Study and Application of Concept-Extraction Algorithms in Natural Language Processing (NLP)

Ana-Maria Vintila
Supervised by Professor Brenda Vo

University of New England, Australia

About Me

- Name: Ana-Maria Vintila
- Age: 22
- International citizen, lived and studied in Canada for a majority of my life before returning to native Europe.
- Diligent, thrive in well-defined, scientific domains.
- Interests: Calculus, Scala / Haskell functional programming, Probability (with Wolfram-Mathematica in ProbOnto-style)
- Currently working towards doing Master By Research in NLP and / or Finance using applied maths and Bayesian statistics.
- Envisioning: NLP business to study text for global clients, and for financial analysis for currency exchange markets.

Introduction: Motivation for Text Processing

Introduction: Motivation for Text Processing

- Vast knowledge is trapped in presentation media such as videos, html, pdfs, and paper as opposed to being concept-mapped, interlinked, addressable and reusable at fine grained levels.
- This defeats knowledge exchanges between humans and AI.
- Known that concept mapping enhances human cognition.
- Especially in domain-specific areas of knowledge, better interlinking would be achieved if concepts would be extracted using surrounding context, accounting for polysemy and key phrases. “You shall know a word by the company it keeps” (Firth, 1957).
- Previous models GloVe and Word2Vec motivated recent models to move beyond simple co-occurrence counts to extract meaning.
- ERNIE 2.0 instead “broadens the vision to include more lexical, syntactic and semantic information from training corpora in form of **named entities** (like person names, location names, and organization names), **semantic closeness** (proximity of sentences), **sentence order or discourse relations**” (Sun et al., 2019).
- **Example:** ERNIE 1.0 can associate entire entity names with other terms in a given sentence, while on the same data, BERT lacks this ability.
- **Aim of This Project:** To understand how models make good language representations by inventorying Transformer, ELMo, BERT, Transformer-XL, XLNet, and ERNIE 1.0 in how they leverage entities, polysemy, context for concept extraction.

Word Embeddings

Word Embedding: Definition

Definition: Word Embedding

Word embeddings are unsupervised models that capture semantic and syntactic information about words in a compact low-dimensional vector representation ⇒ useful for reasoning about word usage and meaning (Melamud et al. 2016, p. 1).

- Sentence embeddings, phrase embeddings, character embeddings (for morphology)
- Can capture **vector space semantics**: can express word analogy "man is to woman as king is to queen" with arithmetic on learned word vectors:
  $vector(man) - vector(woman) = vector(king) - vector(queen)$ (Smith, 2019).
- **Mathematically, How Models Make Embeddings:**
  ⇒ a word's conditional probability combines its *embedding* and *context vectors* of surrounding words, with different methods combining them differently.
  ⇒ Then, embeddings are fitted to text by maximizing the conditional probabilities of observed text.

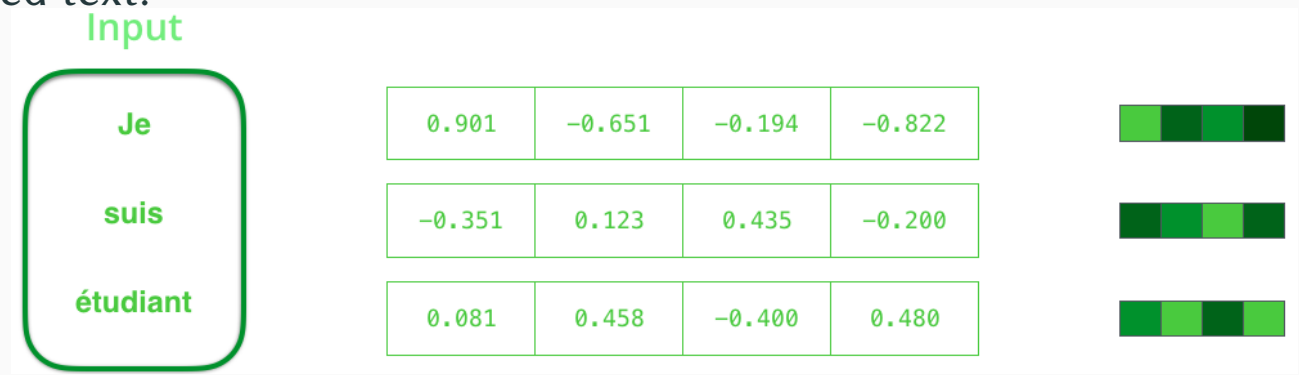

Figure 1: Example Word Embeddings. From *Visualizing Neural Machine Translation Mechanics of Seq2Seq Models with Attention*, by Jay Alammar, 2018. http://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

What is Polysemy?

Definition

Polysemy means a word has multiple senses.

Definition

Distributional hypothesis in NLP says meaning depends on context, and words in same contexts have similar meaning (Wiedemann et al., 2019).

Static Embeddings

- Classic word vectors are called **static embeddings**. They represent each word with a single vector, regardless of its context (Ethayarajh, 2019).

Example

Skip-Gram and Glove produce these "context-free" representations because they use co-occurrence counts, not the more dynamic **language modeling** approach (Batista, 2018).

Alert

All senses of a polysemic word are *collapsed* within a single vector representation (Ethayarajh, 2019).

Confusion!

"Plant"'s embedding would be the "average of its different contextual semantics relating to biology, placement, manufacturing, and power generation" (Neelakantan et al., 2015).

Better: Contextual Embeddings (CWE)

Definition

A contextual word embedding (CWE) captures context using forward and backward history, using a bidirectional language model (biLM) (Antonio, 2019).

Static word embeddings are like "look-up tables" but contextual embeddings have word type information (Smith, 2019).

- Abandoning the idea of using a fixed word sense inventory (to model polysemy) allows CWE's to create create a vector representation for each **word type** in the vocabulary and each **word token** in a context.
- Experimentally superior to static embeddings.
- "sentence or context-level semantics together with word-level semantics proved to be a powerful innovation" in the NLP world (Wiedemann et al., 2019).

Language Models

Language Models

Definition

A **language model** takes a sequence of word vectors and outputs a sequence of predicted word vectors by learning a probability distribution over words in a vocabulary.

They predict words sequentially, unidirectionally, one token at a time, using some context words.

Formally, they compute the conditional probability of a word w_t given a context, such as its previous $n-1$ words, where the probability is: $P(w_t \mid w_{t-1}, ..., w_{t-n+1})$

Types of Language Models

- **n-gram language model:** an n-gram is a sequence of n words. The model finds a word's probability based on frequencies of its constituent n-grams, taking just preceding $n-1$ words as context instead of the entire corpus.
- **neural network model:** uses a linear $W \cdot x + b$ function called a neuron. By applying a nonlinear function $f(\cdot)$ to this equation and by incorporating many hidden layers and by stacking neurons together, a neural network can model any function.
  Has **embedding layer**, **intermediate layers** to transform embeddings, and **final layer**, which often uses softmax function to normalize word embedding matrix to create probability distribution over words.
- **bidirectional language model (biLM):** calculates probability of a token given its *forward* and *backward* tokens. Uses **forward** and **backward language model**s, respectively.

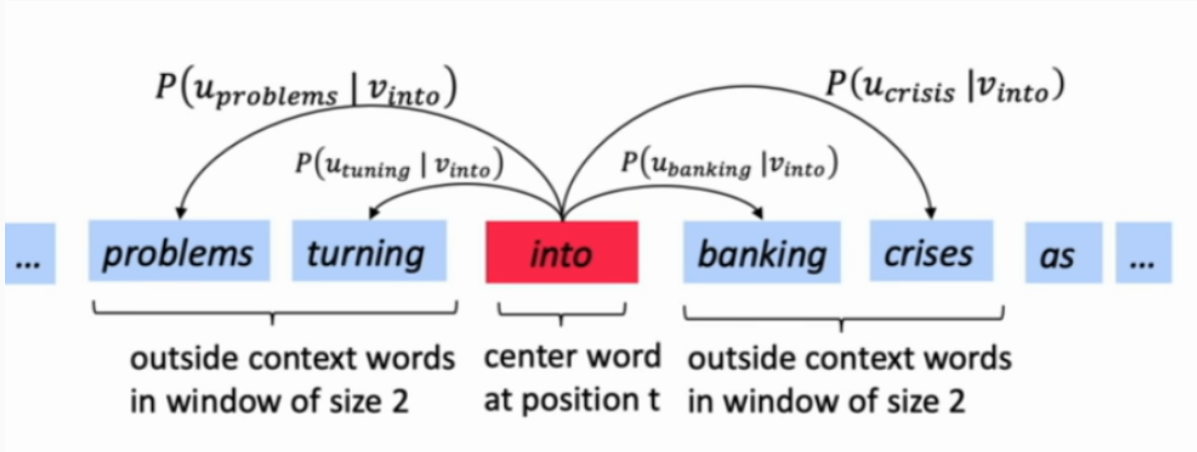

Figure 2: Example Bidirectional Language Model. From *Word2Vec Overview With Vectors*, by CS224n: Natural Language Processing with Deep Learning (Stanford), 2018. https://sangminwoo.github.io/2019-08-28-cs224n-lec1/. Copyright n.d. by n.d.

Word2Vec

Word2Vec: One-Hot Encodings

Definition: One-Hot Encoding

A **one-hot vector encoding** is the simplest type of word embedding where each cell in the vector corresponds to a distinct vocabulary word.

A 1 is placed in the cell marking the position of the word in the vocabulary, and a 0 is placed in all other cells.

Warning!

One-hot encodings cause ...

- high-dimensionality vector representations for large vocabularies, ⇒ increased computational costs.
- similarity between (word) categories cannot be represented.

Word2Vec: Basic Structure of Skip-Gram and CBOW

- Both are neural networks with one hidden layer that is a word embedding with dimension N. (Thus even though the vocabulary size is V, the goal is to learn embeddings with size N).
- input vector is $\vec{x} = (x_1, ..., x_V)$
- output vector is $\vec{y} = (y_1, ..., y_V)$
  - both are one-hot encodings
- At time t, they predict one output word w_{t+j} (whose vector representation is $\vec{y}$), given one input word w_t (whose vector representation is $\vec{x}$).
- **Skip-Gram:** w_{t+j} is the *predicted context word* and w_t is the *input target word*,
- **CBOW:** w_{t+j} is the *predicted target word* and w_t is the *input context word*.
- Notation details below[1]

[1] Vectors v_w and v'_w are two representations of word w. Vector v_w comes from the rows of the *input layer → hidden layer weight matrix* W, and vector v'_w comes from the rows of the *hidden layer → output layer weight matrix* W'. We call v_w the **input vector** and v'_w is the **output vector** of the word w.

Word2Vec: Skip-Gram

- Predicts context words given a single target word.
- Uses a fixed sliding window c, or size of the training context, around a target word, to capture context (bidirectionally) along a sentence.
- Target center word (one-hot encoding) is input to a neural network which updates the vector with values near 1 in cells corresponding to predicted context words (Weng, 2017).

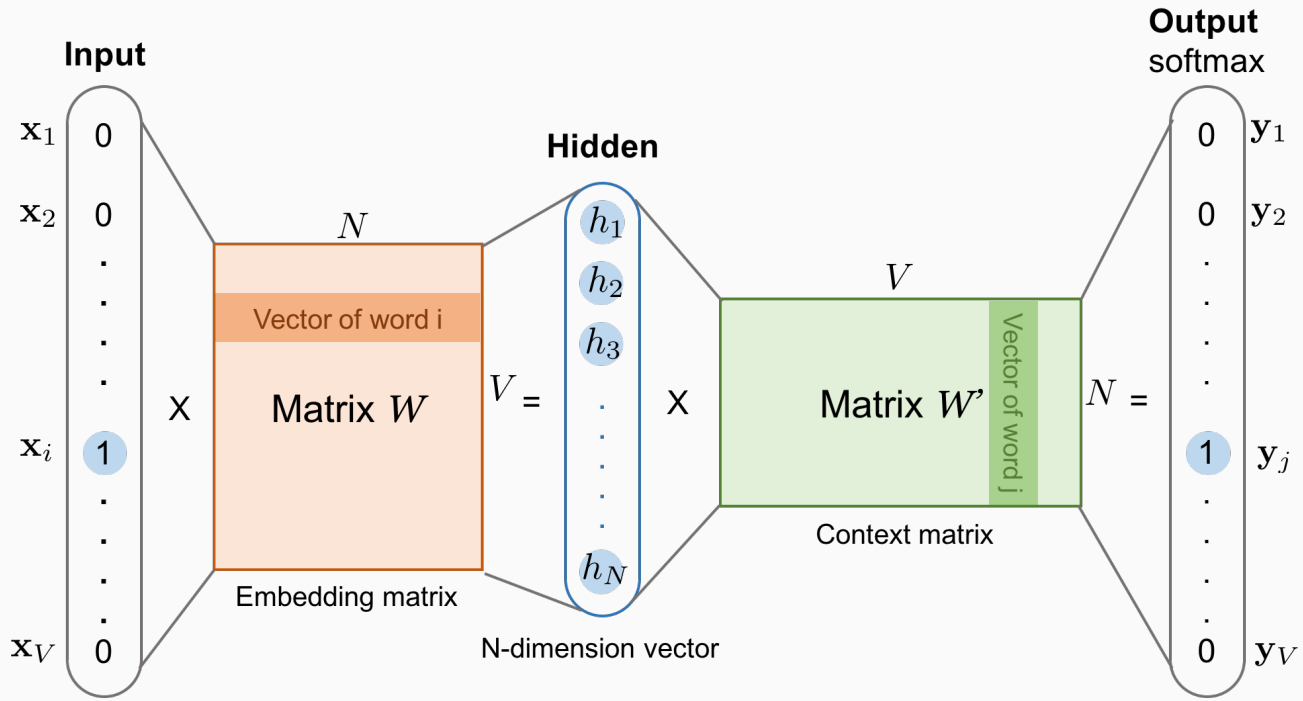

Figure 3: Skip-Gram Model; simplified version, with one input target word and one output context word. From *Learning Word Embeddings*, by Lilian Weng, 2017. https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html. Copyright n.d. by n.d.

Word2Vec: Continuous-Bag-of-Words (CBOW)

- **Continuous bag of words model (CBOW)** is opposite of the Skip-Gram: predicts *target* word based on a *context* word.
- Averages n context words around target word w_t to predict target (in hidden layer calculation): $\vec{h} = \frac{1}{c} W \cdot \left(\overrightarrow{x_1} + \overrightarrow{x_2} + \ldots + \overrightarrow{x_c}\right) = \frac{1}{c} \cdot \left(\overrightarrow{v_{w_1}} + \overrightarrow{v_{w_2}} + \ldots + \overrightarrow{v_{w_c}}\right)$

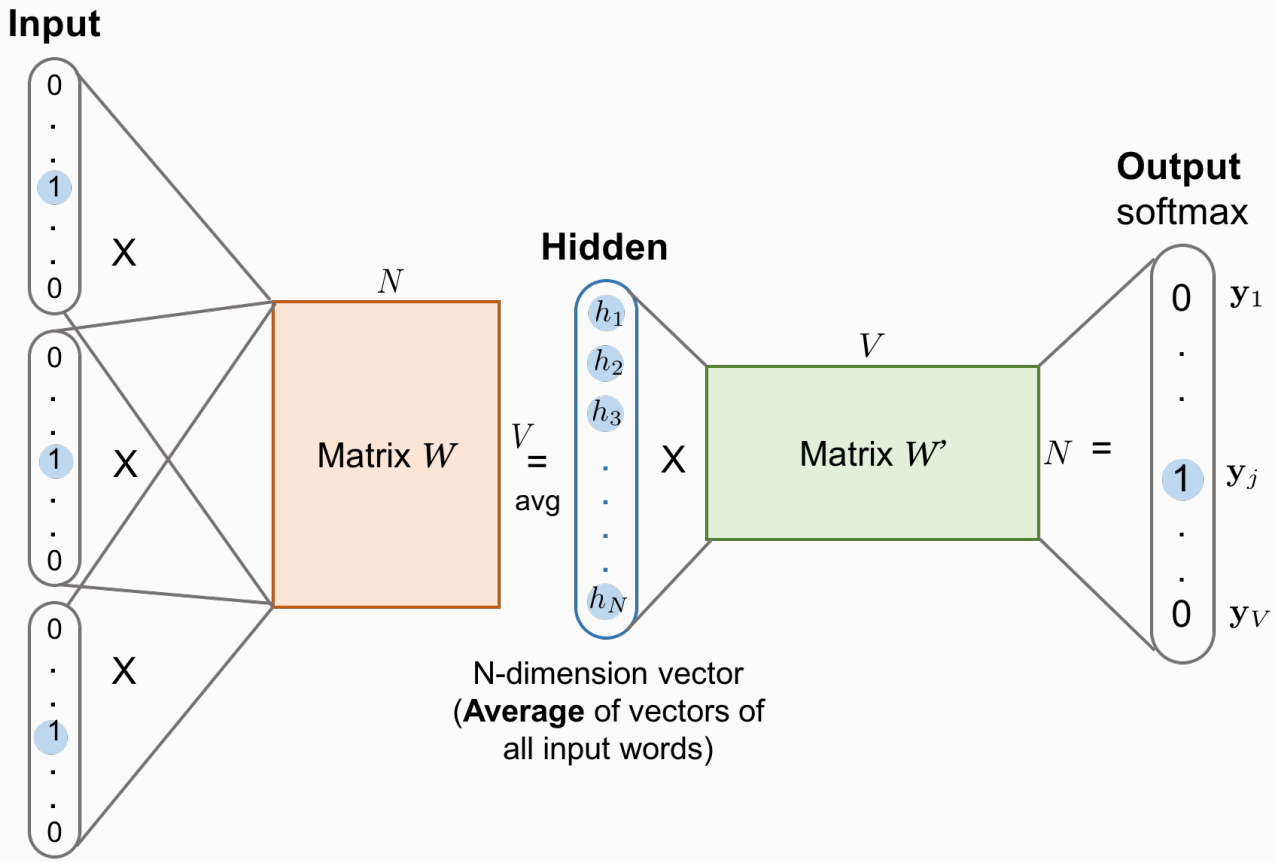

Figure 4: CBOW Model with several one-hot encoded context words at the input layer and one target word at the output layer. From *Learning Word Embeddings*, by Lilian Weng, 2017. https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

Phrase-Learning in Skip-Gram

- **Problem with previous word vectors:** no *phrase representation*
  ⇒ "Canada" and "Air" in a phrase could not be recognized as part of a larger concept and thus combined into "Air Canada" (Mikolov et al., 2013a).

- **Phrase-Skip-Gram Model**:
  - uses *unigram and bigram counts* to make phrases.
    - $S_{phrase} = \frac{C(w_i w_j) - \delta}{C(w_i) C(w_j)}$ where $C(\cdot)$ = count of a unigram w_i or bigram $w_i w_j$ and δ is a discounting threshold to avoid making infrequent words and phrases.
    - Large S_{phrase} indicates the *phrase* is a phrase, less likely a bigram.

- **Result:** linear structure **additive compositionality** of word vectors
  - **Additive Compositionality:** lets some individual words be combined into a phrase and be viewed as a unique *entity*, while a bigram like "this is" should remain unchanged (Mikolov et al., 2013a, p. 5).
    - **How Does It Happen?** product of distributions in loss function acts like an "and" function

Example: Additive Compositionality

If the phrase "Volga River" appears numerously with "Russian" and "river" then:
vector(`"Russian"`)+*vector*(`"river"`) ⇒ *vector*(`"Volga River"`)

GloVe (Global Vectors for Word Representation)

GloVe: Problem With Word2Vec

- **Problem:** Word2Vec uses local not global counts.
  - **Example:** "the" and "cat" may occur frequently but Word2Vec does not know if this is because "the" is a common word or because "the" and "cat" are actually correlated (Kurita, 2018).
- **Reason:**
  - Word2Vec implicitly optimizes over a co-occurrence matrix while streaming over input sentences ⇒ context words are processed equally.
    - Word2Vec's loss function $J = -\sum_i X_i \sum_j P_{ij} \log(Q_{ij})$ is the cross entropy between the *predicted* and *actual word distributions* in the context of word i. [2]
    - GloVe's authors say cross entropy models long-tailed distributions poorly, and here X_i means equal-weighting over words.
  - Kurita (2018a) says there is no inherent justification for equal-streaming over words: GloVe uses *unnormalized* probabilities.

---

[3] $X_i = \sum_k X_{ik}$ is the total number of words appearing in the context of word i

$Q_{ij} = \text{softmax}\left(\texttt{ice} \cdot \texttt{steam}\right)$ is the probability that word j appears in context of word i

GloVe

- **Assumption:** words co-occur when they appear within a fixed sliding window ⇒ reveals corpus-wide, not sentence-level word meaning.

How GloVe Uses Co-Occurrence Ratios [3]

Consider two words `ice` and `steam`.

- **Case 1:** For context words related to `ice` but not "steam" (`solid`) ⇒ expect co-occurrence probability $p_{\text{co}}\left(\texttt{solid} \mid \texttt{ice}\right)$ is much larger than $p_{\text{co}}\left(\texttt{solid} \mid \texttt{steam}\right)$ ⇒ ratio $\dfrac{p_{\text{co}}\left(\texttt{solid} \mid \texttt{ice}\right)}{p_{\text{co}}\left(\texttt{solid} \mid \texttt{steam}\right)}$ gets very large.
- **Case 2:** Conversely, for words related to `steam` but not `ice` (like `gas`) ⇒ the co-occurrence ratio $\dfrac{p_{\text{co}}\left(\texttt{gas} \mid \texttt{ice}\right)}{p_{\text{co}}\left(\texttt{gas} \mid \texttt{steam}\right)}$ should be small.
- **Case 3:** For words w related to both `ice` and `steam` (like `water`) or related to neither (like `fashion`) ⇒ expect the ratio $\dfrac{p_{\text{co}}\left(w \mid \texttt{ice}\right)}{p_{\text{co}}\left(w \mid \texttt{steam}\right)}$ is near one.

[4] GloVe defines word co-occurrence probability as: $p_{\text{co}}\left(w_k \mid w_i\right) = \frac{C(w_i, w_k)}{C(w_i)}$ where $C(w_i, w_k)$ counts the co-occurrence between words w_i and w_k.

Performance: GloVe vs. Word2Vec

fig. 5 shows GloVe's learned embeddings have higher prediction accuracy over those of Skip-Gram and CBOW on **word analogy** and **named entity recognition (NER)**.

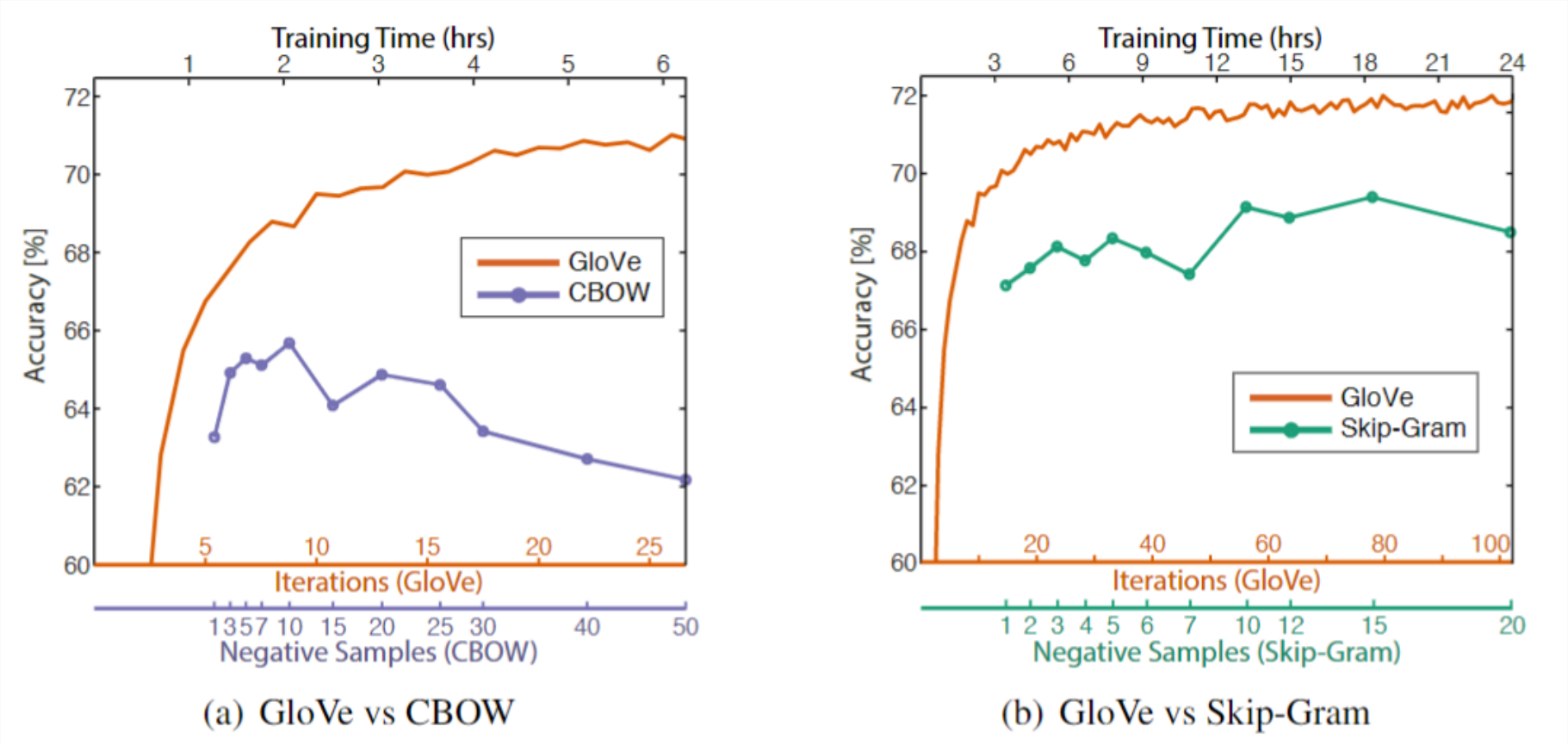

Figure 5: Overall accuracy on word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and Skip-Gram (b). Pennington et al. (2014) train 300-dimensional vectors on the same 6B token corpus from Wikipedia and use a symmetric context window of size 10. From *GloVe: Global Vectors for Word Representation*, by Pennington et al., 2014. https://nlp.stanford.edu/pubs/glove.pdf. Copyright 2014 by Pennington et al.

Sequence-To-Sequence Model

Seq-To-Seq Model (Brief Overview)

- Used for machine translation task. **Encoder** takes in sequence of tokens (words, phrases), processes inputs into a *fixed-length* **context vector**, and sends it to **Decoder** that outputs another sequence of tokens.
- Encoder, Decoder are often RNN's [4]like LSTMs[5]and GRUs[6].
- **Problem:** compressing inputs into **fixed-length vector** causes **long-term dependency problem** (memory loss) since only the *last hidden Encoder state is used*.
- **Solution:** to use **attention mechanism** for selectively focusing on parts of the inputs as required (creates a context vector for *each time step or word*, so all Encoder's output hidden states are used in Decoder)

---

[5]RNN stands for **recurrent neural network**, which has a looping mechanism to share hidden states. Suffers from **long-term dependency problem**

[6]LSTM means **long-short term memory network**, and uses forget gates to regulate memory and information flow

[7]GRU means **gated-recurrent network**, and is a condensed version of LSTM.

Transformer

Transformer: Self-Attention

- Kind of seq-to-seq model used for machine translation; more parallelizable than seq-to-seq via abandoning RNNs and using only **self-attention** to generate sequence of *contextual embeddings*.

Example: Motivation for Self-Attention

"The animal didn't cross the road because it was too tired."

What does "it" refer to? The road or animal?

- **Self-Attention** lets Transformer look at other words to *bake in* their representations into the word "it" while processing the input sequence.
- An **attention function** maps query and key-value pairs to output vector:
  - Query matrix Q rows signify which word to do self-attention ("it") on; Key matrix K rows describe *each* word in the sentence; Value matrix V rows represent rest of the words (other than "it").
  - Final embedding of the word or **output** is a weighted sum of **value** vectors and softmax probabilities of the dot product between query and key vectors:

$$Attention\left(Q, K, V\right) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Cements the values of words on which to focus while drowning out irrelevant words.

Transformer: Multi-Head Attention

Definition: Multi-Head Attention

A **multi-head attention mechanism** comprises of several self-attention heads.

Enables the Transformer to "jointly attend to information from different representation subspaces at different positions."

A single attention head cannot do this because of averaging (Vaswani et al., 2017).

- Instead of calculating attention once, multi-head attention does (1) self attention many times in parallel on the projected dimensions, (2) concatenates the independent attention outputs, and (3) once again projects the result into the expected dimension to give a final value (Vaswani et al., 2017; Weng, 2018).
- More attention heads means Transformer can focus on different words; while encoding "it", one attention head looks at "the animal" while another focuses on "tired" ⇒ representation of "it" includes some of all words.

Transformer: Positional Encodings

Definition: Positional Encoding

A **positional encoding** follows a specific, learned pattern to identify word position or the distance between words in the sequence (Alammar, 2018b).

$$PosEnc_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PosEnc_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

where pos = a position, i = a dimension.

- Transformer has no recurrence, so can't yet see *order in the input sequence*.
- Positional encodings act in place of recurrence mechanism to inject relative / absolute token position info so Transformer can see *sentence order* when taking inputs.

Otherwise ...

... the sentences "I like dogs more than cats" and "I like cats more than dogs" would encode the same meaning (Raviraja, 2019).

Transformer: More Layers

- **Positionwise feed-forward layer** is a kind of **feed-forward neural network (FFN)**, and is "position-wise" since the FFN is applied to each position separately and identically.
- **Residual Connection:** a sub-layer in Encoder / Decoder stacks meant for harmonizing gradient optimization procedure.
- **Masked Multi-Head Attention:** key component in Decoder stack. Is attention with **masking** to prevent positions from attending to subsequent positions (while decoding word embedding $\overrightarrow{w_i}$, the Decoder is not allowed to see words $\overrightarrow{w_{>i}}$ past position i, only words $\overrightarrow{w_{\leq i}}$, so no "cheating" occurs (Ta-Chun, 2018).
- **Encoder:** is bidirectional RNN that concatenates *forward and backward* hidden states to get bidirectional context: $h_t = \left\{\overrightarrow{h}_t^T ; \overleftarrow{h}_t^T\right\}^T, t = 1, ..., T_x$. [7]
- **Decoder:** neural network generates hidden states $s_t = \text{Decoder}\left(s_{t-1}, y_{t-1}, c_t\right)$ for times $t = 1, ..., m$ [8]

---

[8] Note: arrows here denote the direction of the network rather than vector notation.

[9] Context vector $c_t = \sum_{i=1}^{n} \alpha_{ti} \cdot h_i$ is a sum of the hidden states of the input sentence, weighted by alignment scores (same calculation as in the seq-to-seq model)

Transformer: Encoder and Decoder Stack in Detail

- Encoder and Decoder are each composed of N identical **layers** or **stack**.
- A **residual connection layer** then layer normalization surround each sub-layer.

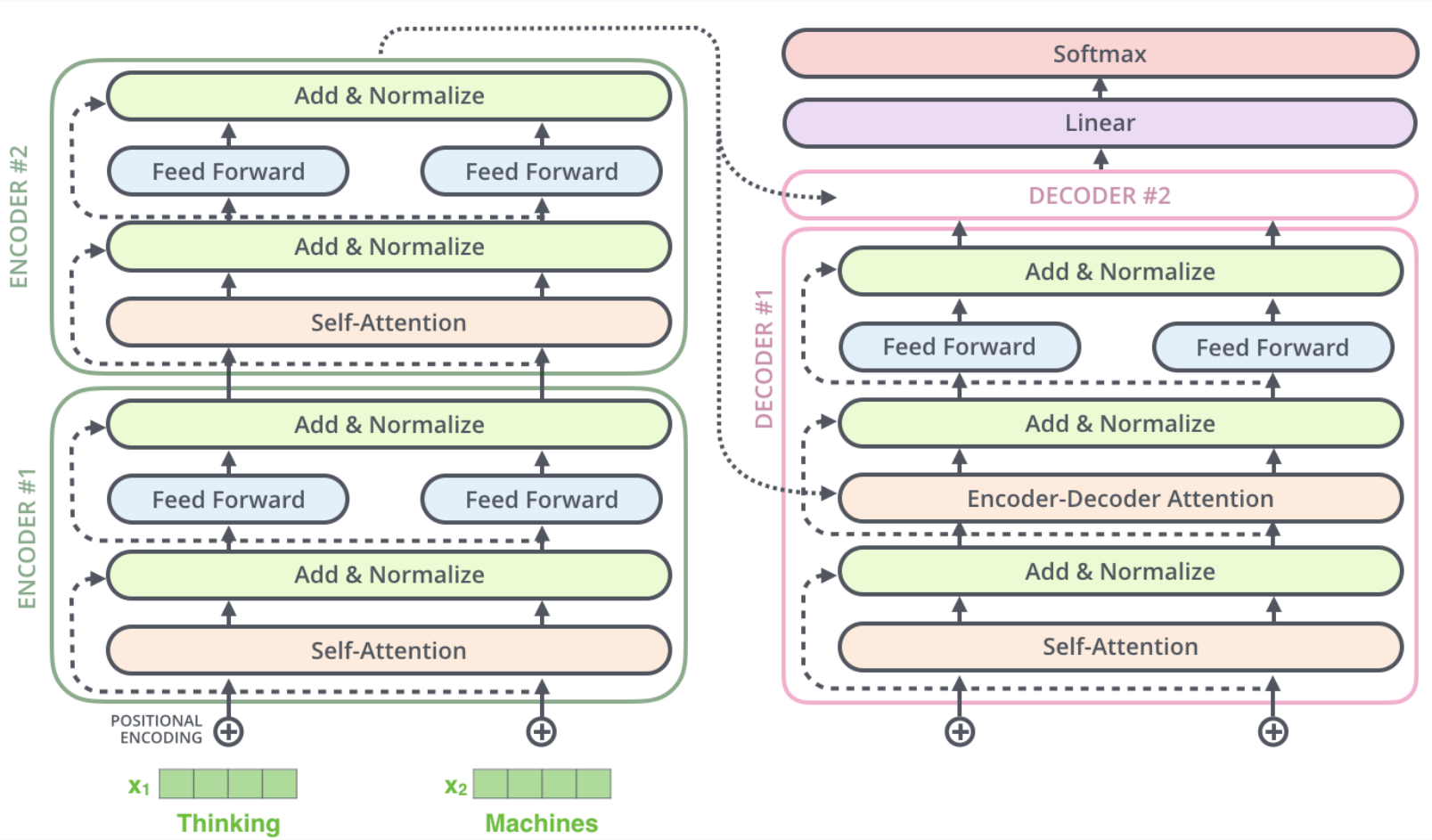

Figure 6: Encoder layer contains: (1) Multi-head attention, (2) Position-wise feed forward layer. Decoder layer contains: (1) Masked multi-head attention, (2) Encoder-Decoder attention, (3) Position-wise feed forward layer. From *The Illustrated Transformer*, by Alammar, 2018. https://jalammar.github.io/illustrated-transformer/. Copyright 2018 by Alammar.

Transformer: How Do We Predict a Word?

Generally ...

Decoder stack ⇒ Linear layer ⇒ Softmax layer.

- **Linear Layer** neural network projects Decoder's float vector to larger dimension "logit vector" (each cell holds a score corresponding to each unique vocabulary word.)
- **Softmax Layer** then converts the Linear Layer's scores into probabilities via the softmax function.

To find the predicted word: the cell with highest probability is chosen ⇒ its word is the "predicted" word.

ELMo (Embeddings from Language Models)

ELMo: Motivation

Motivation for ELMo: Remember **polysemy**? Contextual embeddings create one vector per word type in a context.

Example

If instead models used only word and character embedding, the homonyms "book" (text) and "book" (reservation) are assigned the *same vector representation even though these are different words.*

This vector representation may of course be created using context (as from Word2Vec or GloVe) but contextual meanings get *collapsed into a single representation.*

ELMo: Structure

- **ELMo** uses bidirectional language model (biLM) to make *deep* word embeddings (derived from all its internal layers)
- Higher-level LSTM layers capture contextual meaning (useful for supervised word sense disambiguation (WSD)).
- Lower layers capture syntax information (useful for part of speech tagging (POS)).
- Mixing these signals let ELMo's embeddings select the types of semi-supervision most needed for each end task ⇒ ELMo embeddings are thus richer than traditional word vectors.
- **Task-Specific:** ELMo is a task-specific combination of the intermediate layer representations in the biLM (collapses forward and backward hidden states into single embedding for a specific task by weighting the biLM layers): [9]

$$\mathbf{ELMo}_k^{task} = E\left(R_k; \theta^{task}\right) = \gamma^{task} \sum_{j=0}^{L} s_j^{task}\ \mathbf{h}_{kj}^{LM}$$

[10] the vector $\mathbf{s}^{task} = \left\{s_j^{task}\right\}$ of softmax-normalized weights and task-dependent scalar parameter γ^{task} allow the model for the specific $task$ to scale the entire $\mathbf{ELMo}_k^{task}$ vector. The index k corresponds to a k-th word, and index j corresponds to the j-th layer out of L layers. Here, h_{kj}^{LM} is the output of the j-th LSTM for word k, and s_j is the weight of h_{kj}^{LM} used to compute the representation for word k.

ELMo: Example and Key Feature

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Table 1: Nearest neighbors to "play" using GloVe and the context embeddings from a biLM. From *Table 4 in Deep Contextualized Word Representations*, by Peters et al., 2018.
https://arxiv.org/pdf/1802.05365.pdf. Copyright 2018 by Peters et al.

- table 1 displays words nearest to polysemous word "play" found using GloVe vs biLM.
- GloVe's neighbors have different parts of speech, like verbs ("played", "playing"), and nouns ("player", "game") and only in the sport sense.
- biLM's nearest neighbor sentences from "play" CWE show clear difference between *both* the parts of speech *and* word sense of "play".
  - last row: input sentence has noun / acting sense of "play" and this is matched in the nearest neighbor sentence

ELMo key feature: learn context using part of speech tagging (POS) and word sense disambiguation (WSD).

BERT (Bidirectional Encoder Representations from Transformers)

BERT: Motivation

- **Problem with ELMo:** shallowly combines "independently-trained" biLMs
- **Problem with OpenAI GPT:** uses a left-to-right construction, where each token only attends to past tokens in the self attention layers of the Transformer ⇒ cannot represent bidirectional context ⇒ does poorly on *sentence-level* and *token-level* tasks (question answering (QA))
- **BERT's Solution:** train "deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers" (Devlin et al., 2019).

BERT: Input Embeddings

1. *WordPiece* **token embeddings:** The *WordPiece* tokenization subdivides words to smaller units (instead of tokenizing by natural separations)
   - to handle rare, unknown words (Weng, 2019) and reduce vocabulary size while increasing amount of data available per word.
   - **Example:** if "play" and "**ing" and "**ed" are present in the vocabulary but "playing" and "played" are not, then these can be recognized by their sub-units.
2. **Segment embeddings:** are arbitrary spans of text (packing sentence parts). NOTE: Transformer-XL respects sentence boundaries.
3. **Positional embeddings:** as in ordinary Transformer (to inject word order information).

BERT Framework: MLM and NSP

BERT does **pre-training** (trains on *unlabeled data* over different tasks), and **fine-tuning** (to use pre-training parameters to train over *labeled data* for nlp tasks)

This improves over language models that just predict next tokens given context.

Pre-training involves:

Masked language model (MLM):

- **Motivation:** bidirectional conditioning causes information leakage (a word can implicitly "see itself" letting the model trivially guess the target word in a multi-layered context (Devlin et al., 2019)).
- **Goal:** randomly masks some input tokens to predict original masked word using only its context.
- **Effect of MLM:** fuses left and right context to get *deep* bidirectional context (unlike ELMo's shallow left-to-right language model (Devlin et al., 2019)).

Next Sentence Prediction (NSP):

- **Motivation:** to capture *sentence-level* information ⇒ to do well in question-answering (QA) and natural language inference (NLI) tasks
- **Goal:** task that finds if sentence is the next sentence of the other.

BERT: General Experimental Results

- Using a simple accuracy measure, Munikar et al. (2019) found that a pre-trained BERT model fine-tuned for **sentiment analysis (SA)** task outperformed complex models (RNN, CNNs).
- This proves **transfer learning** is possible with BERT's deep contextual bidirectional language model.

Model	SST-2		SST-5	
	All	Root	All	Root
Avg word vectors [9]	85.1	80.1	73.3	32.7
RNN [8]	86.1	82.4	79.0	43.2
RNTN [9]	87.6	85.4	80.7	45.7
Paragraph vectors [2]	–	87.8	–	48.7
LSTM [10]	–	84.9	–	46.4
BiLSTM [10]	–	87.5	–	49.1
CNN [11]	–	87.2	–	48.0
$\text{BERT}_{\text{BASE}}$	94.0	91.2	83.9	53.2
$\text{BERT}_{\text{LARGE}}$	**94.7**	**93.1**	**84.2**	**55.5**

Table 2: Accuracy (%) of several models on **sentiment classification (SC)** SST dataset. BERT has highest accuracy scores. From *Table B.II in Fine-Grained Sentiment Classification Using BERT*, by Munikar et al., 2019. https://arxiv.org/pdf/1910.03474.pdf. Copyright 2019 by Munikar et al.

Probing BERT: BERT Learns Dependency Syntax

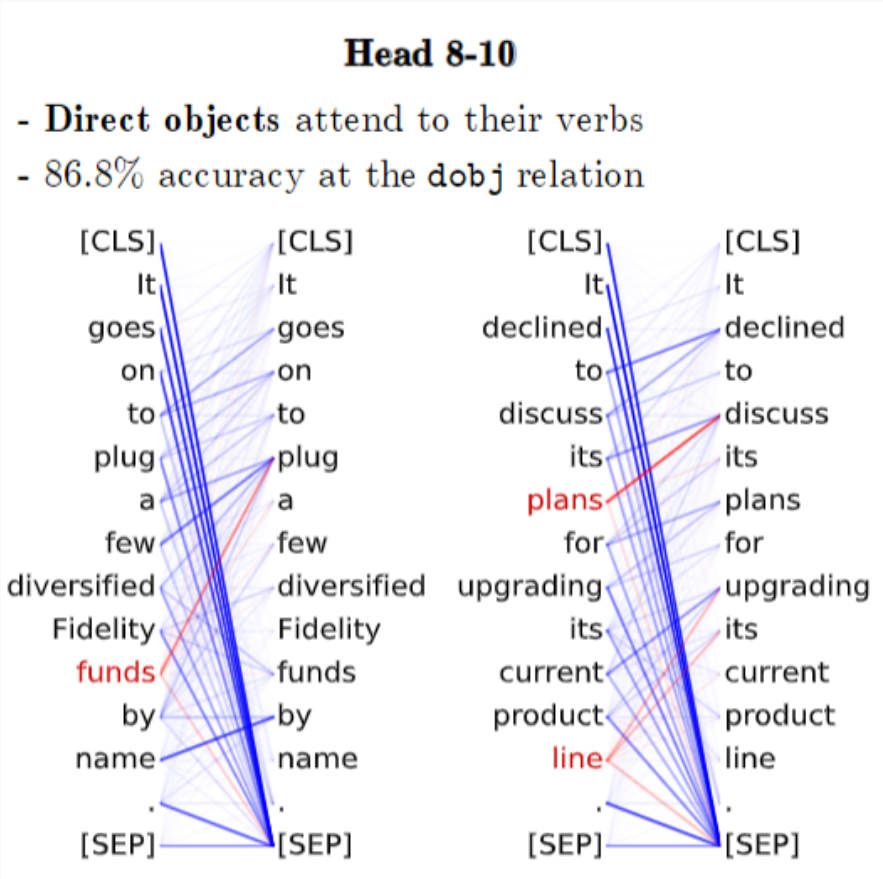

Figure 7: BERT attention heads capture syntax. In heads 8-10, direct objects are found to attend to their verbs. Line darkness indicates attention strength. Red indicates attention to/from red words, to highlight certain attentional behaviors. From *What Does BERT Look At? An Analysis of BERT's Attention*, by Clark et al., 2019. https://arxiv.org/abs/1906.04341.

Clark et al. (2019) found ...

- BERT's attention heads detect "direct objects of verbs, determiners of nouns, objects of prepositions, and objects of possessive pronouns with > 75% accuracy."
- Attention heads 8-10 in fig. 7 learn how direct objects attend to their verbs.
- BERT learns this using only *self-supervision*.

Probing BERT: BERT's Limitation in Segment Representation

- BERT does not use separator tokens ([SEP]) to gather segment-level information.
- BERT uses [SEP] as "no-op" or stub operations for attention heads when the head is not needed for a current task.
- How? Why? Authors investigated in two ways ...
  - If this were true, attention heads processing [SEP] should attend broadly over entire segment to make the segment vectors. But in fig. 7, heads 8-10 show direct objects attend to their verbs, and all other words attend to the [SEP] token.
  - Gradient measures: show much the attention to a token would change BERT's outputs. Attention to [SEP] increases in layers 5-10 (fig. 8), WHILE the gradients for attention to [SEP] decrease here (fig. 9) ⇒ attending to [SEP] does not significantly change BERT's outputs.

So, BERT's attention heads attend to [SEP] when they have no other job, not to gather segment-level information.

Transformer-XL by design improves this.

(continued) Probing BERT: BERT's Limitation in Segment Representation

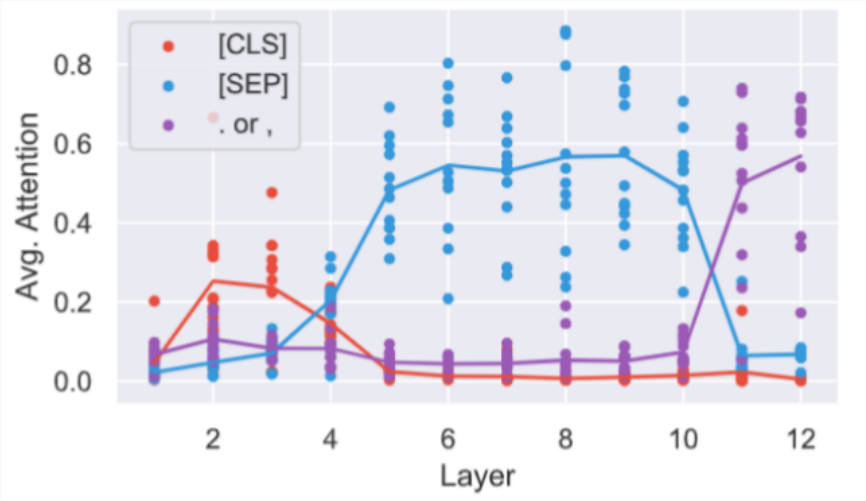

Figure 8: BERT's attention heads in layers 6-10 spend more than half the average attention to separator tokens; deep heads attend to punctuation, while middle heads attend to [SEP], and early heads attend to [CLS]. From *What Does BERT Look At? An Analysis of BERT's Attention*, by Clark et al., 2019. https://arxiv.org/abs/1906.04341. Copyright 2019 by Clark et al.

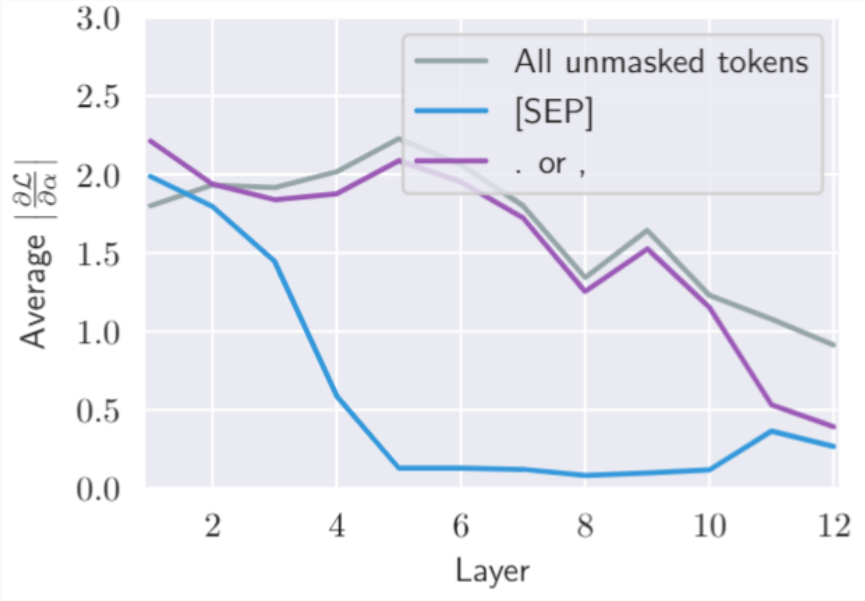

Figure 9: Gradient-based estimates for attention to separator and punctuation tokens. Authors "compute the magnitude of the gradient of the loss from the masked-language model (MLM) task with respect to each attention weight." From *What Does BERT Look At? An Analysis of BERT's Attention*, by Clark et al., 2019. https://arxiv.org/abs/1906.04341. Copyright 2019 by Clark et al.

BERT's Attempt at Polysemy

- Wiedemann et al. (2019) compared the contextual embeddings (CWE)s of ELMo and BERT on **word sense disambiguation (WSD)** task ⇒ BERT places polysemic words into distinct regions according to their senses, while ELMo cannot (stronger word sense separation in BERT embeddings).

- BERT did well when the text had ...
  - *vocabulary overlap* "along the bank of the river" (input text) and "along the bank of the river Greta" (nearest neighbor found by BERT).
  - *semantic overlap* "little earthy bank" (input) and "huge bank [of snow]" (nearest neighbor found by BERT)

- BERT struggled when text had *vocabulary and semantic overlap at the same time*
  - False prediction 1: BERT predicted "land bank" as in a *supply or stock* while the correct sense of "land bank" was *financial institution*
  - False prediction 2: correct word sense of "balloon" is a verb while BERT predicted a noun sense, so it did not even get the word class correct.
  - False prediction 3: correct sense of "watch" was *to look attentively* while BERT predicted its sense as *to follow with the eyes or the mind; observe.*

Transformer-XL

Problem with Transformer

Warning: Fixed-Length Context

Transformers have **fixed-length context** (limited context dependency), meaning the largest dependency distance between characters is limited by input length.

Also, natural semantic boundaries formed by sentences are *not* respected.

⇒ Transformers lose contextual information

⇒ Transformers forget words from a few sentences ago (Dai et al., 2019)

⇒ **Context-Fragmentation Problem**

Problem with Transformer: Fixed-Length Context Illustrated

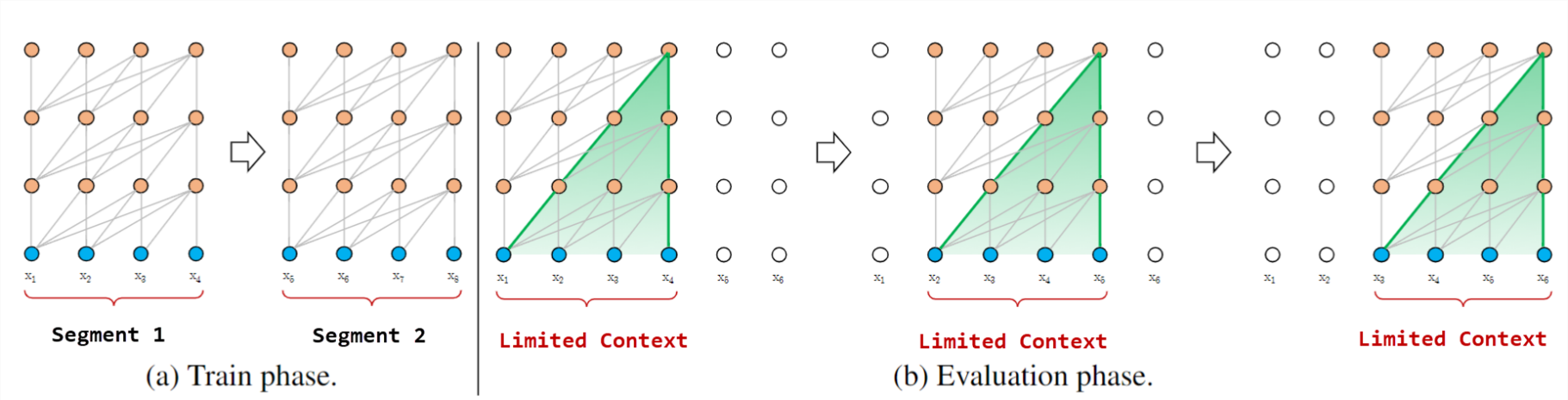

Figure 10: Vanilla Transformer with segment embedding length = 4. Training the model in fixed-length segments while disregarding natural sentence boundaries results in the *context fragmentation problem*: during each evaluation step, the Transformer consumes a segment embedding and makes a prediction at the last position. Then at the next step, the segment is shifted right by one position only, and the new segment must be processed from scratch, so there is no context dependency for first tokens of each segment and between segments. From *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*, by Dai et al., 2019.
https://arxiv.org/pdf/1901.02860.pdf. Copyright 2019 by Dai et al.

Motivation for Transformer-XL

- **Transformer-XL** (extra long) learns longer dependencies without "disrupting temporal coherence" (Dai et al., 2019).
- Doesn't chop sentences into arbitrary **fixed lengths**!
- Transformer-XL *respects natural language boundaries* like sentences and paragraphs, helping it gain richer context over sentences, paragraphs, and even longer texts like documents.
- Transformer-XL is composed of **segment-level recurrence mechanism** and **relative positional encoding** method (to fix **context fragmentation** and represent longer-spanning dependencies)

Transformer-XL: Segment-Level Recurrence Mechanism

When a segment is being processed, each hidden layer receives two inputs:

- the previous hidden layer outputs of the *current segment* (like vanilla transformer, visible as gray arrows in fig. 11)
- the previous hidden layer outputs of the *previous segment* (green arrows in fig. 11).

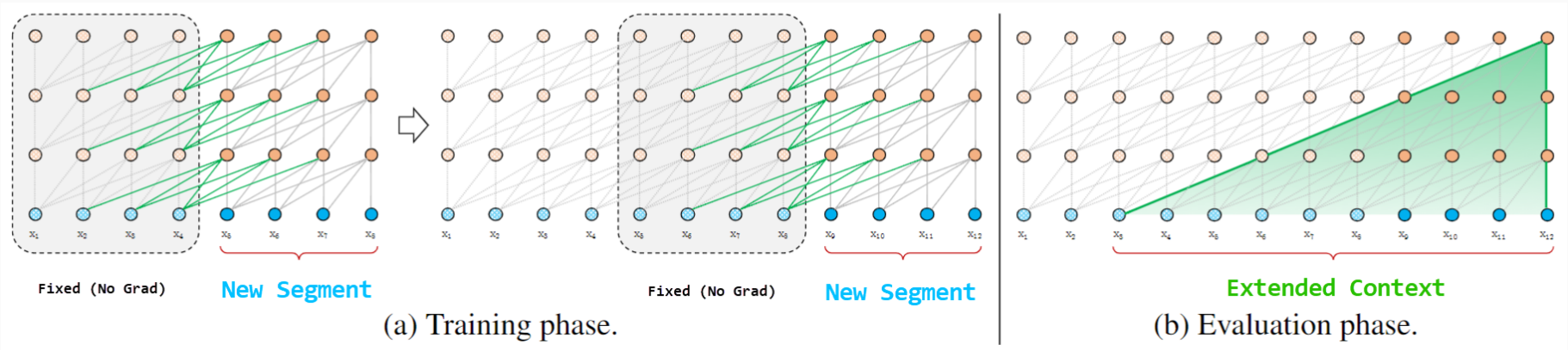

Figure 11: Segment level recurrence mechanism at work: the hidden state for previous segment is *fixed* and *stored* to later be reused as extended context while new segment is processed. Like in Transformer, gradient updates (training) still occurs within a segment, but extended context includes historical information. From *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*, by Dai et al., 2019. https://arxiv.org/pdf/1901.02860.pdf. Copyright 2019 by Dai et al.

Transformer-XL: Relative Positional Encoding

Problem when using Segment-Level Recurrence

How can positional word order be kept coherent when reusing hidden states?

Standard Transformer uses positional encodings (use absolute distance between tokens) .

⇒ Applying to Transformer-XL caused consecutive word embedding sequences to be associated with the same positional encoding.

⇒ Tokens from different segments had the same positional encodings

⇒ Transformer-XL couldn't distinguish the difference in positions of consecutive input tokens.

⇒ Defeated the purpose of positional encodings!

Solution: Relative Positional Encodings

- uses *relative* distance between tokens
- injects this into *each* attention score of each layer (rather than before first layer).

Transformer-XL: Experimental Results

Ablation study: Isolating effects of **segment-level recurrence mechanism** with different encoding schemes (Shaw (2018) uses relative, and Vaswani / Al-Rfou use absolute).

- **KEY:** best results obtained using *both* segment-level recurrence and relative positional encoding.

Remark	Recurrence	Encoding	Loss	PPL init	PPL best	Attn Len
Transformer-XL (128M)	✓	Ours	Full	**27.02**	**26.77**	**500**
-	✓	Shaw et al. (2018)	Full	27.94	27.94	256
-	✓	Ours	Half	28.69	28.33	460
-	✗	Ours	Full	29.59	29.02	260
-	✗	Ours	Half	30.10	30.10	120
-	✗	Shaw et al. (2018)	Full	29.75	29.75	120
-	✗	Shaw et al. (2018)	Half	30.50	30.50	120
-	✗	Vaswani et al. (2017)	Half	30.97	30.97	120
Transformer (128M)†	✗	Al-Rfou et al. (2018)	Half	31.16	31.16	120
Transformer-XL (151M)	✓	Ours	Full	23.43	**23.09**	**640**
					23.16	450
					23.35	300

Table 3: Ablation study for segment-level recurrence on the WikiText-103 data set. **PPL best** (model output) means perplexity score obtained using an optimal backpropagation training time length. **Attn Len** (model input) is the shortest possible attention length during evaluation to achieve the corresponding PPL best. From *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*, by Dai et al., 2019. https://arxiv.org/pdf/1901.02860.pdf. Copyright 2019 by Dai et al.

XLNet

XLNet: Problems with BERT

- An **autoregressive language model (AR)** autoregressively estimates the probability distribution of a text sequence by factorizing a likelihood using tokens *before* a timestep, or tokens *after* a timestep. A neural network is trained to model either the forward or backward distribution ⇒ cannot model bidirectional context.
- An **autoencoding language model (AE)** like BERT is like a masked language model. It masks tokens, predicts them and does not estimate densities like AR ⇒ can learn bidirectional contexts.
- **BERT's problems:**
  - **False Independence Assumption:** BERT factorizes its log likelihood probability assuming all masked tokens are rebuilt independently of each other (so BERT ignores long-term dependencies within texts)
  - **Data Corruption:** Masked tokens do not appear in real data during fine-tuning, so since BERT uses them in pre-training, a discrepancy arises between these two steps.

XLNet: Example of BERT's False Independence Assumption

Example: BERT predicting tokens independently

"I went to the `[MASK] [MASK]` and saw the `[MASK] [MASK] [MASK]`."

Two ways to fill this are:

"I went to *New York* and saw the *Empire State building*," or

"I went to *San Francisco* and saw the *Golden Gate bridge*."

But BERT might incorrectly predict something like: "I went to *San Francisco* and saw the *Empire State building*."

Independence assumption + predicting masked tokens simultaneously ⇒ BERT fails to learn their interlocking dependencies ⇒ weakens the "learning signal" (Kurita, 2019b).

XLNet: Motivation

To keep benefits of both autoencoding and autoregressive modeling while avoiding their issues...

1. XLNet adopts an AR model so that probability of a token can be factored via product rule, eliminating BERT's false independence assumption.
2. XLNet uses **permutation language model** to capture bidirectional context AND **two-stream attention** to adapt its Transformer to create target-aware predictions.

XLNet; Permutation Language Model

Created so that: a model can be trained to use **bidirectional context** while avoiding masking and its resulting problem of independent predictions.

Definition: Permutation Language Model

Like language models, a **permutation language model** predicts unidirectionally.

But instead of predicting in order, the model predicts tokens in a random order.

Is forced to accumulate bidirectional context by finding dependencies between *all* possible input combinations.

(NOTE: only permutes factorization order, not order of word inputs)

XLNet: Target-Aware Predictions

- **Problem:** Trying to merge permutation language model and Transformer made XLNet's target predictions blind to the permutation positions generated by the permutation language model.
- Fault is due to Transformer: while predicting a token at a position, the model masks the token's embedding AND also its *positional* encoding ⇒ Transformer remains blind about the position of the target it should be predicting ⇒ sentence cannot be accurately represented (since positions like the beginning of a sentence have different distributions from other positions in the sentence.)
- **Solution: Target-awareness:** authors adapted the predictive distribution to take the target position as an argument ⇒ now can create target-aware embeddings.
- **Another problem:** aforementioned problem with Transformer creates a contradiction: (1) to predict the content token x_{z_t}, only the position is needed, not the content x_{z_t}, and (2) to predict all other tokens x_{z_j}, the content token is needed.
- **Two-Stream Attention:** uses two sets of hidden states (content stream and query stream) to create an overall hidden state.
  - **Content-Stream Attention:** encodes *context* like an ordinary Transformer, along with the *content* (prediction) token x_{z_t}
  - **Query-Stream Attention:** encodes *context* with target token's *position* but NOT content (prediction) token x_{z_t} (to evade the contradiction).

XLNet: Relative Segment Encodings

- XLNet adopts Transformer-XL's idea of relative encodings.
- BERT's segment embeddings distinguish words belonging to different segments.
- XLNet's segment embeddings encode if two words are *within the same segment* rather than *which specific segments the words are from* $\Rightarrow$ can apply XLNet to tasks that intake arbitrarily many sequences.

XLNet: Conceptual Difference with BERT

Illustrating the conceptual difference between XLNet and BERT from a model training standpoint:

Example: Conceptual Difference between XLNet and BERT

Take the list of words $\left[\texttt{New}, \texttt{York}, \texttt{is}, \texttt{a}, \texttt{city}\right]$.

Prediction tokens: $\left[\texttt{New}, \texttt{York}\right]$

XLNet and BERT must maximize the log-likelihood: $\log P(\texttt{New York} \mid \texttt{is a city})$.

Assumption: XLNet uses the factorization order $\left[\texttt{is}, \texttt{a}, \texttt{city}, \texttt{New}, \texttt{York}\right]$

Then each of their loss functions are:

$$\begin{aligned} \mathcal{J}_{\text{BERT}} &= \log P\Big(\texttt{New} \mid \texttt{is a city}\Big) + \log P\Big(\texttt{York} \mid \texttt{is a city}\Big) \\ \mathcal{J}_{\text{XLNet}} &= \log P\Big(\texttt{New} \mid \texttt{is a city}\Big) + \log P\Big(\texttt{York} \mid \texttt{New}, \texttt{is a city}\Big) \end{aligned} \tag{1}$$

Result: XLNet learns a stronger dependency than BERT between the pairs New and York (Dai et al., 2019).

ERNIE 1.0: Enhanced Representations Through Knowledge Integration

ERNIE: Motivations

- Previous models (Word2Vec, GloVe, BERT) make embeddings via context and co-occurrence ⇒ fail to use prior knowledge (in sentence ordering + nearness) to capture relationships between entities.

Example: ERNIE's Entity Capturing Skills

Consider the following training sentence:

"Harry Potter is a series of fantasy novels written by J. K. Rowling."

Using co-occurring words "J.", "K.", and "Rowling", BERT is limited to predicting the token "K." but utterly fails at recognizing the whole entity *J. K. Rowling*.

A model could use simple co-occurrence counts to predict the missing entity *Harry Potter* even without using long contexts...

... but it would not be using the *relationship between the novel name and its writer*.

- **ERNIE to the rescue!** ERNIE can extrapolate the relationship between the *Harry Potter* entity and *J. K. Rowling* entity using implicit knowledge of words and entities ⇒ predicts Harry Potter is a series written by J. K. Rowling (Sun et al., 2019a).

ERNIE: Phrase and Entity-Level Masking

- ERNIE uses a Transformer Encoder coupled with **entity-level masking** and **phrase-level masking** (to encode prior knowledge in conceptual units like phrases and entities) ⇒ learns longer semantic dependencies, has better generalization, adaptability.
- **Phrase-level masking:** A phrase is a "small group of words or characters acting as a conceptual unit" (Sun et al., 2019a). ERNIE chunks sentences to find phrase boundaries, then masks and predicts them.
- **Entity-level masking:** name entities contain "persons, locations, organizations, products." Often include conceptual information. ERNIE parses the entities from a sentence, masks them, then predicts all slots within entities, as shown in fig. 12.

Sentence	Harry	Potter	is	a	series	of	fantasy	novels	written	by	British	author	J.	K.	Rowling
Basic-level Masking	[mask]	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	J.	[mask]	Rowling
Entity-level Masking	Harry	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]
Phrase-level Masking	Harry	Potter	is	[mask]	[mask]	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]

Figure 12: ERNIE uses basic masking to get word representations, followed by phrase-level and entity-level masking. From *ERNIE: Enhanced Representation Through Knowledge Integration*, by Sun et al., 2019. https://arxiv.org/pdf/1904.09223.pdf. Copyright 2019 by Sun et al.

ERNIE: Using Knowledge Learning To Fill-In-The-Blanks on Named Entities

Case	Text	ERNIE	BERT	Answer
1	"In September 2006, ___ married Cecilia Cheung. They had two sons, the older one is Zhenxuan Xie and the younger one is Zhennan Xie."	Tingfeng Xie	Zhenxuan Xie	Tingfeng Xie
4	"Australia is a highly developed capitalist country with ___ as its capital. As the most developed country in the Southern Hemisphere, the 12th largest economy in the world and the fourth largest exporter of agricultural products in the world, it is also the world's largest exporter of various minerals."	Melbourne	(Not a city name)	Canberra
6	"Relativity is a theory about space-time and gravity, which was founded by ___."	Einstein	(Not a word in Chinese)	Einstein

Table 4: Comparing ERNIE to BERT on Cloze Chinese Task. From *Figure 4 in ERNIE: Enhanced Representation Through Knowledge Integration*, by Sun et al., 2019. Copyright 2019 by Sun et al.

- Case 1: ERNIE predicts the correct father name entity based on prior knowledge in the article while BERT simply memorizes one of the sons' name, completely ignoring any relationship between mother and son.
- Case 4, 6: BERT fills the slots with characters related to the sentences but not with the semantic concept, while ERNIE predicts correct entities.
- Case 4: ERNIE predicts the wrong city name, though it still understands the semantic type.

ERNIE's contextual knowledge understanding is far superior to BERT's!