Natural language understanding with transformers

Lecture slides

Outline

- Typology of NLU tasks
- Sentiment analysis
- Natural language inference
- Named entity recognition
- Question answering
- Sentence encoders
- Information retrieval
- Dialogue systems

Outline

- Typology of NLU tasks
- Sentiment analysis
- Natural language inference
- Named entity recognition
- Question answering

Types of NLU problems

Types of NLU problems

NLU = natural language understanding = extracting meta data from texts

Single text classification
- spam detection
- sentiment analysis
- tagging articles by topic
- toxic comment identification
- intent classification in dialogue systems

Token classification
- named entity recognition (NER)
- part of speech tagging
- slot filling in dialogue systems
- toxic spans detection
- in-text question answering
- grammatical errors detection (and sometimes correction)

Relations between texts
- paraphrase detection
- natural language inference (NLI)
- matching documents with queries
- translation identification
- multiple choice question answering

Relations between tokens
- dependency or constituency parsing
- coreference resolution
- extraction of relations between entities
- nested named entity recognition

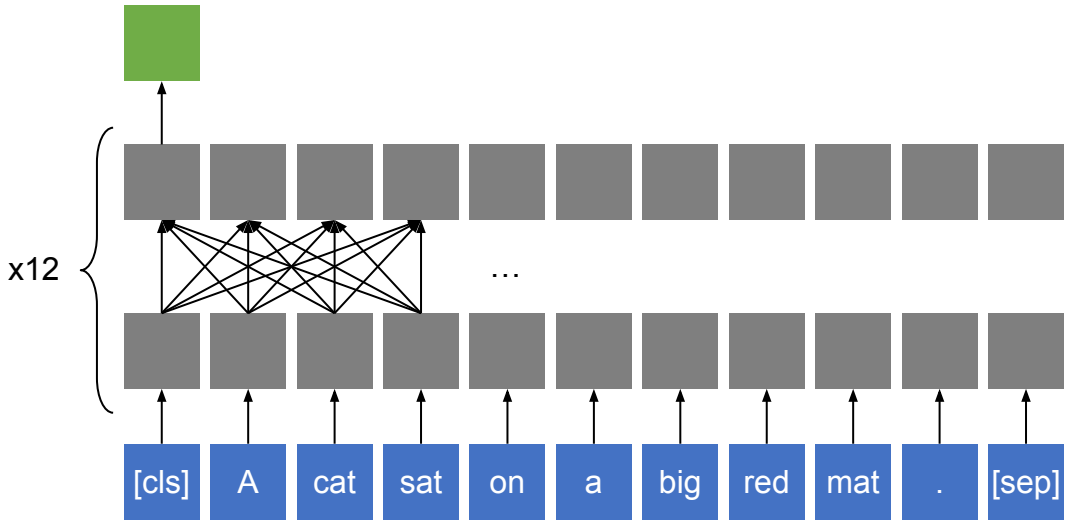
NLP problems NOT related to NLU

- Text-to-text (sequence-to-sequence = seq2seq) problems
  - translation
  - summarization
  - dialogue response generation
  - generative question answering
- Text-to-structure problems: seq2seq with a special output format
  - text to code (e.g. SQL, SPARQL, various programming languages)
  - text to meaning representation (e.g. AMR)
- These problems typically require text decoders
  - We will consider them next time
  - Today, we focus on the problems that can be solved with an encoder-only model (such as BERT)

Architectures for typical NLU problems

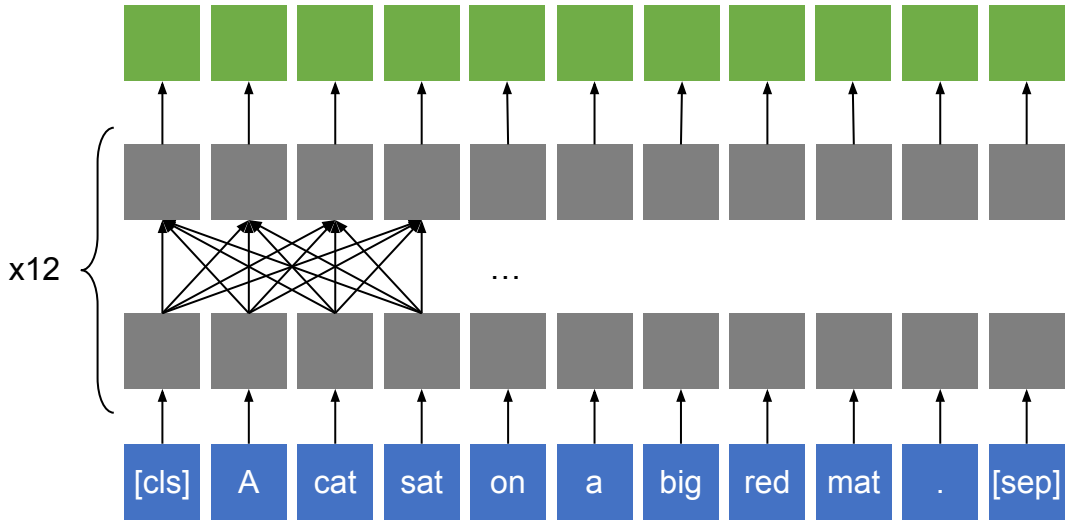
BERT* for <u>sequence classification</u>

- Add a linear layer on top of **the first token**
- Fine-tune the whole model
  - The model will automatically learn to deliver all necessary information to the first token



BERT* for <u>token classification</u>

- Add a linear layer on top of **each token**
  - Use the same weights for each token
- Fine-tune the whole model

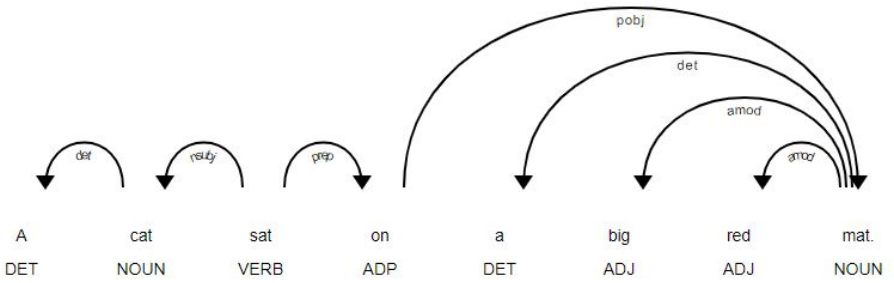


* Instead of BERT, any other pretrained transformer encoder can be applied

Biaffine layers for relations between tokens

The goal: predict pairwise relations between tokens

- Use a binary classifier for each pair of tokens (or multiclass, to predict relation type)
- Biaffine layer is a simple way to combine two vectors



	A	cat	sat	on	A	big	red	mat
A	0	1	0	0	0	0	0	0
cat	0	0	1	0	0	0	0	0
sat	0	0	0	0	0	0	0	0
on	0	0	1	0	0	0	0	0
a	0	0	0	0	0	0	0	1
big	0	0	0	0	0	0	0	1
red	0	0	0	0	0	0	0	1
mat	0	0	0	1	0	0	0	0

Biaffine function for predicting the (i, j) arc:

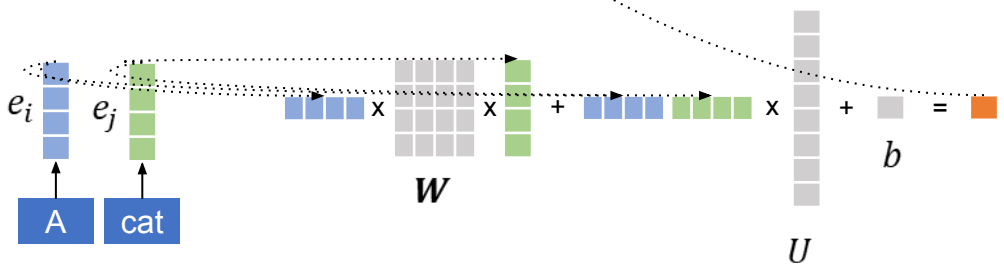
$$score_{ij} = e_i^T \boldsymbol{W} e_j + \left(e_i \oplus e_j\right)^T U + b$$

e_i, e_j – contextual token embeddings

Parameters: $\boldsymbol{W}$ – a matrix, U – a vector, b – a scalar

"Affine" means "linear with bias".

The function above is affine w.r.t. each argument, i.e. biaffine.



Source: Deep Biaffine Attention for Neural Dependency Parsing, Dozat et al, 2017

Sentiment analysis

Sentiment analysis

- "Sentiment analysis" is an umbrella term for multiple tasks:
  - Predict general polarity of a text (positive/neutral/negative)
    - *Coronet has the best lines of all day cruisers.*
    - *Pastel-colored 1980s day cruisers from Florida are ugly.*
  - Predict polarity w.r.t. a specific object of its *aspect*
    - *I was tempted to buy this product as I really like its design, but its price is not good.*
    - *Chris Craft is better looking than Limestone, but Limestone projects seaworthiness and reliability.*
  - Predict fine-grained emotions (joy/sadness/fear/anger/surprise etc.)
- Diverse levels of difficulty
  - In simple cases, lists of keywords suffice
  - In complex cases, one has to deal with negations, collocations, sarcasm etc.

Sources: Stanford cs224u course, Wikipedia

Related classification problems

There are multiple other emotional phenomena that occur in natural texts and can be detected.

Their degree of difficulty is diverse.

- Subjectivity (Pang and Lee 2008)
- Bias (Recasens et al. 2013; Pryzant et al. 2020)
- Stance (Anand et al. 2011)
- Hate-speech (Nobata et al. 2016)
- Microaggressions (Breitfeller et al. 2019)
- Condescension (Wang and Potts 2019)
- Sarcasm (Khodak et al. 2017)
- Deception and betrayal (Niculae et al. 2015)
- Online trolls (Cheng et al. 2017)
- Polarization (Gentzkow et al. 2019)
- Politeness (Danescu-Niculescu-Mizil et al. 2013)
- Linguistic alignment (Doyle et al. 2016)

Source: Stanford cs224u course

Tips for sentiment analysis (and similar problems)

- *AutoModelForSequenceClassification* will do the job well
- Sometimes, up-sampling less frequent classes helps
  - Of course, it depends on the choice of metrics
- Both multiclass classification and regression work well
  - To convert between different sentiment scales, an ordered logistic regression head may be used
  - Sometimes, extra classes like “mixed sentiment” are required
- Data augmentation may help a lot
  - Randomly remove “too easy” features (e.g. punctuation, smileys)
  - Check how prediction change under simple transformations*

* A good example of such methodology is in Beyond Accuracy: Behavioral Testing of NLP models with CheckList by Ribeiro et al (2020)

Aspect-based sentiment analysis

- Analyze “Great food but the service is dreadful”
- Aspect extraction: given a text, detect “food” and “service”
  - Multilabel text classification: for each possible aspect (from a fixed set), predict whether it is present in the text
  - Text pair classification (NLI-style): given an aspect and a text, predict whether it is relevant (or even its sentiment)
  - Token classification (NER-style): highlight tokens that describe aspects
- Opinion extraction: given a text and an aspect, classify opinion
  - Single text classification, with aspect in special tags
  - Text pair classification (NLI-style)

Source: https://arxiv.org/abs/1910.00883, https://arxiv.org/abs/2011.00169

Natural language inference

Natural language inference

The goal of NLI: detect logical relation between two texts

Premise	Relation	Hypothesis
A turtle danced.	entails	A turtle moved.
turtle	contradicts	linguist
Every reptile danced.	neutral	A turtle ate.
Some turtles walk.	contradicts	No turtles move.
James Byron Dean refused to move without blue jeans.	entails	James Dean didn't dance without pants.
Mitsubishi Motors Corp's new vehicle sales in the US fell 46 percent in June.	contradicts	Mitsubishi's sales rose 46 percent.
Acme Corporation reported that its CEO resigned.	entails	Acme's CEO resigned.

Source: Stanford CS224U course, slides.

Why NLI is important

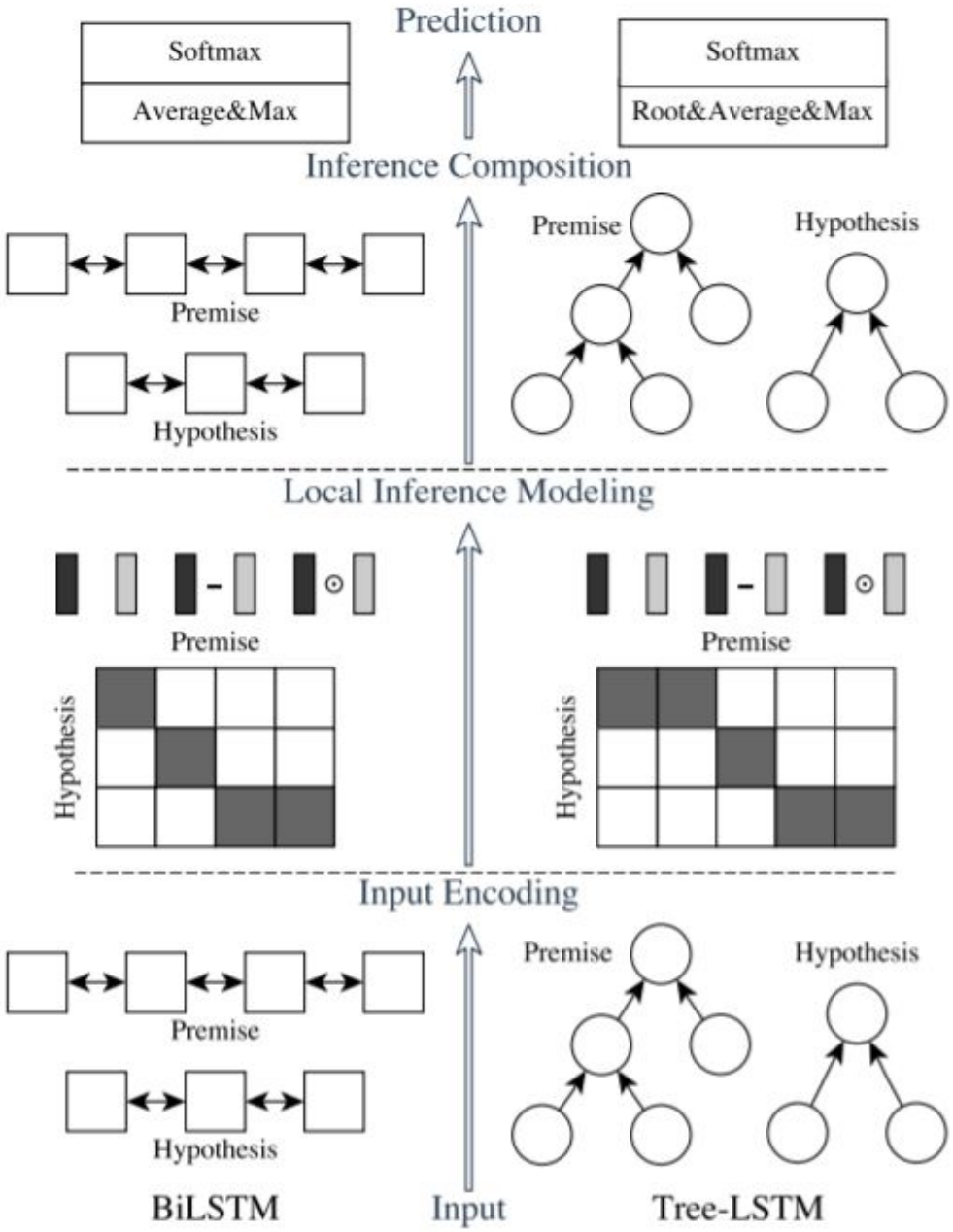
- Diagnostics
  - Evaluate a model's understanding of various linguistic phenomena, logic and world knowledge
    - For example, such a task is a part of the SuperGLUE benchmark
- Applications
  - Controlling quality of generated texts in various tasks
  - Paraphrase detection
  - Zero-shot or few-shot text classification
    - Does *"The IAU downgrade Pluto as a dwarf planet"* imply *"This is science news"*?
  - Pre-training for similar tasks

Task	**NLI framing**
Paraphrase	text ≡ paraphrase
Summarization	text ⊏ summary
Information retrieval	query ⊐ document
Question answering	question ⊐ answer
	Who left? ⇒ *Someone left*
	Someone left ⊐ *Sandy left*

Sources: Stanford CS224U course (slides); Wang et al, 2021, Entailment as Few-Shot Learner; https://gluebenchmark.com/diagnostics

NLI before transformers

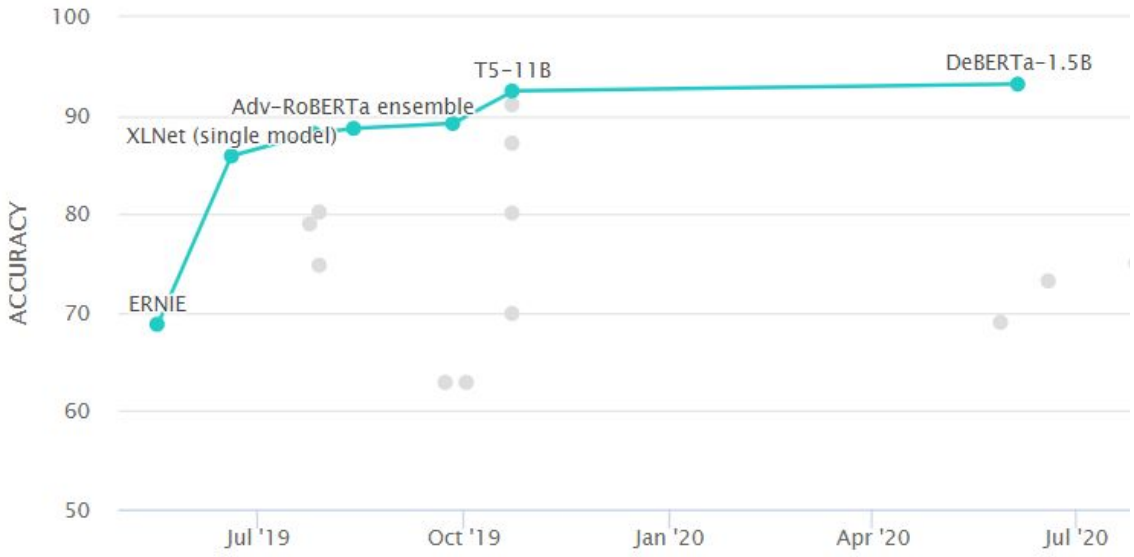
- Before transformers, NLI required special architectures, such as this one:
  - Compute contextual embeddings
  - Compute local interactions between each pair of terms in two texts
    - E.g. use a kind of attention
  - Update local embeddings using the information from interactions
  - Use pooling to get a fixed size vector
  - Predict the result with a MLP
- Transformers do all of this “naturally”



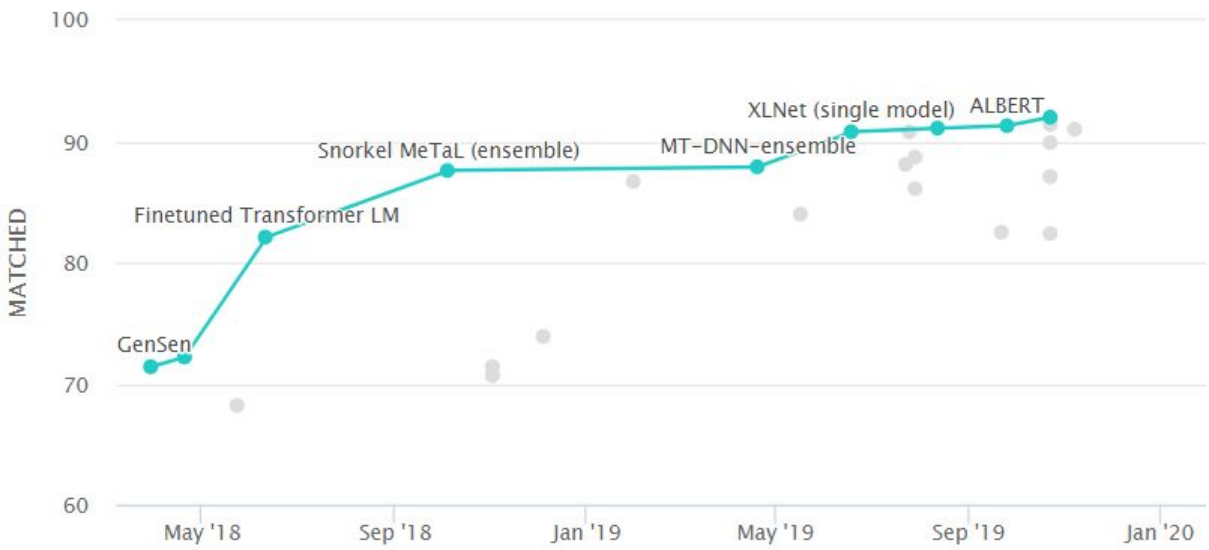
Chen et al, 2017, *Enhanced LSTM for Natural Language Inference*

NLI benchmarks

RTE (part of GLUE and SuperGLUE)



MultiNLI



- Transformers perform significantly better than alternative architectures
- Traditional benchmarks saturate fast

Source: https://paperswithcode.com/task/natural-language-inference

Adversarial NLI: a moving target

If standard NLI benchmarks are to easy, on can create an intentionally difficult dataset, using human-in-the-loop data collection

Repeat:

- Ask humans to create examples on which the current model makes mistakes
- Train a new model using these new difficult examples

Results:

- New difficult datasets
- Improvement even on the old datasets

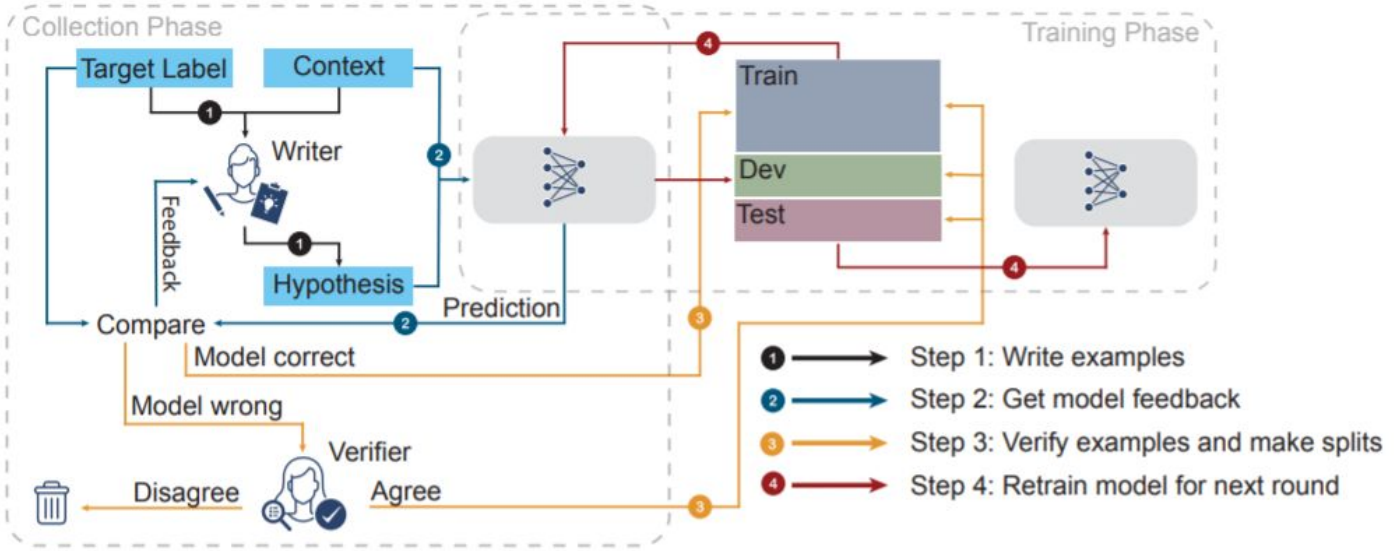


Figure 1: Adversarial NLI data collection via human-and-model-in-the-loop enabled training (HAMLET). The four steps make up one round of data collection. In step 3, model-correct examples are included in the training set; development and test sets are constructed solely from model-wrong verified-correct examples.

Nie et al, 2020, Adversarial NLI: A New Benchmark for Natural Language Understanding; https://dynabench.org/tasks/nli

Named entity recognition

Named entity recognition

- NER is the problem of finding named entities* within a text

- Typically, IO, BIO or BIOES schemes are used for labelling each token
  - BIO helps to distinguish adjacent entities
  - BIOES provides additional information (redundant but sometimes helpful) about entity boundaries

	Alexander	Sergeyevich	Pushkin	was	born	in	Moscow	in	1799	year
IO	PER	PER	PER	O	O	O	GPE	O	DATE	DATE
BIO	B-PER	I-PER	I-PER	O	O	O	B-GPE	O	B-DATE	I-DATE
BIOES	B-PER	I-PER	E-PER	O	O	O	S-GPE	O	B-DATE	E-DATE

* “Named entity” means proper names. However, in practice we train models to recognize whatever type of entity we need, not necessarily named ones.

Model and visualization: Spacy. Labeling schemes: Cho et al, 2013, Named Entity Recognition with Multiple Segment Representations

Tips for NER

- For some tokens, targets can be masked out
  - E.g. for multi-token words, only first tokens of each word are often used
  - In PyTorch, label id -100 is used to ignore it in loss computation
- Evaluate performance with F1 score *on entity level*
  - This is the simplest approach, but it ignores partially overlapping entities
- Sometimes, a CRF layer on top of a transformer is reported to help

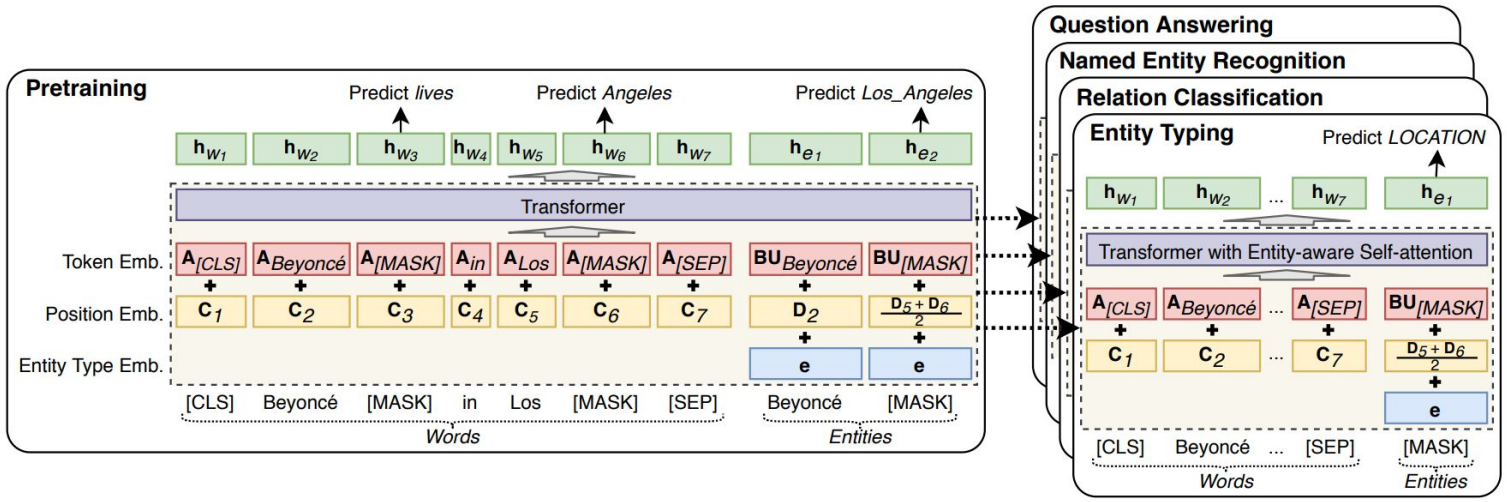
Few-shot NER

- The NER problem is (relatively) easy, but the data is often scarce
  - Typical applications may require non-standard entities and many types
  - Manual annotation for NER is labor-intensive
- Augmentation is a possible solution
  - E.g. modifying non-NER spans or replacing entities with same-type ones
- Active learning is a possible solution
  - Apply the model to a corpus, identify the texts on which it is most uncertain, label them, retrain the model, repeat.
    - Uncertainty can be estimated with Monte Carlo dropout
- Other methods: KNN over embeddings, pretraining on large noisy corpora, self-training

(1) Active Learning for Sequence Tagging with Deep Pre-trained Models and Bayesian Uncertainty Estimates, Shelmanov et al, 2021
(2) Few-Shot Named Entity Recognition: A Comprehensive Study, Huang et al, 2020

LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention

- LUKE is trained to predict randomly masked words and entities using a large amount of entity-annotated corpus obtained from Wikipedia.
- The model utilizes an entity-aware self-attention mechanism, which considers the type of the tokens (words or entities) when computing attention scores.

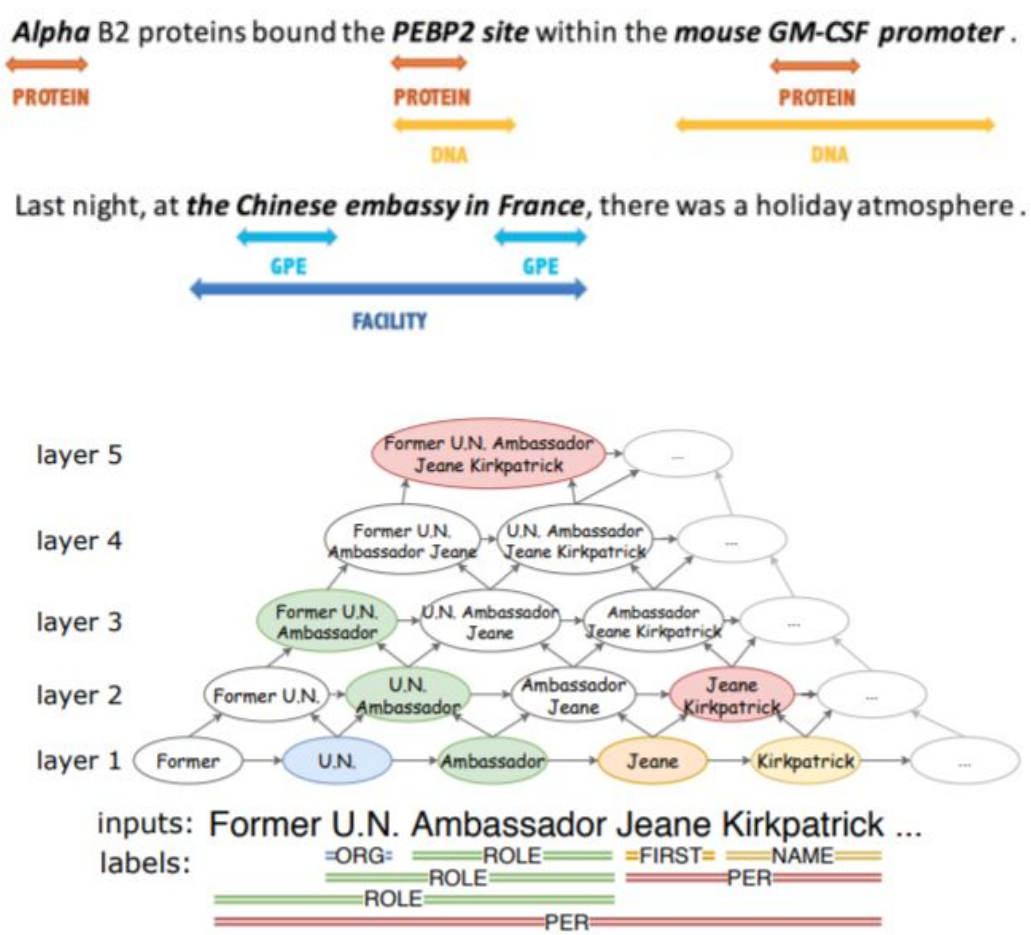


$$e_{ij} = \begin{cases} \mathbf{K}\mathbf{x}_j^\top \mathbf{Q}\mathbf{x}_i, & \text{if both } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are words} \\ \mathbf{K}\mathbf{x}_j^\top \mathbf{Q}_{w2e}\mathbf{x}_i, & \text{if } \mathbf{x}_i \text{ is word and } \mathbf{x}_j \text{ is entity} \\ \mathbf{K}\mathbf{x}_j^\top \mathbf{Q}_{e2w}\mathbf{x}_i, & \text{if } \mathbf{x}_i \text{ is entity and } \mathbf{x}_j \text{ is word} \\ \mathbf{K}\mathbf{x}_j^\top \mathbf{Q}_{e2e}\mathbf{x}_i, & \text{if both } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are entities} \end{cases}$$

Source: (1) https://arxiv.org/abs/2005.07150, (2) https://aclanthology.org/2020.acl-main.525/, (3) https://arxiv.org/abs/1910.11476

Recognition of nested or overlapping entities

- Sometimes, named entities nest or overlap, so the problem cannot be stated as *flat* sequence labelling
- Alternative formulations:
  - Scoring each span with a biaffine classifier (as in dependency parsing)[1]
  - Add extra layers ("pyramid") to predict entity labels for n-grams[2]
  - Re-formulate the problem as question answering, where entity type is the question[3]
    - Because same-type entities still can overlap, an additional layer for filtering candidate spans is required



Source: https://arxiv.org/pdf/2010.01057.pdf

Question answering

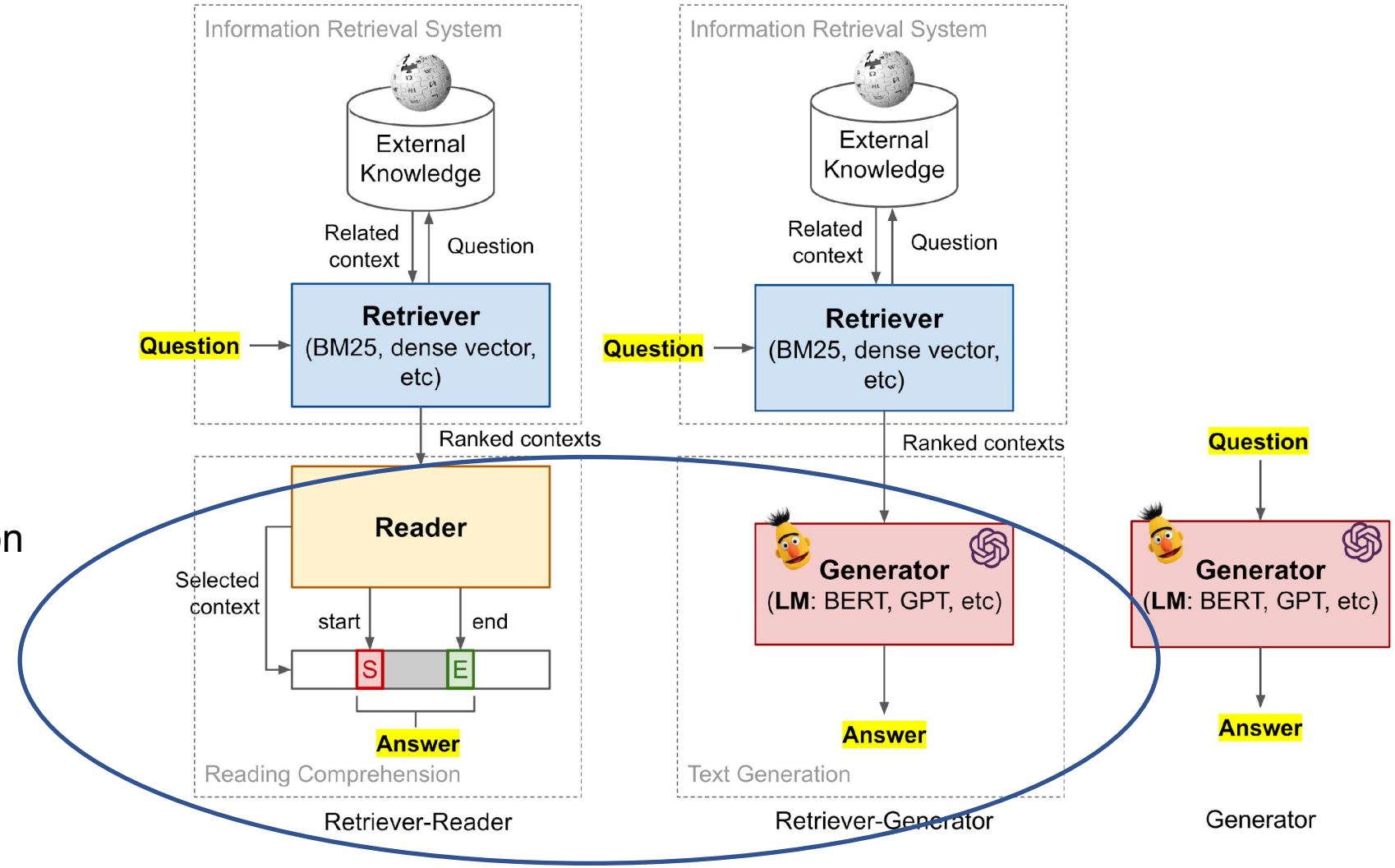
Typical QA problems and benchmarks

- Extractive QA
  - SQuAD: passage + question; the answer is within the passage
- Open generative QA
  - CoQA: passage + dialogue with questions; the answer is free-form text, but for each answer there is evidence within the passage
  - TriviaQA: question + long evidence documents; the answer is free-form text
- Closed generative QA
  - Just questions are given
- KBQA
  - Question + knowledge base, the answer should be selected or generated

Sources: https://github.com/danqi/acl2020-openqa-tutorial, https://huggingface.co/tasks/question-answering

Question answering system architectures

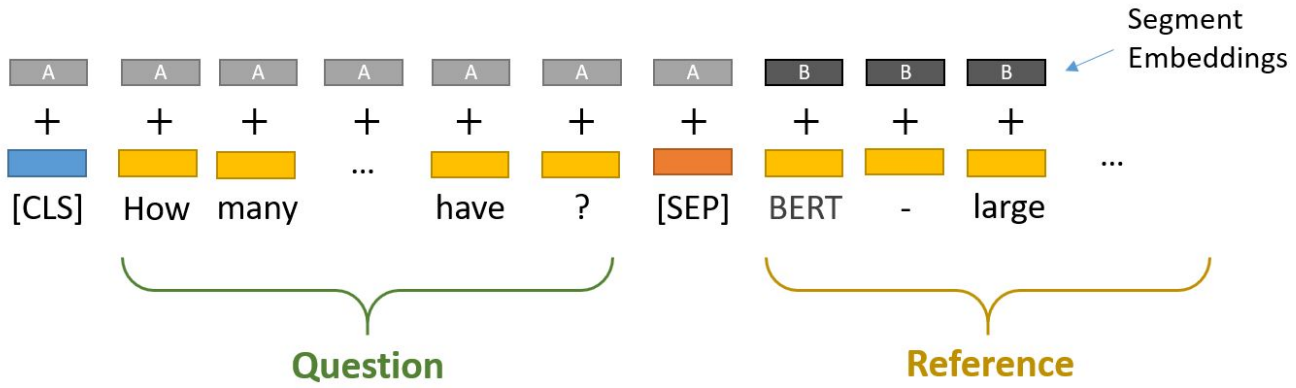
Reading comprehension



Source: https://lilianweng.github.io/posts/2020-10-29-odqa/

Extractive question answering

- Used for SQuAD-like tasks: given the question and the context, select the segment of the context which is the right answer
  - If there is no answer, select the first zero-length segment (i.e. [CLS] token)
- Inputs: the question + separator + the context
- Outputs: probabilities of each segment
- Evaluation: exact match or F1 over words



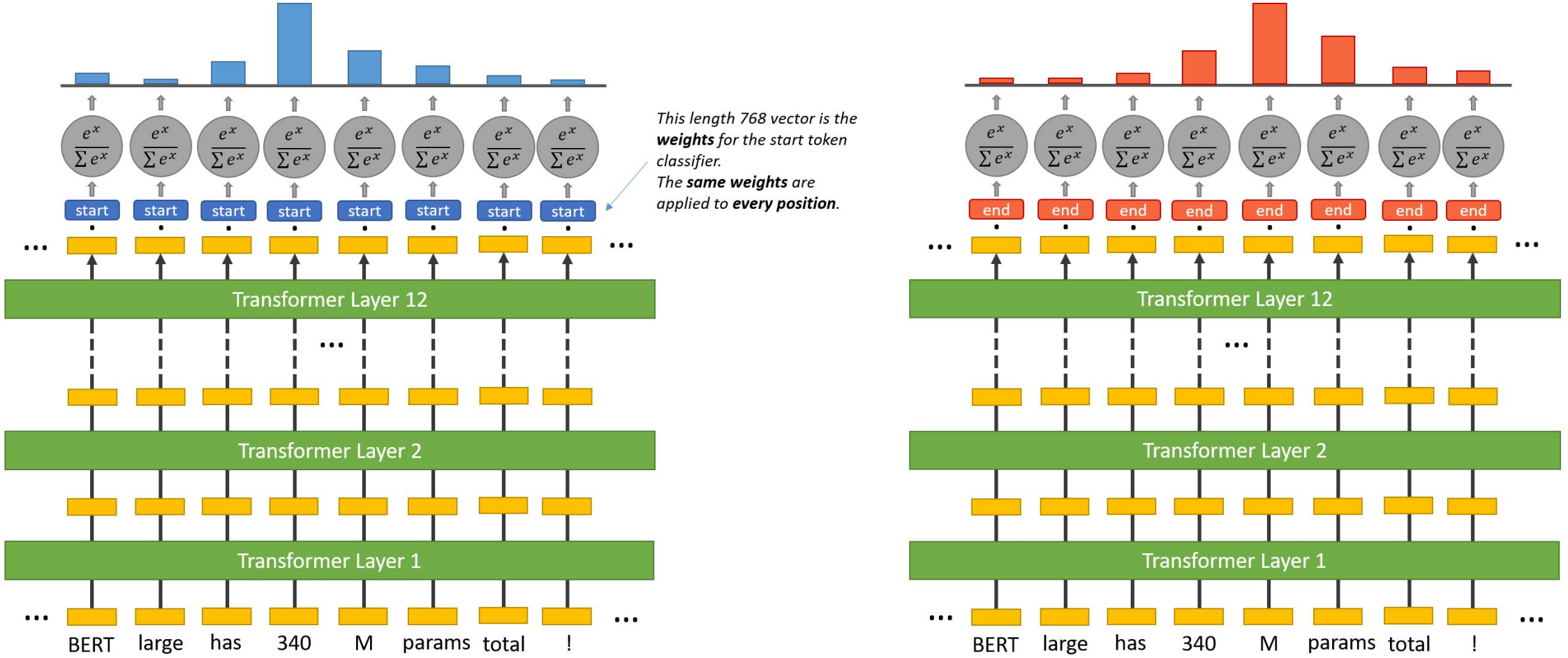
Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

Source: https://mccormickml.com/2020/03/10/question-answering-with-a-fine-tuned-BERT/

Extractive question answering

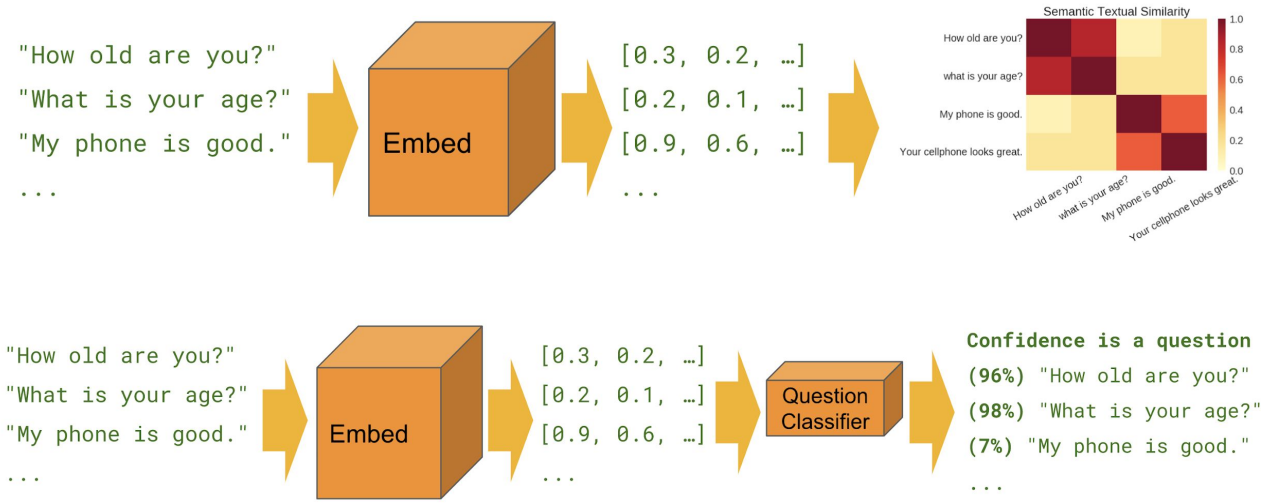
The most likely answer is the span with the maximal product of start and end probabilities.



Sentence encoders

Sentence encoders

- A sentence encoder is a model that converts a sentence (or another short text) to one fixed-size vector
  - BERT instead turns a text into a sequence of vectors (one per token)
    - But if we pool them (average or the embedding of [CLS]), it is a sentence encoder
- Applications:
  - Semantic similarity of sentences
    - Dot product or cosine similarity
    - Used e.g. for fast semantic search
  - Features for text classification
    - Works well in few-shot setting, especially with KNN
  - Cross-lingual transfer



Sources: https://tfhub.dev/google/universal-sentence-encoder/4; Cer et al, 2018, Universal Sentence Encoder.

Training sentence encoders

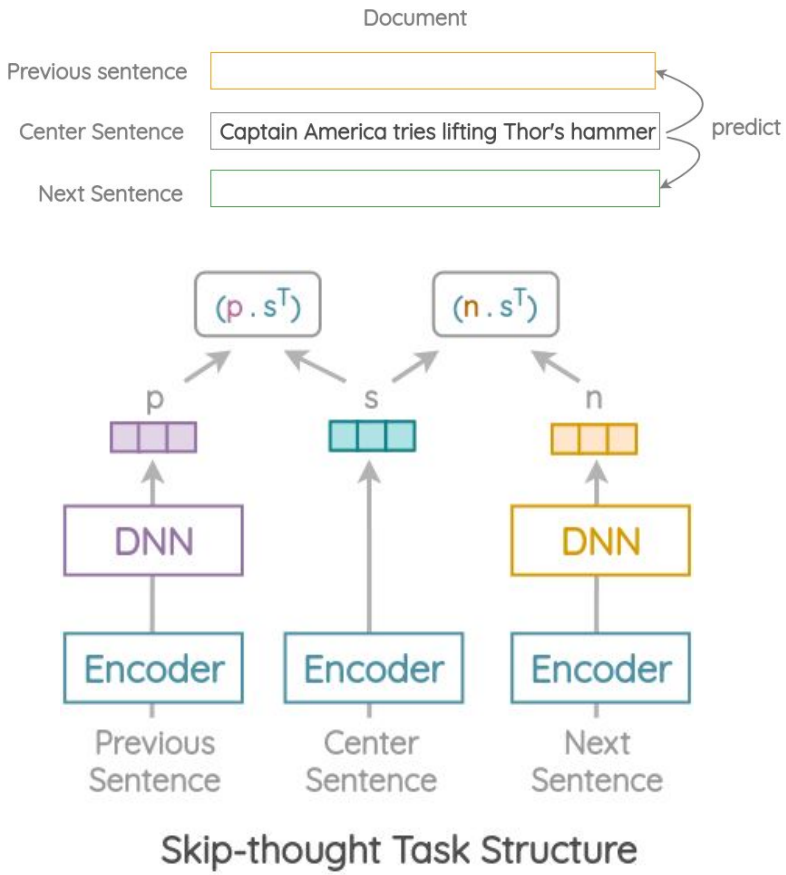
- Backbone model: transformer encoder[12345], CNN[13]
- Pretraining: MLM[245], translation MLM[5]
- Training objectives:
  - Unsupervised contrastive learning[4]
    - Positive examples are created by augmentation (simply applying dropout)
    - All other sentences in the batch are negative examples
  - Translation ranking[35]
    - Positive examples are sentences with the same meaning in another language
  - NLI[1234]
    - entailment for positive examples, contradiction for negative examples
  - Skip-thought[1], predicting dialogue response[13], various classification tasks[1]

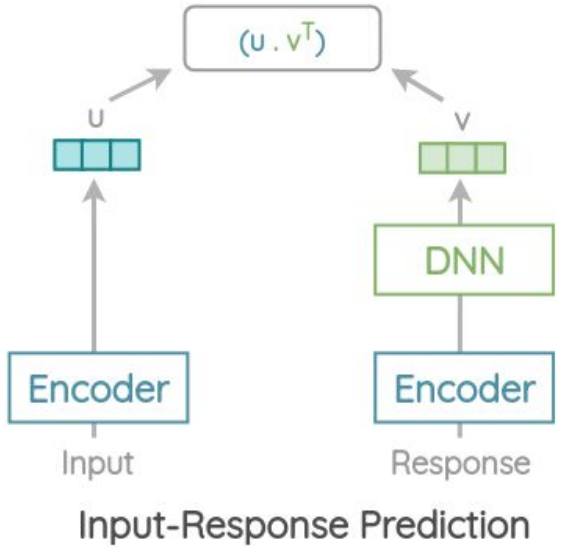
$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\frac{e^{sim(x_i,y_i)/\tau}}{\sum_{j=1}^{N} e^{sim(x_i,y_j)/\tau}}$$

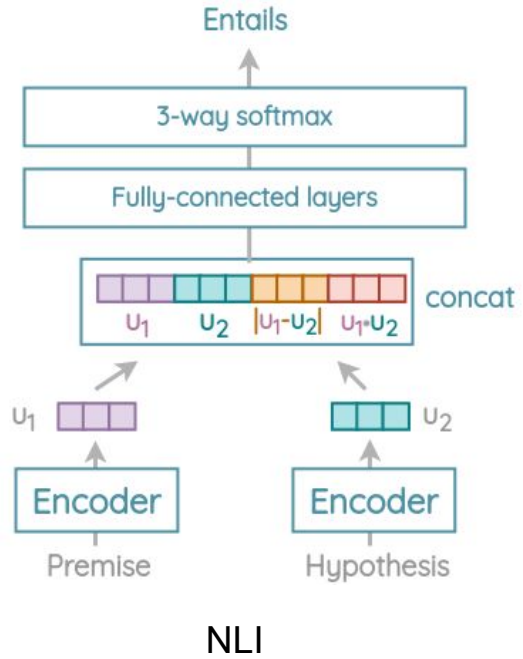
A typical loss for contrastive learning: softmax with (x_i, y_i) pairs as positive examples, (x_i, y_j) as negatives

1. Cer et al, 2018, Universal Sentence Encoder
2. Reimers et al, 2019, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
3. Yang et al, 2019, Multilingual Universal Sentence Encoder for Semantic Retrieval
4. Gao et al, 2021, SimCSE: Simple Contrastive Learning of Sentence Embeddings
5. Feng et al, 2022, Language-agnostic BERT Sentence Embedding

Some pretraining tasks for USE







Source: https://amitness.com/2020/06/universal-sentence-encoder/

Evaluating sentence encoders

Sentence encoders can be evaluated on multiple tasks:

- Probing tasks: evaluate linguistic capacities
  - Predicting sentence length, dependency tree depth, word order, verb tense, etc.
- Downstream tasks: evaluate applications
  - Sentiment analysis (MR, CR, SST)
  - Subjectivity classification (SUBJ)
  - Opinion polarity (MPQA)
  - Question type detection (TREC)
  - NLI (SICK, SNLI)
  - Semantic similarity (STS, SICK)
  - Image-caption retrieval (COCO)

Model	MR	CR	SUBJ	MPQA	TREC	SST	MRPC
English Models							
InferSent	81.1	86.3	92.4	**90.2**	88.2	84.6	76.2
Skip-Thought LN	79.4	83.1	93.7	89.3	–	–	–
Quick-Thought	**82.4**	86.0	94.8	**90.2**	92.4	**87.6**	**76.9**
USE_{Trans}	82.2	84.2	**95.5**	88.1	93.2	83.7	–
Multilingual Models							
m-USE_{Trans}	78.1	**87.0**	92.1	89.9	**96.6**	80.9	–
LaBSE	79.1	86.7	93.6	89.6	92.6	83.8	74.4

A table from Feng et al, 2022, Language-agnostic BERT Sentence Embedding

- Sentence encoders are used as fixed feature extractors
  - In contrast with other benchmarks such as SuperGLUE, where models are typically fine-tuned on each evaluation problem

Conneu et al, 2018, SentEval: An Evaluation Toolkit for Universal Sentence Representations

Some applications of sentence encoders

- Classifiers over fixed embeddings for few-shot classification or cross-lingual transfer
  - Efficient Intent Detection with Dual Sentence Encoders
  - Cross-language sentiment analysis of European Twitter messages during the COVID-19 pandemic
  - EMET: Embeddings from Multilingual-Encoder Transformer for Fake News Detection
  - Deep Learning Models for Multilingual Hate Speech Detection
- Retrieval of relevant sentences and paragraphs
  - ReQA: An Evaluation for End-to-End Answer Retrieval Models
  - Intelligent Translation Memory Matching and Retrieval with Sentence Encoders
  - WikiMatrix: Mining 135M Parallel Sentences in 1620 Language Pairs from Wikipedia

Text retrieval

Information retrieval

= Finding material that fulfills an information need
from within a large collection of unstructured documents.

Expression of Information Need	Potential Query	Potential Collection
Find related literature	The full text of the BERT paper	ACL anthology; arXiv CL
Recommend me a TV show to watch	[no explicit query!]	Netflix shows
Find every relevant patent	Boolean query with technical terms	U.S. Patents
Buy a new laptop	Short conversation: system asks questions to ascertain your criteria	E-commerce platforms

Source: http://web.stanford.edu/class/cs224u/

Non-neural search

- The problem:
  - Find (and rank) the most relevant documents related to the query
  - Do it fast!
- The basic solution: inverted index
  - Split the documents into words
  - Create the mapping from words to document ids
  - Consider only documents that contain the words from the query
  - Rank documents by TF-IDF, BM25, etc.
  - By remembering word positions in the document, we can look up by word n-grams

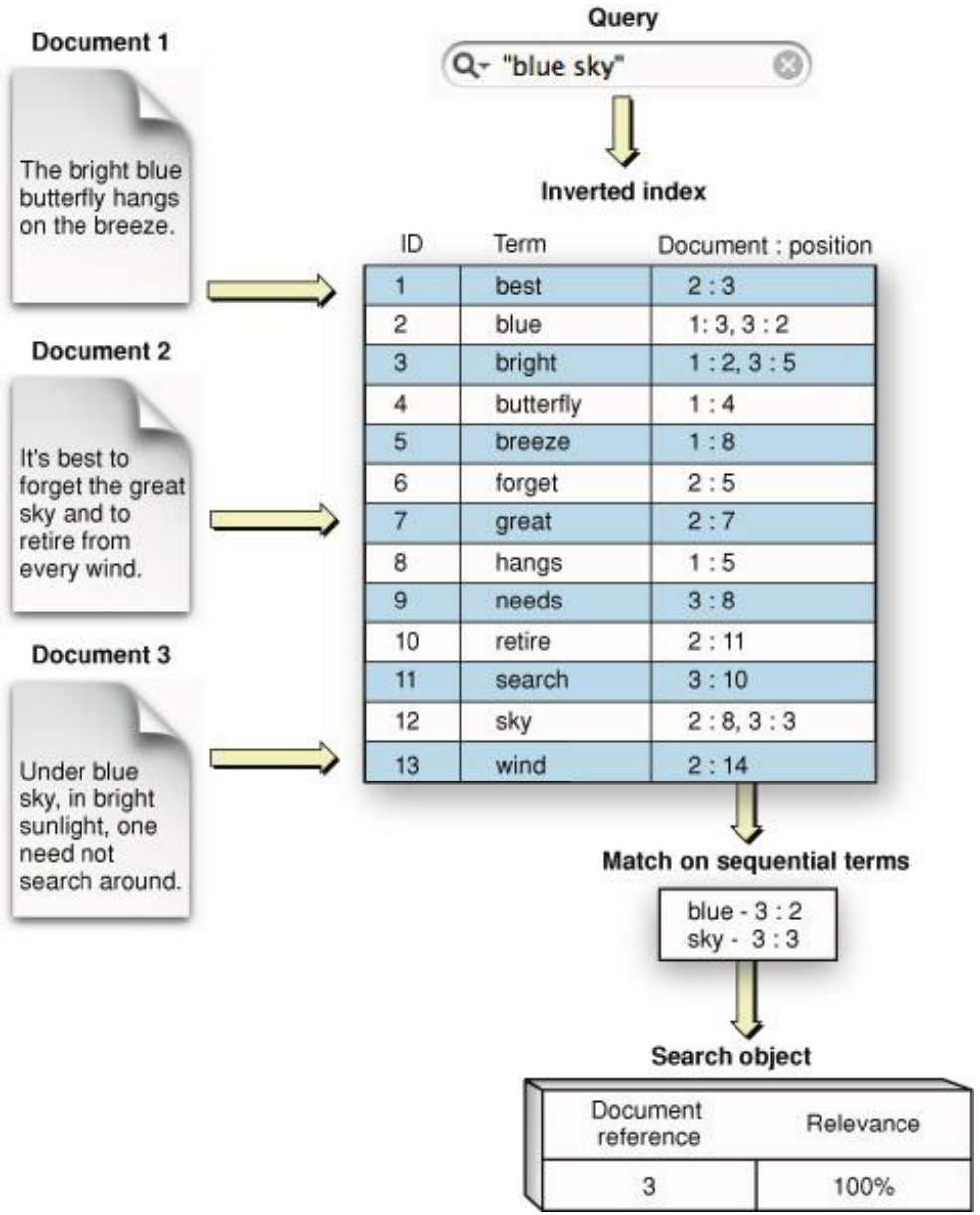


Image source: https://spec-zone.ru/RU/OSX/documentation/UserExperience/Conceptual/SearchKitConcepts/searchKit_basics/searchKit_basics.html

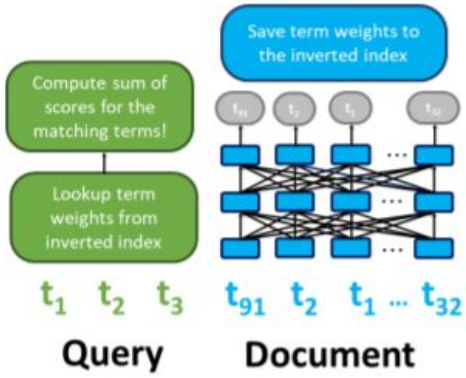
Transformers for text retrieval

- Retrievers:
  - Relevance = cosine_similarity(encoder(query), encoder(passage))
  - Passage embeddings can be precomputed once
  - Query and passage can be encoded with the same or different models (“Siamese network”)
  - Cosine similarity is equivalent to Euclidean distance □ fast (approximate) nearest neighbor lookup is possible (e.g. FAISS)
- Rerankers:
  - Relevance = classifier(encoder(query + passage))
  - Works slower but better than retrievers due to cross-attention between query and passage
- The reranker can be combined with the reader
- The retriever or reranker can be trained jointly with the reader

Source: https://github.com/danqi/acl2020-openqa-tutorial/

Late document-query interaction

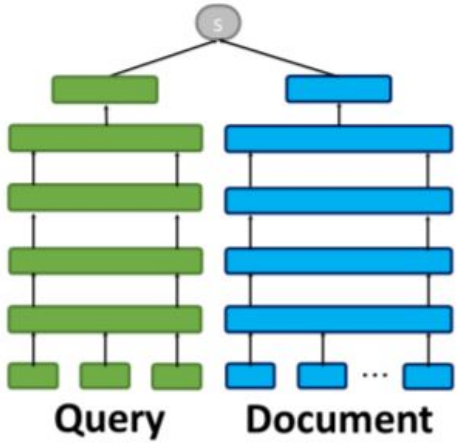
Alternative neural ranking paradigms



(a) Learned Term Weights

✔ Independent Encoding

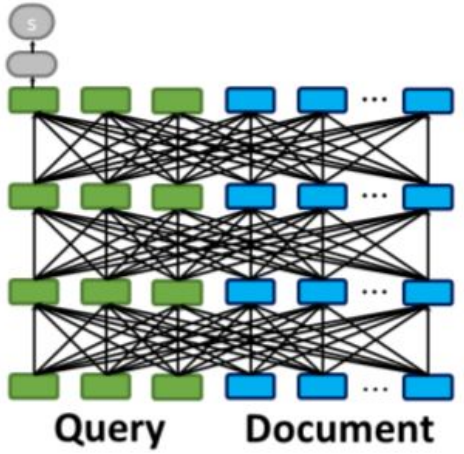
✘ Bag-of-Words Matching



(b) Representation Similarity

✔ Independent, Dense Encoding

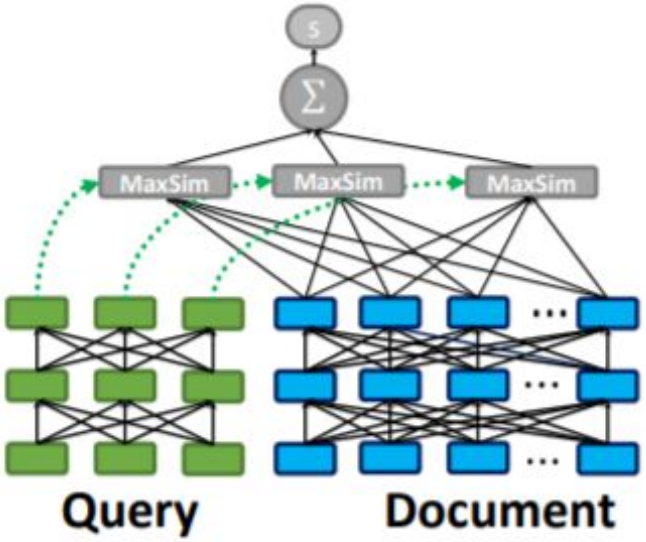
✘ Coarse-Grained Representation



(c) Query–Document Interaction

✔ Fine-Grained Interactions

✘ Expensive Joint Conditioning



(d) Late Interaction

(i.e., ColBERT)

✔ Independent Encoding

✔ Fine-Grained Representations

✔ End-to-End Retrieval (pruning!)

Sources: http://web.stanford.edu/class/cs224u/slides/CS224U-IR-part-5.pdf
https://arxiv.org/abs/2004.12832 (ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT)

Dialogue systems

Typical tasks of dialogue systems

- Search (≈single-turn dialogue)
  - Responses should be relevant
  - Typically solved with retrieval or QA tools
- General conversation
  - Responses should be engaging and safe
  - Typically solved with generative or retrieval-based chatbots
- Goal-oriented scenarios
  - Responses should solve the user’s problem quickly and correctly
  - Accurate understanding of requests is extremely important
  - Typically solved with request parsing and lots of business logic

General conversation (chit-chat)

- The goal: support conversation on any topic
- Generative approach
  - Previous dialogue --> model --> response
  - Before GPTs, such models were difficult to train
  - Responses can be (theoretically) very diverse
  - Quality control is difficult
- Retrieval approach
  - Previous dialogue + candidate response --> model --> estimate of relevance
    - With Siamese networks, responses can be selected from millions of candidates
  - The pool of possible responses can be collected and filtered in advance
- Retrieved and generated responses can be pooled together and reranked

Goal-oriented scenarios

- «Wake me up tomorrow at half past seven»
  - Intent (=type of the request): `set_alarm`
  - Slots (= entities supported by the intent):
    - `time: 07:30`
    - `date: tomorrow`
- Fulfilling requests: handwritten business logic + working with APIs
- Responses are typically generated by filling templates
- Challenges:
  - Understanding the context («The day after tomorrow as well»)
  - Developing NLU without real data from users (for new scenarios)

Multiclass text classification

Named entity + role recognition

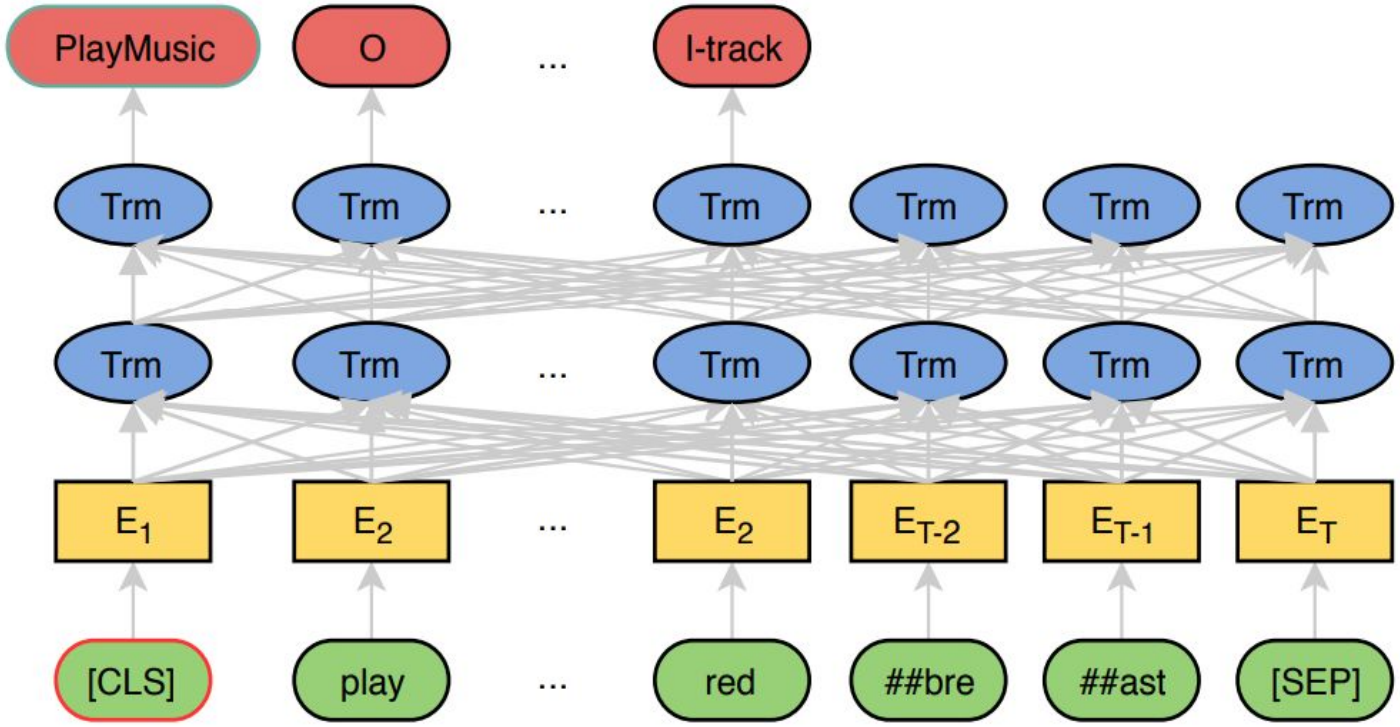
Dialogue management

- Slot-filling
  - Frame is a set of slots with a questions for each one
  - A scenario is triggered by user (by a specific intent)
    - Some slots can be inherited from the previous dialogue
  - The bot asks questions until all required slots are filled, then fulfills the request
- Belief-state
  - State is the current frame with all accumulated slots
  - Some systems support a distribution over alternative states
- Dialogue acts
  - Act is a general type of intent (hello, inform, request, confirm, etc.)
  - Detecting acts can help choosing the right action
- Dialogue policy: choosing the next action of the bot
  - E.g. whether to ask for confirmation, suggest extra options, or fulfill the request
  - In some academic systems, a probabilistic policy is learned by supervised or reinforcement learning
  - In most industrial systems, policies are deterministic and rule-based

Source: https://web.stanford.edu/~jurafsky/slp3/

BERT for Joint Intent Classification and Slot Filling

- The intent is predicted based on the hidden state of the first special token [CLS]
- For slot filling, we feed the final hidden states of other tokens into a softmax layer to classify over the slot filling labels
- Train model in multitask mode



Sources: https://arxiv.org/pdf/1902.10909.pdf

Example: schema guided dialogue state tracking challenge

Each service is described by a schema

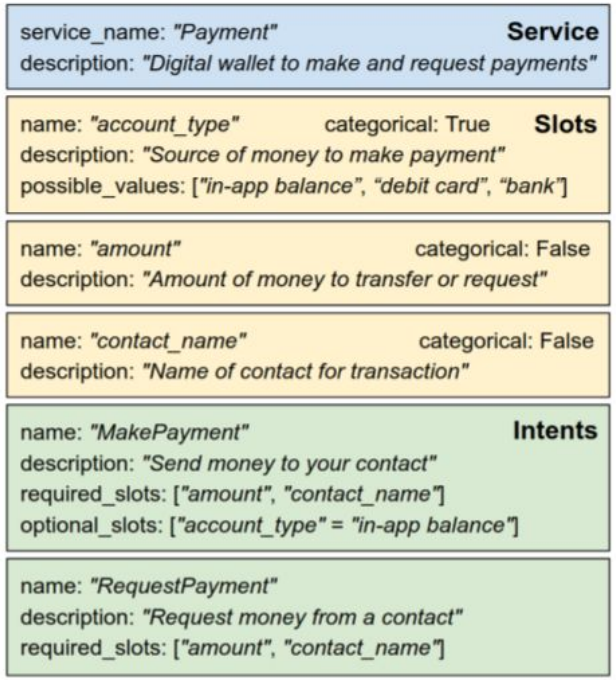


Figure 1: Example schema for a digital wallet service.

Dialogue state tracking task:

- Identify the service (not in this particular challenge)
- Classify the current intent
- Extract the slots provided and/or requested by user

A possible solution: combination of sequence classification (for intents and categorical slots) and sequence tagging (for free-form slots) based on BERT embeddings of dialogue and schema.

Sources: https://arxiv.org/abs/1909.05855, https://arxiv.org/abs/2002.01359

Summary

- Encoder-based models (such as BERT) are good for NLU:
  - Classification of texts, tokens, or relations between them
  - Encoding texts into informative vectors
- Many diverse NLP problems can be reformulated as sequence or token classification and solved by standardized models
- Systems based on transformer encoders are widely used for practical applications, such as information retrieval or chatbots