There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are sub-optimal for sentence-level tasks, and could be very harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions.

There are two main ways to use pre-trained language models for specific tasks:

1. Feature-Based Approach (Example: ELMo)

- Think of this like using a language model as a helper rather than changing it.
- The model (ELMo) is trained first on a large dataset and learns how words are related.
- Then, we **extract** the learned word representations (embeddings) from ELMo and **add them as extra features** to another model designed for a specific task (like sentiment analysis or question answering).
- The main model for the task is separate and does not update the pre-trained model much.

💡 **Analogy**: It's like using a dictionary. You don't rewrite the dictionary every time you use it; you just look up meanings and apply them.

2. Fine-Tuning Approach (Example: OpenAI GPT)

- Here, instead of just using the pre-trained model's outputs, we **train it further on the specific task**.
- The entire model, including pre-trained parameters, is **fine-tuned** for a new task.
- GPT (Generative Pre-trained Transformer) uses a **left-to-right** structure, meaning it only considers words that came **before** in a sentence when making predictions.
- This left-to-right approach is **good for generating text**, but **bad for tasks that require full sentence understanding**, like question answering or text classification.

💡 **Analogy**: Instead of using a dictionary, imagine learning a language and practicing by using new words in different sentences until you fully understand them.

Key Differences

Feature	Feature-Based (ELMo)	Fine-Tuning (GPT)
How it's used?	Uses pre-trained features in a separate model	Fine-tunes the entire model on the task
Model updates?	Pre-trained model stays mostly unchanged	The entire model is updated for the task
Architecture?	Bi-directional LSTMs (context from both sides)	Left-to-right Transformer (only looks at past words)

Best for?	Structured NLP tasks (e.g., named entity recognition, POS tagging)	Text generation, language modeling
Limitation?	Requires additional task-specific models	Cannot see the full sentence at once (bad for QA, classification)

How BERT Solved This Issue

- **BERT (Bidirectional Encoder Representations from Transformers)**, introduced in 2018, **fixed the limitations of GPT's left-to-right approach**.
- Instead of processing text in only one direction, BERT **looks at both past and future words at the same time** (bi-directional).
- This helps for **tasks that require full sentence understanding**, like sentiment analysis or question answering.
- BERT also introduced **"masked language modeling"**, where random words are hidden in training so the model learns to predict them using surrounding words.
- This makes BERT much better at understanding context than GPT's left-to-right model.

💡 **Analogy**: If GPT reads a sentence like a novel (word by word), BERT reads it like a puzzle, looking at all the words together to understand the full meaning.

ELMo vs. BERT: Understanding the Key Differences

Both **ELMo** and **BERT** are pre-trained language models that improve NLP tasks, but they are fundamentally different in **architecture, training methods, and how they generate word representations**. Let's break it down in detail.

1. ELMo (Embeddings from Language Models)

📜 **Introduced by:** AllenNLP (Peters et al., 2018)
💡 **Main Idea:** Uses pre-trained deep **bi-directional LSTMs** to generate word representations, which are then **used as features** in another model for downstream tasks.

ELMo Architecture

1. **Bi-directional LSTM:**
   - Uses **two LSTMs**—one reads text from left to right (**forward LSTM**), and the other reads from right to left (**backward LSTM**).

   - This ensures that the model captures **both past and future context** when generating word embeddings.
2. **Stacked LSTM Layers:**
   - ELMo stacks **two layers of LSTMs** (deep LSTMs), which helps in capturing more abstract representations of words.
3. **Pre-Training Objective (Language Model Training):**
   - Trained on **large corpora** like Wikipedia by **predicting the next word** (forward) and the **previous word** (backward).
   - This helps it learn the context in both directions.
4. **Feature-Based Approach:**
   - After pre-training, ELMo generates **contextual word embeddings** that are passed into a separate task-specific model.
   - These embeddings are used as **features**, but the core ELMo model **is not fine-tuned** for each task.

💡 **Key Strengths of ELMo:**
✅ **Captures context** because it uses bi-directional LSTMs.
✅ Works well for **sequence labeling tasks** (like Named Entity Recognition, POS tagging).
✅ **Can be combined with other models** (e.g., it can be plugged into an LSTM-based classifier).

❌ **Weaknesses of ELMo:**

- LSTMs process sequences **sequentially** (word by word), which is **slower** than parallel processing in Transformers.
- Even though it is bi-directional, **it does not deeply integrate both directions** like BERT does.
- **Feature-based approach** requires extra task-specific architectures.

---

2. BERT (Bidirectional Encoder Representations from Transformers)

📜 **Introduced by:** Google AI (Devlin et al., 2018)
💡 **Main Idea:** Uses **Transformers** instead of LSTMs and is **fully bidirectional**, allowing it to deeply understand the context of words.

BERT Architecture

1. **Transformer-Based Model:**

    - Unlike ELMo’s LSTMs, BERT is built on **Transformer encoders** (self-attention mechanisms).
    - This allows it to process **entire sequences at once** (parallel processing), making it **much faster** than LSTMs.
2. **Bidirectional Context:**
    - Instead of separate forward and backward LSTMs, BERT **reads the entire sentence at once** using a self-attention mechanism.
    - This allows it to use **both past and future words** to predict missing words (deep bi-directionality).
3. **Pre-Training Objectives:**
    - **Masked Language Model (MLM):**
        - Instead of predicting the next word like ELMo, BERT randomly **masks** words and forces the model to predict them using the surrounding context.
    - **Next Sentence Prediction (NSP):**
        - BERT is trained to understand relationships between two sentences by predicting whether the second sentence follows the first.
4. **Fine-Tuning Approach:**
    - Unlike ELMo, BERT is **fine-tuned** on downstream tasks (such as Question Answering, Text Classification).
    - The entire model, including **pre-trained parameters, is updated** for each specific task.

💡 **Key Strengths of BERT:**
✅ **Fully bidirectional** (better contextual understanding than ELMo).
✅ **Uses self-attention** (can process entire sequences in parallel, much faster than LSTMs).
✅ **Works well for both sentence-level and word-level tasks** (unlike OpenAI GPT, which is only left-to-right).
✅ **Fine-tuned per task**, leading to **better performance than feature-based models like ELMo**.

❌ **Weaknesses of BERT:**

- More computationally expensive than ELMo due to the **Transformer architecture**.
- Requires a **large dataset and GPU power** to fine-tune effectively.

3. Key Differences Between ELMo and BERT

Feature	ELMo (Feature-Based)	BERT (Fine-Tuning)
Architecture	Uses **bi-directional LSTMs**	Uses **Transformer encoders**

Directionality	Bi-directional (but forward and backward are separate)	Fully **deep bidirectional**
Pre-Training Task	Predicts next/previous word (separately)	**Masked Language Model (MLM) + Next Sentence Prediction (NSP)**
How it's used?	Word embeddings are extracted and used as features	Fine-tuned end-to-end for specific tasks
Context Handling	Uses LSTMs (sequential processing)	Uses self-attention (parallel processing)
Task Adaptation	Needs extra models for each task	Directly fine-tuned for each task
Efficiency	Slower due to sequential LSTM processing	Faster due to Transformer parallelization
Best for?	Sequence tagging (NER, POS tagging)	Text classification, QA, summarization, etc.

4. Why is BERT Better Than ELMo?

1. **Better Bidirectionality:**
   - ELMo has separate forward and backward LSTMs, meaning it doesn't integrate both directions deeply.
   - BERT processes **the entire sentence at once**, learning richer representations.
2. **Faster Processing with Transformers:**
   - LSTMs in ELMo process words one by one (sequentially), making them **slower**.
   - BERT's Transformer model **processes all words in parallel**, improving speed and scalability.
3. **More Effective Pre-Training Objective:**
   - ELMo only predicts the next/previous word.
   - BERT's **Masked Language Model** forces the model to learn **deeper contextual relationships**.
4. **Fine-Tuning vs. Feature-Based Approach:**
   - ELMo provides word embeddings that need an **extra model** to make predictions.
   - BERT can be **fine-tuned end-to-end**, leading to better performance on a wide range of tasks.

5. When Would You Choose ELMo Over BERT?

✅ **If you have limited computational resources** → ELMo is less expensive to use.
✅ **If you only need word embeddings** → ELMo gives good contextual word representations.
✅ **If you are working on tasks like POS tagging or NER**, where LSTM-based embeddings work well.

🚀 **But for most NLP applications today, BERT is the superior choice** because it is **more powerful, deeply bidirectional, and adaptable to many tasks**.

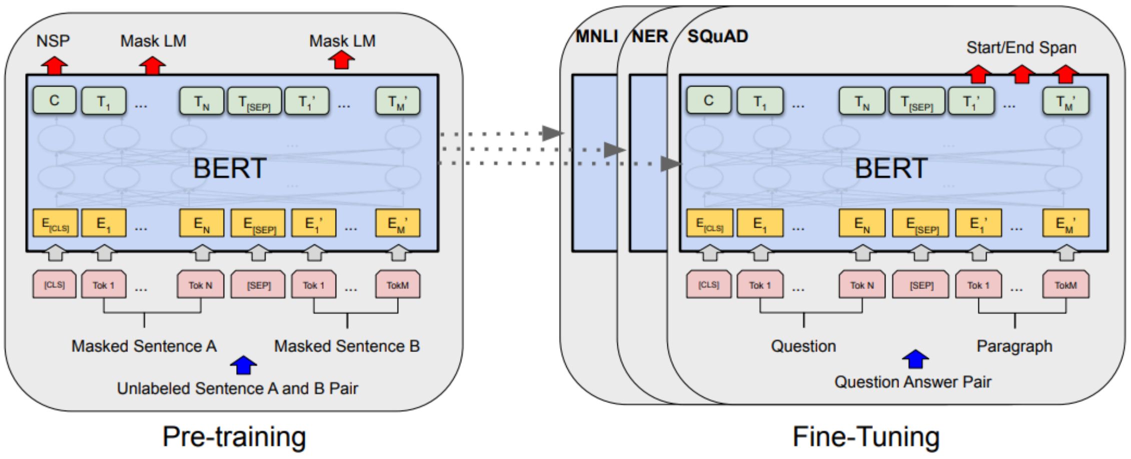

Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

This image illustrates the **BERT (Bidirectional Encoder Representations from Transformers) training process**, which consists of two main stages: **Pre-training** and **Fine-tuning**.

1. Pre-training (Left Side of the Image)

- BERT is first **pre-trained** on a large corpus of text using **unsupervised learning**.
- It learns contextual representations of words using two tasks:
  - **Masked Language Model (Mask LM)**:
    - Some words in the input sentences are randomly **masked** (hidden), and BERT is trained to predict these missing words.
  - **Next Sentence Prediction (NSP)**:

        - BERT is trained to determine whether **Sentence B follows Sentence A** in the original text.
- The input consists of **two unlabeled sentences (Sentence A and B pair)**, which are tokenized and processed.
- Each token is embedded using:
    - **Token embeddings** (E_1, E_2, …)
    - **Segment embeddings** (to distinguish Sentence A from Sentence B)
    - **Positional embeddings** (to capture word order)

2. Fine-tuning (Right Side of the Image)

- After pre-training, BERT is **fine-tuned** for specific tasks by adding output layers.
- Example fine-tuning tasks shown in the image:
    1. **MNLI (Multi-Genre Natural Language Inference) & NER (Named Entity Recognition)**
        - BERT is adapted to classify sentence relationships and recognize named entities.
    2. **SQuAD (Stanford Question Answering Dataset)**
        - BERT is fine-tuned for **Question Answering (QA)**.
        - Given a **question and a paragraph**, BERT learns to predict the **start and end span** of the correct answer.

Key Takeaways

- **Pre-training**: BERT learns general language understanding from a large dataset.
- **Fine-tuning**: BERT is specialized for specific tasks like classification, NER, and QA.
- This approach enables **transfer learning**, where the knowledge from pre-training helps improve performance on downstream tasks.

Pre-training Phase (Left Side)

BERT is trained on a **large corpus of text** using **unsupervised learning** with two primary tasks:

1. **NSP (Next Sentence Prediction)**
    - The **C** token (classification token) at the beginning helps in classifying whether **Sentence B follows Sentence A**.
    - BERT is trained to predict whether two sentences are logically connected.
2. **Mask LM (Masked Language Model)**
    - BERT learns to predict missing words in sentences by randomly **masking words** and trying to reconstruct them.
3. **Tokens and Embeddings:**
    - **C** → Classification token (**[CLS]**) used for sentence-level classification tasks.

    - **T_1, T_2, …, T_n** → Tokens representing words in the input sentences.
    - **TSEP** → Separator token (**[SEP]**) used to differentiate Sentence A from Sentence B.
    - **E_1, E_2, …, E_n** → Word **Embeddings** representing individual token meanings.
    - **FCLS** → Feature embedding of the **[CLS]** token used for classification tasks.

Fine-tuning Phase (Right Side)

Once BERT is pre-trained, it is fine-tuned for specific **NLP (Natural Language Processing) tasks**.

1. **MNLI (Multi-Genre Natural Language Inference)**
    - A dataset used for sentence-pair classification to determine the relationship between two sentences.
2. **NER (Named Entity Recognition)**
    - A task where BERT identifies entities (e.g., names, locations, dates) in text.
3. **SQuAD (Stanford Question Answering Dataset)**
    - A dataset used for **Question Answering (QA)**.
    - Given a **paragraph** and a **question**, BERT is trained to find the **start and end span** of the correct answer.
4. **Question Answering (QA) Task:**
    - **Start/End Span** → The model predicts the position (span) of the correct answer within the paragraph.

Key Tokens in BERT's Input Representation

- **[CLS]** → A special classification token, used for tasks like NSP and text classification.
- **[SEP]** → A special separator token, used to distinguish Sentence A from Sentence B.
- **Tok 1, Tok N, Tok M** → Tokenized words from the input text.

Summary of Abbreviations

Abbreviation	Meaning
BERT	Bidirectional Encoder Representations from Transformers
NSP	Next Sentence Prediction
Mask LM	Masked Language Model

C	Classification Token (**[CLS]**)
T_1, T_2, ...	Tokenized Words
T[SEP]	Separator Token (**[SEP]**)
E_1, E_2, ...	Embeddings for tokens
F[CLS]	Feature vector of the **[CLS]** token
MNLI	Multi-Genre Natural Language Inference (Sentence-Pair Classification)
NER	Named Entity Recognition
SQuAD	Stanford Question Answering Dataset
QA	Question Answering
Start/End Span	Predicted position of the correct answer in QA tasks

---

Conclusion

This image shows how **BERT learns representations in the pre-training stage** and how these representations are **fine-tuned for specific NLP tasks**. Let me know if you need more clarification! 🚀

3 BERT

We introduce BERT and its detailed implementation in this section. There are two steps in our framework: *pre-training* and *fine-tuning*. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. The question-answering example in Figure 1 will serve as a running example for this section.

In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A.[3] We primarily report results on two model sizes: $\mathbf{BERT_{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M) and $\mathbf{BERT_{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M).

$\mathrm{BERT_{BASE}}$ was chosen to have the same model size as OpenAI GPT for comparison purposes. Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Transformer uses constrained self-attention where every token can only attend to context to its left.[4]

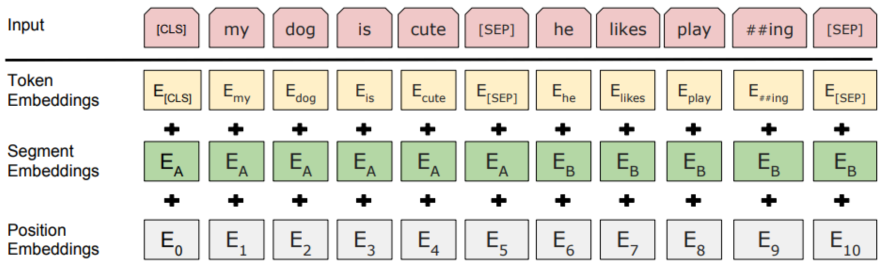
Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8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

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
BERT$_{\text{LARGE}}$	96.6	92.8
BERT$_{\text{BASE}}$	96.4	92.4
Feature-based approach (BERT$_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

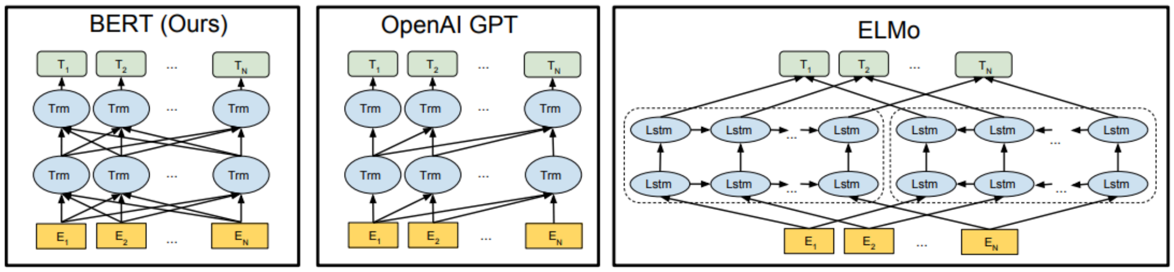

Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.