Word Embeddings – Semantic Space of Words

- **Geometric relationship** between two words (i.e. the vector encoding) should reflect the **semantic relationship** between two words.

- Word embeddings are vector representations of words that map words into a structured geometric space.

- Word embeddings result in low-dimensional vectors. Example: Word2Vec, GLoVe.

- They can be learned from data!
Learning Word Embeddings

• There is no perfect word embedding that is applicable to any task

• Word embedding for different languages will be different

• Recognizing that word embedding for a specific task say, IMDB Movie Reviews, will look different from the word embedding for another task say, legal document classification.

• It is possible to learn a new embedding for every new task. Learning word embedding can be integrated into the NN’s overall task.

• It is also possible to use a pre-trained word embedding. Word2Vec is one such embedding. GloVe (Global Vectors for Word Representation, Stanford University, 2014)

RNN Model with Embedding

```python
import tensorflow as tf
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256, mask_zero=True)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model=keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
model.summary()
```

After 10 epochs (~2 hours): Accuracy: 98.7%, Validation Accuracy: 87%

It is also possible to use pre-built word embeddings (like Word2Vec, GloVe, etc.)
Machine Translation with NNs

December 2009

October 2018

August 2022

November 2023

Encoder & Decoder NNs

- NN models for Machine Translation

- Uses LSTM (Long Short Term Memory) units. As sentences get too long, the encoder tends to “forget”/lose memory. LSTMs fix this.

- >30 million human-translated pairs of sentences are used to train the network

- Networks also employ several improvements/tweaks.
NNs for NLP Architectures

- **Representing words** and **word order** is important in NNs for NLP tasks.
- Representing Words as Vectors: One-hot encoding, Word2Vec, Word Embedding
- Inputting a word at a time ignores word ordering.
- RNNs enable word sequence modeling, but only go *so far*.
- **Transformers** (hybrid approach) track word order information and pay attention to different parts of a sentence without the use of RNNs.

Transformers, 2017

- **Sequence to sequence models** – processing words as a sequence

  Models learn their own features (like word embedding and word order) using raw word sequences.

- 2016-17, RNNs were all the rage for NN sequence models for NLP
- Transformers replaced many RNNs
  “Attention is all you need” by Vaswani, *et al*, 2017
Sequence to Sequence Models

- Given an input sequence of words (in English)
  Output a sequence of words (in Spanish)

- **Encoder-Decoder model**
  
  **Encoder** turns an input sequence into an intermediate representation.
  
  **Decoder** is trained to predict the next token \((i)\) in the output sequence by looking at (1) Previous output sequence \((0..i-1)\) and (2) the encoded input sequence.

---

Sequence to Sequence Models

- Given an input sequence of words (in English)
  Output a sequence of words (in Spanish)

- **Encoder-Decoder model**
  
  **Encoder** turns an input sequence into an intermediate representation.
  
  **Decoder** is trained to predict the next token \((i)\) in the output sequence by looking at (1) Previous output sequence \((0..i-1)\) and (2) the encoded input sequence.
Transformers: Key Components/Ideas

• Encoder-Decoder

• Positional Encoding
  (preserves positional information in input sequence)

• Attention Mechanism (*aka* Neural Attention)

Transformers: Positional Encoding

• Since there are no recurrent layers in a transformer, an explicit positional encoding is added to the embedding vector.

[Diagram of word embedding vectors and positional encoding vectors]
Transformers: Attention

• Idea: “pay” more attention to important features in input

• Compute importance scores for a set of features. Method of computation varies by approach.

• Makes features context aware

• Example:

Maxpooling: selects one feature in a spatial region (nxn)-all or nothing attention

Transformers: Self Attention

• Creates context aware word/token representations starting with word embeddings.

1. Compute relevancy scores between vectors for “station” and every other word in the sequence.

2. Compute sum of all vectors in the sentence weighted by relevancy scores.

3. Sum up the weighted scores to create a context aware vector representation of the word (“station”).

The process is repeated for every word in the sentence producing a new sequence of vector encoding of the sentence.
Transformers: Multi-headed-Attention

- The output sequence produced by the attention mechanism is concatenated (called, a **head**).

- Outputs from all the heads is concatenated into a vector that represents the input sequence.

**Transformer: Encoder**
Transformer: Encoder+Decoder

- This is a full sequence to sequence transformer architecture.
- The encoder produces context aware representations of each input token.
- The decoder reads in $0..i-1$ tokens already produced and outputs the $i$th token.

It uses neural attention to identify tokens in input sequence that may be closely related to the token it is trying to predict.

Transformer: Applications

- Can be used for any sequence-sequence task
  - **Machine Translation**: Convert text in a source language into text in a target language.
  - **Text Summarization**: Convert a long document into a shorter version that retains important information.
  - **Question Answering**: Convert an input question into an answer.
  - **Chatbots**: Convert a dialog prompt into a reply to this prompt, or convert a history of a conversation into the next reply in the conversation.
  - **Text Generation**: Convert a text prompt into a paragraph that completes the prompt.
  - Etc.
A short History of Transformers

Language Modeling

• The problem:
  Given a sequence of words \(w_j, w_2, ..., w_{i-1}\)

  Predict \(w_i\)

  where \(w_1 ... w_n \in \{\text{vocabulary of words}\}\)

  i.e.

  \[ P(w_i | w_{j}, w_2, ..., w_{i-1}) \]

• Example,

  Input: the cat sat on the

  Output: the cat sat on the mat

  (97%)

• Any system that that can do this prediction is called a language model.

• A language model is a probabilistic model of language.
Language Model: Word N-Gram Models

- \( P(w_i \mid w_1, w_2, ..., w_{i-1}) \) depends on i-1 previous words

This is called an i-gram model.

Unigram is words frequencies of every word in a language
Bigram is pair frequencies
Trigram is 3-word frequencies

Given: He likes
Output: He likes being

- Large Ngram LLMs used for Machine translation (2005)

Applications

- Google Search
- Next word prediction in smart phone texts
- Writing assistants
- Etc.
Language Modeling Using Transformers

- Train a language model (using encoders)
- Enter some initial text
- System generates the next word (using decoder)
- Append generated word to input text...repeat.
Large Language Models (LLMs)

• Trained on massive amounts of data (e.g. LLAMA-2 used 10TB)

• Involve billions of parameters (e.g. LLAMA-2 has 70 billion)

• Use large amounts of computational resources (e.g. LLAMA-2 used 6000 GPUs, took ~12 days, cost over $2million)

Example LLMs: OpenAI’s GPT models (3. and 4.9 used in ChatGPT), Google’s PaLM (used in Bard), Meta’s LLaMa and BLOOM, Ernie 3.0 Titan, Anthropic’s Claude 2.

LLMs Training

• Pre-Training
  Uses copious amounts of text (e.g. scrapped from the entire web)
  Text is huge, but low quality, raw. Results in a Base Model.

• Fine Tuning
  Uses smaller but high quality domain specific text (e.g. human generated and labelled text/documents).
  Training on this text is built on top of the pre-trained transformer. This is also called alignment.
  The result is an Assistant Model. Cheaper, faster (takes ~ 1 day). Undergoes evaluation and incorrect responses are fixed (by humans, adding to training data).

• Fine Tuning (RLHF)
  Have the transformer generate multiple responses, humans select good candidate answers. This is called Reinforcement Learning with Human Feedback (RLHF)

• Tool Integration
  In the future, LLMs are being evolved into tool use capabilities. For example, a chat assistant that can draw plots by generating Python Matplotlib code, or doing web searches to get additional facts/data.
Example Instructions to Human Labelers

Ranking LLMs

See: Chatbot Arena: https://chat.lmsys.org/
Vocabulary

Assistant Model
Attention
Base Model
Encoder
Decoder
Geometric Space
GLoVe
Head
Large Language Model
Language Modeling
Multi-Head Attention
Neural Attention
NGrams
Positional Encoding
RLHF
RNNs
Self-Attention
Semantic Space
Semantic Space
Sequence to Sequence Models
Transformers
Word Embedding
Word2Vec

References

• A. Karpathy: *Introduction to Large Language Models*. 2023. YouTube Video: https://www.youtube.com/watch?v=zjkBMFhNj_g
• *Word Embedding Demo*: https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/