CS447: Natural Language Processing

http://courses.grainger.illinois.edu/cs447

Lecture 27: Intro to Large Language Models

Julia Hockenmaier

juliahmr@illinois.edu

ILLINOIS

Today's class

Recap: Using RNNs for various NLP tasks

From static to contextual embeddings: ELMO

Recap: Transformers

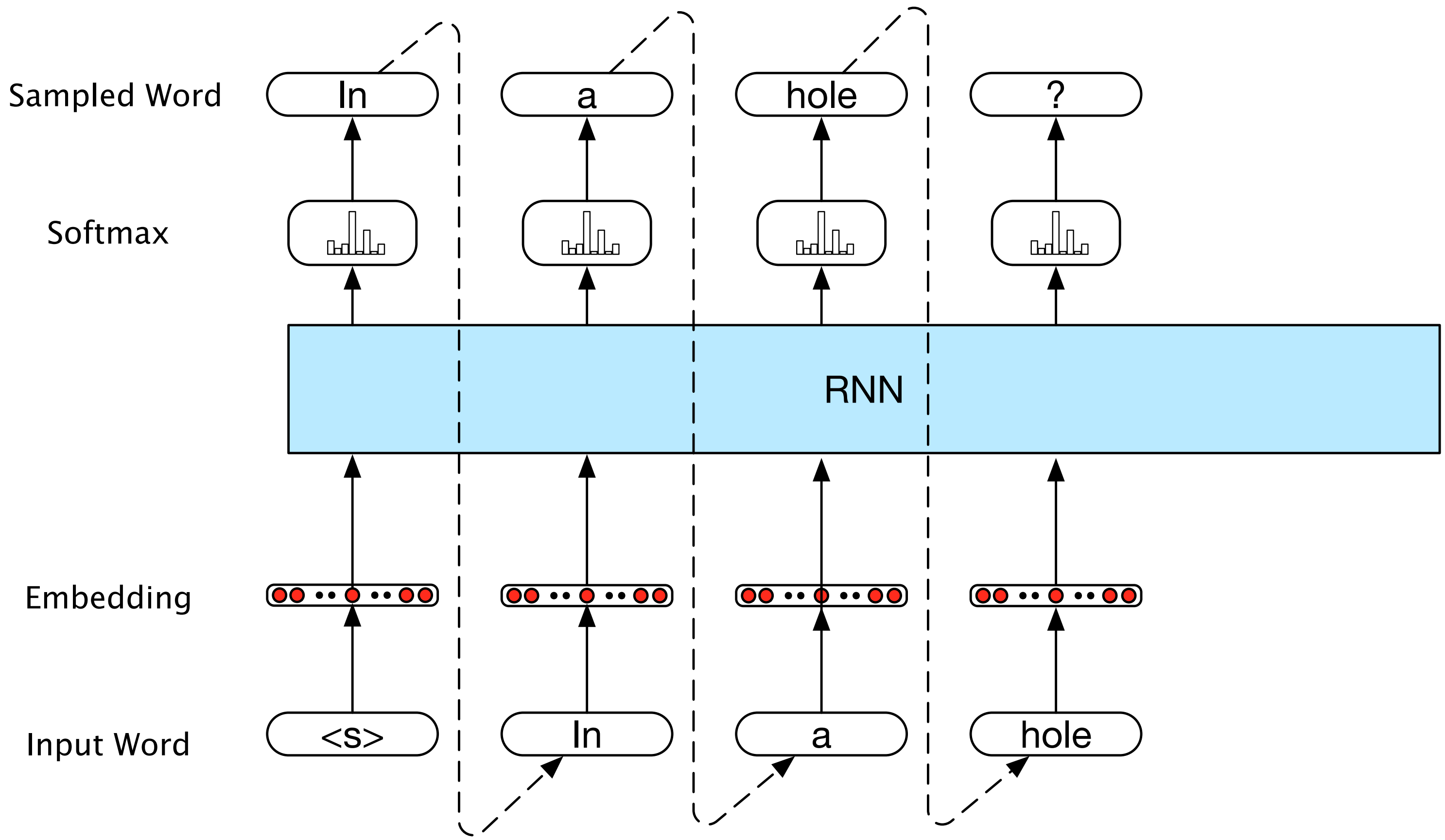
Subword tokenizations

Early Large Language Models (GPT, BERT)

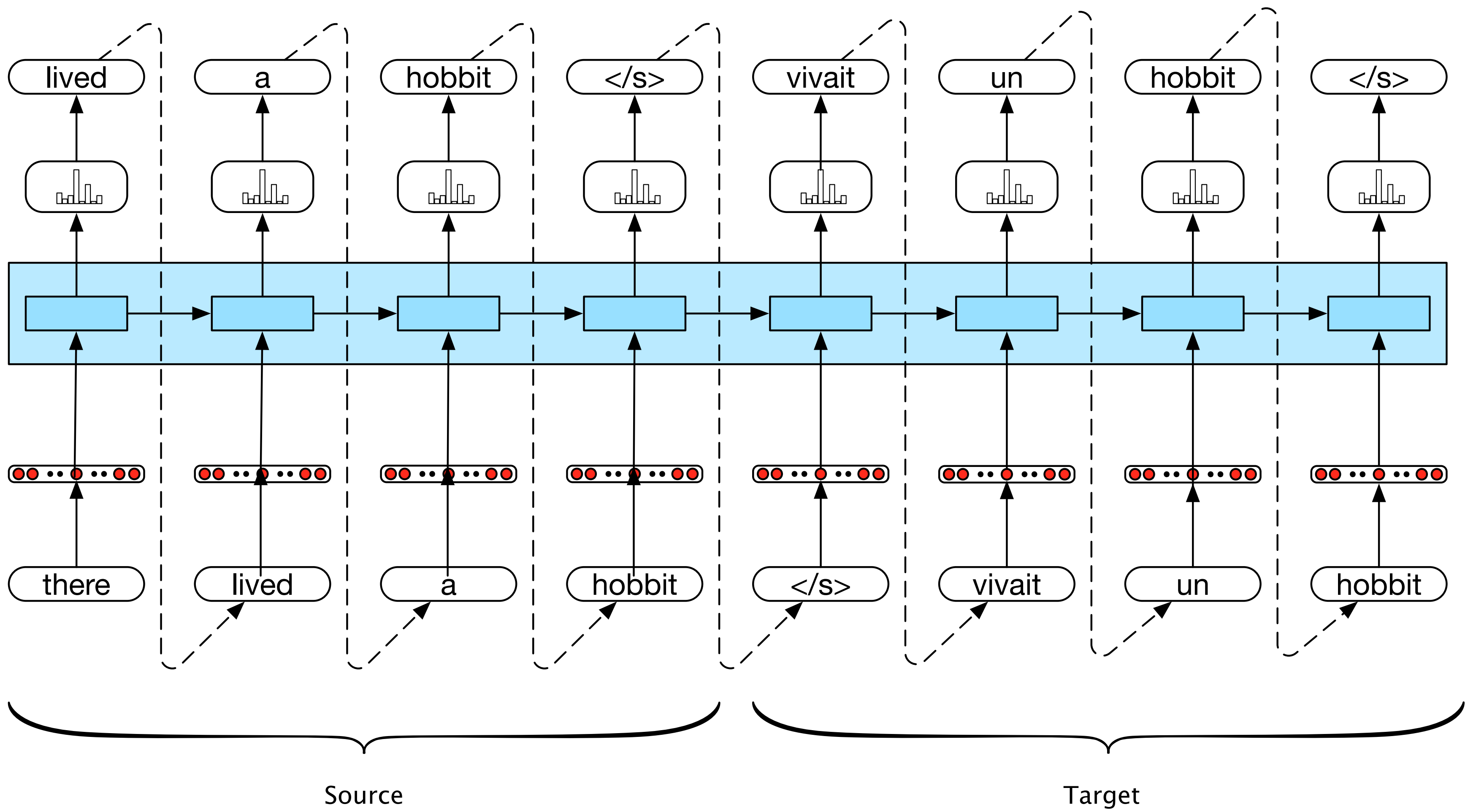
Recap: Using RNNs for different NLP tasks

RNNs for language generation

AKA “autoregressive generation”



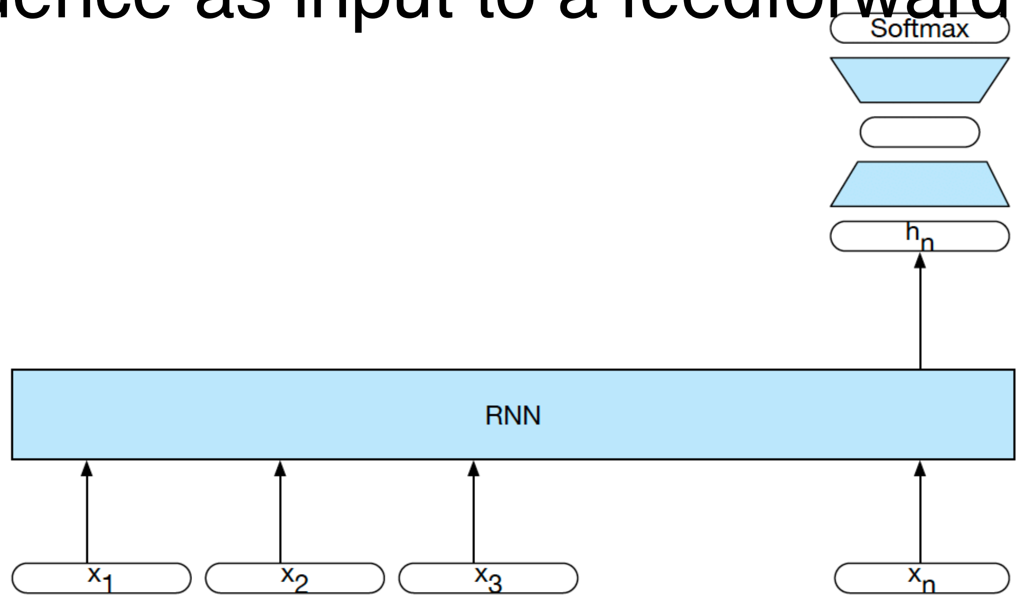
An RNN for Machine Translation



RNNs for sequence classification

If we just want to assign **one label** to the entire sequence, we don't need to produce output at each time step, so we can use a simpler architecture.

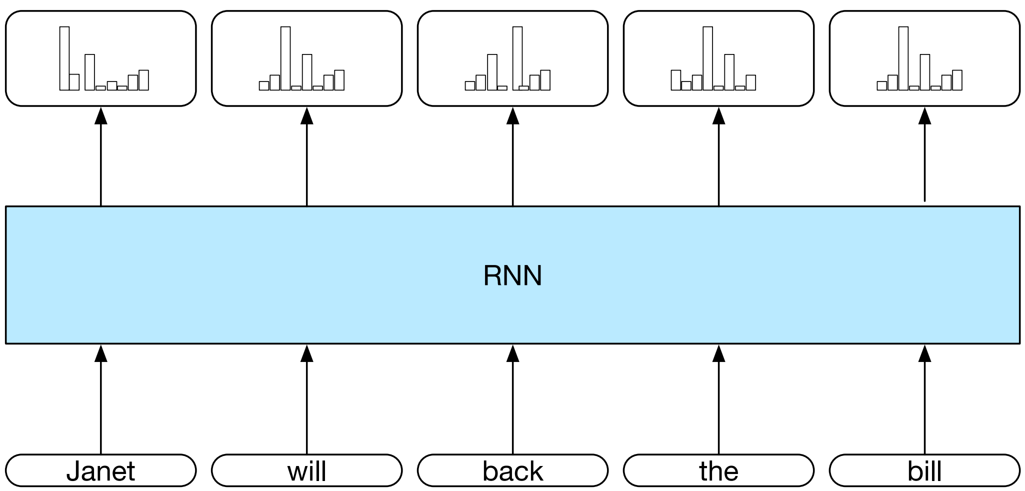
We can use the hidden state of the last word in the sequence as input to a feedforward net:



Basic RNNs for sequence labeling

Sequence labeling (e.g. POS tagging):
Assign **one label to each element** in the sequence.

RNN Architecture:
Each time step has a distribution over output classes



Extension: add a CRF layer to capture dependencies among labels of adjacent tokens.

ELMo

Embeddings from Language Models

Replace static embeddings (lexicon lookup) with **context-dependent embeddings** (produced by a **neural language model**)

=> **Each token**'s representation is a **function of the entire input sentence**, computed by a deep (multi-layer) bidirectional language model

=> Return for each token a **(task-dependent) linear combination of its representation across layers**.

=> Different layers capture different information

Peters et al., NAACL 2018

ELMo

Pre-training:

— Train a **multi-layer bidirectional language model** with character convolutions on **raw text**

— **Each layer** of this language model network computes **a vector** representation **for each token**.

— **Freeze the language model** parameters.

Fine-tuning (for each task)

Train **task-dependent softmax** weights to **combine the layer-wise representations** into **a single vector for each token** *jointly* with **a task-specific model that uses those vectors**

ELMo’s input token representations

The input token representations are purely **character-based:** a character CNN, followed by linear projection to reduce dimensionality

“2048 character n-gram convolutional filters with two highway layers, followed by a linear projection to 512 dimensions”

Advantage over using fixed embeddings:
no UNK tokens, any word can be represented

ELMo's bidirectional language models

Forward LM: a deep LSTM that goes over the sequence from start to end to predict token t_k based on the prefix $t_1 \ldots t_{k-1}$:

$$p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s)$$

Parameters: token embeddings Θ_x LSTM $\overrightarrow{\Theta}_{LSTM}$, softmax Θ_s

Backward LM: a deep LSTM that goes over the sequence from end to start to predict token t_k based on the suffix $t_{k+1} \ldots t_N$:

$$p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)$$

Train these LMs jointly, with the same parameters for the token representations and the softmax layer (but not for the LSTMs)

$$\sum_{k=1}^{N} \left(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right)$$

ELMo's output token representations

Given **an input token representation x**$_k$,
each layer *j* of the LSTM language models computes
a vector representation h$_{k,j}$ **for every token** *k*.

With *L* layers, ELMo represents each token as *L* vectors $\mathbf{h}_{k,l}^{LM}$

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

where $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$ and $\mathbf{h}_{k,0}^{LM} = \mathbf{x}_k$

ELMo learns **softmax weights** s_j^{task} and a **task-specific scalar** γ^{task}
to collapse these *L* vectors into **a single task-specific token vector:**

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

Results

ELMo gave improvements on a variety of tasks:

— question answering (SQuAD)
— entailment/natural language inference (SNLI)
— semantic role labeling (SRL)
— coreference resolution (Coref)
— named entity recognition (NER)
— sentiment analysis (SST-5)

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

ELMo:

ELMo showed that **contextual embeddings** are very useful: it outperformed other models on many tasks

- ELMo embeddings could also be concatenated with other token-specific features, depending on the task

ELMo requires training a task-specific softmax and scalar to predict how best to combine each layer

- Not all layers were equally useful for each task

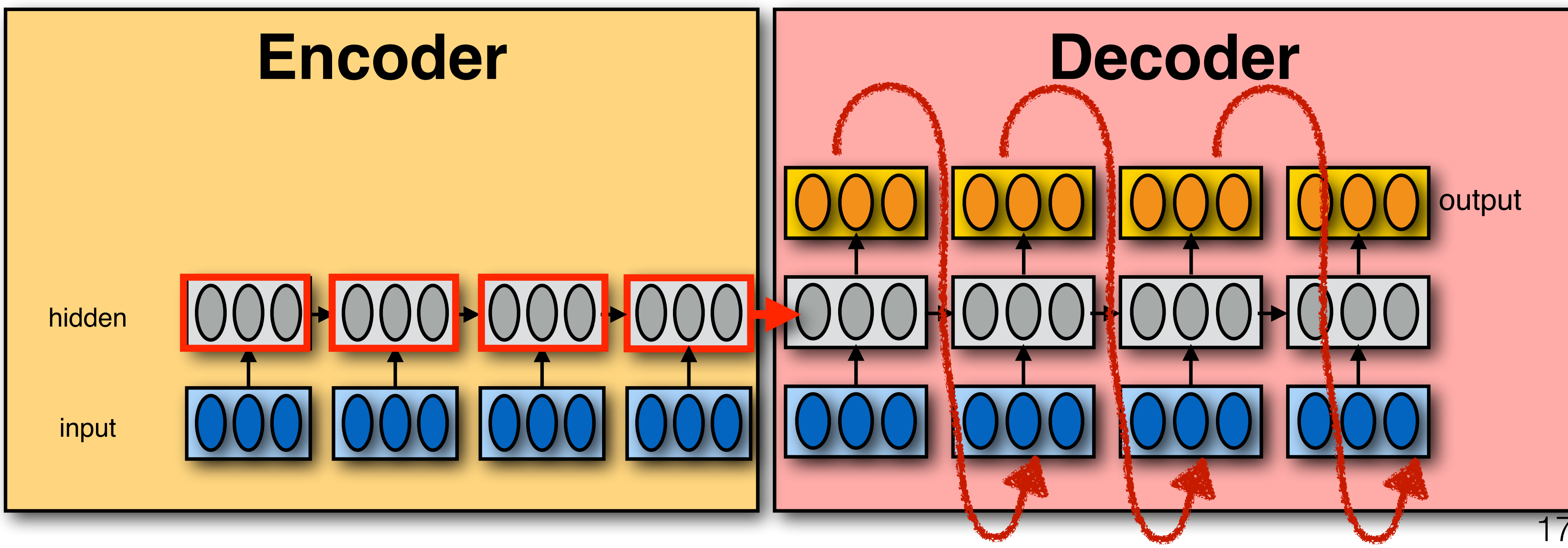
Recap: Seq2seq, Transformers

Encoder-Decoder (seq2seq) model

The **decoder** is a language model that generates an output sequence **conditioned on the input** sequence.

— **Vanilla RNN**: condition on the **last** hidden state

— **Attention**: condition on **all** hidden states



17

Transformers use Self-Attention

Attention so far (in seq2seq architectures):

In the ***decoder*** (which has access to the complete input sequence), compute attention weights over ***encoder*** positions that depend on each ***decoder*** position

Self-attention:

If the ***encoder*** has access to the complete input sequence, we can also compute attention weights over ***encoder*** positions that depend on each ***encoder*** position

self-attention:

encoder

For each ~~*decoder*~~ position t…,

…Compute an attention weight for each *encoder* position s

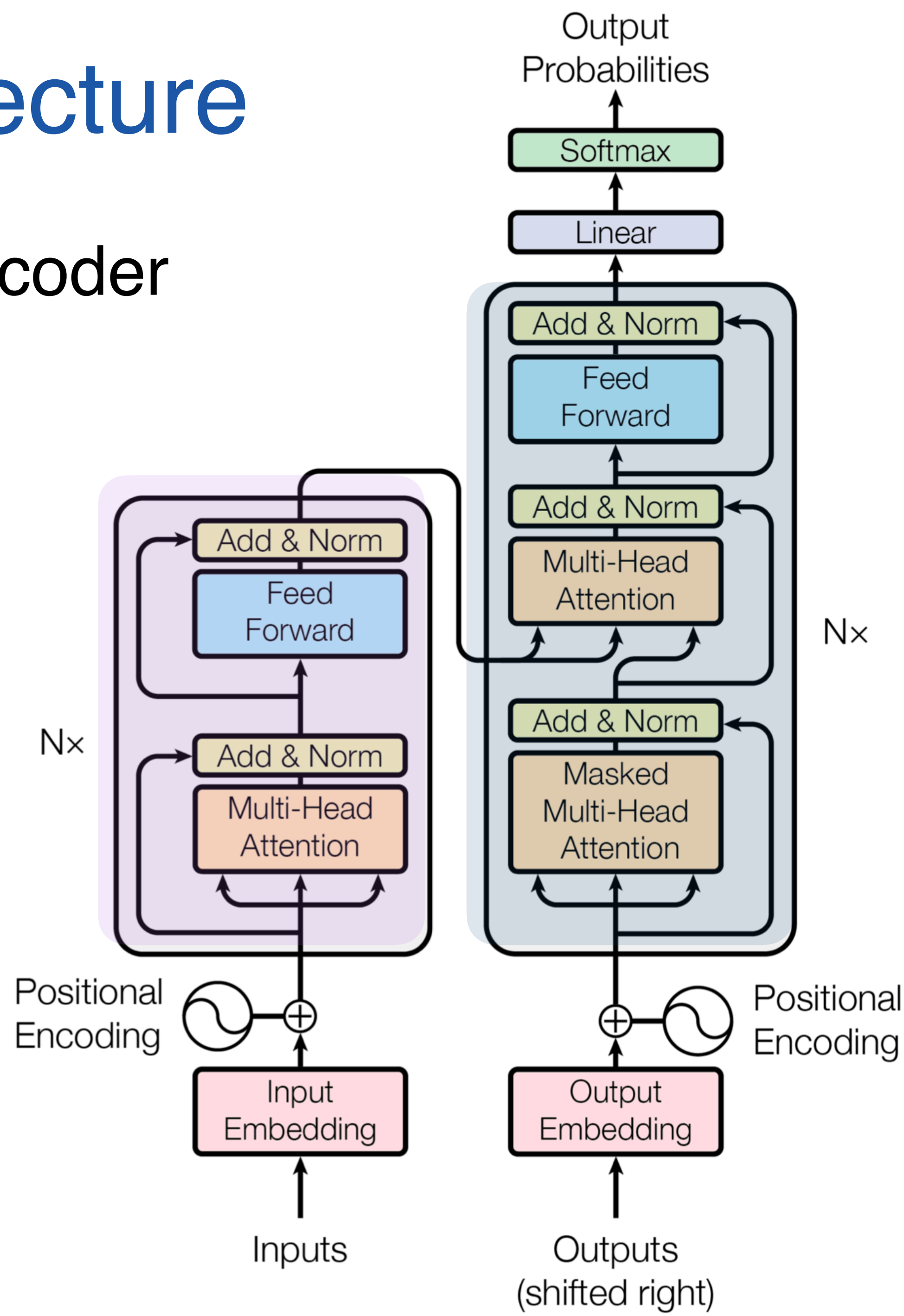
…Renormalize these weights (that depend on t) w/ softmax to get a new weighted avg. of the input sequence vectors

Transformer Architecture

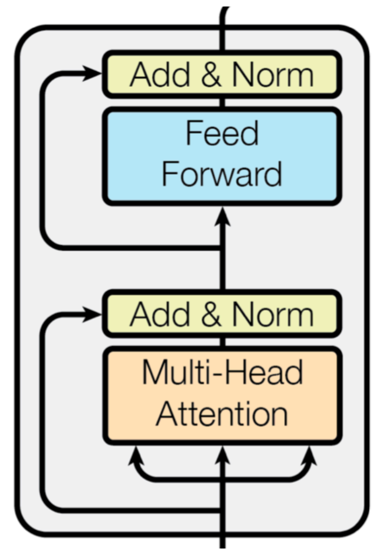
Non-Recurrent Encoder-Decoder architecture

— No hidden states
— Context information captured via attention and positional encodings
— Consists of stacks of layers with various sublayers

Vaswani et al, NIPS 2017



Encoder Vaswani et al, NIPS 2017



A stack of **N=6 identical layers**

All layers and sublayers are 512-dimensional

Each layer consists of **two sublayers**

— one **multi-head self attention** layer

— one **position-wise feed forward** layer

Each sublayer is followed by an **"Add & Norm"** layer:

… a **residual connection** $\mathbf{x} + \mathrm{Sublayer}(\mathbf{x})$

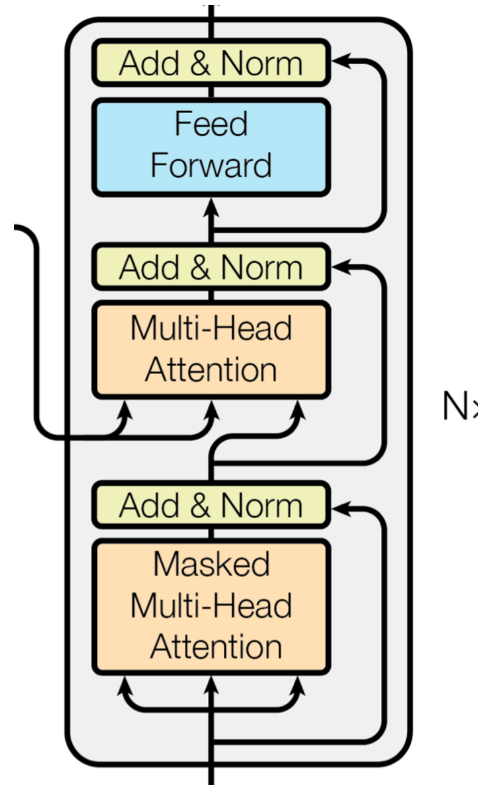
(the input $\mathbf{x}$ is added to the output of the sublayer)

… followed by a **normalization step**

(using the mean and standard deviation of its activations)

$$\mathrm{LayerNorm}\big(\mathbf{x} + \mathrm{Sublayer}(\mathbf{x})\big)$$

Decoder Vaswani et al, NIPS 2017



A stack of N=6 identical layers

All layers and sublayers are 512-dimensional

Each layer consists of **three** sublayers

— one **masked multi-head self attention layer** over **decoder** output (masked, i.e. ignoring future tokens)

— one **multi-headed attention layer** over **encoder** output

— one **position-wise feed forward layer**

Each sublayer has a residual connection and is normalized: $\mathrm{LayerNorm}\big(\mathbf{x} + \mathrm{Sublayer}(\mathbf{x})\big)$

Subword Tokenization

BPE Tokenization (Sennrich et al, ACL 2016)

BytePair Encoding (Gage 1994): a compression algorithm that iteratively replaces the most common pair of adjacent bytes with a single, unused byte

BPE tokenization: introduce new tokens by merging the most common adjacent pairs of tokens

- Start with all characters, plus a special end-of-word character
- Introduce new token by merging the most common pair of adjacent tokens.
  (Assumption: each individual token will still occur in a different context, so we will also keep both tokens in the vocabulary)
- **Machine translation:** train one tokenizer across both languages (better generalization for related languages)

Wordpiece tokenization (Wu et al, 2016)

Part of Google's LSTM-based Neural Machine Translation system (https://arxiv.org/pdf/1609.08144.pdf)

Segment words into **subtokens** (with special word boundary symbols to recover original tokenization)

Input: Jet makers feud over seat width with big orders at stake
Output: _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

Training of Wordpiece:

Specify desired number of tokens, D

Add word boundary token (at beginning of words)

Optimization task: greedily merge adjacent characters to improve log-likelihood of data until the vocabulary has size D.

Subword Regularization (Kudo, ACL 2018)

Observation: Subword tokenization can be ambiguous

Can this be harnessed?

Approach: Train a (translation) model with (multiple) subword segmentations that are sampled from a character-based **unigram language model**

Training the unigram model:

Start with an overly large seed vocabulary V (all possible single-character tokens and many multi-character tokens)

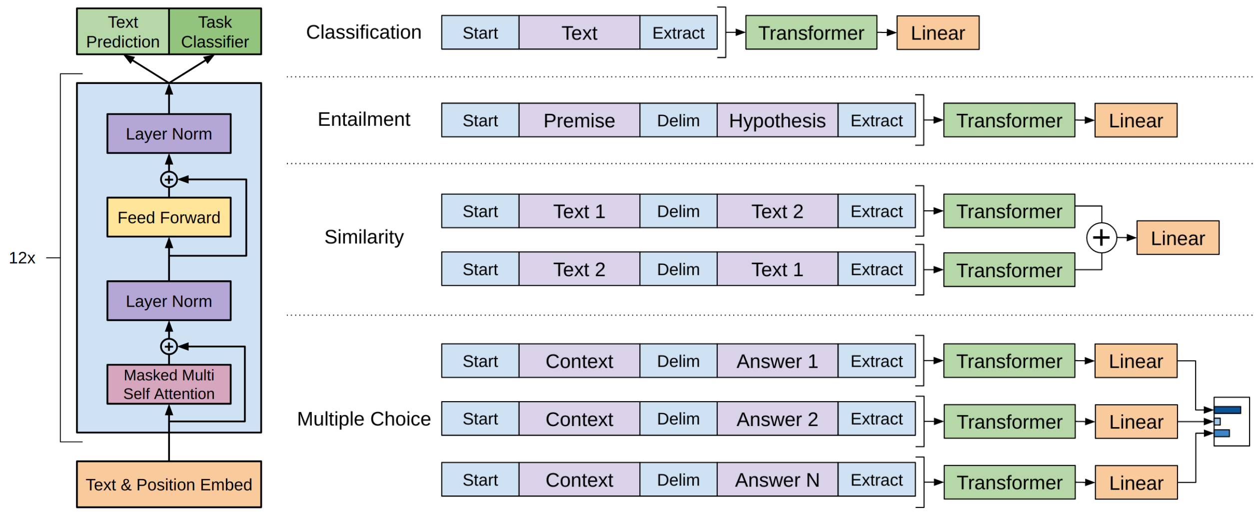
Randomly sample a segmentation from the unigram model

Decide which multi-character words to remove from V based on how the likelihood decreases by removing them

Stop when the vocabulary is small enough.

GPT

Generative Pre-Training (Radford et al, 2018)



Auto-regressive 12-layer transformer **decoder**

Each token only conditioned on preceding context

BPE tokenization (|V| = 40K), 768 hidden size, 12 attention heads

Pre-trained on raw text as a language model

(Maximize the probability of predicting the next word)

Fine-tuned on labeled data (and language modeling)

Include new **start**, **delimiter** and **end** tokens,
plus **linear** layer added to last layer of **end token** output.

BERT

BERT (Devlin et al, NAACL 2019)

Fully <u>bidirectional</u> transformer encoder

BERT$_{base}$: 12 layers, hidden size=768, 12 att'n heads (110M parameters)

BERT$_{large}$: 24 layers, hidden size=1024, 16 attention heads (340M parameters)

Input: sum of token, positional, segment embeddings

Segment embeddings (A and B): is this token part of sentence A (before SEP) or sentence B (after SEP)?

[CLS] and **[SEP]** tokens: added during pre-training

Pre-training tasks:

– Masked language modeling

– Next sentence prediction

BERT Input

	[CLS]	Sentence A				[SEP]	Sentence B				[SEP]
Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

Pre-training tasks

BERT is jointly pre-trained on two tasks:

Next-sentence prediction: [based on CLS token]

Does sentence B follow sentence A in a real document?

Masked language modeling:

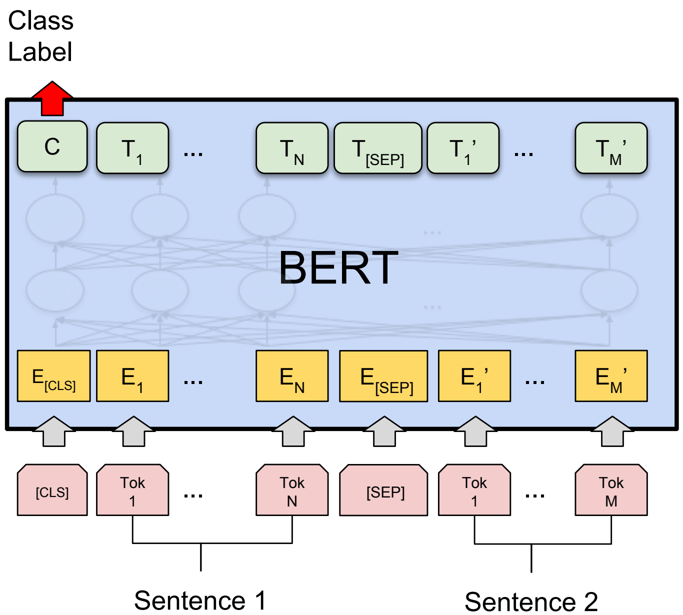
15% of tokens are randomly chosen as **masking tokens**

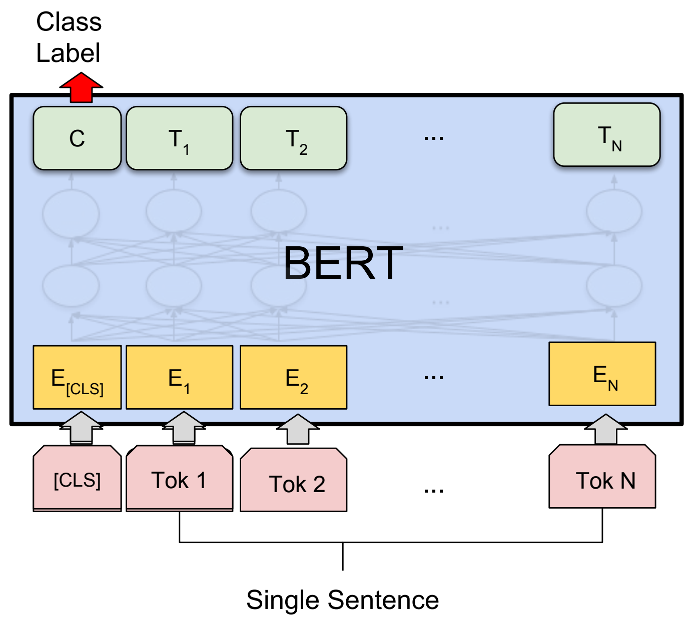
10% of the time, a masking token remains unchanged

10% of the time, a masking token is replaced by a random token

80% of the time, a masking token is replaced by **[MASK]**, and the **output layer has to predict the original token**

Using BERT for classification



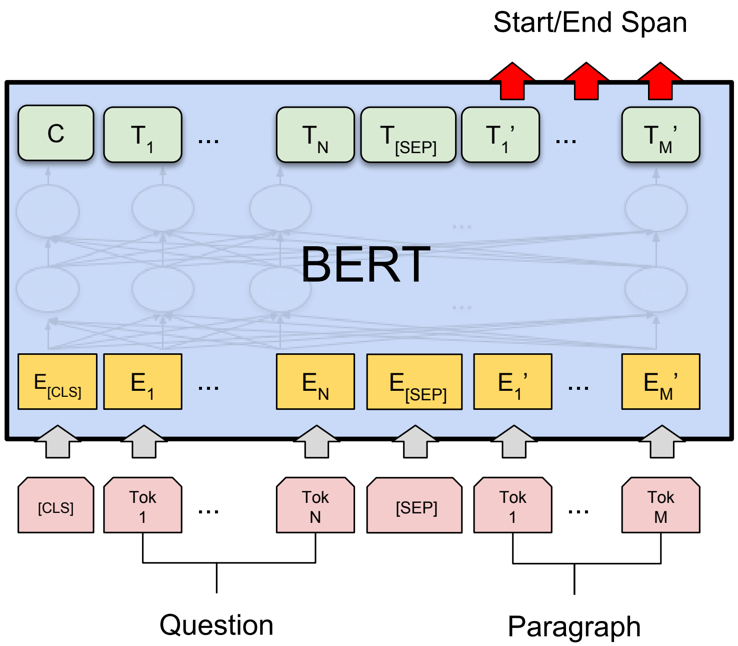


Sentence Pair Classification

Single Sentence Classification

Add a **softmax classifier** on final layer of **[CLS]** token

Using BERT for Question-Answering



Input: [CLS] question [SEP] answer passage [SEP]

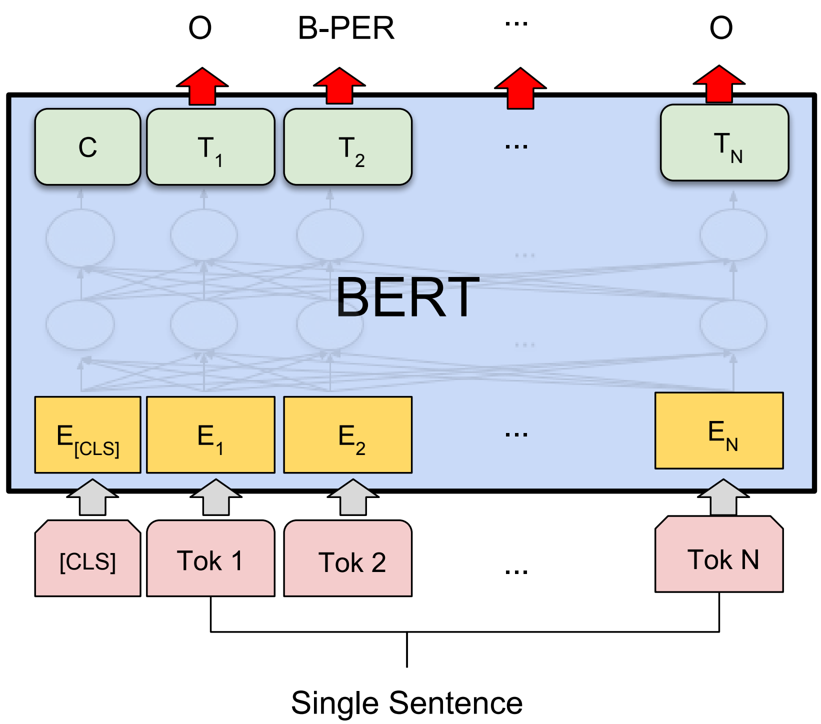
Learn to predict a **START** and an **END token** on answer tokens

Represent START and END as H-dimensional vectors *S*, *E*

Find the most likely start and end tokens in the answer by computing a softmax over the dot product of all token embeddings T_i and S (or E)

$$P(T_i \textit{ is start}) = \frac{\exp(T_i \cdot S)}{\sum_j \exp(T_j \cdot S)}$$

Using BERT for Sequence Labeling



Add a **softmax classifier** to the tokens in the sequence

Fine-tuning BERT

To use BERT on any task, it needs to be fine-tuned:

— Add any new parts to the model
(e.g. classifier layers)

This will add **new parameters** (initialized randomly)

— Retrain the entire model (update all parameters)

More compact BERT models (Turc et al., 2019)

Pre-training and fine-tuning works well on much smaller BERT variants

https://arxiv.org/abs/1908.08962

Additional improvements through knowledge distillation:

- **Pre-train** a compact model ('student') in the standard way
- Train/Fine-tune a large model ('teacher') on the target task
- **Knowledge distillation** step:
  Train the student on noisy task predictions made by teacher
- Fine-tune student on actual task data

Students can have more layers (but smaller embeddings) than models trained in the standard way

BERT Variants

RoBERTA (Liu et al. 2019)

Investigates **better pre-training** for BERT

Found that BERT was undertrained.

Optimizes hyperparameter choice.

Evaluates next-sentence prediction task

RoBERTA outperforms BERT on several tasks.

Pre-training improvements:

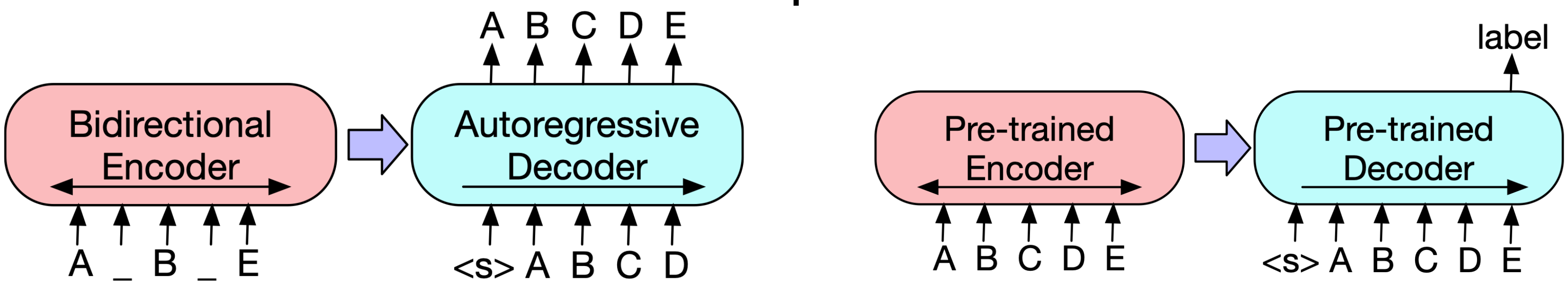
- **Dynamic masking:** randomly change which tokens in a sentence get masked (BERT: same tokens in each epoch)
- **Much larger batch sizes** (2K sentences instead of 256)
- **Use byte-level BPE, not character level BPE**

BART (Lewis et al., ACL 2020)

Combines **bidirectional encoder** (like BERT) with **auto-regressive** (unidirectional) **decoder** (like GPT)

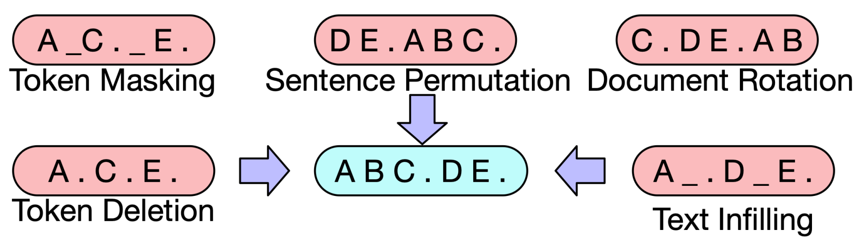
Used for **classification**, **generation**, **translation**

Uses final token of decoder sequence for classification tasks.



Pre-training: corrupts (encoder) input with **masking**, **deletion, rotation, permutation, infilling.**

Decoder needs to recover original input



SentenceBERT (Reimers & Gurevych, EMNLP 2019)

For tasks that require scoring of **sentence pairs**

(e.g. semantic textual similarity, or entailment recognition)

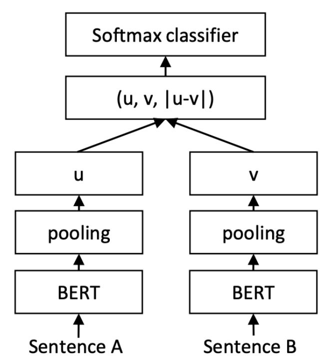
Motivation: BERT treats sequence pairs as one (long) sequence, but cross-attention across O(2n) words is very slow.

SentenceBERT Solution: **Siamese network**

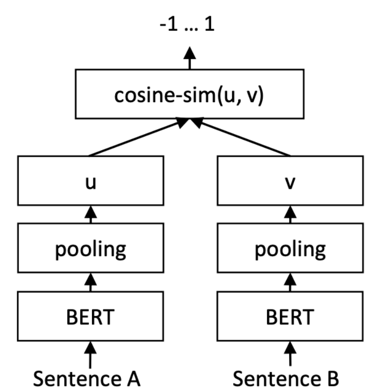
Run BERT over each sentence independently

Compute **one vector** (**u** and **v**) **for each sentence** by (mean or max) pooling over word embeddings or by using CLS token

Classification tasks: concatenate **u**, **v**, and **u–v**, use as input to softmax



Similarity tasks: use the cosine similarity of **u** and **v** as similarity score



Training: start with BERT, fine-tune Siamese model on task-specific data