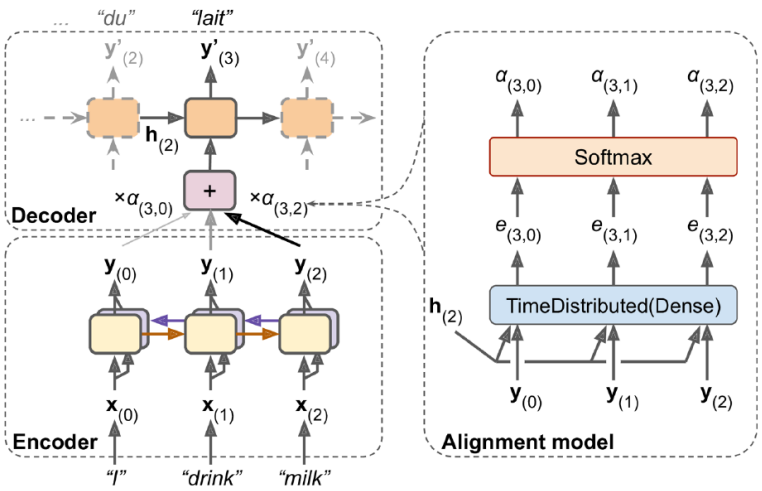


Figure 7: Neural machine translation with attention mechanism

Algorithm

1. Decoder computes a weighted sum of all the encoder outputs; this determines which words to focus upon at this time step.
2. Using Figure 7, $\alpha_{(t,i)}$ is the weight of the i^{th} encoder output at the t^{th} decoder time step.
3. If $\alpha_{(3,2)} > \alpha_{(3,0)} > \alpha_{(3,1)}$, then the decoder will focus attention on word, "milk" than the other two words (at this time step).
4. Apart modification, decoder behaves as in the simple NMT of Figure 2.
5. The block on the right generates the weights; it is the Attention layer or alignment model.

Alignment model

Alignment model

1. Time-distributed dense layer (with single neuron) receives all encoder output, concatenated with the decoder's previous hidden state as input
2. Output of dense layer is a score (or energy) for each encoder output (e.g. $e_{(3,2)}$)
3. Score is a measure of how well aligned is each output with the decoder's previous hidden state
4. Final weight ($\alpha_{(3,2)}$) is generated by the softmax layer (this is called, additive attention)
5. The alignment can also be measured by simple inner (i.e. dot) product (this is the multiplicative attention) (also called Luong attention after the first author of paper that introduced it)

Alignment model

Three models of alignment mechanism

$$\tilde{h}_{(t)} = \sum_i \alpha_{(t,i)} y_{(i)}$$

$$\text{with,} \quad \alpha_{(t,i)} = \frac{\exp(e_{(t,i)})}{\sum_{i'} \exp(e_{(t,i')})}$$

$$\text{and} \quad e_{(t,i)} = \begin{cases} h_{(t)}^T y_{(i)} & \text{dot} \\ h_{(t)}^T W y_{(i)} & \text{general} \\ v^T \tanh(W[h_{(t)}; y_{(i)}]) & \text{concat} \end{cases}$$

Transformer

Transformer - description

1. Encoder (left) -Decoder (right) pair.
2. Nx implies both encoder and decoder are stackable.
3. Input to encoder: word ID (shape: [batch size, max input sentence length]) passed through embedding to generate 512-dimensional representation.
4. Encoder output shape: [batch size, max input sentence length, 512].

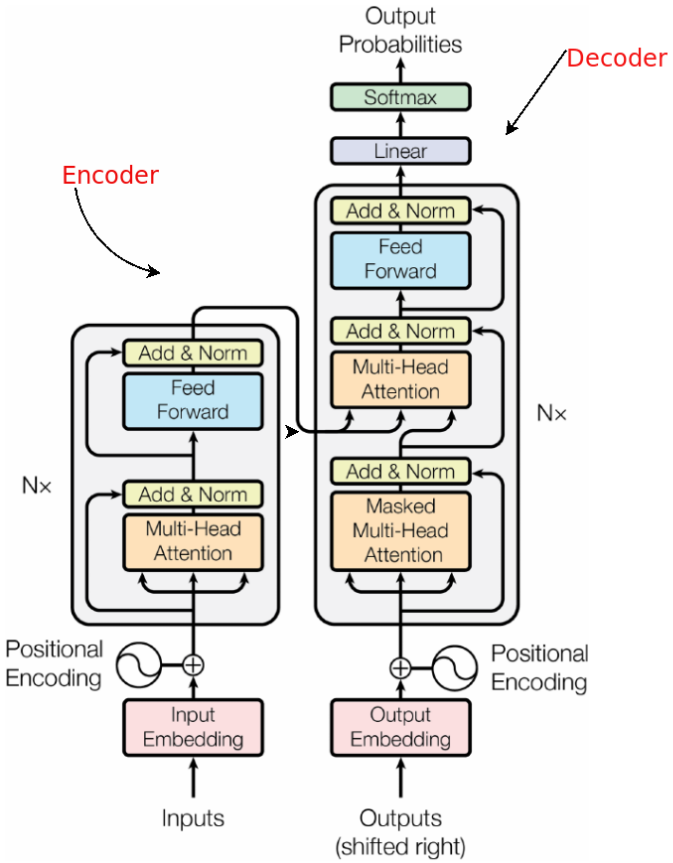


Figure 8: Transformer: "Attention is all you need" (Géron, 2019)

Transformer

Transformer - description (cont'd)

1. Decoder (right) takes target sentence (shifted by one time step to the right) as input during training.
2. Second input to decoder is the output of the encoder
3. Decoder outputs a probability for each possible next word at each time step (output shape: [batch size, max output sentence length, vocabulary length]).
4. During inference, decoder has no access to targets, so it is fed previously output words (starting with start-of-sentence token).

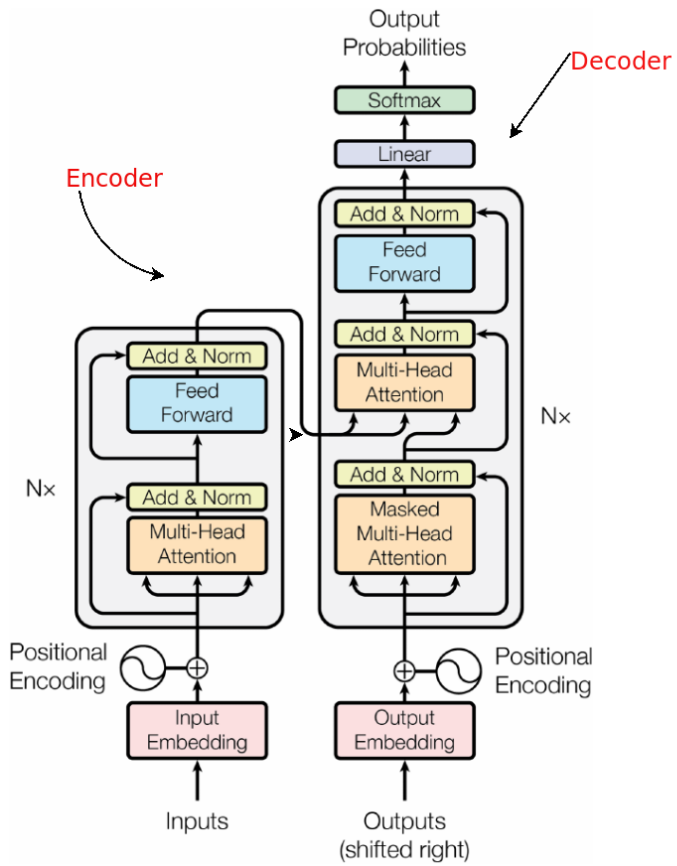


Figure 9: Transformer: "Attention is all you need" (Géron, 2019)

Transformer

Transformer - description (cont'd)

1. Basic Encoder-Decoder pair has two embedding layers and 5 skip connection ($5 \times N$ with stacking).
2. Skip connections are followed by layer normalization and feed-forward modules (two dense layers; first with ReLu; second with no activation)
3. Decoder output layer is a dense layer with softmax activation.
4. All layers are time-distributed, so ech word is treated independently of all others.
5. Word-word relationship and position within sentence are respectively encoded by multi-head attention and positional encoding modules

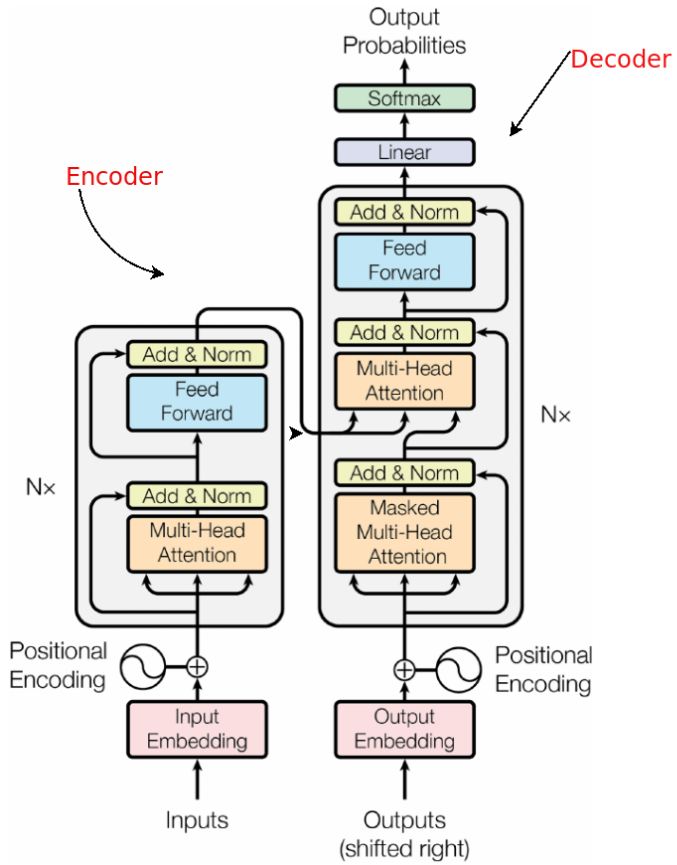


Figure 10: Transformer: "Attention is all you need" (Géron, 2019)

Transformer

Transformer - positional embedding

1. Positional embedding encodes the position of each word, and adds the i^{th} embedding to the word embedding of the i^{th} word.
2. Positional embedding can be learned or obtained in a deterministic formula.
3. Positional embedding matrix P is deterministically obtained as:

$$P_{p,2i} = \sin(p/10000^{2i/d})$$
$$P_{p,2i+1} = \cos(p/10000^{2i/d})$$

where, p is the word position, i is index of the embedding, and d is the maximum dimension of each word representation.

Transformer - Multi-Head Attention

1. Multi-head attention module depends on the Scaled Dot-Product Attention layer.
2. Assume Encoder learned to encode the sentence They played chess with attributes:

Table 1: possible encoding - value-key dictionary

Token (word)	Part of speech
They	pronoun (also subject of the sentence)
played	verb
chess	noun

3. After decoding They, a *subject*, decoder "decides" to decode a *verb*; it should look up (using a "query") in the encoding (map) for the "value" corresponding to the "key" *verb*.
4. Since the encoding of the attributes is vectorized representation, decoder must find the appropriate "value" by computing an approximation - a Scaled Dot-Product.

Transformer

Transformer - Scaled dot-product

1. Compute similarity between "query" and "key"s in the dictionary and use softmax to ensure scores sum to one.
2. In the example being considered, the "key" corresponding to *verb* should have score close to one
3. A weighted sum of the corresponding "values" should be close to the representation of "played".
4. Transformer uses dot-product

Attention

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_{\text{keys}}}}\right)\boldsymbol{V}$$

- $\boldsymbol{Q}$: matrix of queries ($[n_{\text{queries}}, d_{\text{keys}}]$)
- $\boldsymbol{K}$: matrix of keys ($[n_{\text{keys}}, d_{\text{keys}}]$); n_{keys} is number of key-value pairs.
- $\boldsymbol{V}$: matrix of values ($[n_{\text{keys}}, d_{\text{values}}]$)
- Final output ($[n_{\text{queries}}, d_{\text{values}}]$) has one row per query; each row is a query result (i.e. weighted sum of values).

Transformer

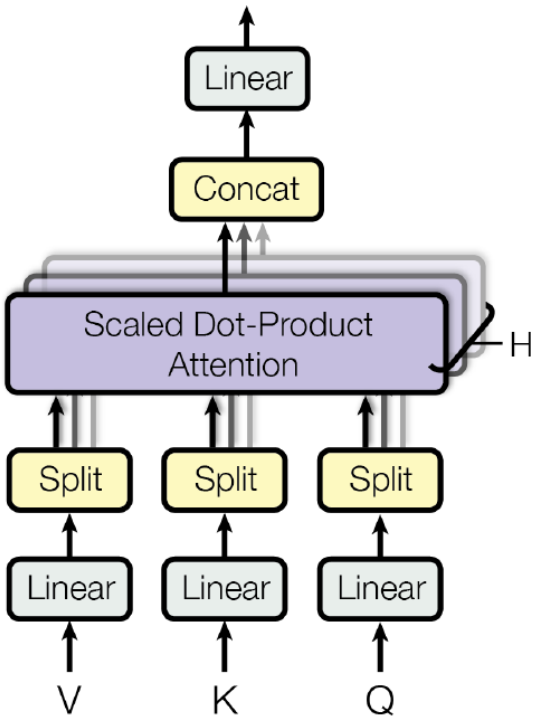


Figure 11: Multi-Head Attention layer architecture (Géron, 2019)

Multi-Head Attention

- It is a collection of Scaled Dot-Product Attention modules.
- Starts with linear transformation of the values, keys, and queries (i.e., a time-distributed Dense layer with no activation function).
- The values, keys and queries are split and passed to each Head (i.e. Scaled Dot-Product Attention module).
- The model uses multi-head to project the word representation into different subspaces, each focusing on a subset of the word's characteristics.

Tensorflow (2.12) API

```
    tfm.nlp.layers.Transformer(
    num_attention_heads,
    intermediate_size,
    intermediate_activation,
    dropout_rate=0.0,
    attention_dropout_rate=0.0,
    output_range=None,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    use_bias=True,
    norm_first=False,
    norm_epsilon=1e-12,
    intermediate_dropout=0.0,
    attention_initializer=None,
    **kwargs
)
```

Some questions

- How do we apply transformer architecture to other tasks apart from NLP?
- What is the equivalent of words in other tasks?
- Do we need to interpret the task as sequence-to-sequence task?

Bibliography

Chollet, F. (2021). *Deep Learning with Python* (2nd ed.). Shelter Island, NY, USA: Manning Publishing Co. Ltd.

Géron, A. (2019). *Hands-on machine learning with scikit-learn keras and tensorflow: Concepts, tools and techniques to build intelligent systems* (2nd ed.). CA, USA: O'Reilly Media, Inc.