In my previous NLP post on word embedding, the introduced embeddings are not context-specific — they are learned based on word concurrency but not sequential context. So in two sentences, "*I am eating an apple*" and "*I have an Apple phone*", two "apple" words refer to very different things but they would still share the same word embedding vector.

Despite this, early adoption of word embeddings in problem-solving is to use them as additional features for an existing task-specific model and in a way the improvement is bounded.

CoVe (Dependent on Pre-training) Supervised

CoVe (McCann et al. 2017), short for **Contextual Word Vectors**, is a type of word embeddings learned by an encoder in an attentional seq-to-seq machine translation model. Different from traditional word embeddings introduced here, CoVe word representations are functions of the entire input sentence.

Use CoVe in Downstream Tasks

The hidden states of NMT encoder are defined as **context vectors** for other language tasks:

$$\text{CoVe}(x) = \text{biLSTM}(\text{GloVe}(x))$$

The paper proposed to use the concatenation of GloVe and CoVe for question-answering and classification tasks. GloVe learns from the ratios of global word co-occurrences, so it has no sentence context, while CoVe is generated by processing text sequences is able to capture the contextual information.

$$v = [\text{GloVe}(x); \text{CoVe}(x)]$$

Given a downstream task, we first generate the concatenation of GloVe + CoVe vectors of input words and then feed them into the task-specific models as additional features.

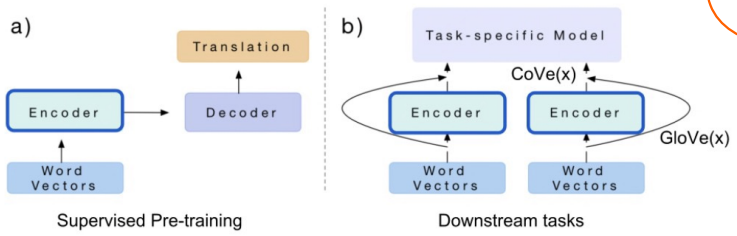


Fig. 2. The CoVe embeddings are generated by an encoder trained for machine translation task. The encoder can be plugged into any downstream task-specific model. (Image source: original paper)

Summary: The limitation of CoVe is obvious: (1) pre-training is bounded by available datasets on the supervised translation task; (2) the contribution of CoVe to the final performance is constrained by the task-specific model architecture.

ELMo (Unsupervised)

ELMo, short for **Embeddings from Language Model** (Peters, et al, 2018) learns contextualized word representation by pre-training a language model in an *unsupervised* way.

Bidirectional Language Model

The bidirectional Language Model (**biLM**) is the foundation for ELMo. While the input is a sequence of n tokens, $(x_1, \dots, x_n)$, the language model learns to predict the probability of next token given the history.

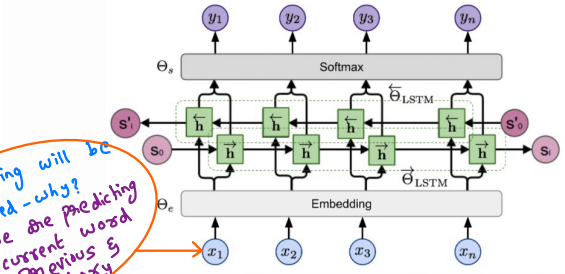
In the forward pass, the history contains words before the target token,

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1})$$

In the backward pass, the history contains words after the target token,

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i+1}, \dots, x_n)$$

The predictions in both directions are modeled by multi-layer LSTMs with hidden states $\overrightarrow{\mathbf{h}}_{i,\ell}$ and $\overleftarrow{\mathbf{h}}_{i,\ell}$ for input token x_i at the layer level $\ell = 1, \dots, L$. The final layer's hidden state $\mathbf{h}_{i,L} = [\overrightarrow{\mathbf{h}}_{i,L}; \overleftarrow{\mathbf{h}}_{i,L}]$ is used to output the probabilities over tokens after softmax normalization. They share the embedding layer and the softmax layer, parameterized by Θ_e and Θ_s respectively.



Masking will be applied - why? As we are predicting the current word given previous & Next history

Fig. 3. The biLSTM base model of ELMo. (Image source: recreated based on the figure in "Neural Networks, Types, and Functional Programming" by Christopher Olah.)

The model is trained to minimize the negative log likelihood (= maximize the log likelihood for true words) in both directions:

$$\mathcal{L} = -\sum_{i=1}^{n} \Big(\log p(x_i \mid x_1, \dots, x_{i-1}; \Theta_e, \overrightarrow{\Theta}_{\text{LSTM}}, \Theta_s) + \log p(x_i \mid x_{i+1}, \dots, x_n; \Theta_e, \overleftarrow{\Theta}_{\text{LSTM}}, \Theta_s) \Big)$$

Use ELMo in Downstream Tasks

Similar to how CoVe can help different downstream tasks, ELMo embedding vectors are included in the input or lower levels of task-specific models. Moreover, for some tasks (i.e., SNLI and SQuAD, but not SRL), adding them into the output level helps too.

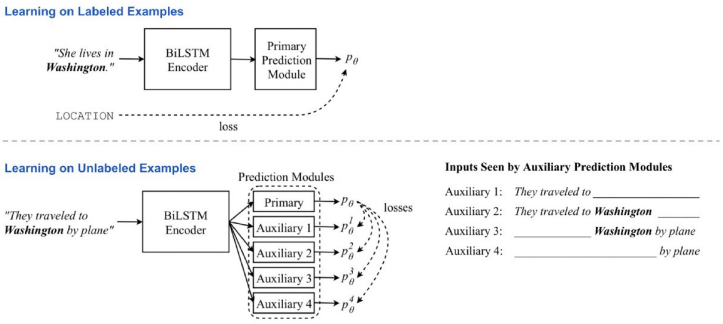
The improvements brought up by ELMo are largest for tasks with a small supervised dataset. With ELMo, we can also achieve similar performance with much less labeled data.

Summary: The language model pre-training is unsupervised and theoretically the pre-training can be scaled up as much as possible since the unlabeled text corpora are abundant. However, it still has the dependency on task-customized models and thus the improvement is only incremental, while searching for a good model architecture for every task remains non-trivial.

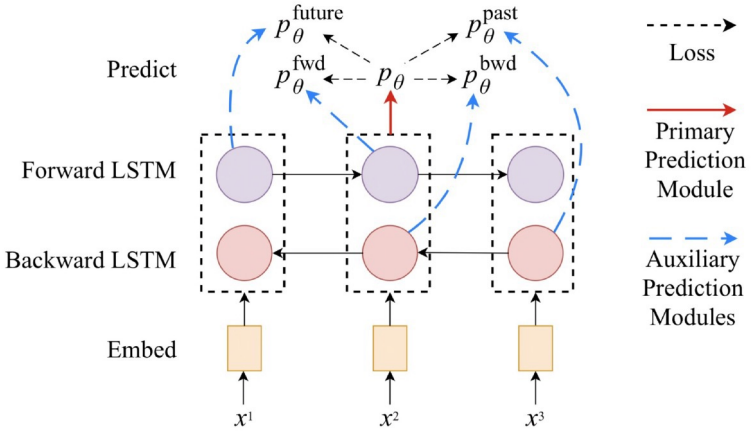
Both can be used in a real-app like Question-Answering Sys/Sentiment Analysis (Positive/Negative)

Cross-View Training (CVT)

Describe CVT and highlight ideas that that could be useful in other application domains!



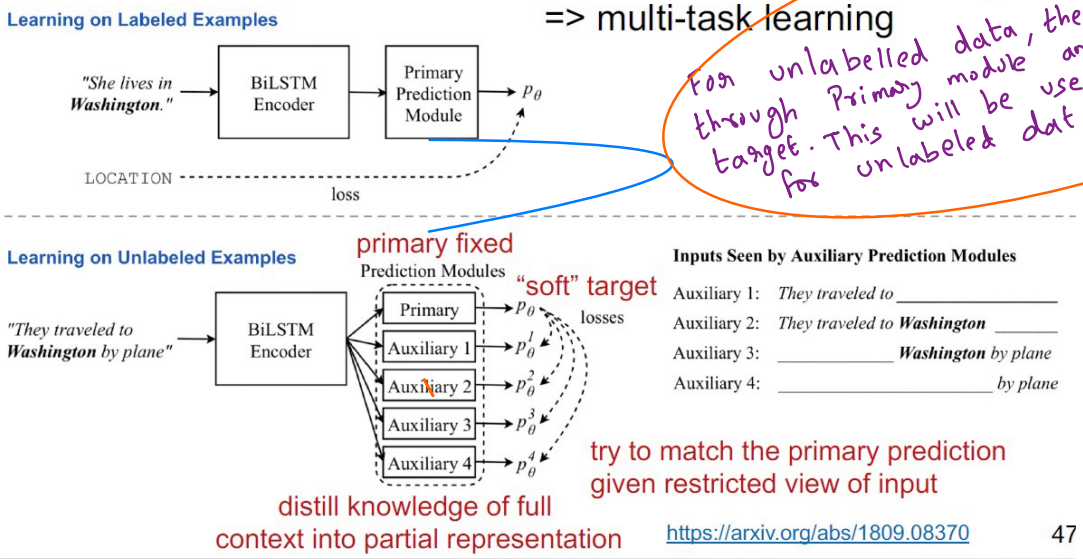
CVT Example: Sequential Tagging



https://arxiv.org/abs/1809.08370

Cross-View Training (CVT)

- combines unsupervised (pre-)training and task-specific supervised learning
  => semi-supervised learning
  => multi-task learning



For unlabelled data, the Bi-LSTM output goes through Primary module and gives a soft target. This will be used as true-target for unlabeled dat & back-Propagated

https://arxiv.org/abs/1809.08370

CVT Multi-Task Learning

- several extra primary prediction models
- same BiLSTM encoder
- supervised training:
  - 1 task randomly selected at each step
  - update primary predictor + encoder
- unsupervised training:
  - all tasks in parallel (soft targets)
  - update encoder + auxiliary predictors
- encourages better generality
- produces cross-task labels

CVT Example: Machine Translation

- primary prediction module:
  - unidirectional LSTM decoder with attention
- auxiliary tasks:
  1. apply dropout on attention weight vector by randomly zeroing out some values
  2. predict the future word in target sequence

soft target: best predicted sequence by (fixed) beam search with primary decoder

GPT

Following the similar idea of ELMo, OpenAI **GPT**, short for **Generative Pre-training Transformer** (Radford et al., 2018), expands the unsupervised language model to a much larger scale by training on a giant collection of free text corpora. Despite of the similarity, GPT has two major differences from ELMo.

1. The model architectures are different: ELMo uses a shallow concatenation of independently trained left-to-right and right-to-left multi-layer LSTMs, while GPT is a multi-layer transformer decoder.
2. The use of contextualized embeddings in downstream tasks are different: ELMo feeds embeddings into models customized for specific tasks as additional features, while GPT fine-tunes the same base model for all end tasks.

This model applies multiple transformer blocks over the embeddings of input sequences. Each block contains a masked *multi-headed self-attention* layer and a *pointwise feed-forward* layer. The final output produces a distribution over target tokens after softmax normalization.

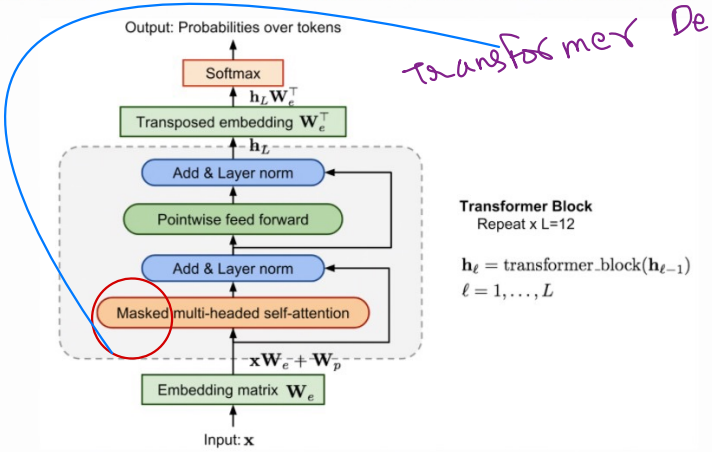


Fig. 7. The transformer decoder model architecture in OpenAI GPT.

The loss is the negative log-likelihood, same as ELMo, but without backward computation. Let's say, the context window of the size k is located before the target word and the loss would look like:

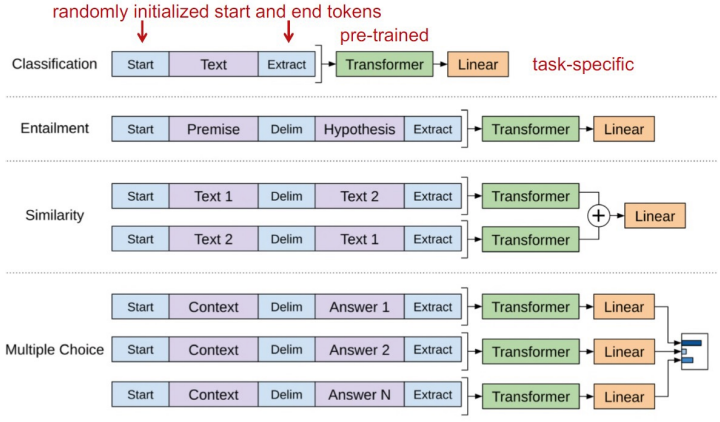
$$\mathcal{L}_{\text{LM}} = -\sum_i \log p(x_i \mid x_{i-k}, \ldots, x_{i-1})$$

GPT Byte Pair Encoding (BPE)

- data compression algorithm (from 1990s)
- used to encode the input sequences
- find best word segmentation by iteratively and greedily merging frequent pairs of characters (considering word boundaries!)
- run on the dictionary extracted from a text, with each word being weighted by its frequency

1-) Build vocab from text corpus
2-) Separate each char in word by space & add extra token at the end.
3) Count freq of tokens in corpus
4) get counts of pairs
5) Merge all occurences of most freq pair.

GPT Tasks



lo & behold, low, lower, lowest

[lo, low, er, est]

[lo, we, r, st, be, ho, ld]

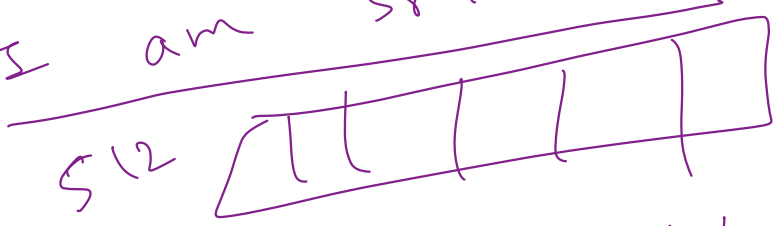
lo → 4

we → 2

lo(2) → 4

low(3) → 3

I am srinath.



I am srinath

I/w
am
sr
in
at
h/w

I/w/w
am/w
sri

1024

512

n=512

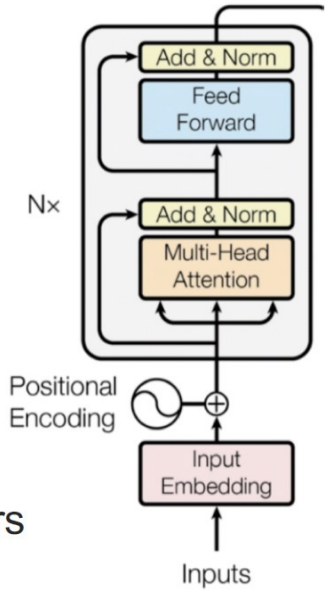
BERT (Bidirectional Encoder Representations from Transformers)

Transformer Encoder

- multi-layer bidirectional Transformer encoder

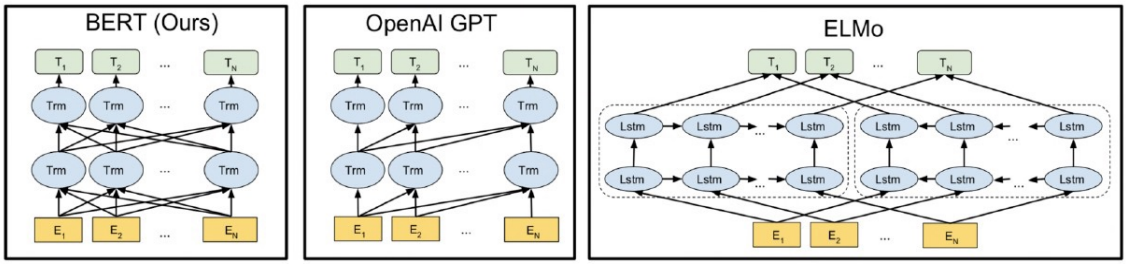
- trained with 2 auxiliary tasks:

1. mask language model (MLM):
   randomly mask 15% of tokens
   (80 [MASK], 10% random, 10% same)
2. next sentence prediction:
   binary classifier for sentence pairs

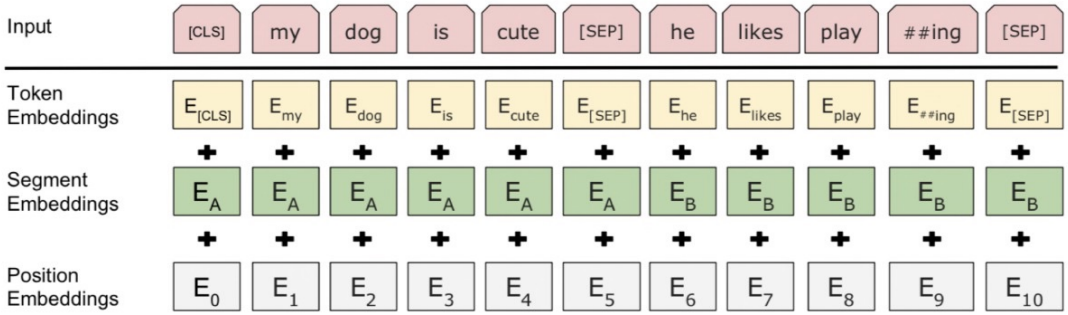


https://arxiv.org/abs/1706.03762

BERT vs. GPT vs. ELMo LM



BERT: Input Embedding



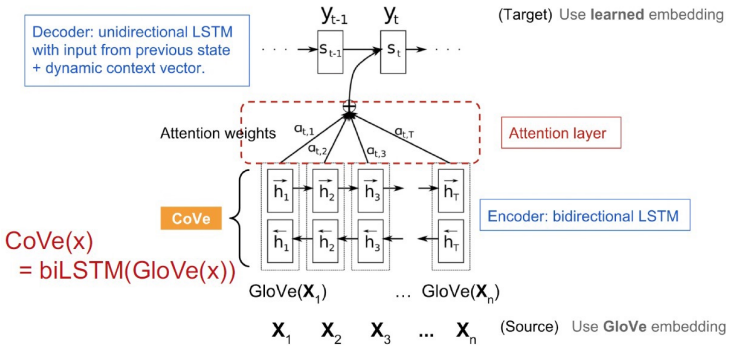
1. divide words into smaller sub-word units
   (more effective to handle rare or unknown words)
2. separate multiple sentences [SEP] & encode IDs
3. add (learned) positional embeddings

Summary

	Base model	pre-training	Downstream tasks	Downstream model	Fine-tuning
CoVe	seq2seq NMT model	supervised	feature-based	task-specific	/
ELMo	two-layer biLSTM	unsupervised	feature-based	task-specific	/
CVT	two-layer biLSTM	semi-supervised	model-based	task-specific / task-agnostic	/
ULMFiT	AWD-LSTM	unsupervised	model-based	task-agnostic	all layers; with various training tricks
GPT	Transformer decoder	unsupervised	model-based	task-agnostic	pre-trained layers + top task layer(s)
BERT	Transformer encoder	unsupervised	model-based	task-agnostic	pre-trained layers + top task layer(s)
GPT-2	Transformer decoder	unsupervised	model-based	task-agnostic	pre-trained layers + top task layer(s)

Contextual Word Vectors (CoVe)

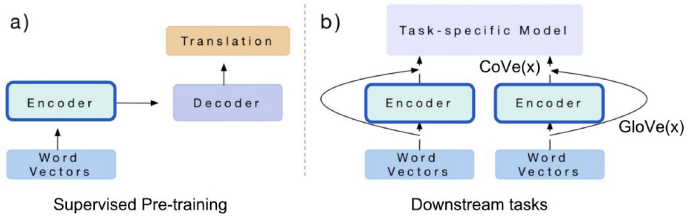
- learned by an encoder in an attentional seq2seq NMT
- functions of entire input sentence



https://arxiv.org/abs/1708.00107
image: https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html

Contextual Word Vectors (CoVe)

- CoVe(x) = biLSTM(GloVe(x))
- concatenate GloVe and CoVe for question-answering and classification tasks

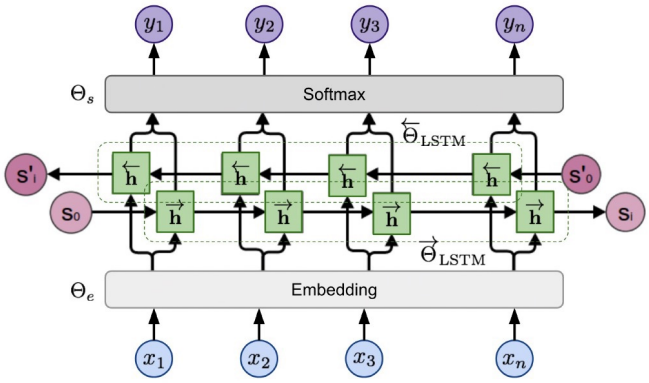


- limitations: 1) pre-training bounded by available datasets on **supervised** translation task
  2) contribution to final performance constrained by **task-specific** model architecture

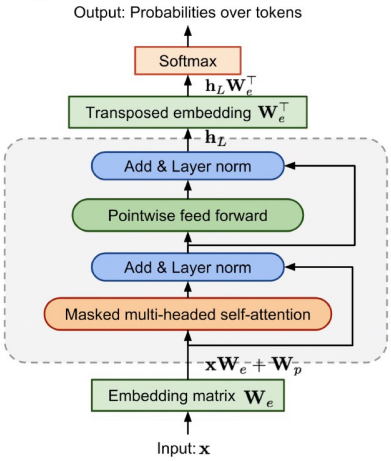
ELMo (Embeddings from Language Model)

- unsupervised task
- bi-directional LM (biLM)
- predict word in sequence given history of
  - words before (forward pass)
  - words after (backward pass)
- using bi-directional multi-layer LSTM

ELMo (Embeddings from Language Model)



OpenAI GPT (Generative Pre-training Transformer)



Transformer Block
Repeat x L=12

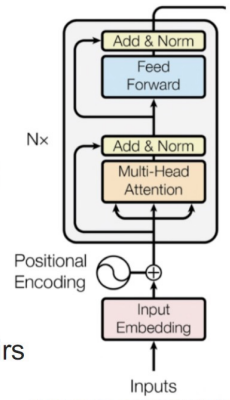
$$\mathbf{h}_\ell = \text{transformer\_block}(\mathbf{h}_{\ell-1}) \quad \ell = 1, \ldots, L$$

GPT vs. ELMo

- different architecture:
  - ELMo: bi-directional multi-layer LSTM
  - GPT: transformer decoder (forward only!)
- fine-tuning for downstream tasks:
  - ELMo: task-specific linear combination of states across layers
  - GPT: same model for all tasks
- input encoding (next slide)

BERT (Bidirectional Encoder Representations from Transformers)

- multi-layer bidirectional Transformer encoder
- trained with 2 auxiliary tasks:
  1. mask language model (MLM): randomly mask 15% of tokens (80 [MASK], 10% random, 10% same)
  2. next sentence prediction: binary classifier for sentence pairs



https://arxiv.org/abs/1706.03762