Neural (Pre-Trained) Language Models

Mausam

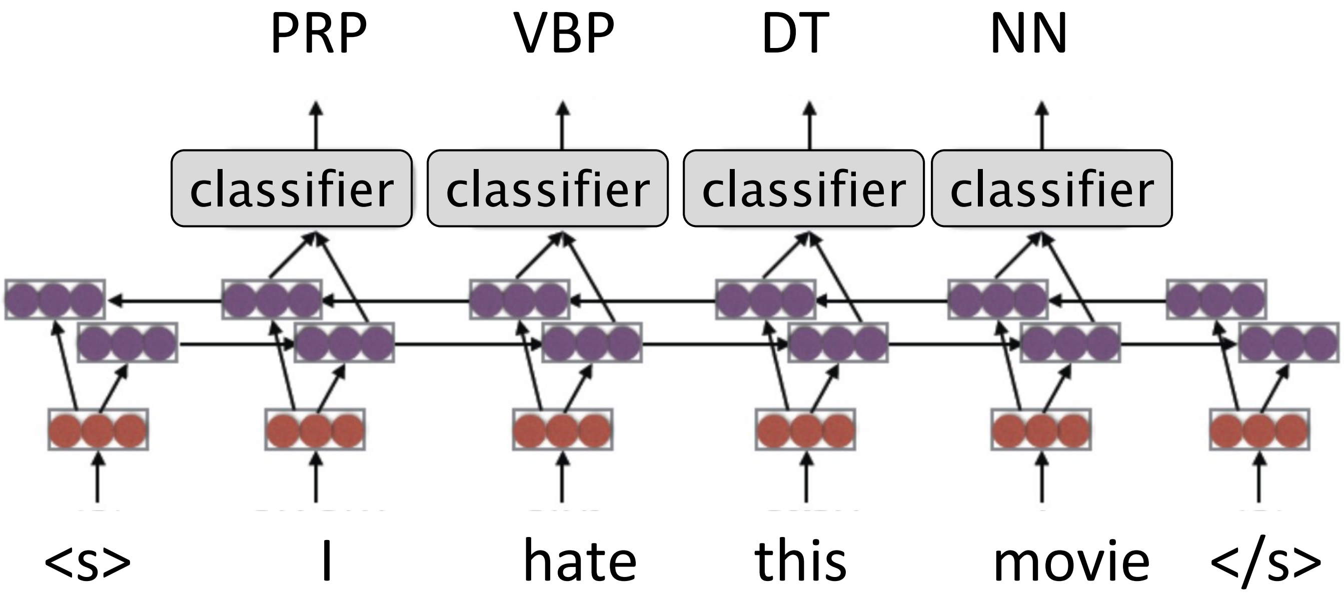
(Based on slides of Yoav Goldberg, Graham Neubig, Jay Allamar and Keshav Kolluru)

Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

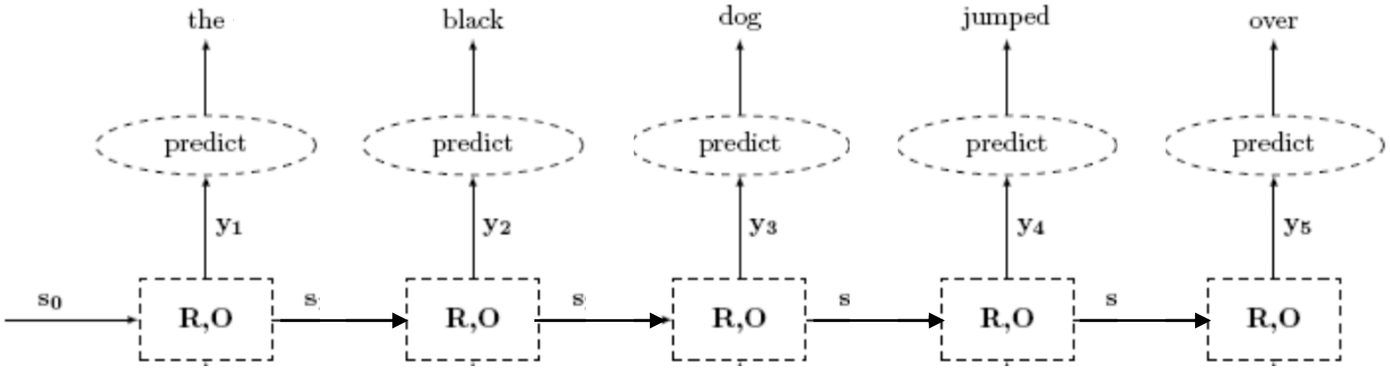
- Pre-trained Language Models: LSTMs (ELMo, GPT)
- Pre-trained Language Models: Transformers (BERT)

Sequence Labeling with (Transducer) BiLSTM



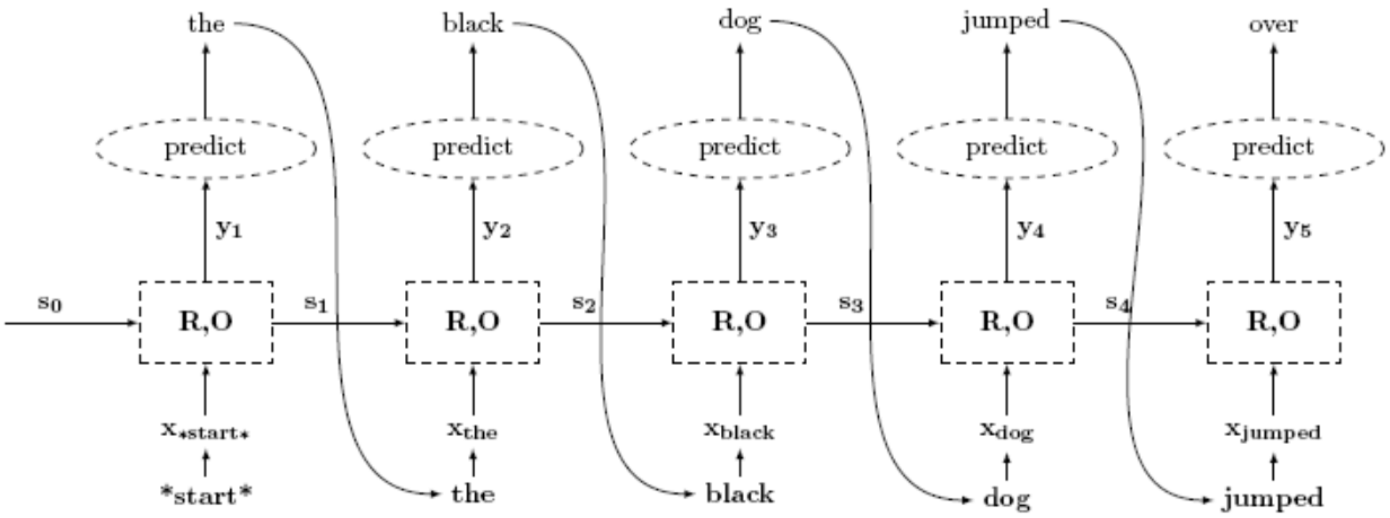
RNN Language Models

- Use LSTMs not BiLSTMs
- When does it stop?
- *Problem: the next word oblivious to exact sentence so far.*



RNN Language Models

- *Training*: an RNN Transducer.
- *Generation*: the output of step i is input to step i+1.
  - Called "Auto-regressive models"



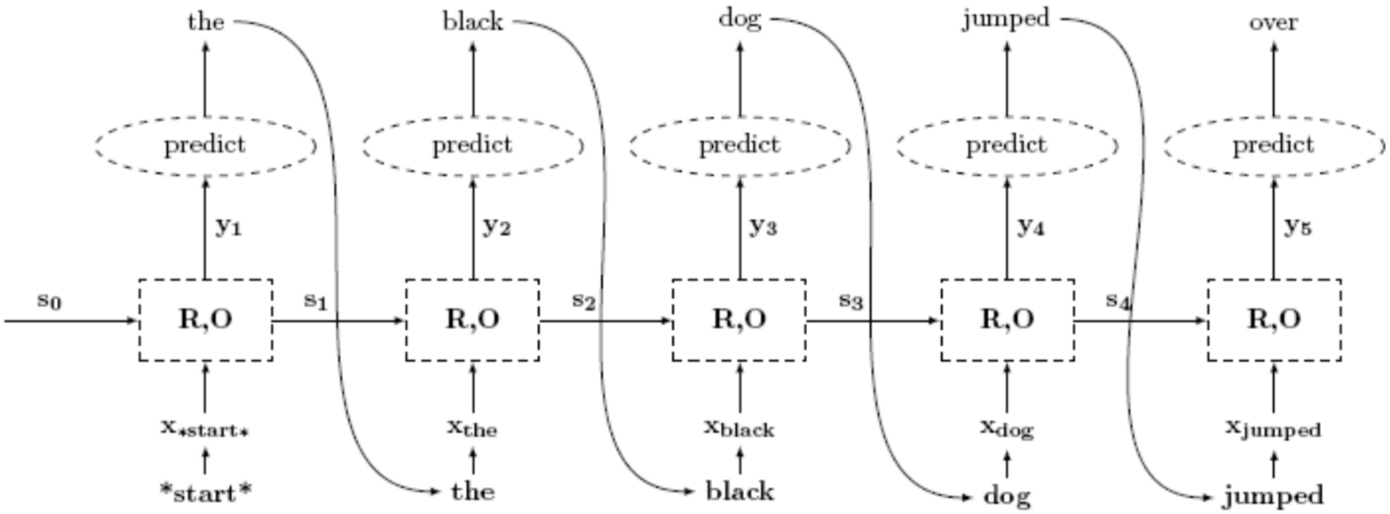
RNN Language Model for generation

- Define the probability distribution over the next item in a sequence (and hence the probability of a sequence).

$$P(w_{1:n}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_{1:2})P(w_4 \mid w_{1:3}) \ldots P(w_n \mid w_{1:n-1})$$

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(t_i = w_i | w_1, \ldots, w_{i-1})$$

RNN Language Models



$$p(t_{j+1} = k \mid \hat{t}_{1:j}) = f(\mathrm{RNN}(\mathbf{\hat{t}_{1:j}}))$$

$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1})$$

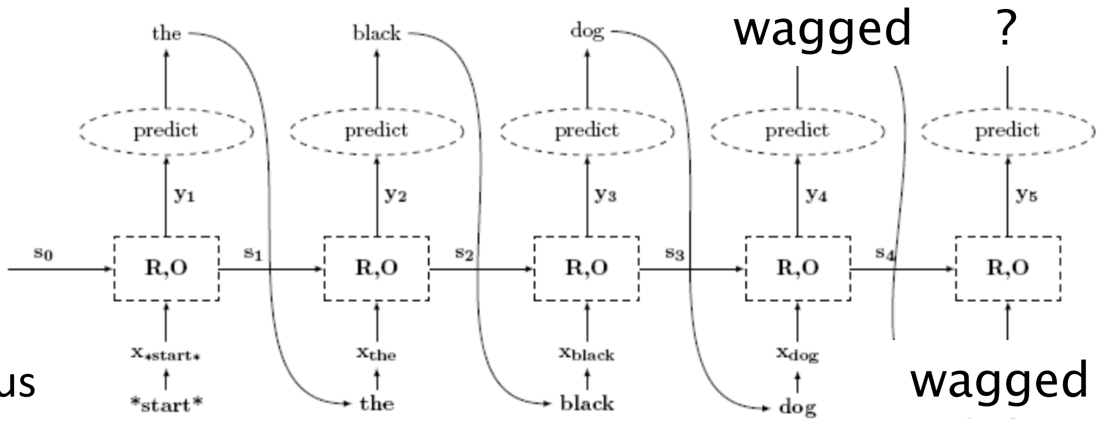
$$p(t_{j+1} = k \mid \hat{t}_{1:j}) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{\hat{t}_j}, \mathbf{s_j})$$

$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1})$$

How to Train this Model?

- Loss function: sum(cross entropy at each prediction)
- Issues with vanilla training
  - Slow convergence. Model instability. Poor skill.
- Simple idea: **Teacher Forcing**
  - Just feed in the *correct* previous tag during training
- Drawback: **Exposure bias**
  - Not exposed to mistakes during training



A Solution to Exposure Bias

- DAgger (Ross et al. 2010) ~ “scheduled sampling”
- Start with no mistakes, and then
  - gradually introduce them using annealing

Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo, GPT)
- Pre-trained Language Models: Transformers (BERT)

Conditioned Language Models

Generating sentences is nice, but what if we want to add some additional conditioning contexts?

Conditioned Language Model

- Not just generate text, generate text according to some specification

Input X	Output Y (**Text**)	Task
Structured Data	NL Description	NL Generation
English	Japanese	Translation
Document	Short Description	Summarization
Utterance	Response	Response Generation
Image	Text	Image Captioning
Speech	Transcript	Speech Recognition

RNN Language Model for **Conditioned generation**

Let's add the condition variable to the equation.

$$P(\mathrm{T}) = \prod_{i=1}^{I} \mathrm{P}(\underbrace{\mathrm{t_i}}_{\text{Next Word}} \mid \underbrace{\mathrm{t_1,\ldots,t_{i-1}}}_{\text{Context}})$$

$$P(\mathrm{T} \mid \mathrm{C}) = \prod_{j=1}^{J} \mathrm{P}(\mathrm{t_i} \mid \underbrace{\mathrm{c}}_{\text{Added Context!}}, \mathrm{t_1,\ldots,t_{i-1}})$$

(a vector)

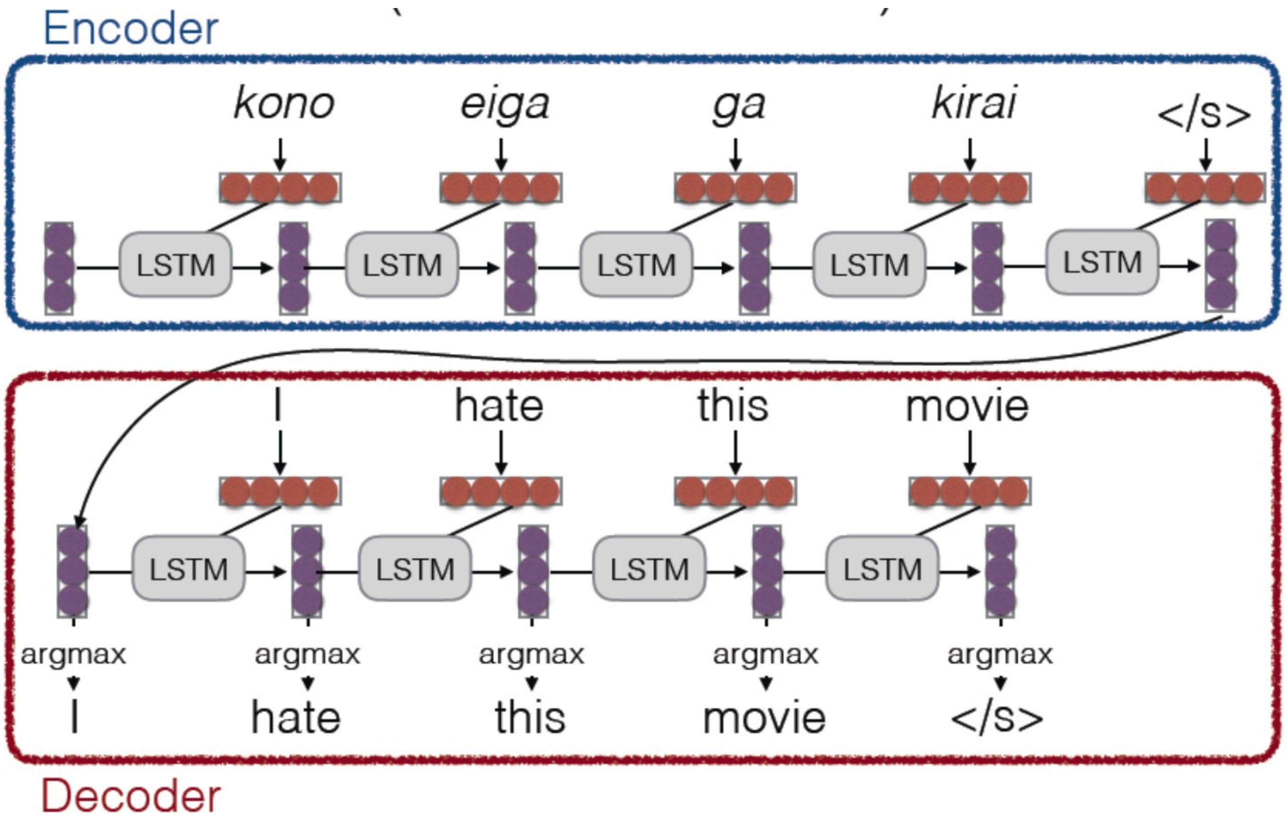
RNN Language Model for **Conditioned generation**

what if we want to condition on an entire sentence?

just encode it as a vector...

$$\mathbf{c} = \mathrm{RNN}^{\mathrm{enc}}(\mathbf{x_{1:n}})$$

A simple Sequence to Sequence conditioned generation

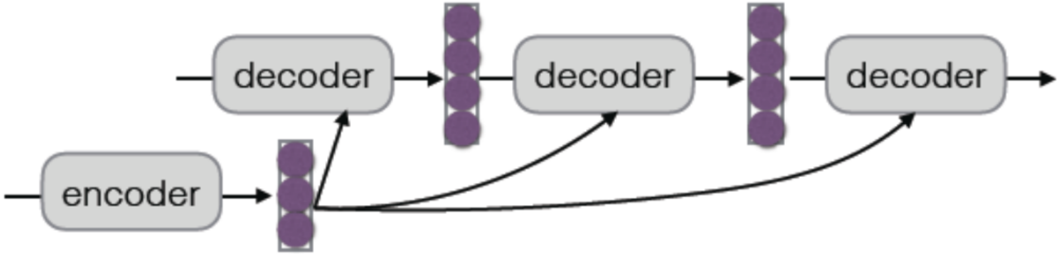


How to Pass Hidden State

- Initialize decoder w/ encoder (Sutskever et al. 2014)

- Transform (can be different dimensions)

- Input at every time step (Kalchbrenner & Blunsom 2013)



RNN Language Model for **Conditioned generation**

Let's add the condition variable to the equation.

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(\mathrm{RNN}(\mathbf{v_{1:j}}))$$

$$\mathbf{v_i} = [\mathbf{\hat{t}_i}, \mathbf{c}]$$

$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}, c)$$

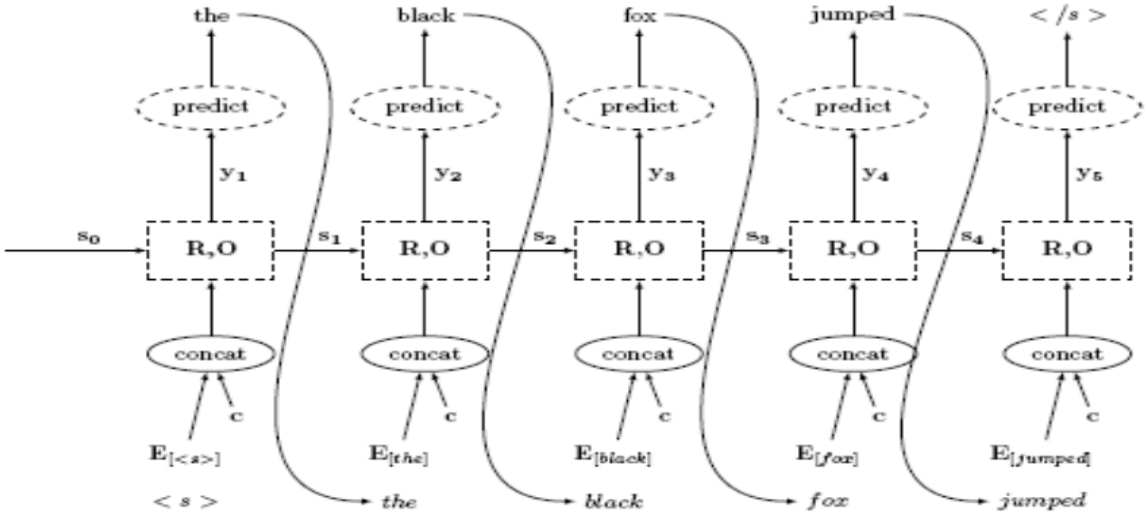
RNN Language Model
for **Conditioned generation**

Let's add the condition variable to the equation.

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(\text{RNN}(\mathbf{v_{1:j}}))$$
$$\mathbf{v_i} = [\mathbf{\hat{t}_i}; \mathbf{c}]$$
$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}, c)$$

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(O(\mathbf{s_{j+1}}))$$
$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\mathbf{\hat{t}_j}; \mathbf{c}])$$
$$\hat{t}_j \sim p(t_i \mid \hat{t}_{1:j-1}, c)$$

RNN Language Model for **Conditioned generation**



$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\hat{\mathbf{t}}_\mathbf{j}; \mathbf{c}])$$

$$\hat{t}_j \sim p(t_i \mid \hat{t}_{1:j-1}, c)$$

RNN Language Model
for **Conditioned generation**

what if we want to condition on an entire sentence?

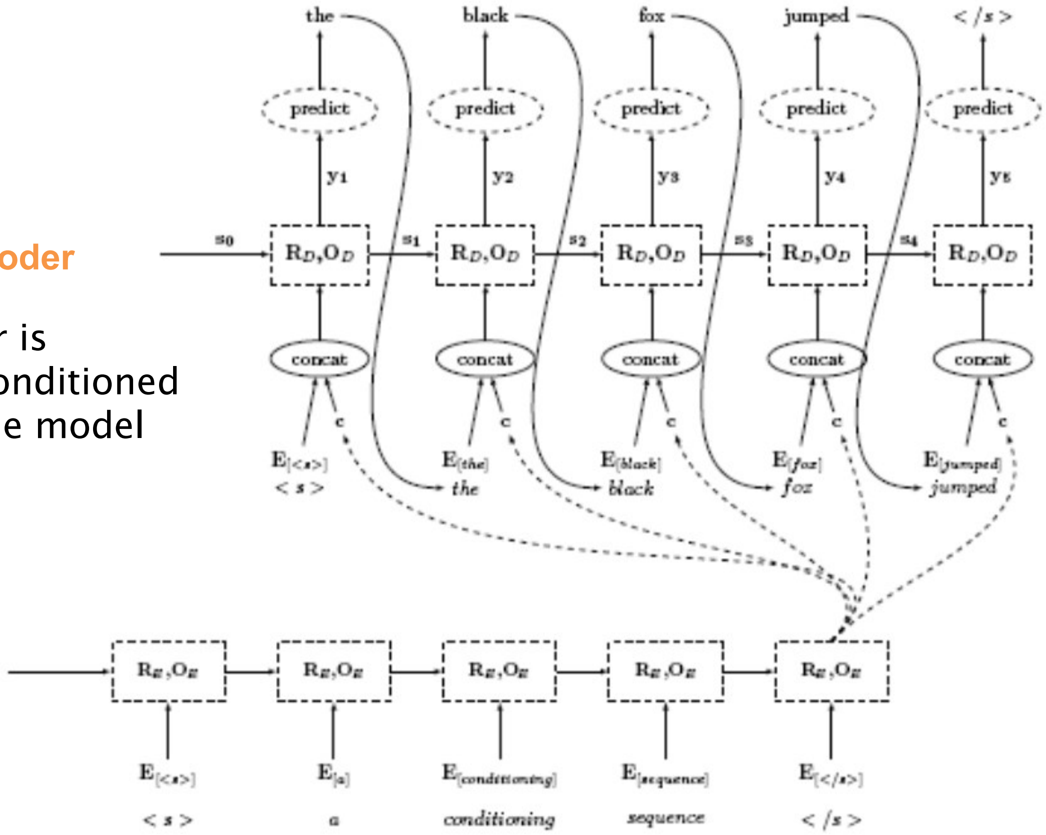
Sequence to Sequence conditioned generation

This is also called
"Encoder Decoder"
architecture.

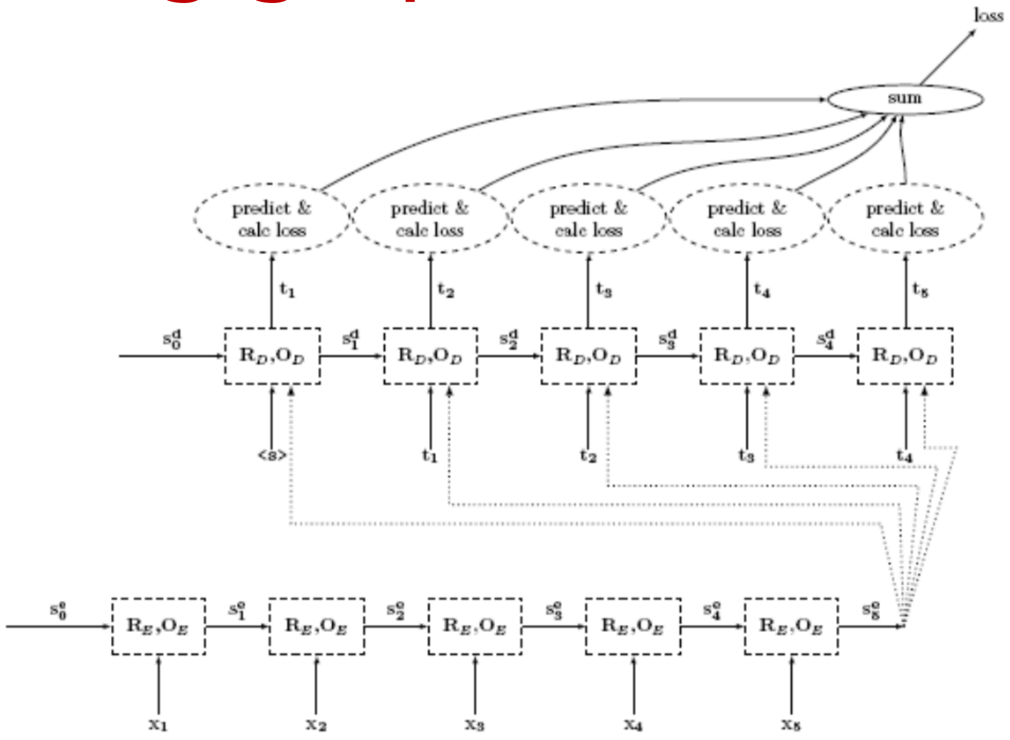
Decoder

Decoder is
just a conditioned
language model

Encoder



Sequence to Sequence training graph



The Generation Problem

We have a probability model, how do we use it to generate a sentence?

Two methods:

- **Sampling:** Try to generate a *random* sentence according to the probability distribution.
- **Argmax:** Try to generate the sentence with the *highest* probability.

Ancestral Sampling

Randomly generate words one-by-one.

```
while y_{j-1} != "</s>":
  y_j ~ P(y_j | X, y_1, ..., y_{j-1})
```

An **exact method** for sampling from P(X), no further work needed.

Greedy Search

One by one, pick the single highest-probability word

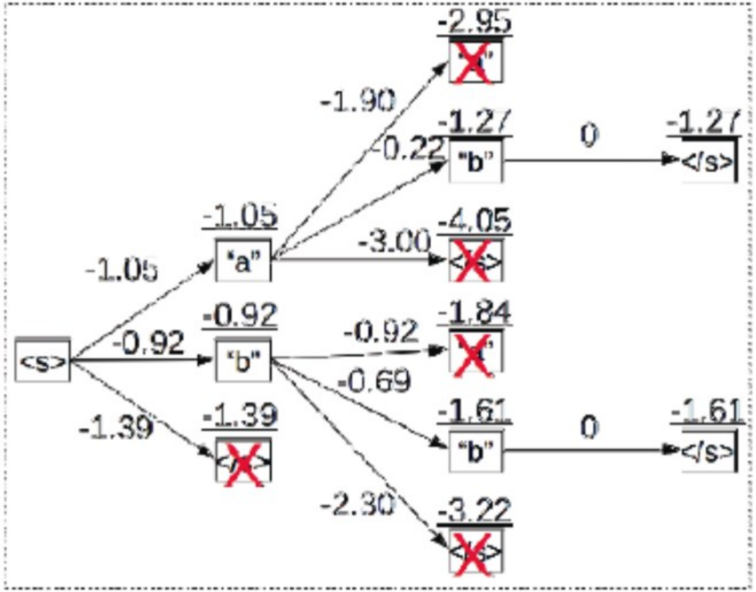
$$
\begin{aligned}
&\text{while } y_{j-1} \mathrel{!=} \text{“</s>”:} \\
&\quad y_j = \operatorname{argmax} P(y_j \mid X, y_1, \ldots, y_{j-1})
\end{aligned}
$$

Not exact, real problems:

- Will often generate the “easy” words first
- Will prefer multiple common words to one rare word

Beam Search

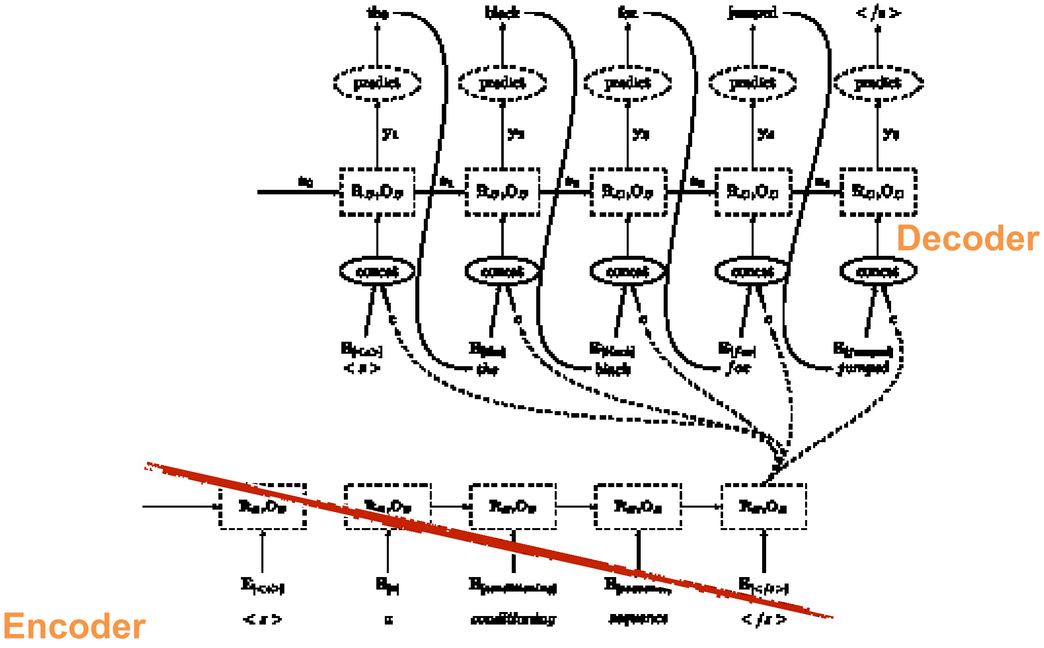
Instead of picking one high-probability word, maintain several paths



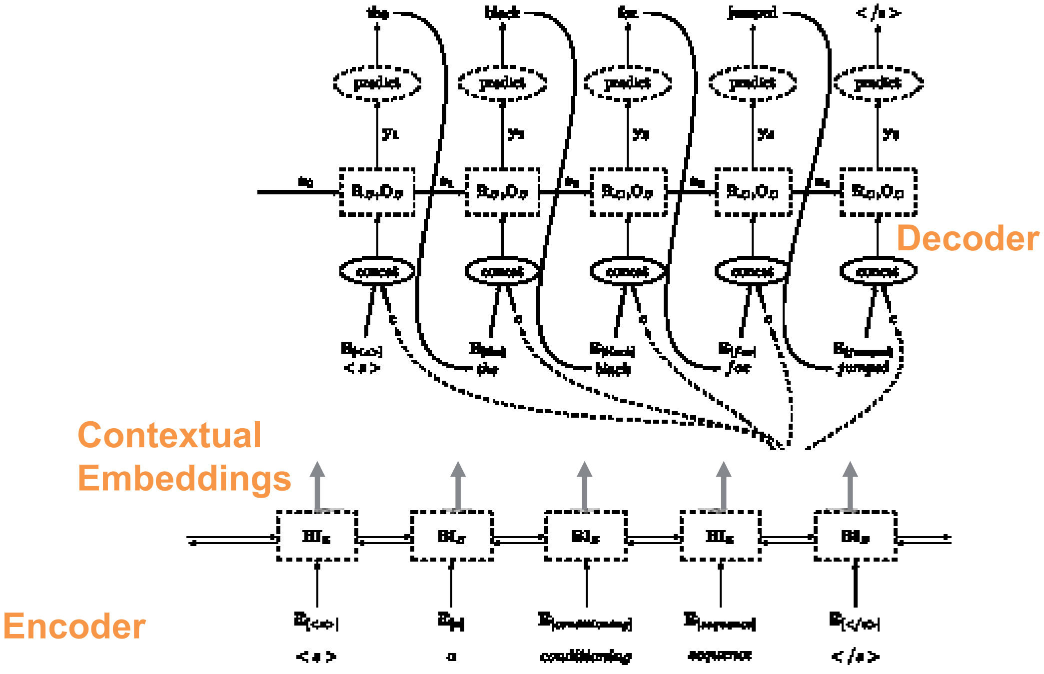
Attention

- Instead of the encoder producing a single vector for the sentence, it will produce a one vector **for each word**.

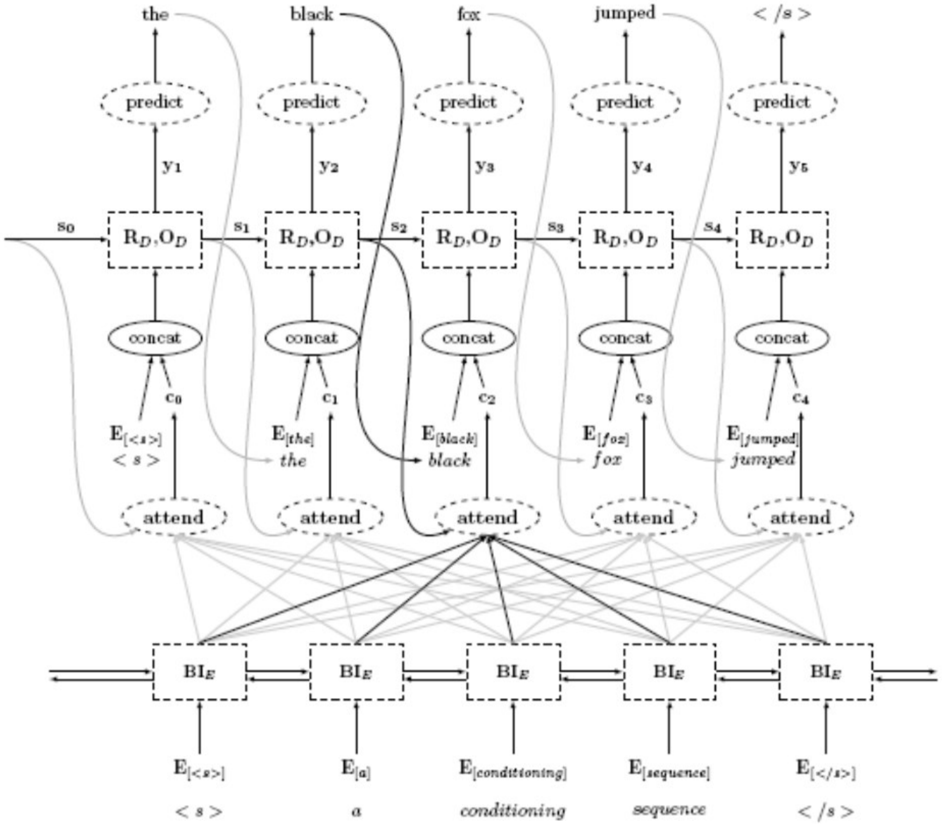
Sequence to Sequence conditioned generation



Sequence to Sequence conditioned generation



encoder-decoder with attention



encoder-decoder with attention

- Encoder encodes a sequence of vectors, $c_1,...,c_n$
- At each decoding stage, an MLP assigns a relevance score to each Encoder vector.
- The relevance score is based on c_i and the state s_j
- Weighted-sum (based on relevance) is used to produce the conditioning context for decoder step j.

encoder-decoder with attention

- Decoder "pays attention" to different parts of the encoded sequence at each stage.
- The attention mechanism is "soft" -- it is a mixture of encoder states.

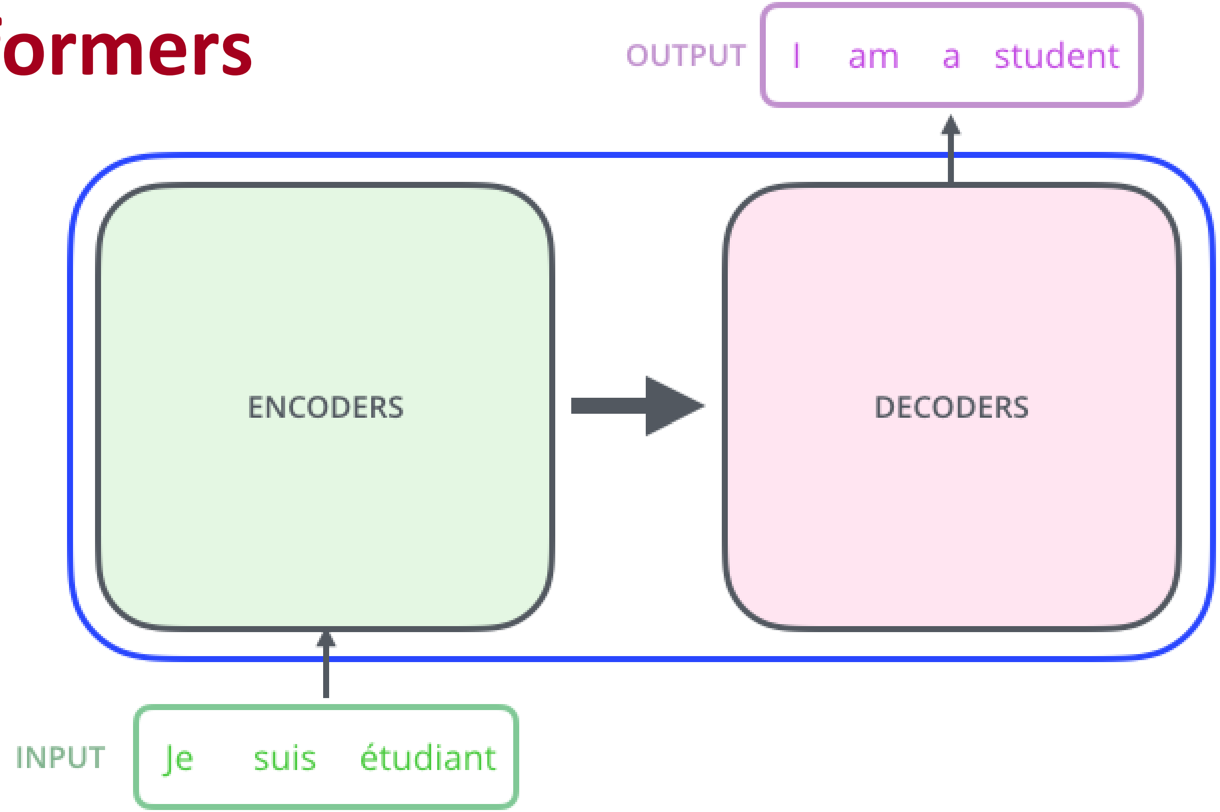
- The encoder acts as a read-only memory for the decoder
- The decoder chooses what to read at each stage

Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo, GPT)
- Pre-trained Language Models: Transformers (BERT)

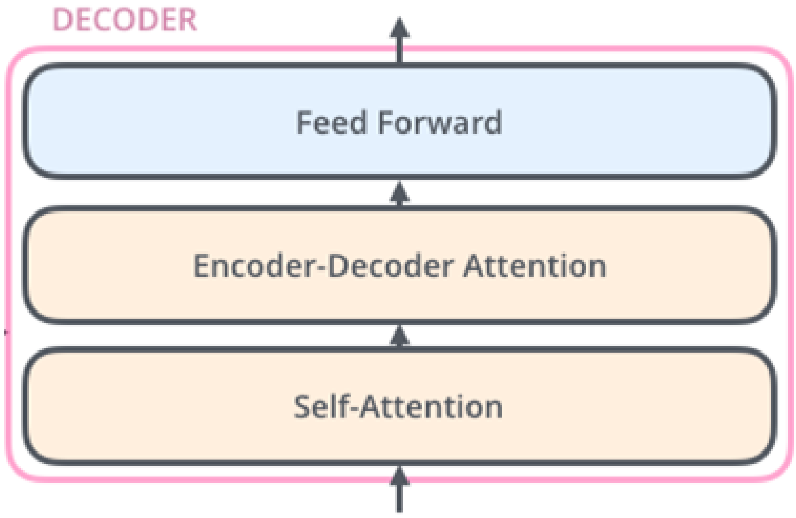
Transformers



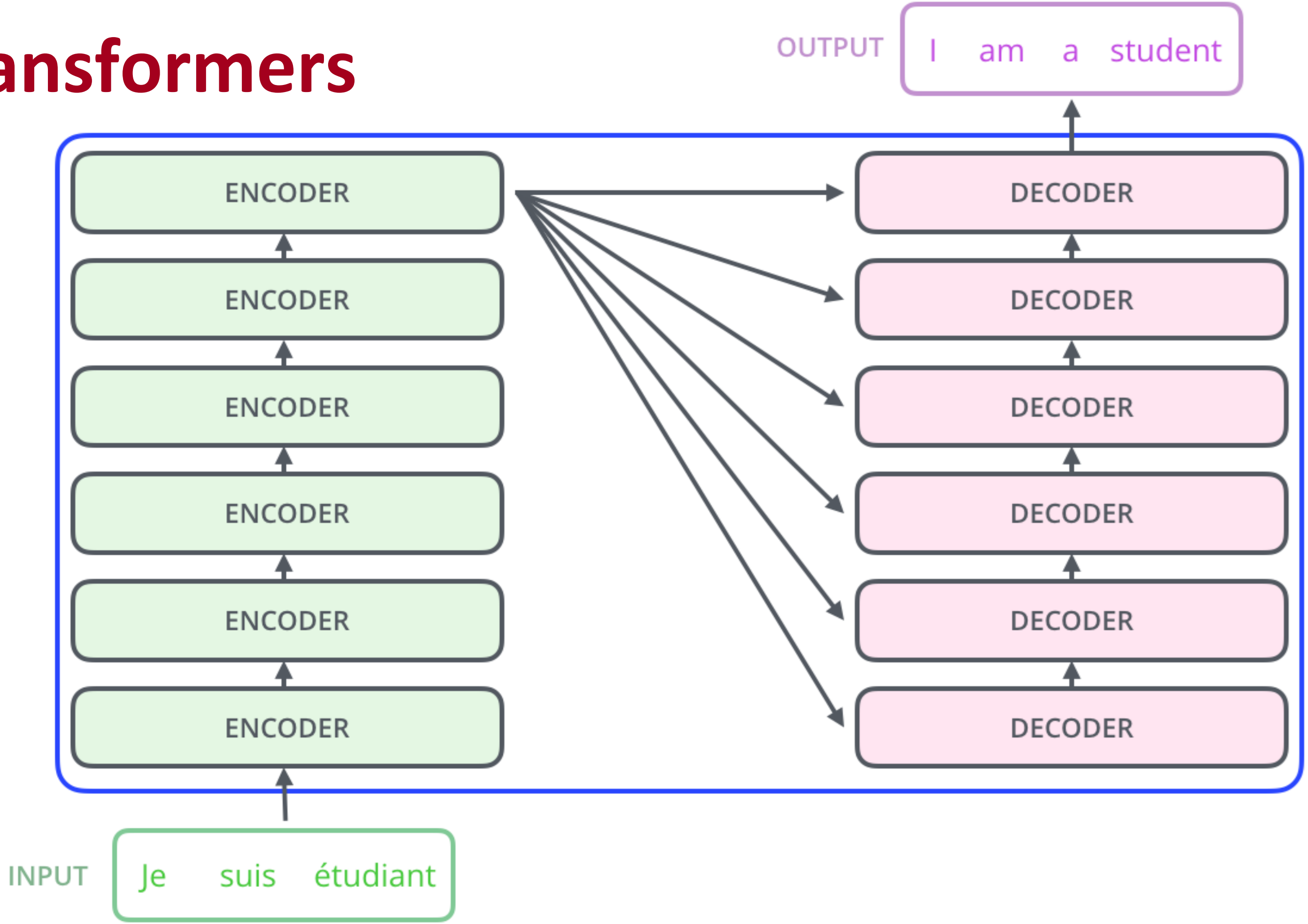
Decoders

Two key differences from encoder:

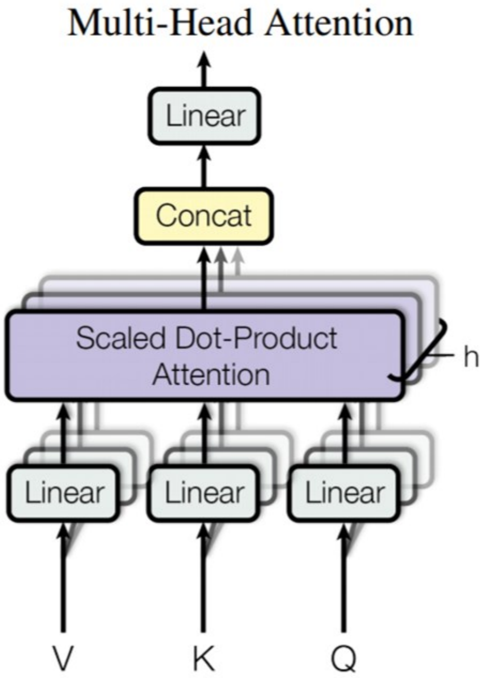
- Self-attention only on words generated uptil now, not on whole sentence.
- Additional encoder-decoder attention layer where keys, values come from last encoder layer.

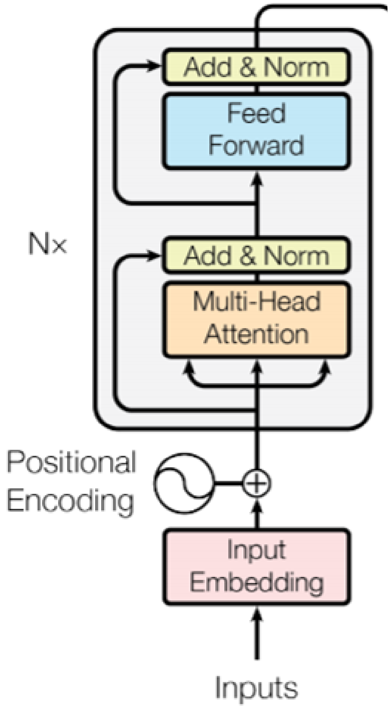


Transformers

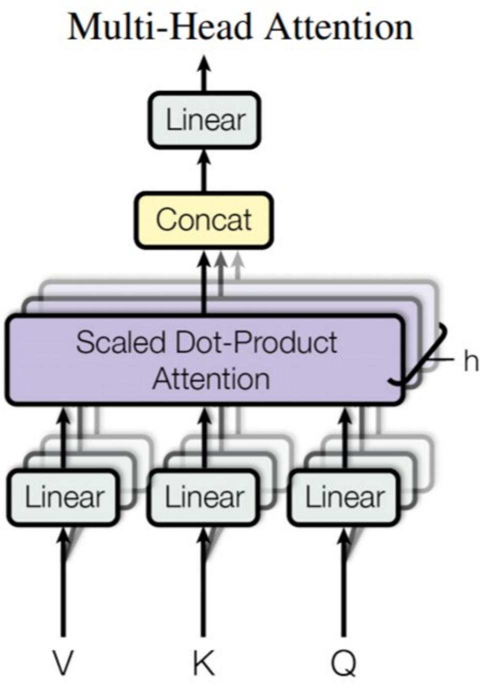


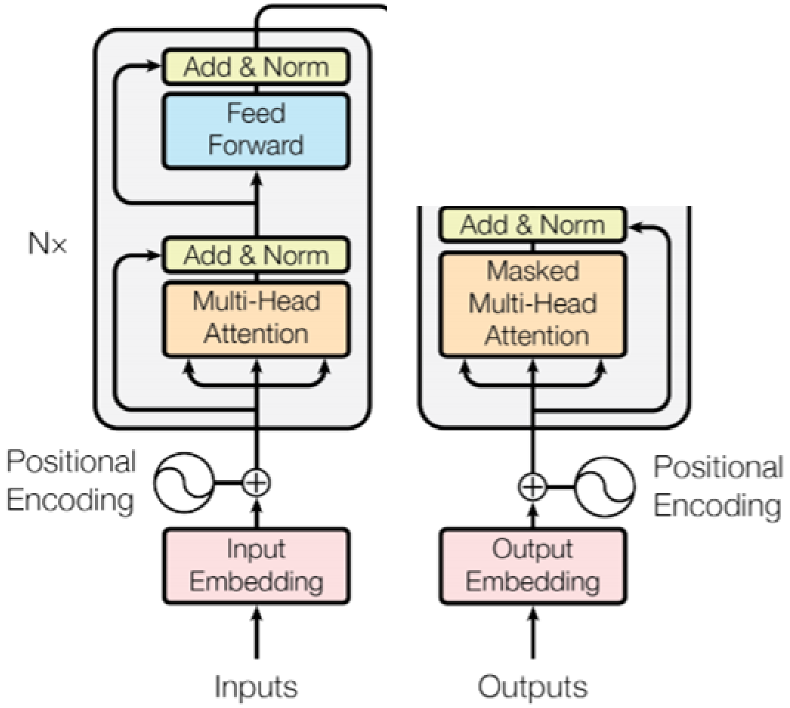
Full architecture with Attention reference



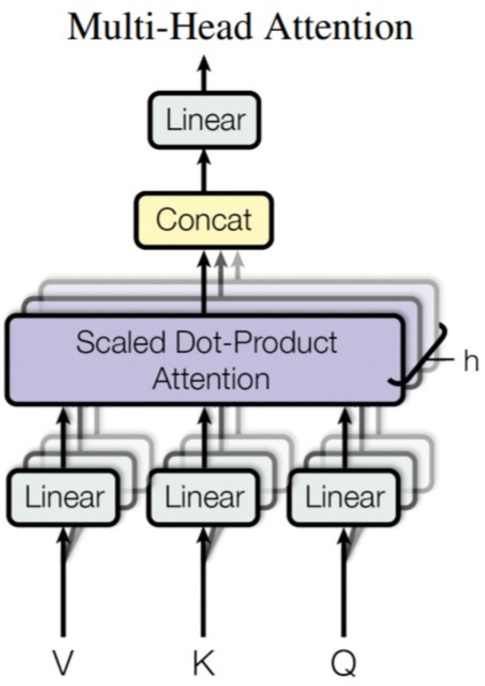


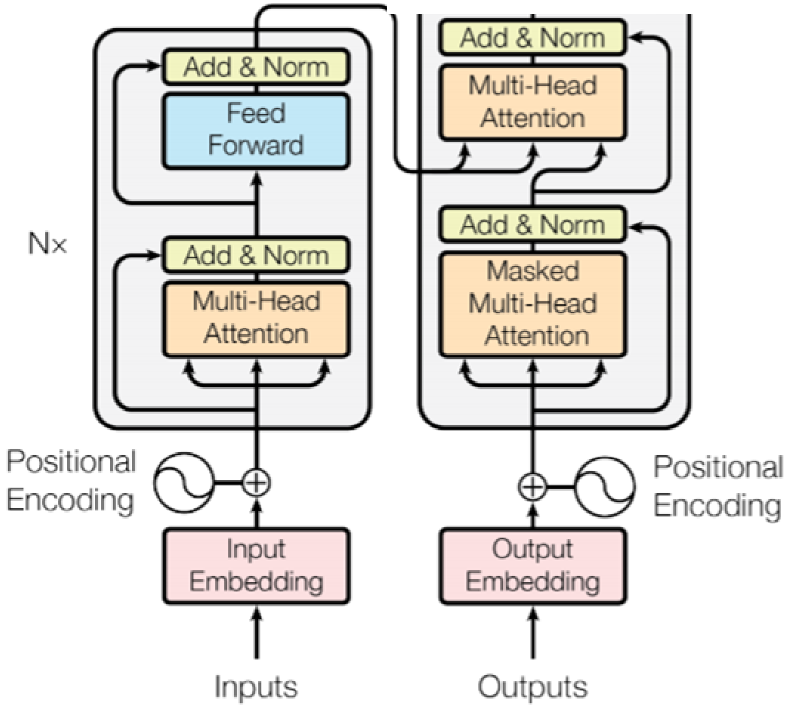
Full architecture with Attention reference



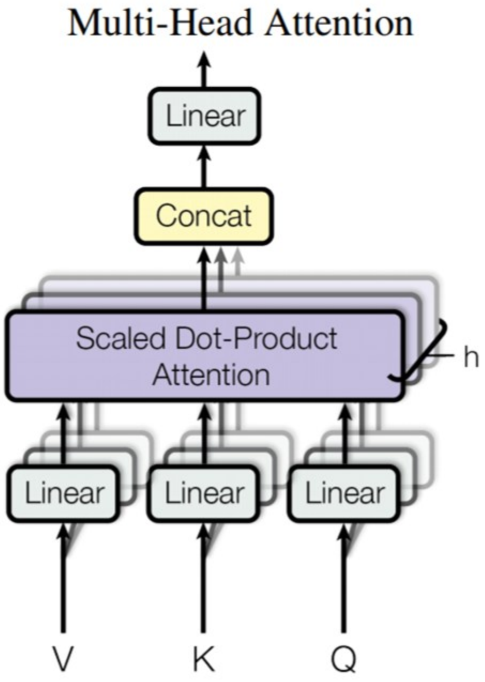


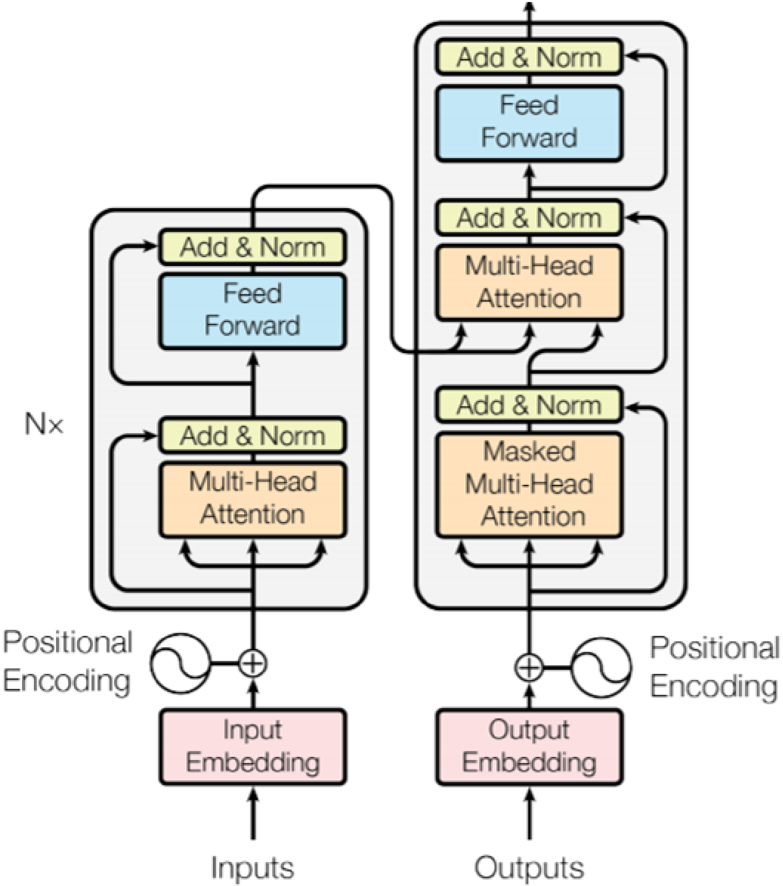
Full architecture with Attention reference



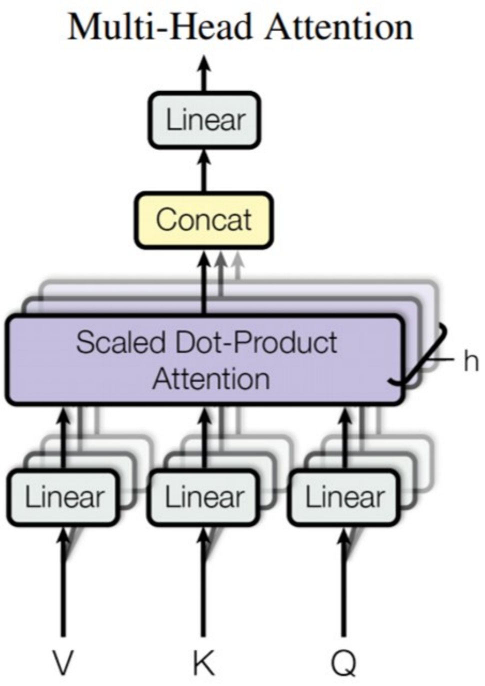


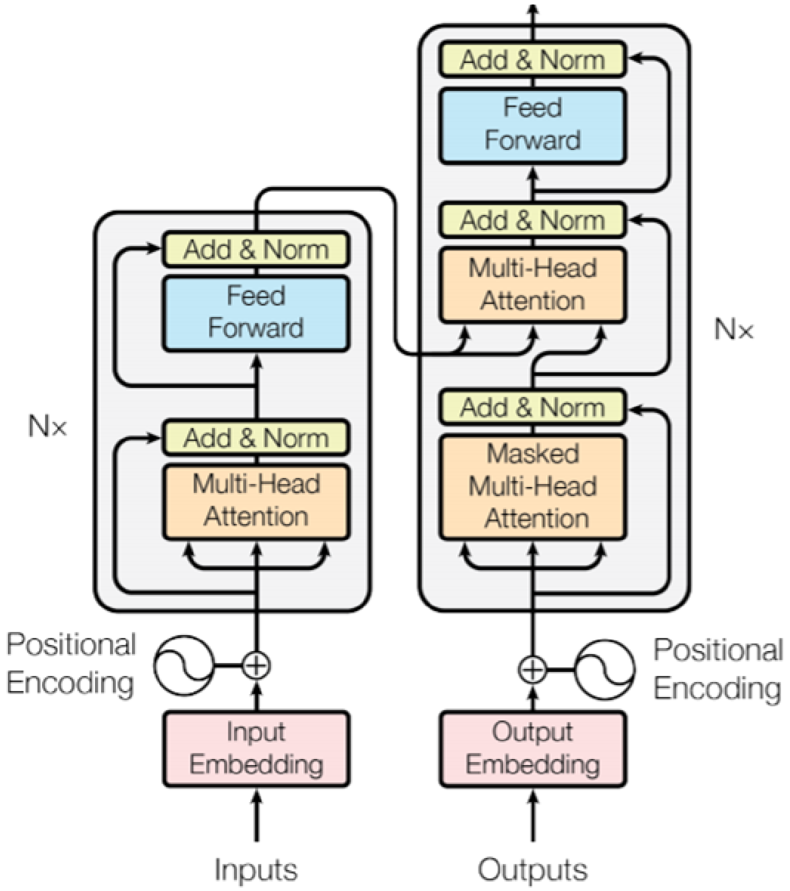
Full architecture with Attention reference



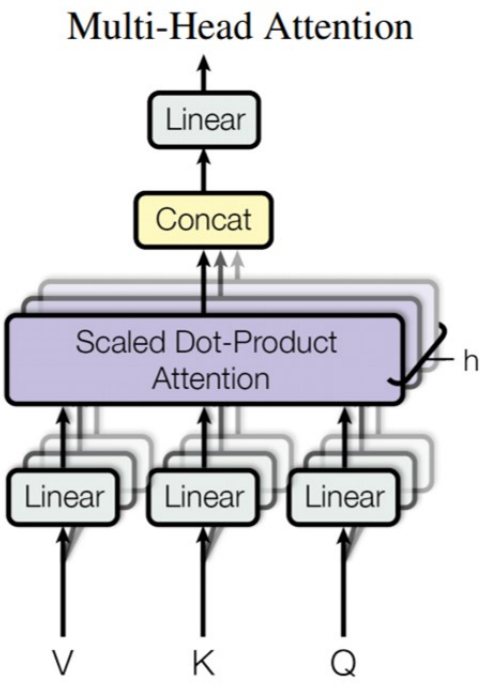


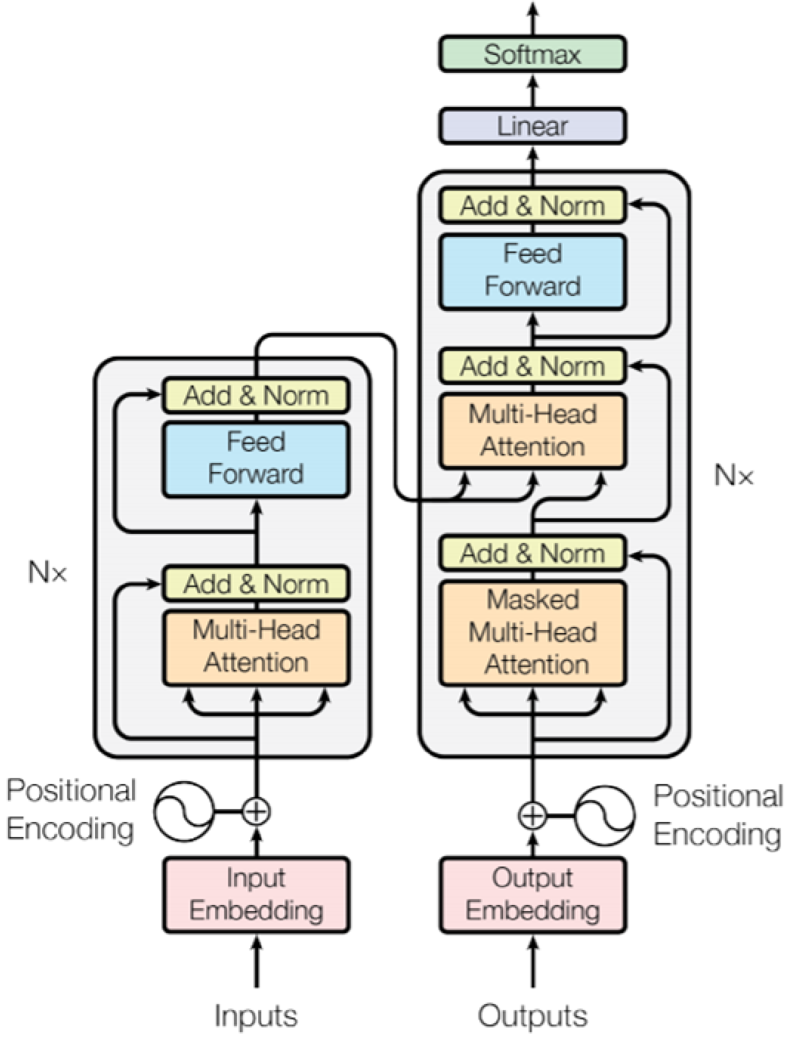
Full architecture with Attention reference





Full architecture with Attention reference





Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo, GPT)
- Pre-trained Language Models: Transformers (BERT)

Pretraining

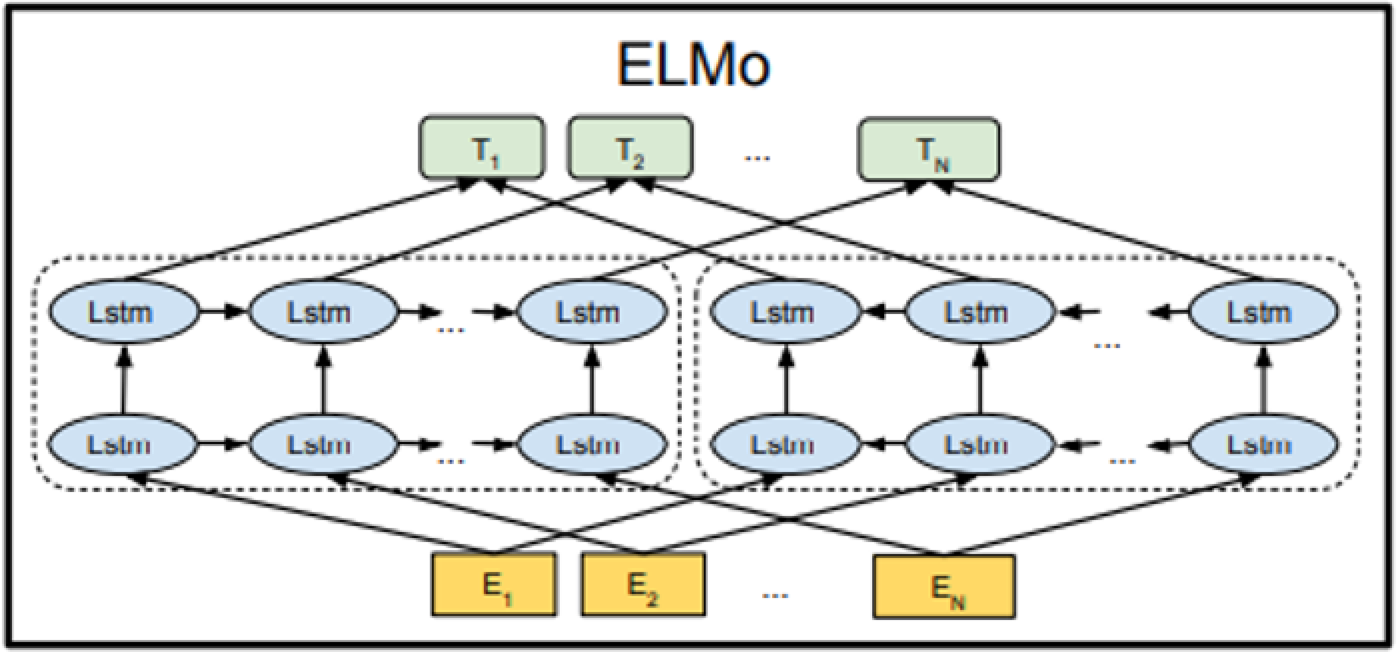
- In NLP, we are interested in solving a variety of end tasks - Question Answering, Search, etc.
- One approach - train neural models from scratch
- Issue - This involves two things
  - Modelling of Syntax and Semantics of the language
  - Modelling of the end-task
- Pretraining: Learns the modelling of syntax and semantics - through another task
- So the model can focus exclusively on modelling of end-task

Pretraining

- Which base task to choose:
  - Must have abundant data available
  - Must require learning of syntax and semantics

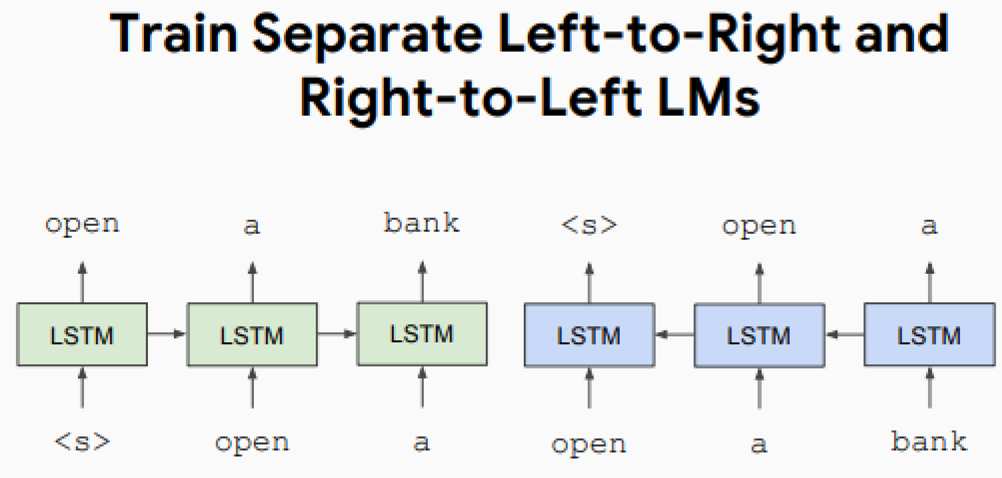
- Solution: Language Modelling
  - Does not require human annotated labels - abundance of sentences
  - Requires understanding of both syntax and semantics to predict the next word in sentence

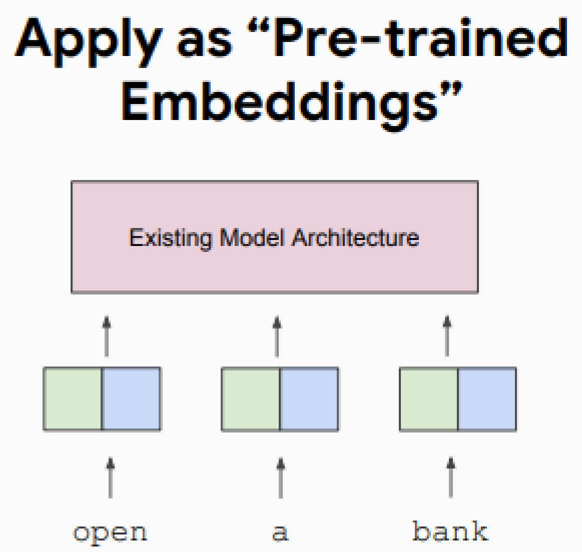
Model 1: ELMo (two LSTMs)



ELMo (Contextualized Embeddings)

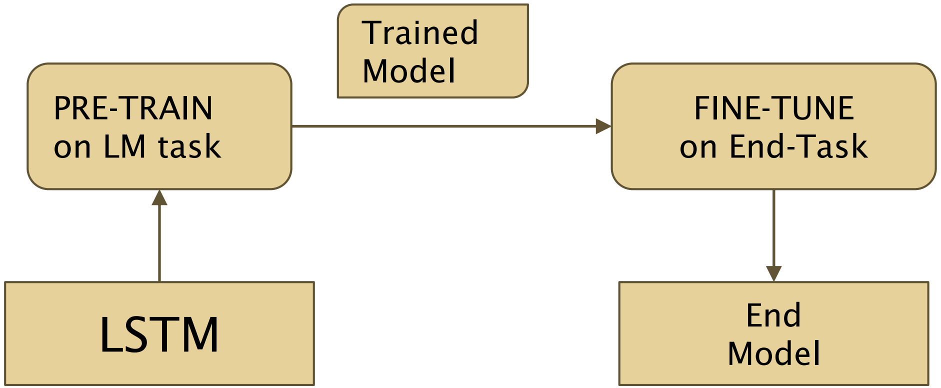
- Bidirectional language modelling: separate forward and backward LSTMs
- Issue: Both LSTMs are not coupled with one another





Reference: https://nlp.stanford.edu/seminar/details/jdevlin.pdf

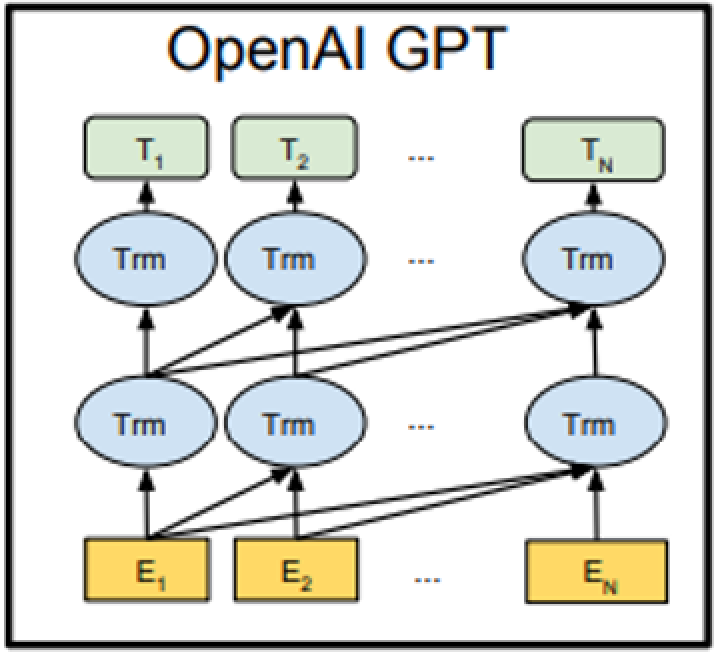
Universal Language Model Fine-tuning for Text Classification



- Uses the same architecture for both pretraining and finetuning
- ELMo is added as additional component to existing task-specific architectures

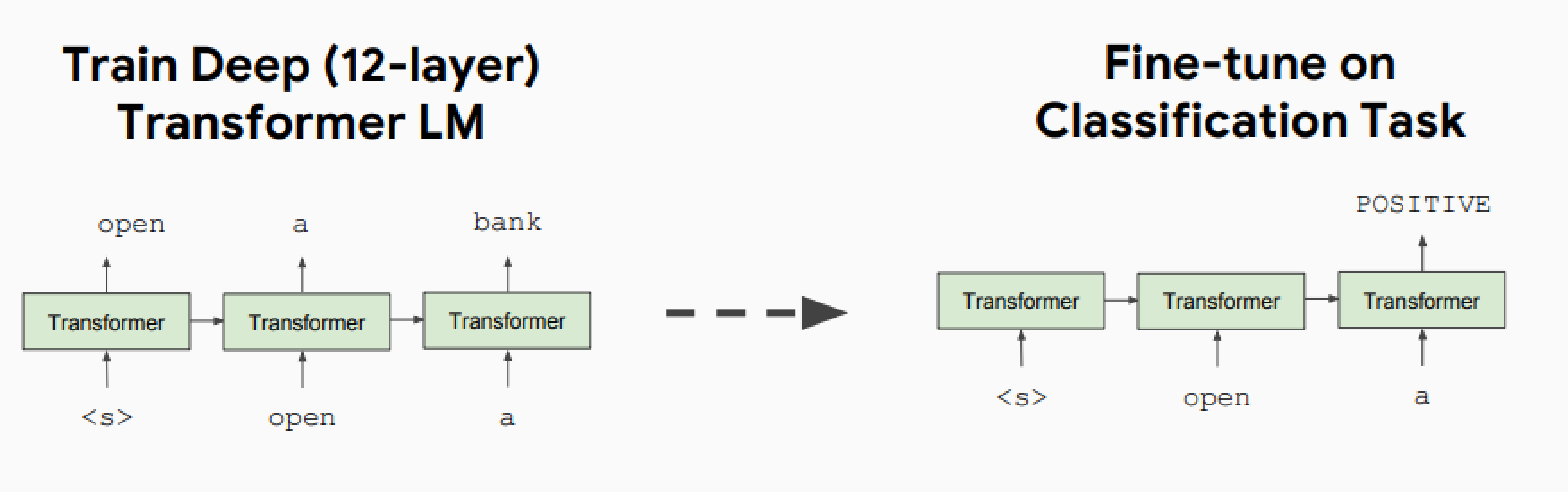
- Introduced the Pretrain-Finetune paradigm for NLP
- Similar to pretraining ResNet on ImageNet and finetune on specific tasks
- Pretrained using Language modelling task
- Finetuned on End-Task (such as Sentiment Analysis)

Model 2: Generative Pre-Training (Transformers)



Generative Pre-Training

- GPT - Uses Transformer decoder instead of LSTM for Language Modeling
- GPT-2 - Trained on larger corpus of text (40 GB) Model size:1.5 B parameters
- Can generate text given initial prompt - “unicorn” story, economist interview



Unicorn Story

SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow. Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo, GPT)
- Pre-trained Language Models: Transformers (BERT)

Model 3: Masked language modeling (BERT)

- GPT/language model task is unidirectional.
- Tasks like classification - we already know all the words –
- Bidirectional context required for end tasks:
  - using unidirectional model is sub-optimal

- **Solution:** Mask out *k*% of the input words, and then predict the masked words
  - We always use $k = 15\%$

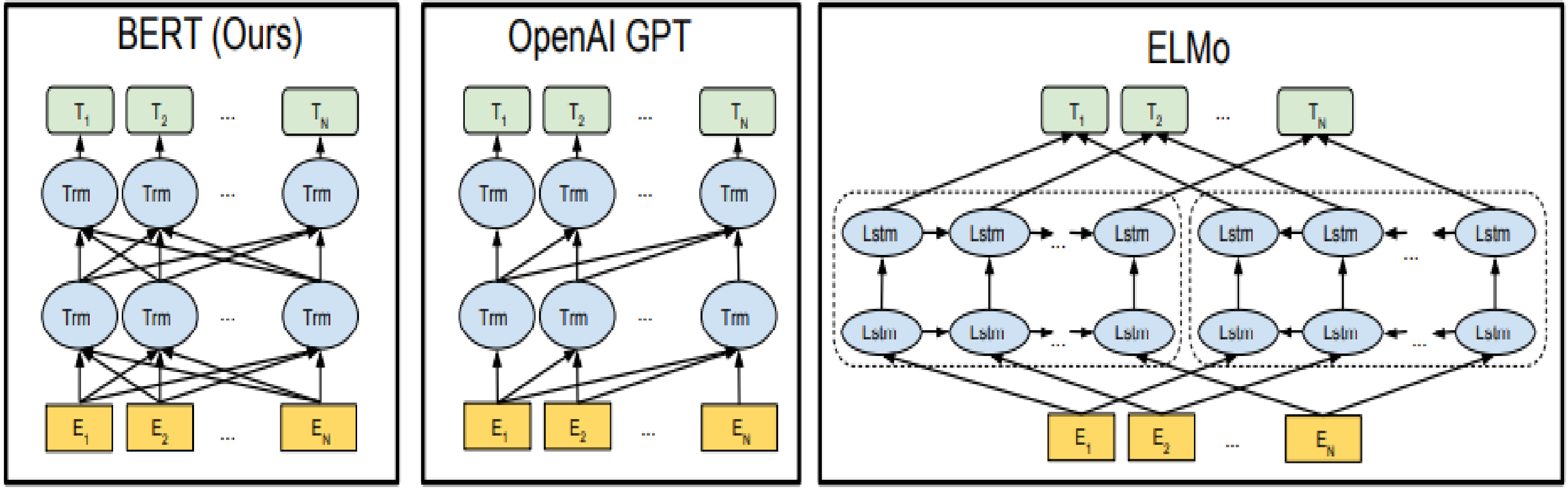
```
                    store          gallon
                      ↑               ↑
the man went to the [MASK] to buy a [MASK] of milk
```

Solution 2: Masked Language Modelling

- Issue with Language modelling - Unidirectional
- Cannot train model on bidirectional context - required for many end tasks
- Solution 2: Masked Language Modelling
    - Randomly mask a word in the sentence
    - Train the model to predict it

*Image Credits: [1]

BERT vs. OpenAI-GPT vs. ELMo



Bidirectional

Unidirectional

De-coupled Bidirectionality

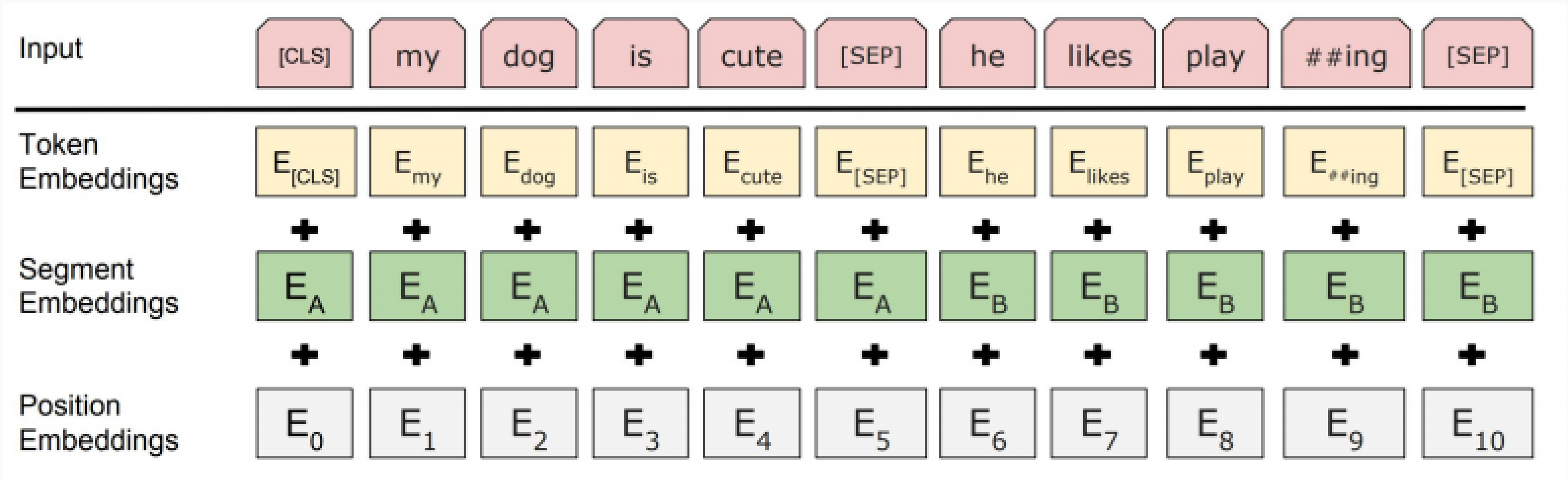
Word-Piece tokenizer

- Middle ground between character level and word level representations
- tweeting → tweet + ##ing
- xanax → xa + ##nax
- Technique originally taken from paper for Japanese and Korean languages from a speech conference

- Given a training corpus and a number of desired tokens D, the optimization problem is to select D wordpieces such that the resulting corpus is minimal in the number of wordpieces when segmented according to the chosen wordpiece model.

Schuster, Mike, and Kaisuke Nakajima. "Japanese and korean voice search." *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012.

Input Representation



- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings

Practical Tips

- Proper modelling of input for BERT is extremely important
    - Question Answering: [CLS] Query [SEP] Passage [SEP]
    - Natural Language Inference: [CLS] Sent1 [SEP] Sent2 [SEP]
    - BERT cannot be used as a general purpose sentence embedder

- Maximum input length is limited to 512. Truncation strategies have to be adopted

- BERT-Large model requires random restarts to work
- Always PRE-TRAIN, on related task - will improve accuracy
- Highly optimized for TPUs, not so much for GPUs

Small Hyperparameter search

- Because of using a pre-trained model - we can’t really change the model architecture any more
- Number of hyper-parameters are actually few:
  - Batch Size: 16, 32
  - Learning Rate: 3e-6, 1e-5, 3e-5, 5e-5
  - Number of epochs to run
- Compare to LSTMs where we need to decide number of layers, the optimizer, the hidden size, the embedding size, etc…
- This greatly simplifies using the model

Implementation for fine-tuning

- Using BERT requires 3 modules
    - Tokenization, Model and Optimizer
- Originally developed in Tensorflow
- HuggingFace ported it to Pytorch and to-date remains the most popular way of using BERT (18K stars)
- Tensorflow 2.0 also has a very compact way of using it - from TensorflowHub
    - But fewer people use it, so support is low

- Keshav’s choice - use HuggingFace BERT API with Pytorch-Lightning
    - Lightning provides a Keras-like API for Pytorch

Self-Supervised Learning

Yann LeCun shared a photo.

30 April 2019 ·

I now call it "self-supervised learning", because "unsupervised" is both a loaded and confusing term.

In self-supervised learning, the system learns to predict part of its input from other parts of it input. In other words a portion of the input is used as a supervisory signal to a predictor fed with the remaining portion of the input.

Self-supervised learning uses way more supervisory signals than supervised learning, and enormously more than reinforcement learning. That's why calling it "unsupervised" is totally misleading. That's also why more knowledge about the structure of the world can be learned through self-supervised learning than from the other two paradigms: the data is unlimited, and amount of feedback provided by each example is huge.

Roberta: A Robustly Optimized BERT Pretraining Approach

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of $\text{BERT}_{\text{LARGE}}$. Results for $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.