Lecture 6

AI and Ethics: Explainability / Weaponizing AI / Concentrating Power / Existential Risk

NLP Applications: Machine translation(Google Translate) / Natural language generation / Web Search / Spam filters / Sentiment Analysis / Chatbots… and many more

History of NLP: Rule Based Machine Translation (~1970) / Late 1980s realized limitations of rule based solutions / 1990s Surge of statistical methods Probabilistic Context Free Grammers (PCFGs) / Brown Corpus large set of language created in 1960s / IBMs Candide / Statistical methods continued until deep learning in 2010s / Inverse Document Frequency (TF-IDF)/ IDF = Log((# of documents) / (Number of documents contain the word)) / TF = (Number of repetitions of word in a document) / (# of Words in a document) / TF-IDF = TF\*IDF

Scalar, 1x1, CPU / Vector, 1xN, GPU, Tensor, NxN, TPU / CUDA (Compute Unified Device Architecture) Nvidia

Transformer FLOPS Eauation / C – the compute required to train a transformer model / N – the size of a model, its parameters / D – the number of tokens that it is trained on / C is about 6\*N\*D

Tau is the clusters throughput and T is the time it takes to train. Therefore tau \* T = C = 6ND

Example: An 82B parameter Korean variant of GPT-3 called HyperCLOVA was trained on 150B tokens using a cluster of 1024 Nvidia A100 GPUs. How long could that take? Solution: The peak float16 FLOPs throughput of A100 is 𝜏𝜏 = 312 teraFLOPs = 3.12e14 FLOPs. The total compute is C = 6 ∙ 8.2e10 ∙ 1.5e11 = 7.38e22. The training must have taken at least T = C / 1024𝜏𝜏 / 86400 = 2.67 days.

Estimated GPU Memory M = ((# of parameters \* 4B) / (32 / Q)) \* 1.2. Q = The amount of bits that should be used for loading the model.

Example: Mistral 7B model Memory needed for parameters = 7 \* 4 \* 1.2 = 33.2GB of memory. In reality, to load the full model, you will need around 40-45GB.

To save memory, can map 32 bit to 16 bit precision. Quantitiztion: map FP numbers to integers, say 4 bit. Saves space. Need to expand backto FP for computation.

Quantization - Example • Given weights: [0.02345, -0.56789, 1.23456, -0.34567]. • Min and max values are -0.56789 and 1.23456, respectively. • Calculate the range: 1.23456 - (-0.56789) = 1.80245. • Quantization levels for 4-bit signed integers are from -8 to 7. • Compute the scale factor: 15 / 1.80245 ≈ 8.3205. • Apply the scale factor to quantize the values. / For 0.02345, multiply by 8.3205 to get 0.195; round to 0. • For -0.56789, multiply by 8.3205 to get -4.723; round to -5. • For 1.23456, multiply by 8.3205 to get 10.272; round to 10 (clamped to 7). • For -0.34567, multiply by 8.3205 to get -2.876; round to -3. • The quantized values are [0, -5, 7, -3]. • These values represent the nearest integers within the quantized range. / Convert back to floating-point using the scale factor. For 0, divide by 8.3205 to get 0.0.For -5, divide by 8.3205 to get approximately -0.6006. For 7, divide by 8.3205 to get approximately 0.8411. For -3, divide by 8.3205 to get approximately -0.3603. Dequantized values approximate the original values.

Key Libraries: NTLK – CPU Based Standalone NLP. PyTorch and TensorFlow

Recurrent NNs and LSTM.

Lecture 7

Intuition Behind TF-IDF Term Frequency (TF): •Measures how frequently a term (word) appears in a document. •Intuition: The more times a word appears in a document, the more important it is in that document. • TF = (Number of repetitions of word in a document) / (# of words in a document) • TF is a local measure (i.e. within a document) IDF: • Measures how important a term is in the entire corpus. • Intuition: A term that appears in many documents is less informative about any particular document. • The logarithm is used to dampen the effect of terms that are very common across all documents. IDF is a Global measure (i.e. across documents)

Byte Pair Encoding BPE / Create a table of all characters including space and their frequencies. At each iteration, look for the most common pairs and merge them. The last character of the first token to be merged cannot be whitespace to prevent merging across words. Stop when a predetermined number of tokens is identified. Can find roots, prefixes, suffixes. Reduces number of tokens. Can possibly use sub words when it finds a word not in the training set (OOV words). Morphological Awareness.

Wordpiece In addition to frequency adds the likelihood of the new symbol when added to the model's existing vocabulary, example BERT

SentencePiece – for languages such as Japanese that do not have sentences.

Tokenizers that provide ids – PyTorch and Tensorflow / Tokenizers without IDs – General text processing – Syntax / POS

ChatGPT – one token corresponds to ~4 characters

Embeding tokenizing does not give meaning, embedding does. Use a multidimensional space to capture all the meanings, similar words have similar embedings / Word2Vec Embeding, Continuous Bag of Words (CBOW) use surrounding words to predict the middle word. Set weights to minimize the error. Skip gram encoding attempts to predict the surrounding words from the central word. Each word is predicted separately. Window size is a Skip Gram hyperparameter. CBOW is faster but Skip Gram is more complete and can capture infrequent words.

The law of similarity is a principle stating that similar items are perceived as a group. Dot product between tow words measures their similarity Then, the dot product of a and b becomes: a⋅b=∑aibi for I = 1 to n = a1b1+a2b2+a3b3. Geometric definition a⋅b=|a||b|cosθ This metric is not influenced by the magnitude or length of the vectors, focusing solely on direction. •A cosine similarity of 1 means words are semantically identical, while 0 means no similarity. •LLMs uses cosine similarity to find the best-matching words and phrases during conversation. Cosine similarity is used in attention mechanisms to weigh the influence of different words.

Embedings: Word2Vec, GloVe (2014) count based, FastText (2016) facebook extension of Word2Vec, ELMo(2018) different embedings based on usage, Transformer (2017)

BERT – Bidirectional Encoder Representations from Transformers - Book Corpus (800 Million Words) and Wikipedia (2,500 Million Words) Model Sizes: BERT-Base: 12 layers (transformer blocks) 110 million parameters BERT-Large: 24 layers (transformer blocks) 340 million parameters / train by masking words / randomly 15% of the words are masked

Lecture 8

NSP/ NSV(Next Sentence Prediction/ Validity ) is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not.

Transformers – tokenizing and embedding is a start. We also need positional encoding - For each position(pos) and dimension(i), the positional encoding is calculated as: PE(pos, 2i)=sin(pos/10,0002i/dmodel), PE(pos, 2i+1)=cos(pos/10,0002i/dmodel) dmodel represents the dimensionality of the embeddings. Incorporation into Embeddings: 1. The resulting positional encoding is added element-wise to the word embedding vector. 2. This combination ensures that the model not only captures the semantic meaning of each word but also understands the syntax and structure of the sentence.

Positional Encoding -- Example •Sentence: "I love to play soccer" - focusing on the word "I" (position 0). •d\_model = 4: Four dimensions for illustrative purposes. GPT 4 has d\_model of 2048. •Calculation for pos=0: • Dimension 0: sin(0/100000/4) = sin(0) =0 • Dimension 1: cos(0/100000/4) = cos(0) =1 • Dimension 2: sin(0/100002/4) = sin(0) = 0 • Dimension 3: cos(0/100002/4)= cos(0) =1 •Result: Positional encoding for "I" is [0, 1, 0, 1].

Word: "love" (position 1). •Calculation for pos=1: • Dimension 0: sin(1/100000/4) = sin(1) • Dimension 1: cos(1/100000/4) = cos(1) • Dimension 2: sin(1/100002/4) = sin(1/10000.5) • Dimension 3: cos(1/100002/4) = cos(1/10000.5) •Result: Positional encoding for "love" shows variation across dimensions.

Attention Mechanism in an attention mechanism allows the model to focus on different parts of the input sequence when making predictions. •Self-attention enables each token to attend to all tokens in the sequence. •The model computes a weighted sum of all input representations based on their relevance. •Attention helps in understanding context and relationships between words in a sentence.

This is Great, but where do the “a”s come from. They are at the core of this. How does the “attention” of a word to another get computed? If you have two values and you want to “measure” how they relate “without” impacting them, take a “copy” of them by multiplying them with some parameter. Note that that parameter has to be the same within the attention head (or else, you will be not measuring/listening equally to the impact of the words to each other).

More than one Attention Head…. Multiple self-attention mechanisms are usually applied in parallel, and this is known as multi-head self-attention. Now H different sets of values, keys, and queries are computed Typically, if the dimension of the inputs xm is D and there are H heads, the values, queries, and keys will all be of size D/H, as this allows for an efficient implementation. The outputs of these self-attention mechanisms are vertically concatenated, and another linear transform Ωc is applied to combine them

BERT • BERT has 12 of these layers, each with 12 attention heads!!. • Each Head is connected parallely to the next head in the same transformer layer, whereas layers are connected in series. • Residual connections are done within the layer (i.e. adding back the input is within the layer, but not across layers).

A picture containing diagram

Description automatically generated

Table

Description automatically generated with medium confidence

Diagram, schematic

Description automatically generated

BERT special tokens [CLS]- begining, [SEP] - separator, [MASK], [PAD], [UNK], [START], [END]

Transfer learning, Pre training, Fine Tuning / Encoders, Decoders, Encoders-Decoders,

LLNs Evaluations: HumanEval, Pass@k Metric – probability top k will pass unit test, ROUGE Score – mis of precsion and recal – F1

Diagram

Description automatically generated