

# Pre-Training Strategies

ELMo and BERT

Tanmoy Chakraborty

Associate Professor, IIT Delhi

<https://tanmoychak.com/>



**Introduction to Large Language Models**



*“You shall know a word by the company it keeps”*

This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:

*“... the complete meaning of a word is always contextual,  
and no study of meaning apart from a complete context  
can be taken seriously.” (J. R. Firth 1935)*

***I record the record***

the two instances of **record** mean different things.

# Background - Contextual Representations

- Word embeddings serve as the foundation for deep learning models in natural language processing.
- **Problem :** Word embeddings (word2vec, GloVe) are used without considering the context in which the words appear.

A bat flew out of the cave.

He hit the ball with a bat.

[ 0.286 , 0.792 , -0.177 , ....]

- **Solution :** Train contextual representations on text corpus

A bat flew out of the cave.

He hit the ball with a bat.

[ -0.107 , 0.109 , -0.542 , ....]

[ 0.349 , 0.271 , 0.130 , ....]

The representation of the word should depend on the context in which it appears.

# Deep contextualized word representations

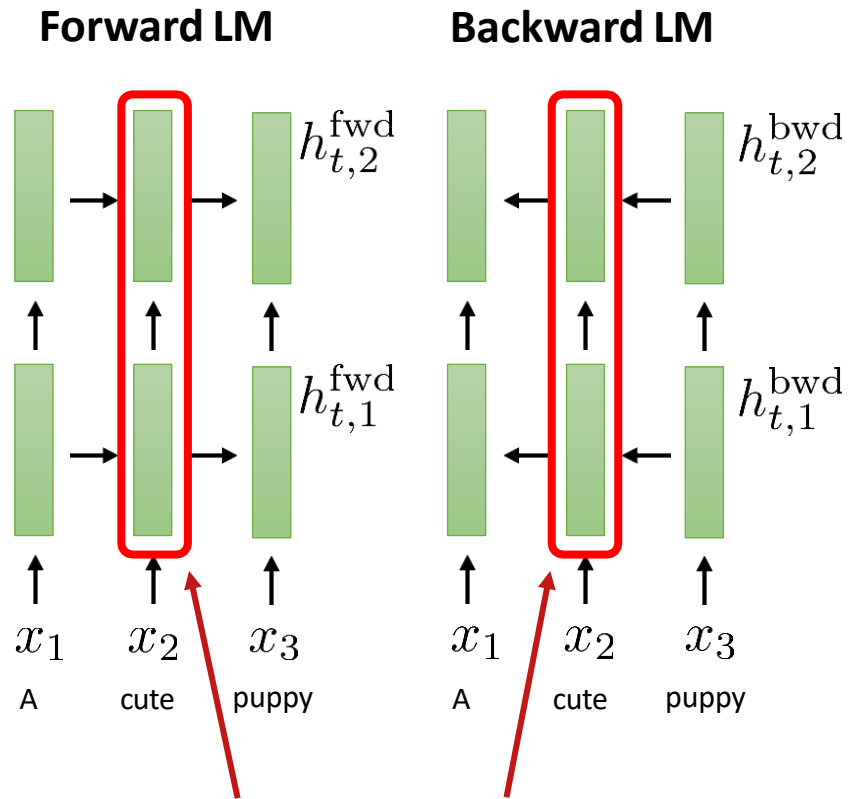
**Matthew E. Peters<sup>†</sup>, Mark Neumann<sup>†</sup>, Mohit Iyyer<sup>†</sup>, Matt Gardner<sup>†</sup>,**  
`{matthewp, markn, mohiti, mattg}@allenai.org`

**Christopher Clark\*, Kenton Lee\*, Luke Zettlemoyer<sup>†\*</sup>**  
`{csquared, kentonl, lsz}@cs.washington.edu`

<sup>†</sup>Allen Institute for Artificial Intelligence

\*Paul G. Allen School of Computer Science & Engineering, University of Washington

# ELMo (Embedding from Language Models)

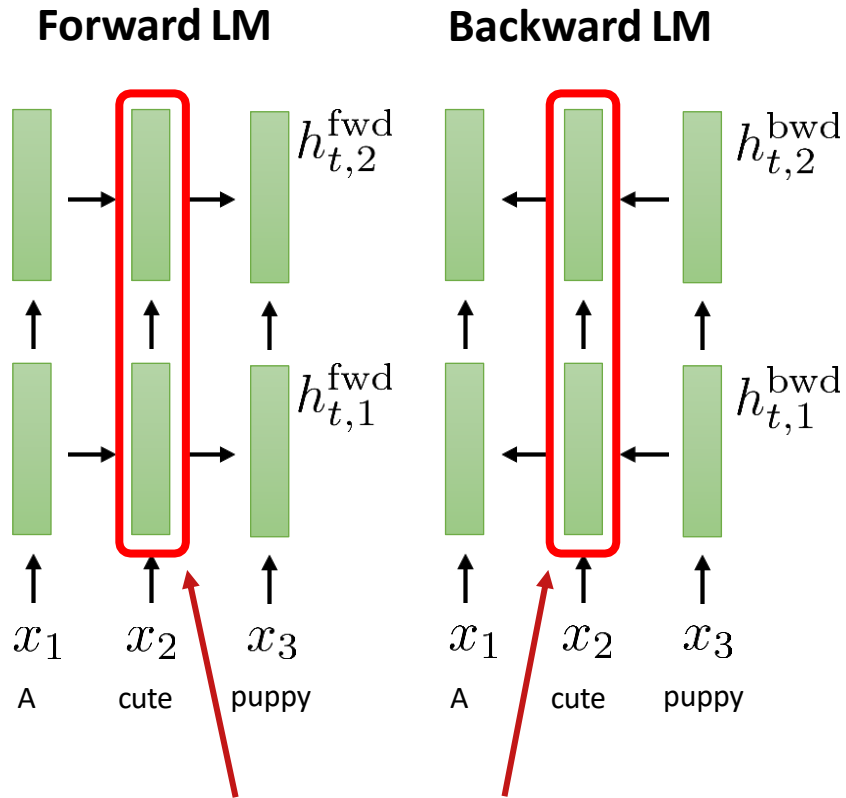


All these hidden states, when combined, represent the word "cute."

Replace static embeddings (lexicon lookup) with **context-dependent embeddings** (produced by a deep 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

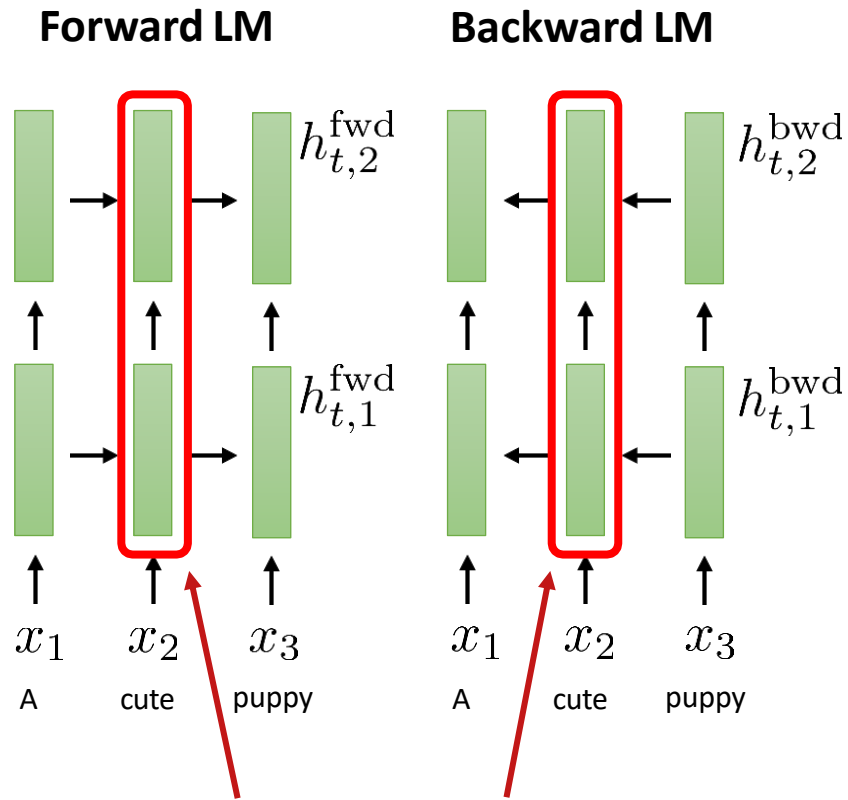
# ELMo Architecture



All these hidden states, when combined, represent the word "cute."

- 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 parameters of the language model.
- 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 Architecture



All these hidden states, when combined, represent the word "cute."

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

$$p(t_k | t_1, \dots, t_{k-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s)$$

Parameters: token embeddings  $\Theta_x$  LSTM  $\vec{\Theta}_{LSTM}$  softmax  $\Theta_s$

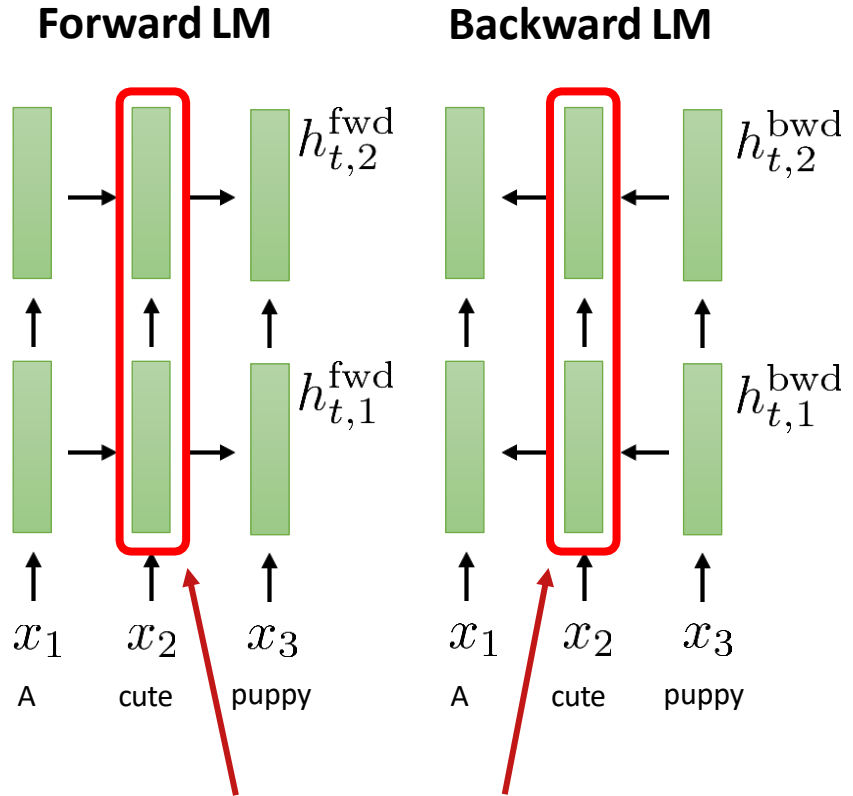
The **backward LM** is a deep LSTM that goes over the sequence from end to start to predict token  $t_k$  based on the suffix  $t_{k+1} \dots t_N$ :

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

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

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

# ELMo's Token Representation



All these hidden states, when combined, represent the word "cute."

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

With  $L$  layers, ELMo represents each token as

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

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

ELMo learns softmax weights  $s_j^{task}$  to collapse these vectors into a single vector and a task-specific scalar  $\gamma^{task}$ :

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

simple version:  $\mathbf{ELMO}_t = [h_{t,2}^{fwd}, h_{t,2}^{bwd}]$  top layer hidden states



# ELMo's Token Representation

- 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

# Evaluation

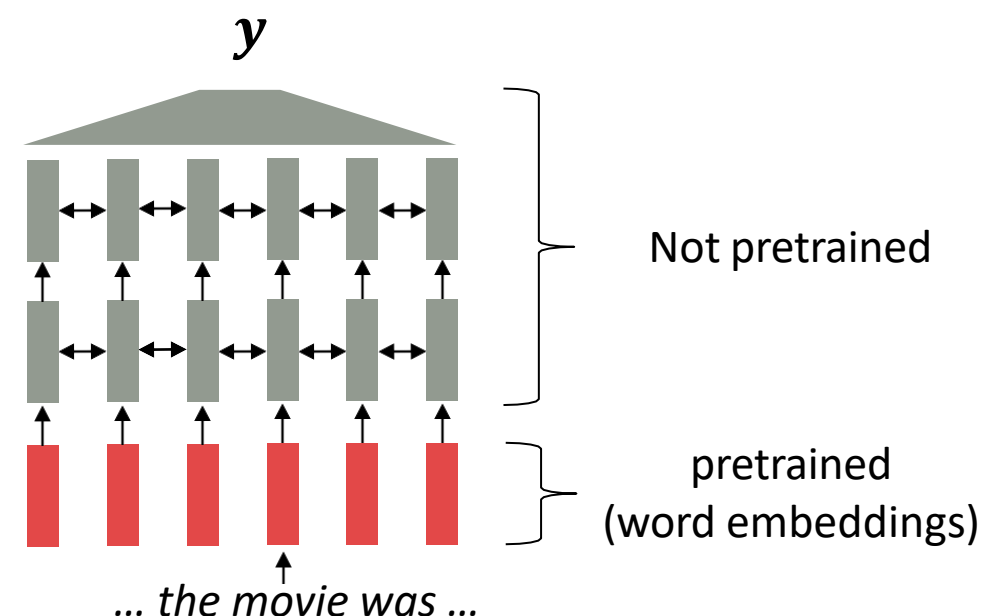
ELMo gave improvements on a variety of tasks:

- 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 $\pm$ 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 $\pm$ 0.19	90.15	92.22 $\pm$ 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 $\pm$ 0.5	3.3 / 6.8%

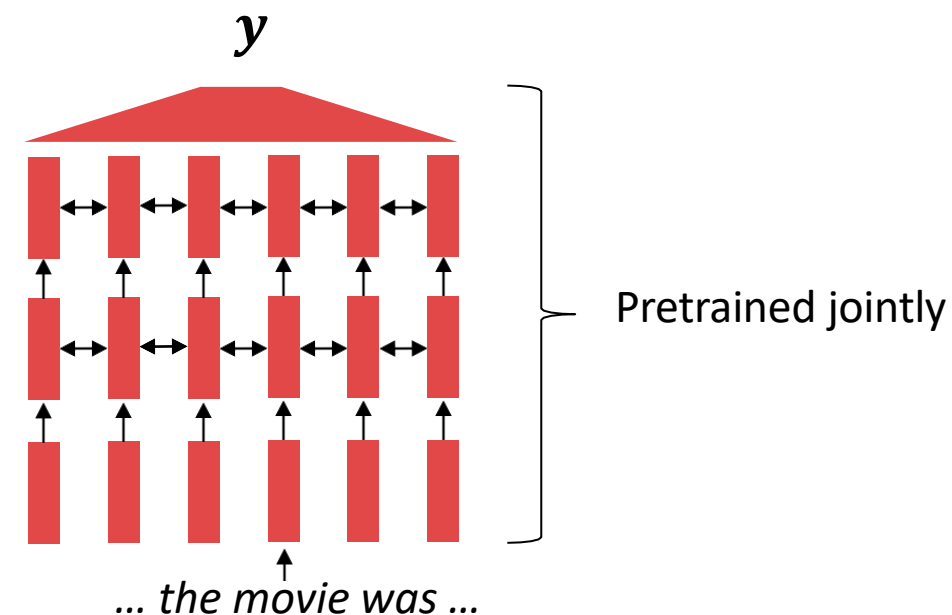
# Where We Were: Pre-trained Word Vectors

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.
- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network **are randomly initialized!**



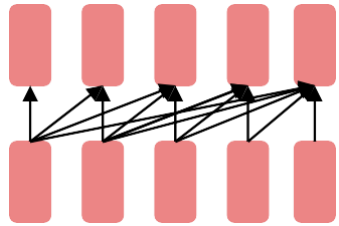
# Pre-trained Word Vectors -> Pre-trained Models

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.
- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from



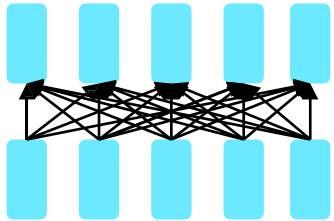
# Pretraining for Three Types of Architectures

The neural architecture influences the type of pretraining, and natural use cases.



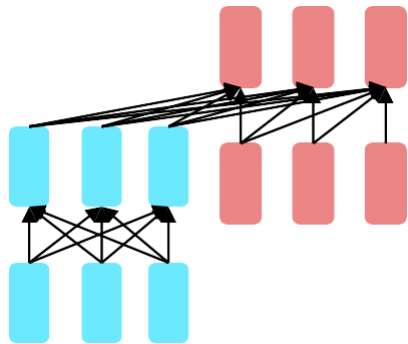
**Decoders**

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



**Encoders**

- Gets bidirectional context – can condition on future!
- How do we pretrain them?



**Encoder-  
Decoders**

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

**BERT:** Bidirectional Encoder Representations from Transformers

# **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**

**Jacob Devlin   Ming-Wei Chang   Kenton Lee   Kristina Toutanova**

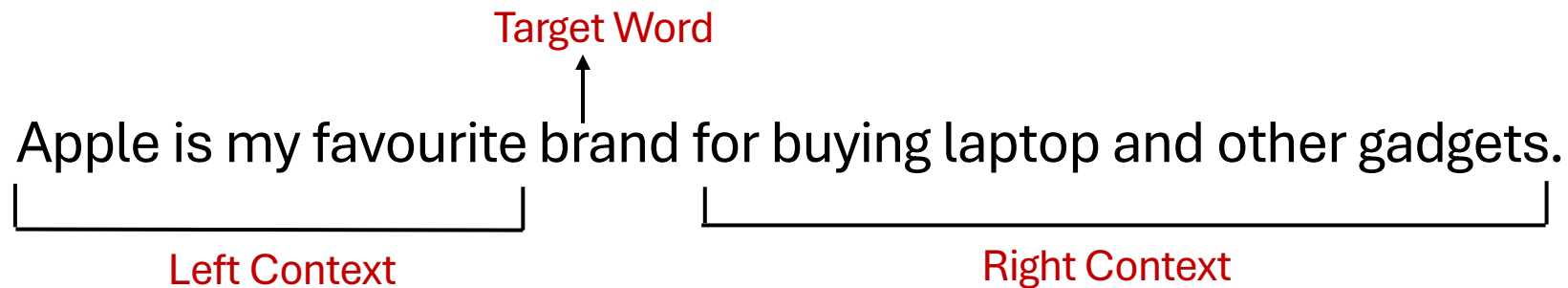
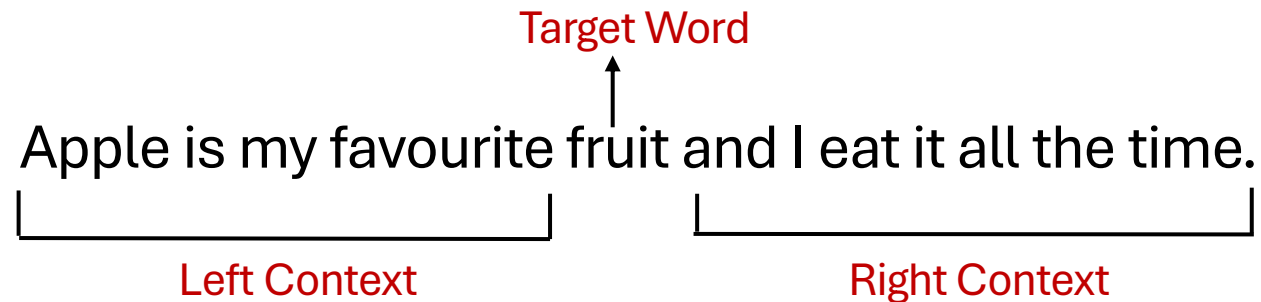
Google AI Language

`{jacobdevlin, mingweichang, kentonl, kristout}@google.com`

Slides are adopted from Jacob Devlin

# Background - Bidirectional Context

- Bidirectional context, unlike unidirectional context, takes into account both the left and right contexts.

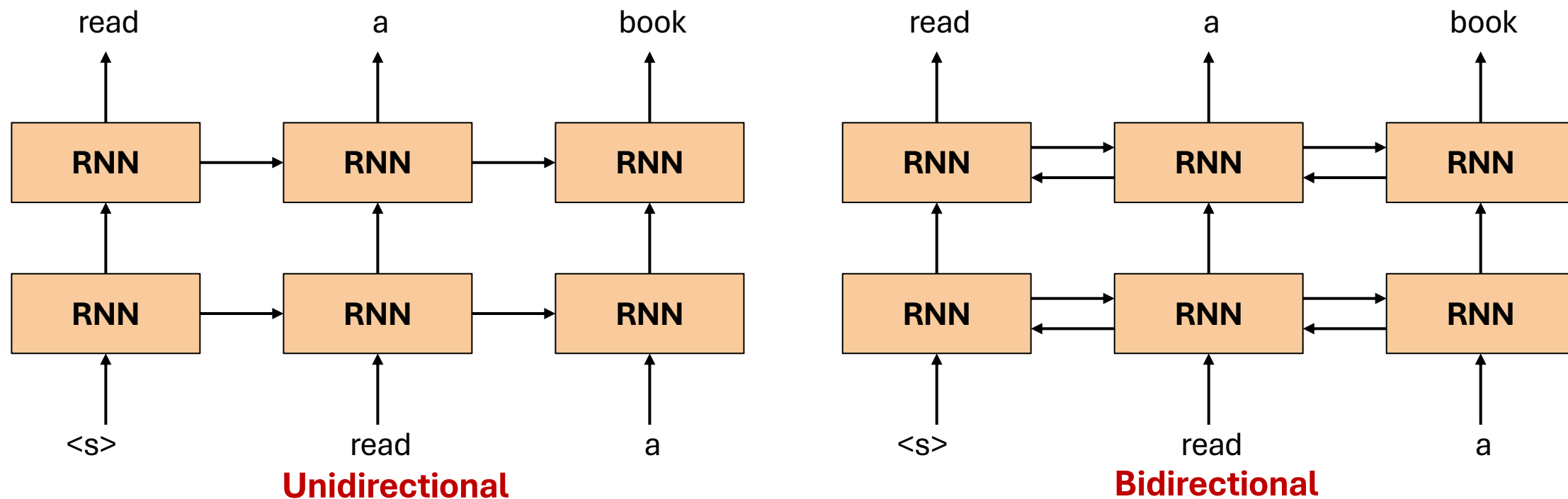


# Motivation

- **Problem with previous methods:**
  - Language models only use left context or right context.
  - But language understanding is **bidirectional**.
- **Possible Issue:**
  - Directionality is needed to generate a well-formed probability distribution.
  - Words can see themselves in a bidirectional model.



# Unidirectional vs. Bidirectional Models



# Masked Language Modelling

- Mask out  $k\%$  of the input words, and then predict the masked words (Usually  $k = 15\%$ ). Example :

I like going to the [MASK] in the evening

↓  
park

- Too little masking: Too expensive to train
- Too much masking: Not enough context
- The model needs to predict 15% of the words, but we don't replace with [MASK] 100% of the time. Instead:

- 80% of the time, **replace with [MASK]**

- Example : like going to the park → like going to the [MASK]

- 10% of the time, **replace random word**

- Example : like going to the park → like going to the store

- 10% of the time, **keep same**

- Example : like going to the park → like going to the park

# Next Sentence Prediction

- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence.

Input = [CLS] I enjoy read [MASK] book ##s [SEP]

I finish ##ed a [MASK] novel [SEP]

Label = IsNext

Input = [CLS] I enjoy read ##ing book [MASK] [SEP]

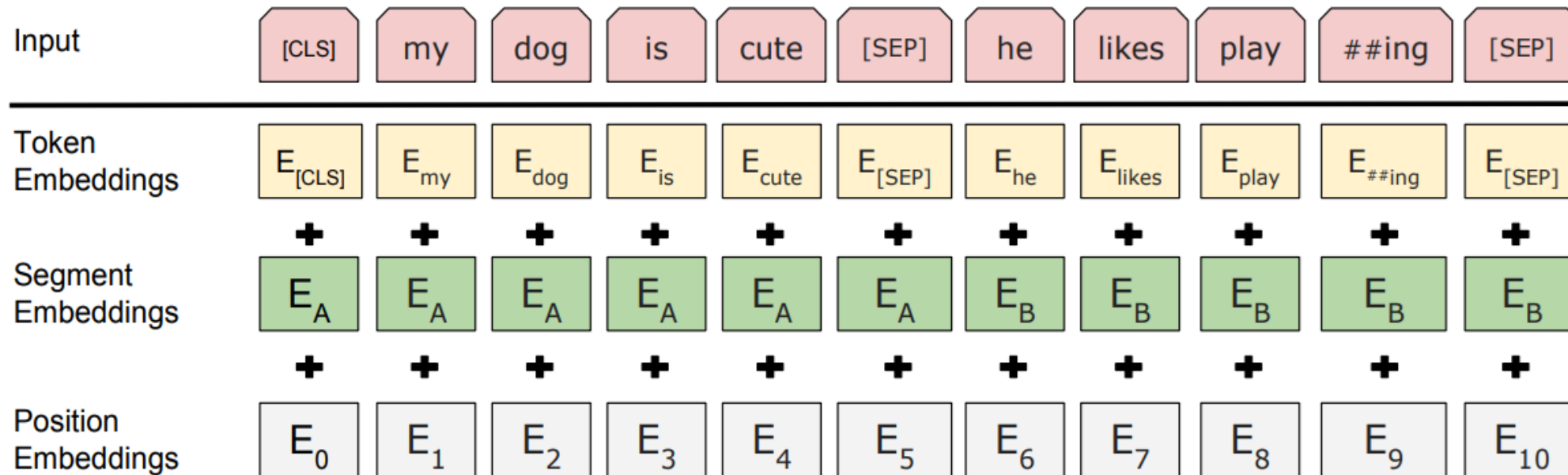
The dog ran [MASK] the street [SEP]

Label = NotNext

- Important for many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI)
- How to choose sentences A and B for pretraining?
  - 50% of the time B is the actual next sentence that follows A (labeled as IsNext)
  - 50% of the time it is a random sentence from the corpus (labeled as NotNext)

# Input Representation

- Use 30,000 WordPiece vocabulary on input.
- For a given token, its input representation is constructed by summing the token embeddings, the segmentation embeddings and the position embeddings.

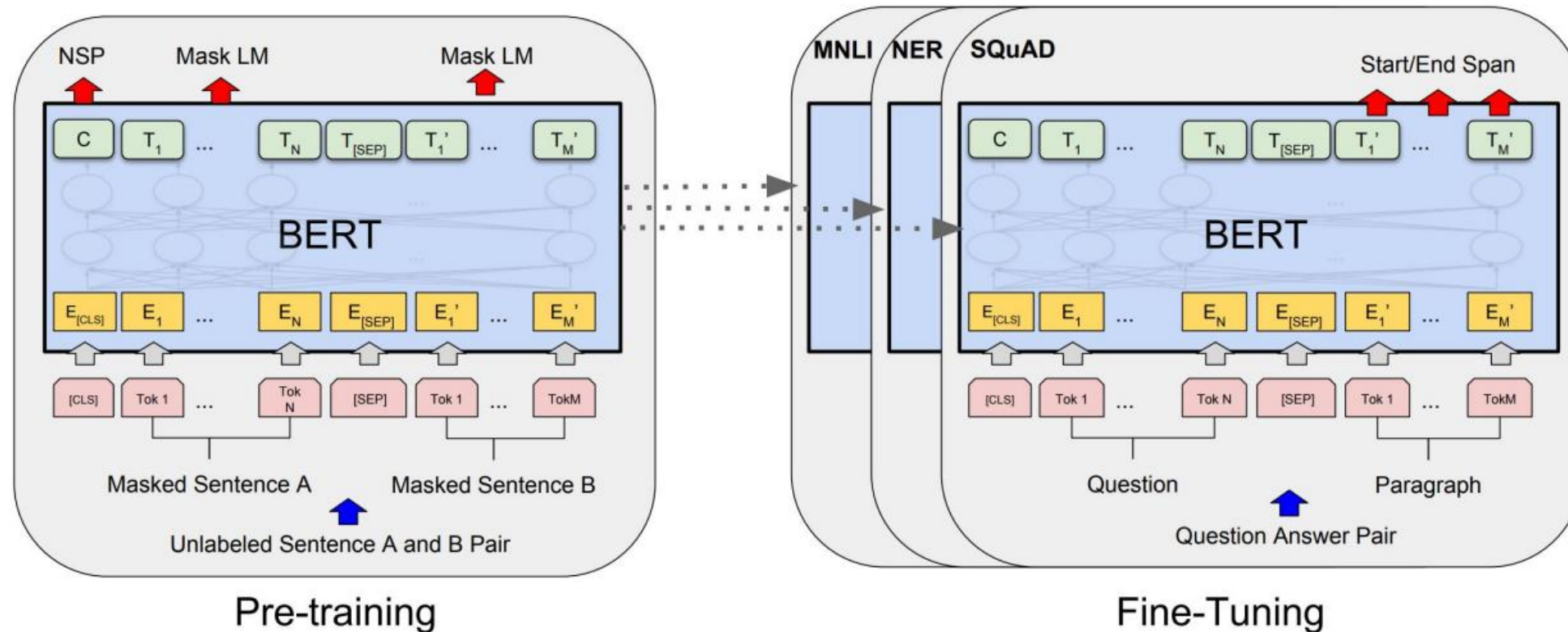


Source of Image : BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., NAACL 2019)

# Training Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences \* 128 length or 256 sequences \* 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

# Fine-Tuning Procedure



# QA Task based Fine-tuning

# BERT: Evaluation

BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI:** natural language inference over question answering data
- **SST-2:** sentiment analysis
- **CoLA:** corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B:** semantic textual similarity
- **MRPC:** Microsoft paraphrase corpus
- **RTE:** a small natural language inference corpus

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>