Transformers for Natural Language Processing

Concepts, Examples & Tutorials

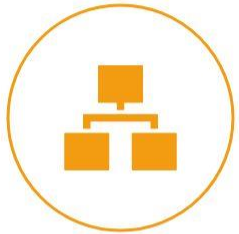
Dipanjan (DJ) Sarkar

Session Agenda

Introduction

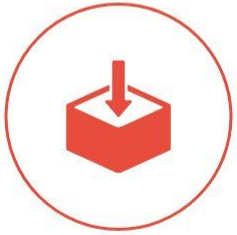
Transfer Learning Brief

Deep Transfer Learning for NLP

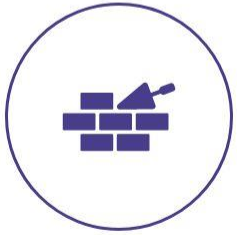


Deep Learning Architecture Fundamentals

Motivation for Transformers

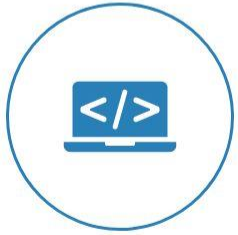


Transformer Essentials



Transformer Model Architectures

Understanding BERT & DistilBERT



Hands-on Tutorials


Introduction

About Me

Dipanjan Sarkar

Data Science Lead, Author, Google Developer Expert - ML

APPLIED MATERIALS®

Springboard

Experts

Machine Learning

Slide Deck and Hands-on Examples

https://bit.ly/transformers_ds


Transfer Learning Brief

Transfer Learning

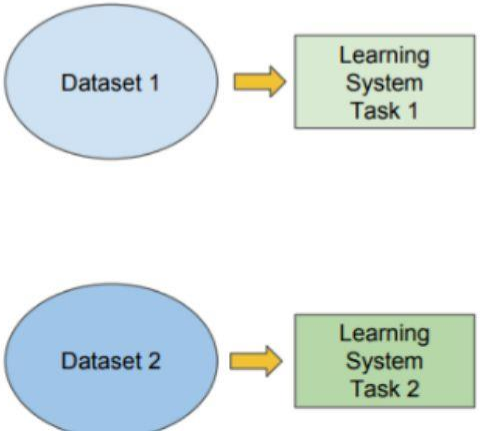
The ability to utilize knowledge learnt in prior tasks to new and novel tasks

- Know how to ride a motorbike ➦ Learn how to ride a car
- Know how to play classic piano ➦ Learn how to play jazz piano
- Know math and statistics ➦ Learn machine learning

Traditional ML vs. Transfer Learning

Traditional ML

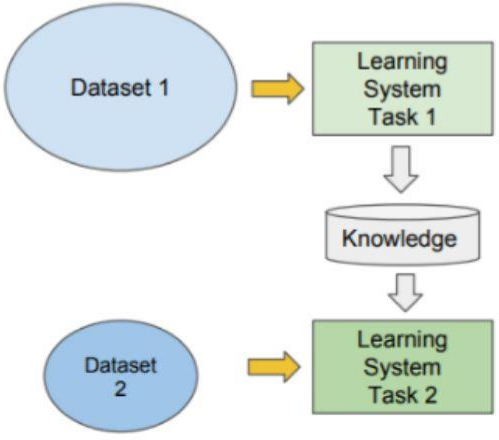
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



Defining Transfer Learning

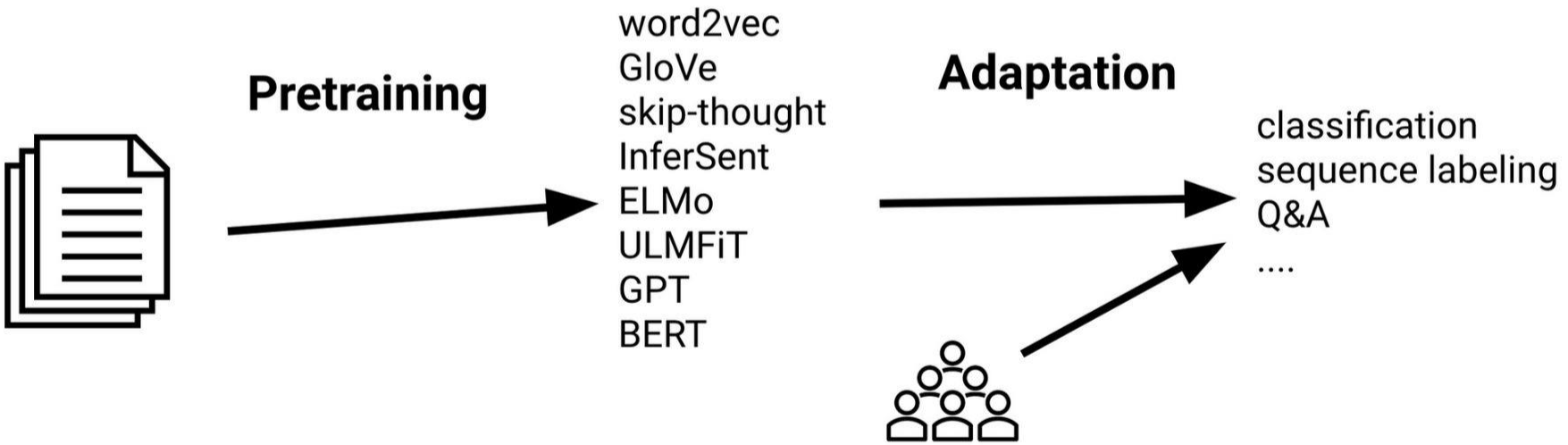
Given a source domain $\mathcal{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in $\mathcal{D}_T$ with the information gained from $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.

RUDER, 2017


Deep Transfer Learning for NLP

Sequential Transfer Learning for NLP

Sequential transfer learning has led to the biggest improvements so far in the field of NLP



RUDER, 2019

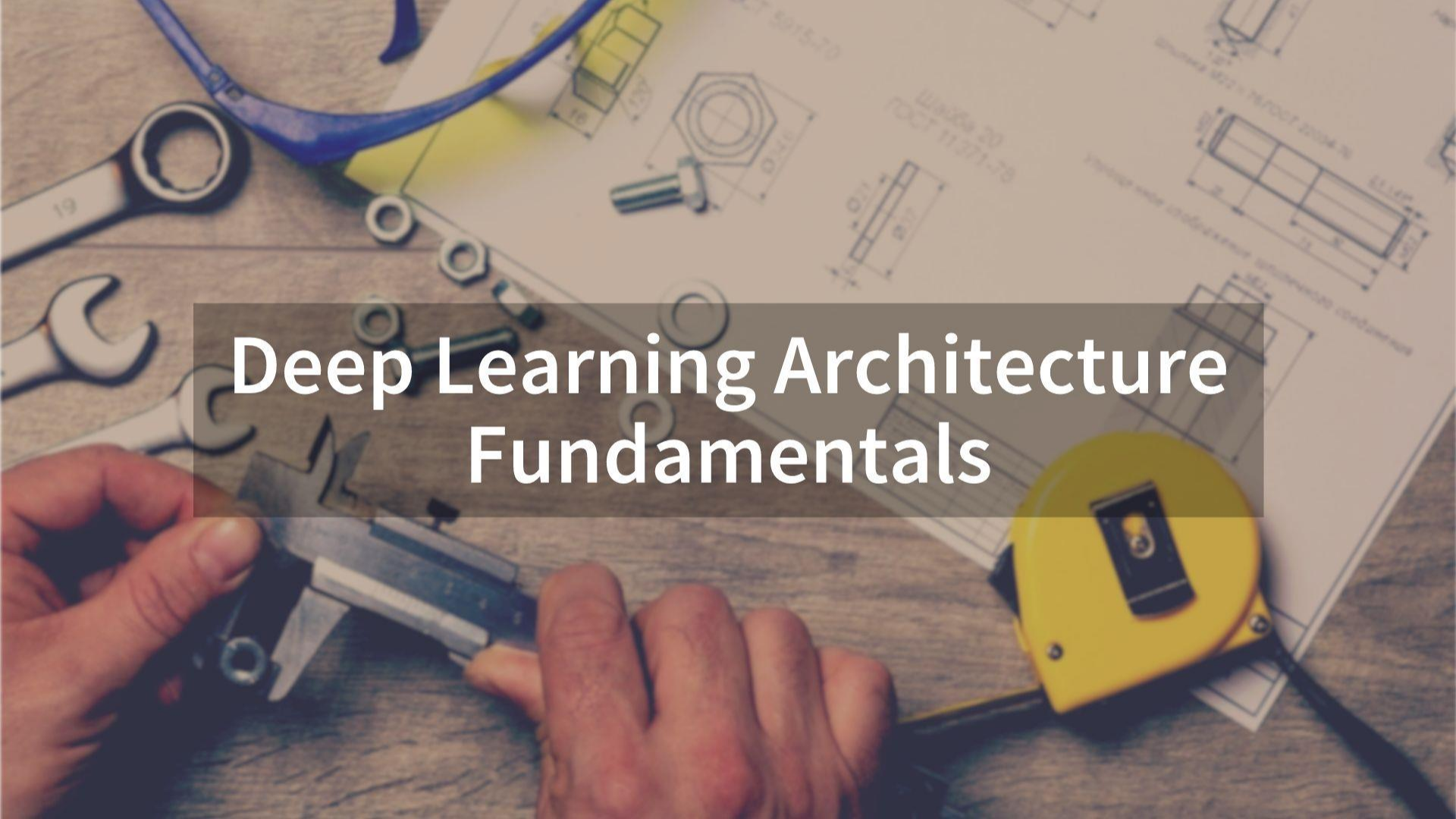
Sequential Transfer Learning for NLP

1 Pre-training

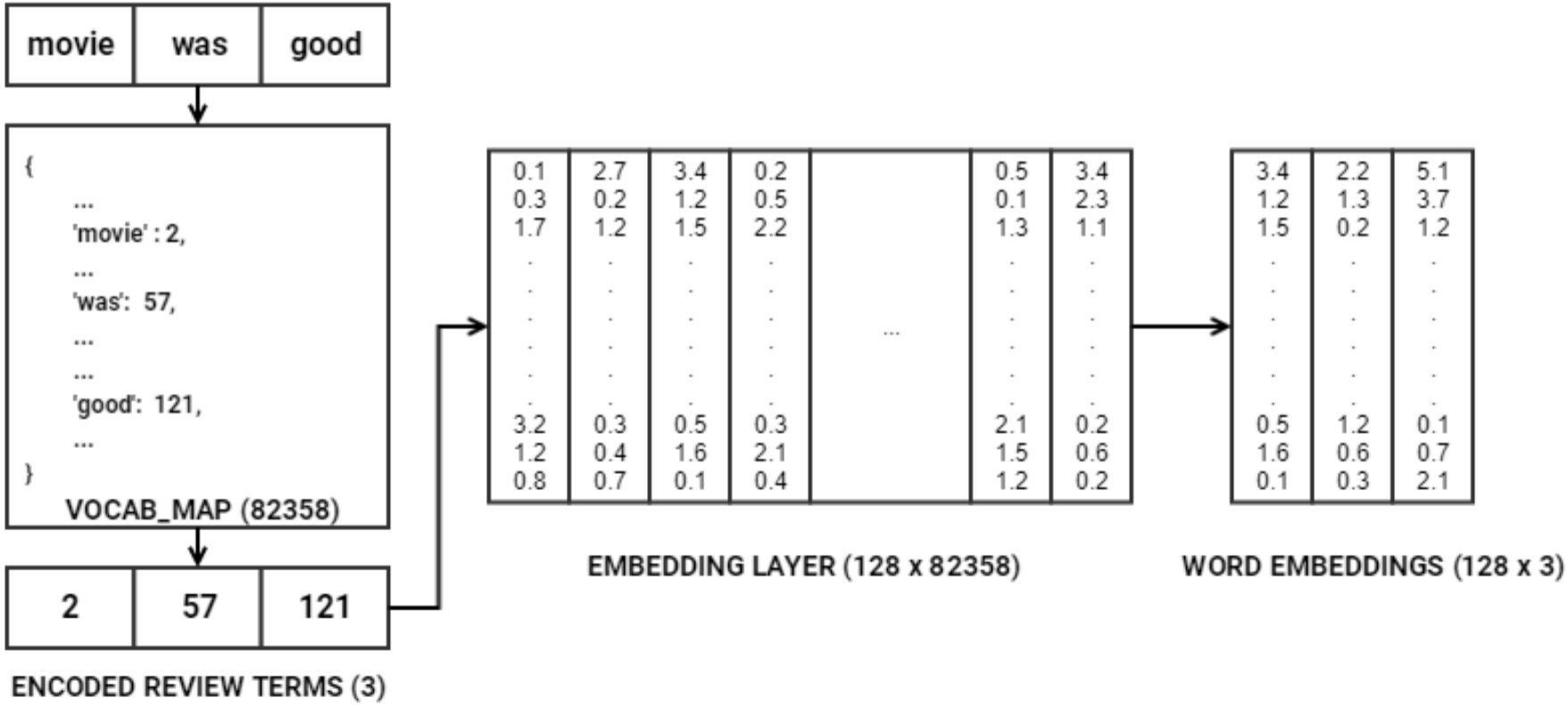
- Word and context-based representations (word2vec, fasttext, elmo)
- Simple Language Models (RNN, LSTMs, Bi-LSTMs)
- Generalized Language Modeling (ULMFiT, GPT, BERT)

2 Adaptation

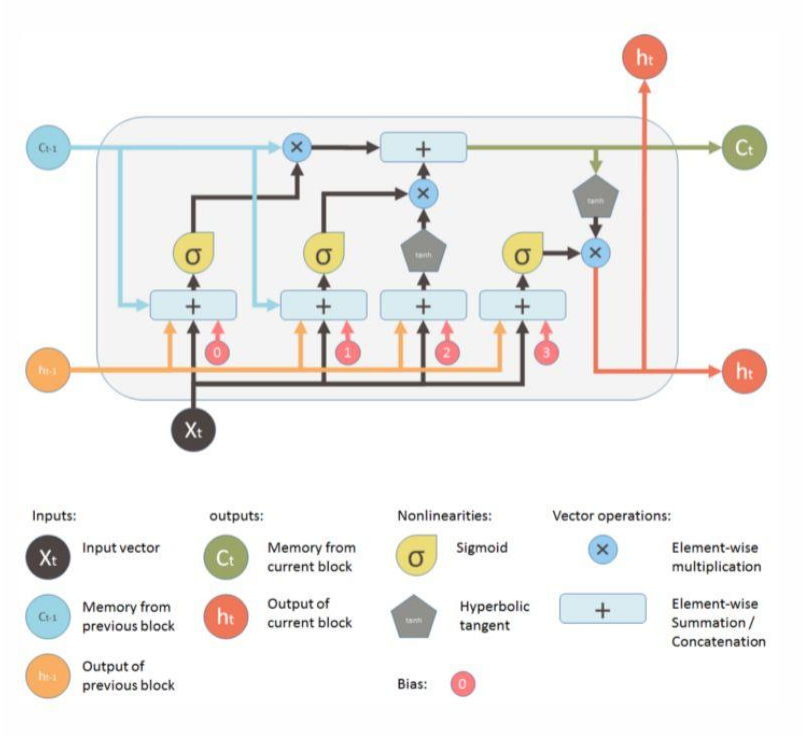
- Keep the model architecture and weights unchanged
- Change and fine-tune model architecture and weights


Deep Learning Architecture
Fundamentals

Embedding Layer

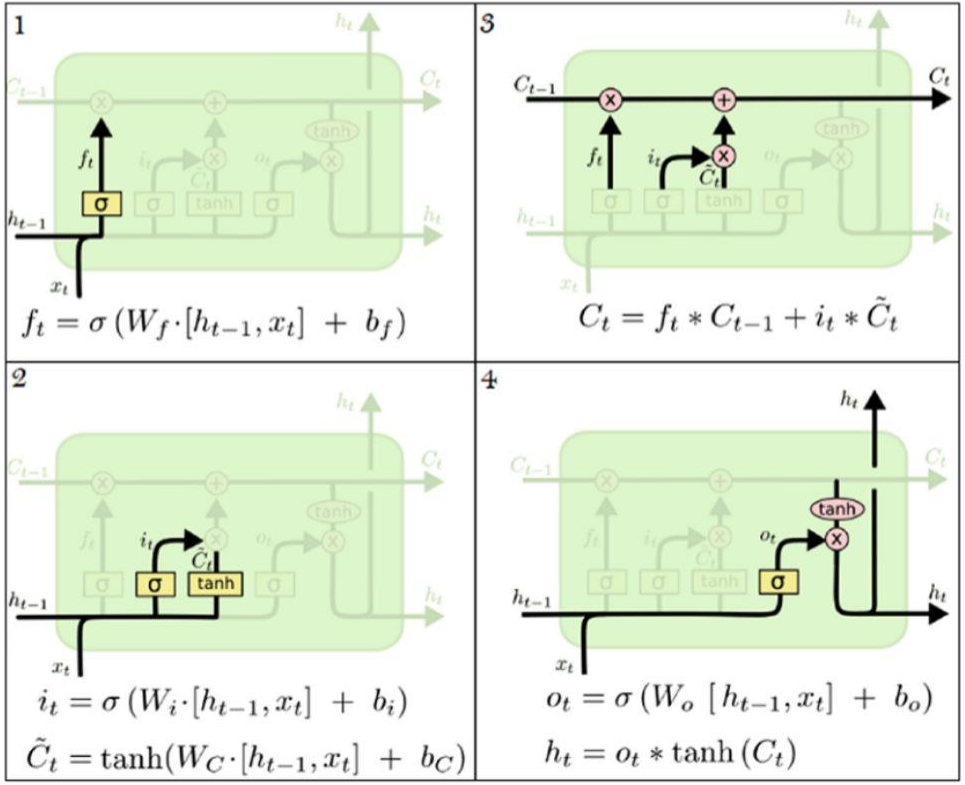


Sequential Models - LSTM

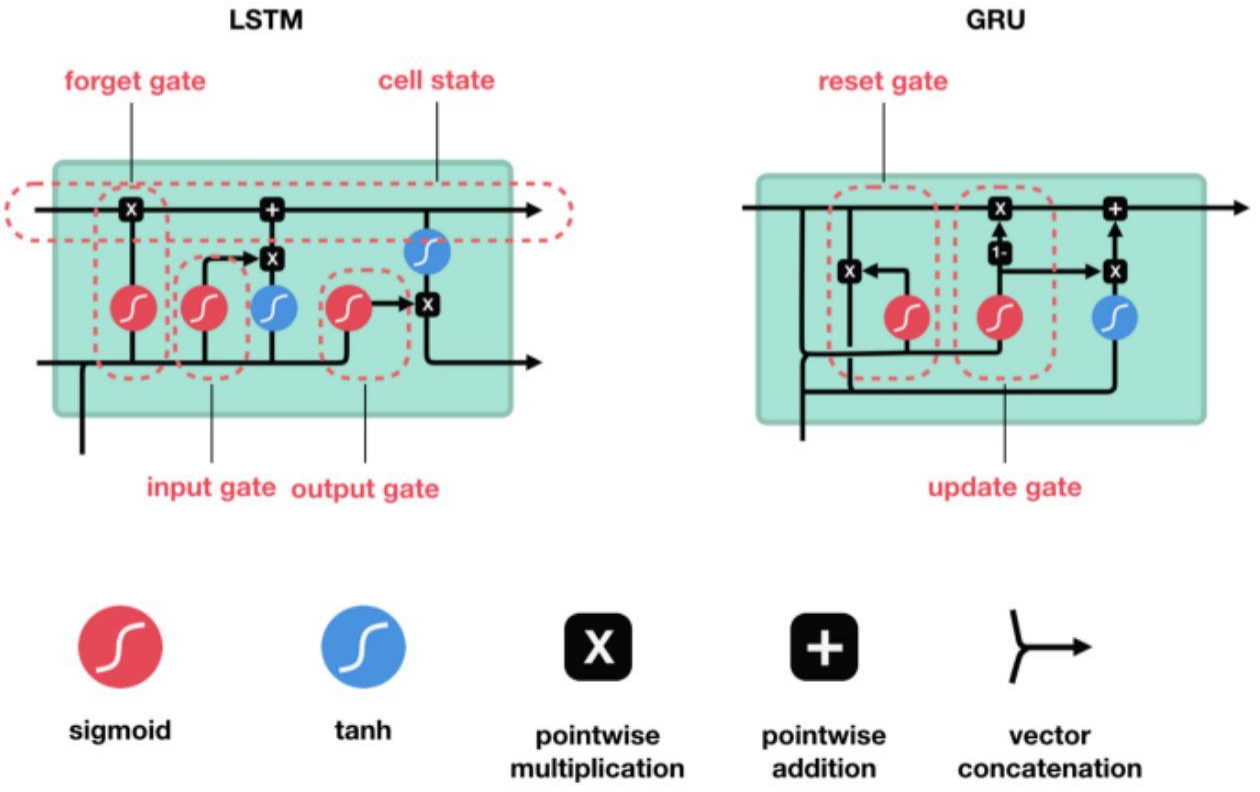


- LSTMs can remember longer sequences of data than RNNs
- The network takes three inputs
  - X_t is the input of the current time step
  - h_{t-1} is the output from the previous LSTM unit
  - C_{t-1} is the "memory" or cell state of the previous unit
- The forget gate decides what is relevant to retain from previous steps
- The input gate decides what information can be added from the current step
- The output gate decides what the next hidden state should be

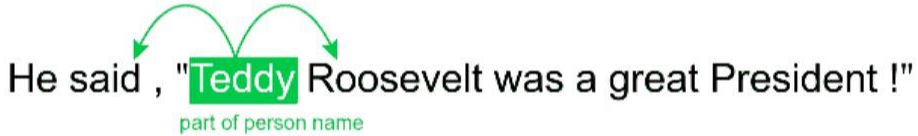
LSTM Cell Operations

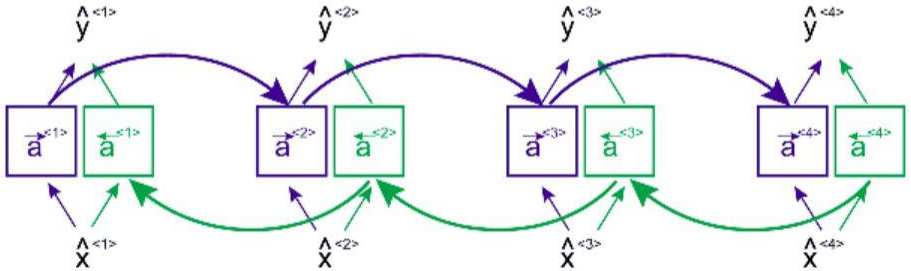


LSTM vs GRU


LSTM
forget gate
cell state
input gate
output gate
GRU
reset gate
update gate
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

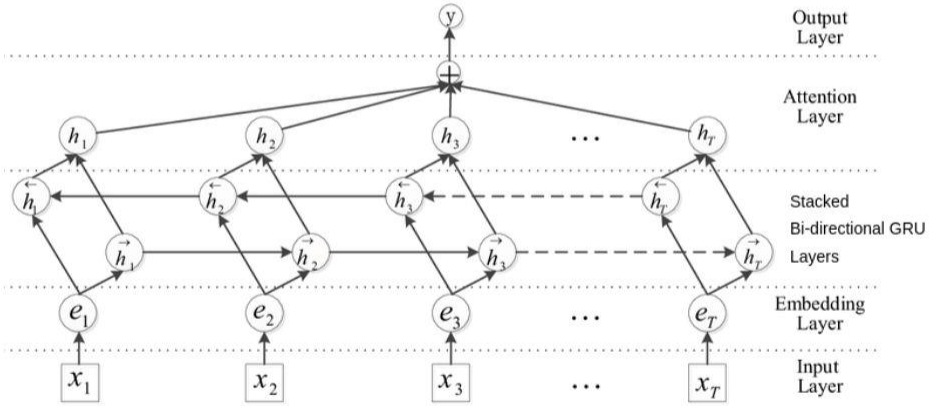
The need for Bi-directional LSTMs





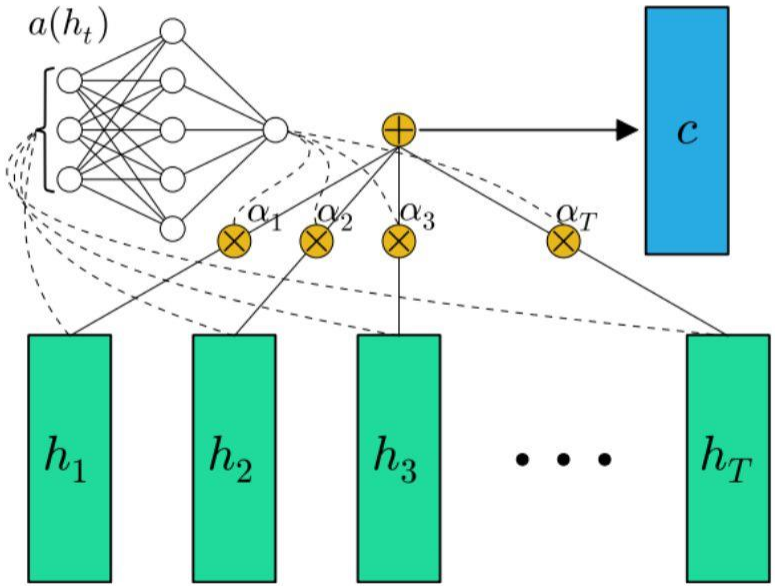
- Bi-directional LSTMs are just putting two independent LSTMs together
- The input sequence is fed in forward order for one LSTM, and in reverse order for the other
- The outputs of the two networks are usually concatenated at each time step
- Preserving information from both past and future helps understand context better

Bi-directional LSTMs + Attention



- Embedding Layer populated with pre-trained embeddings
  - FastText Embeddings
- Bi-directional GRUs or LSTMs
- Global Attention Layer
- FC Dense Layers

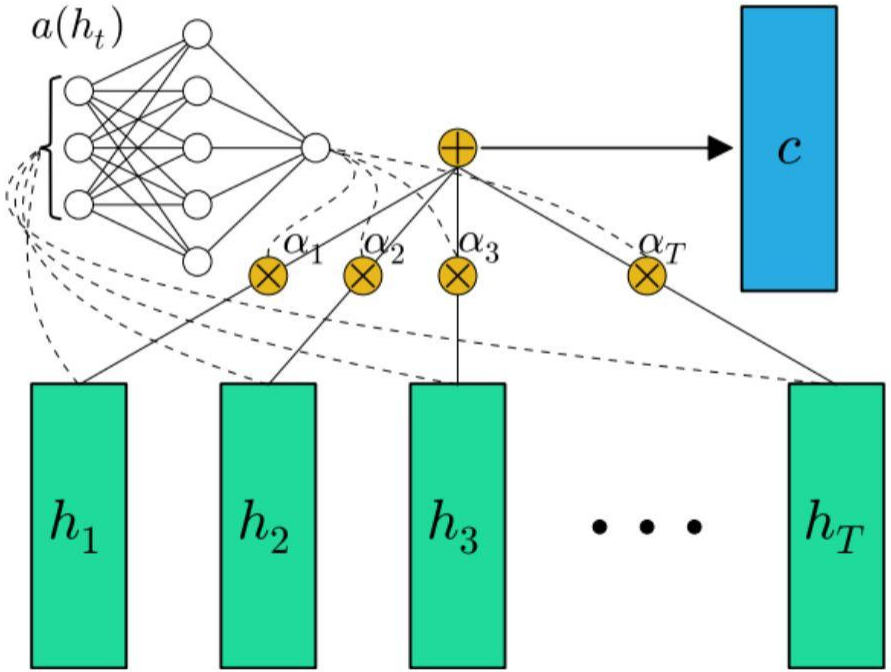
Attention Mechanism!



- Instead of using the output from the last GRU cell, send the entire sequence to a global attention layer
- Vectors from the hidden sequence h_t are fed into the learnable function $a(h_t)$
- Produces a probability vector α_t
- The final context vector c is a weighted average given by $\alpha_t.h_t$

Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Problems, Raffel et al.

Attention Mechanism!



$$e_t = a(h_t) = \tanh(Wh_t + b)$$

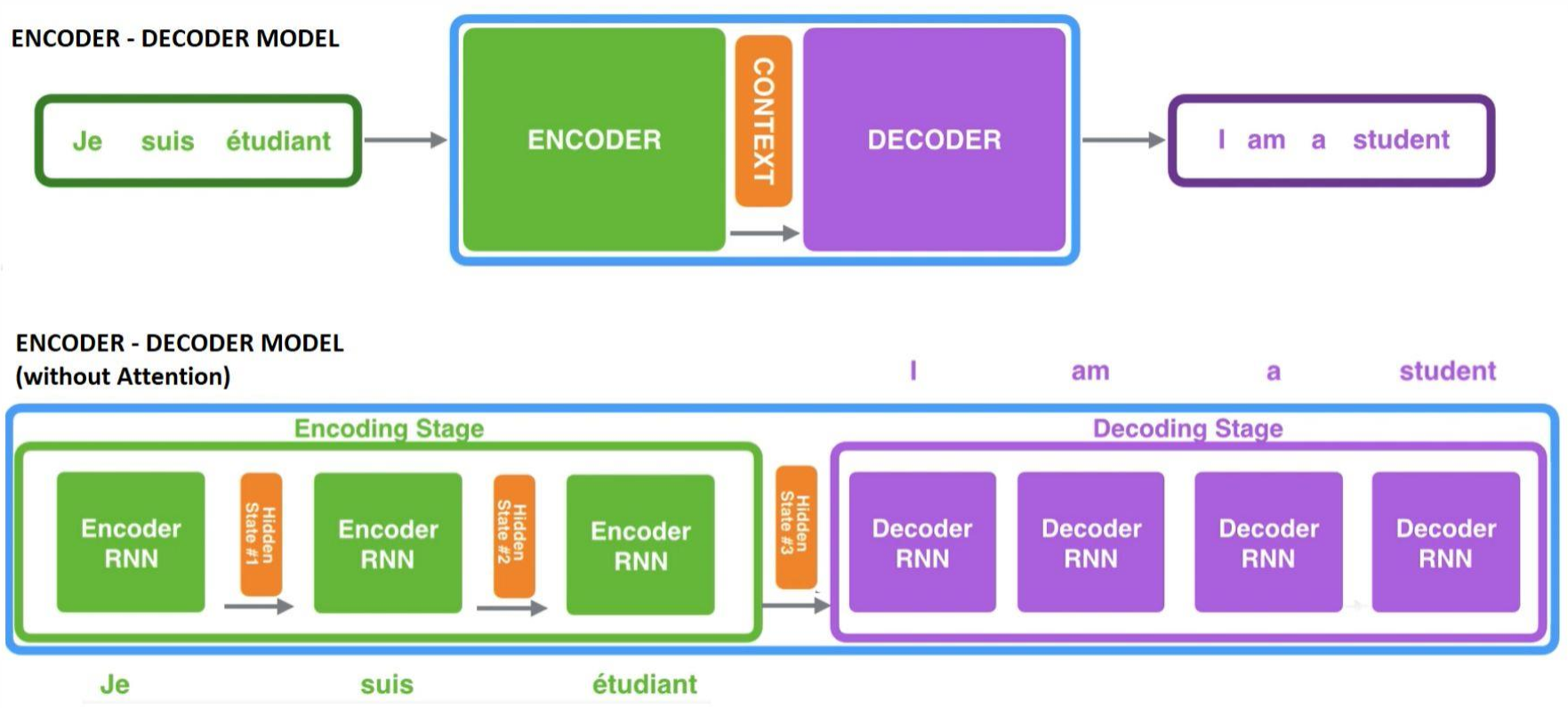
$$\alpha_t = \text{softmax}(e_t) = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)}$$

$$c = \sum_{t=1}^{T} \alpha_t h_t$$

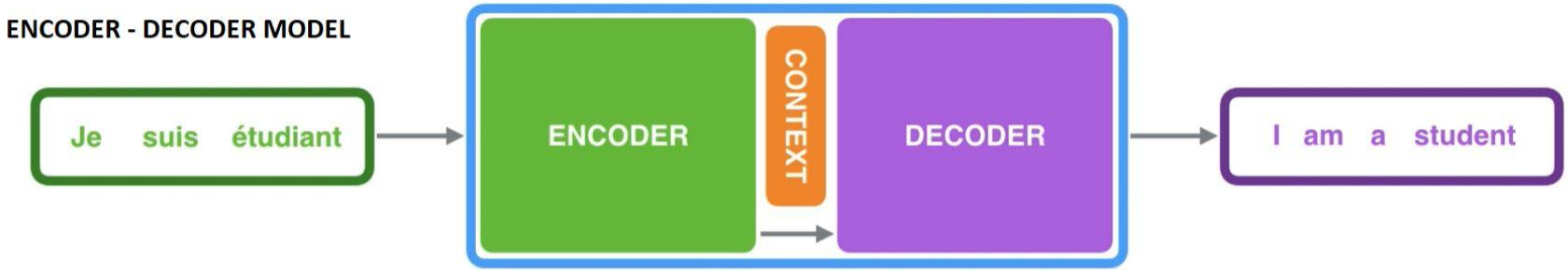
T is the total number of time steps in the input sequence

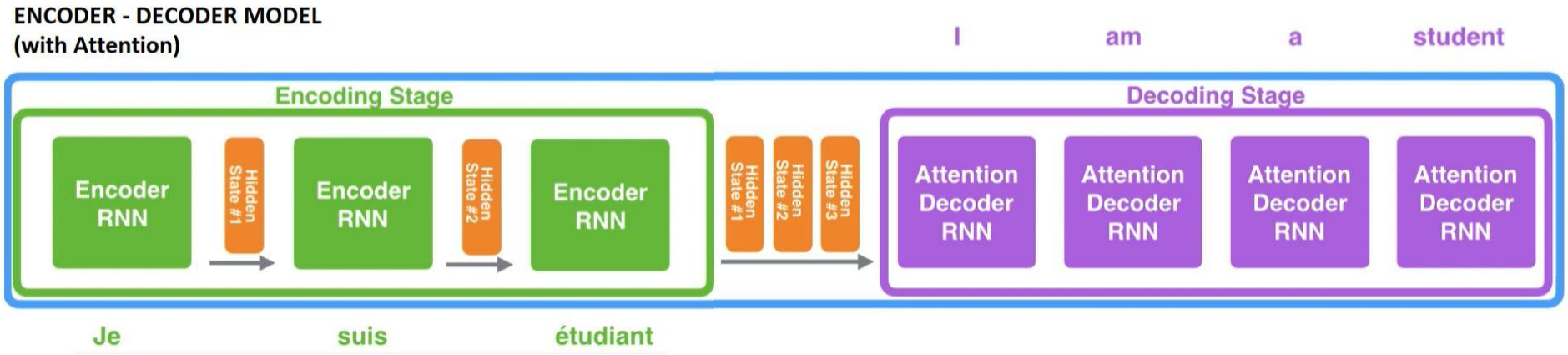

Motivation for Transformers

Encoder - Decoder Model (without Attention)

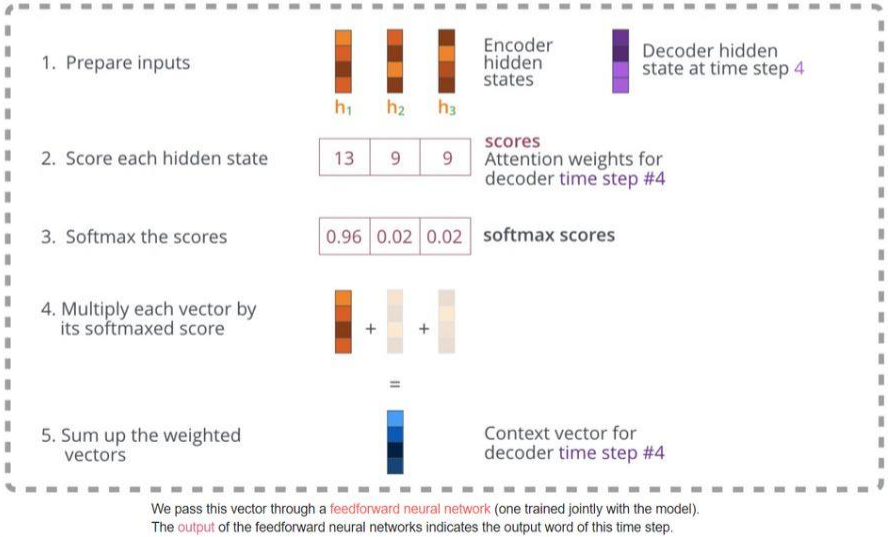


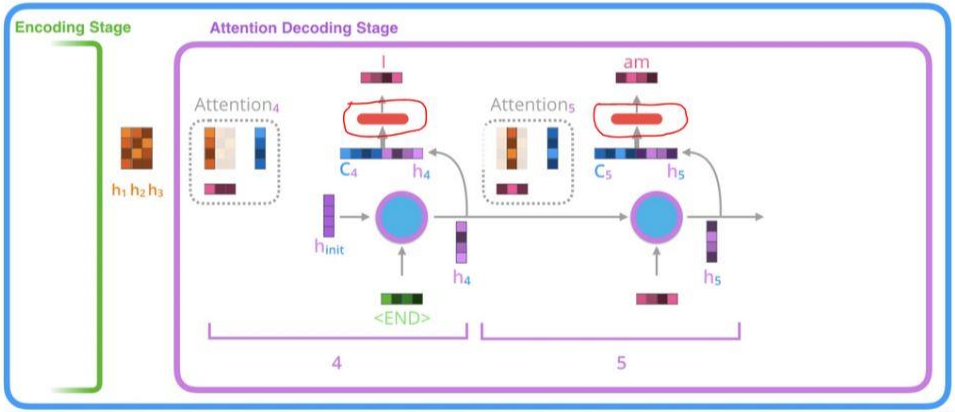
Encoder - Decoder Model (with Attention)



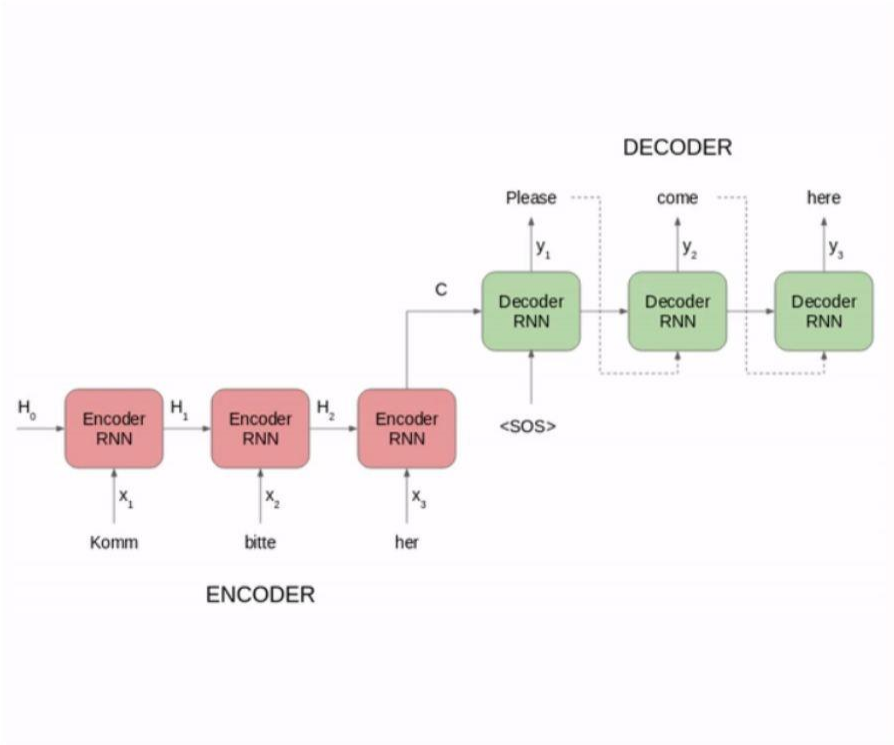


Attention Mechanism - Illustrated



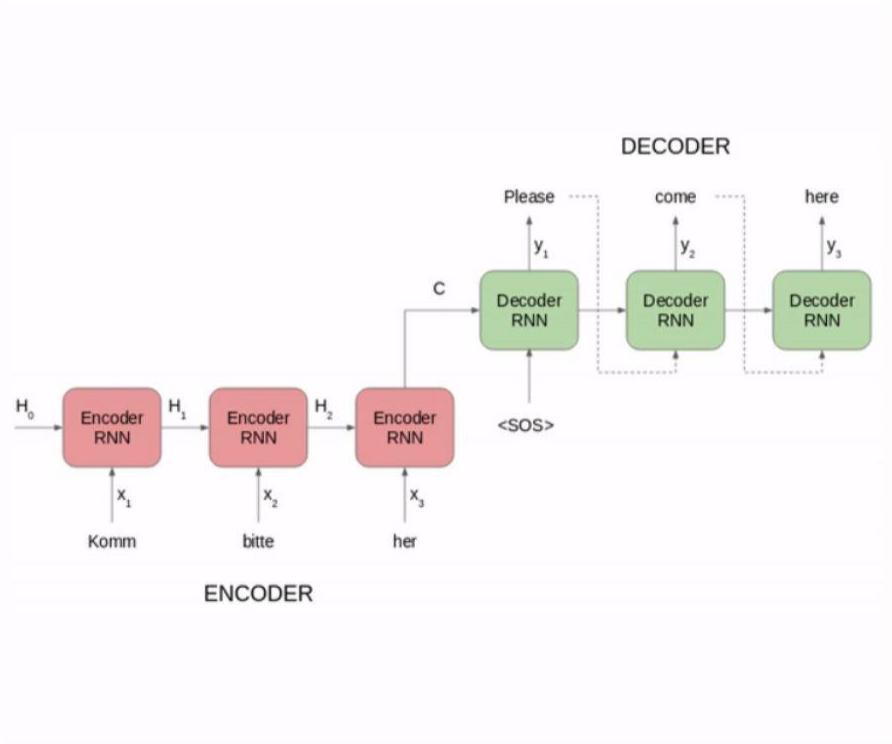


Encoder - Decoder Models - Summary



- Encoder & Decoder blocks consist of multiple RNN cells
- Hidden States are updated at each time step
- Context or Thought Vector from last step encodes information of the complete sequence
- Attention Mechanism uses a weighted sum of all hidden states instead of just the last one

Encoder - Decoder Models - Challenges



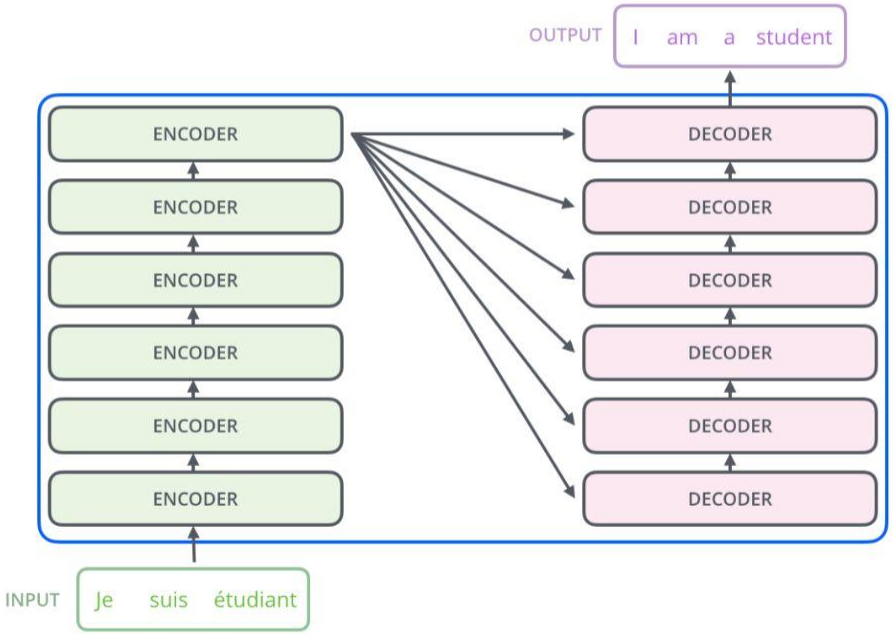
1. Encoding a long sequence without attention into a single context vector loses valuable information
2. Even with Attention, results show it is challenging to handle long sequences & dependencies
3. Sequential model has limitations with parallelizing training


Transformer Essentials

Transformer Model - Architecture

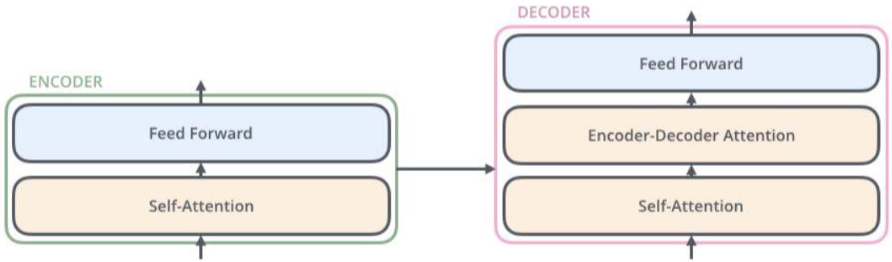
- Layered & Stacked Encoder - Decoder Model
- Relies on Multi-headed Self-Attention & Encoder-Decoder Attention
- No sequential RNN based training (completely parallelized)
- Achieved state-of-the-art performance on several NLP tasks

Transformer Model - Architecture



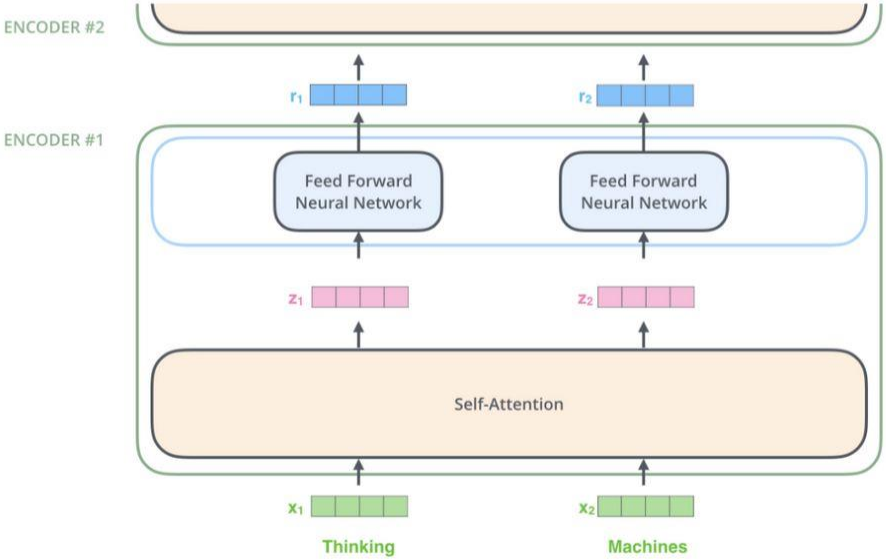
- At-heart a transformer model is a stacked encoder-decoder model
- Has 6-12 stacked encoder and decoder blocks usually based on standard architectures
- Based on the type of transformer model (only encoder \ decoder or both are used)

Transformer Model - Architecture



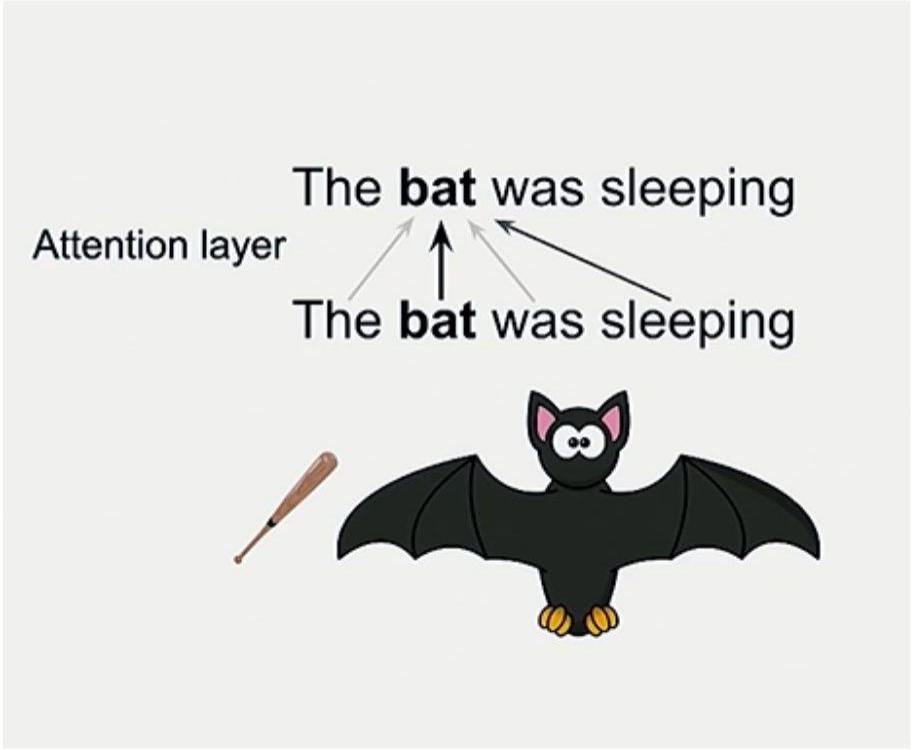
- **The encoder's inputs flow through a self-attention layer**
  - Each word attends (looks at) every other word in this process
- **The outputs of the self-attention layer are fed to a feed-forward neural network**
- **The decoder has similar layers**
- **Between them is an attention layer that helps the decoder focus on relevant parts of the input sentence**
  - This is the encoder-decoder attention layer

Transformer Model - Encoder Flow



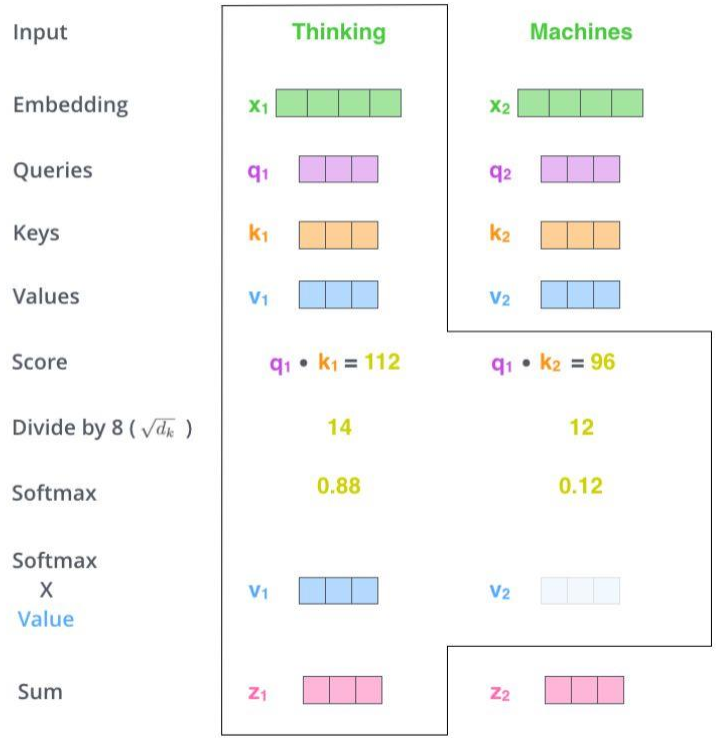
- **Self-attention layers (multi-head) at each block**
  - Each word attends (looks at) every other word in this process
- **The outputs of each self-attention layer are fed to a feed-forward neural network (FFNN)**
- **Output from each FFNN are sent upwards to the next encoder layer (with the same workflow as above)**

Self-Attention: A Simplistic Overview



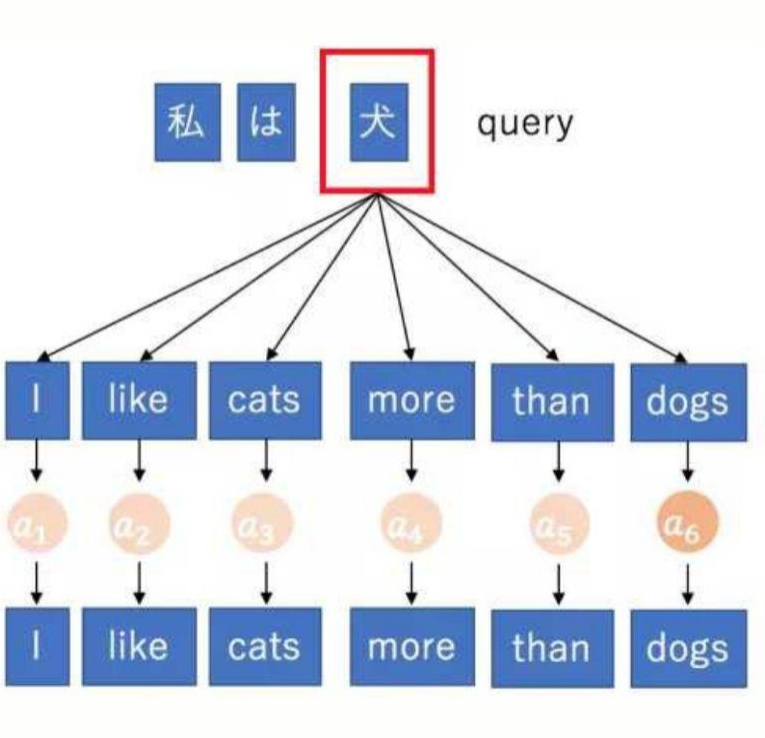
- What does “bat” in this sentence refer to? Is it referring to a baseball bat \ cricket bat or the animal?
- When the model is processing the word “bat”, self-attention allows it to associate ”bat” with “sleeping” strongly
  - This gives it the indication that it could be an animal
- Self attention allows the model to look at other positions in the input sequence for clues that can help lead to a better encoding for each word

Self-Attention: A Detailed Overview



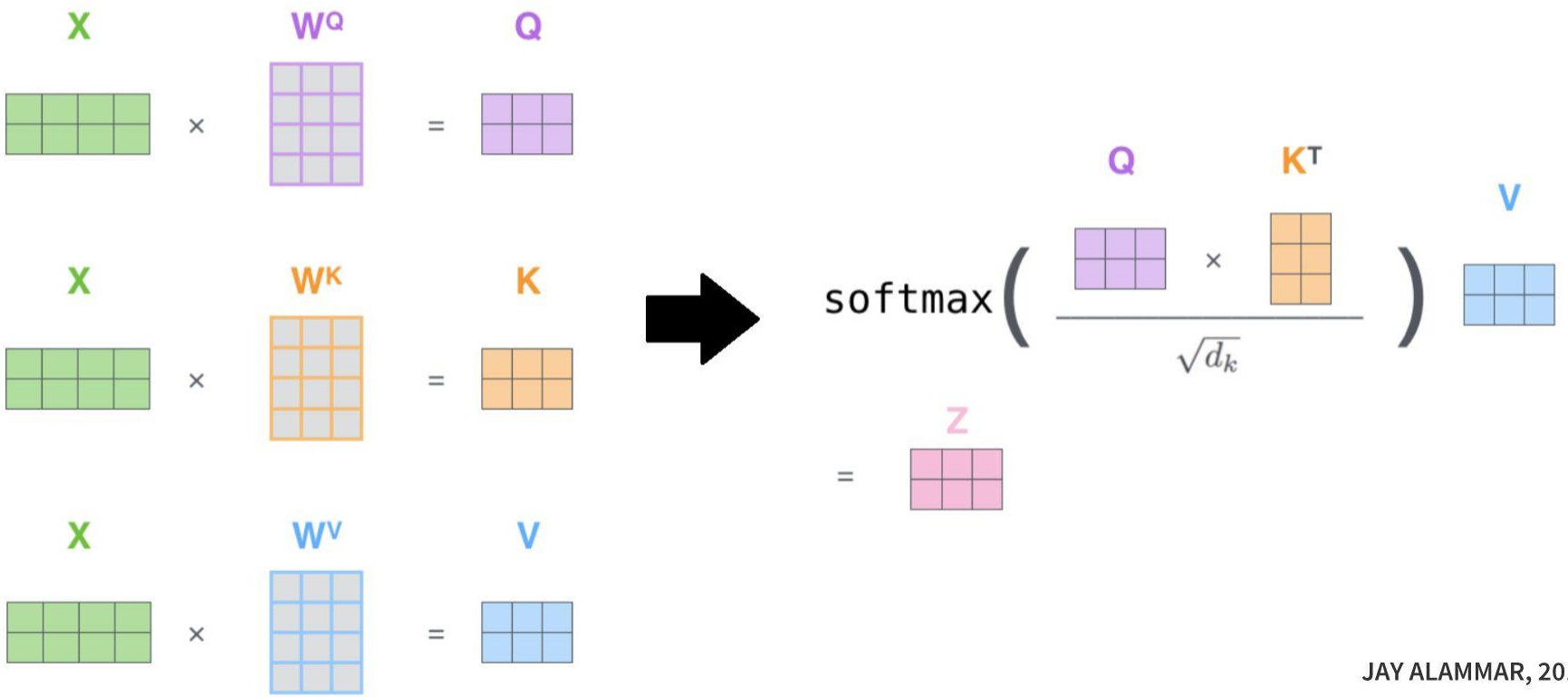
- Embeddings are generated for each word
- Query, Key and Value tensors are trained over epochs
  - Query comes from target sequence
  - Key & Value comes from source sequence
  - Initially K, V can be same
- Importance \ Alignment of each key is computed w.r.t the query to get scores
  - Scores are divided by 8 to stabilize gradients
- Softmax normalizes the scores and computes dot product with values (source sequence) to create the sum vector
- Result vector is sent to next FFNN

Self-Attention: A Detailed Overview



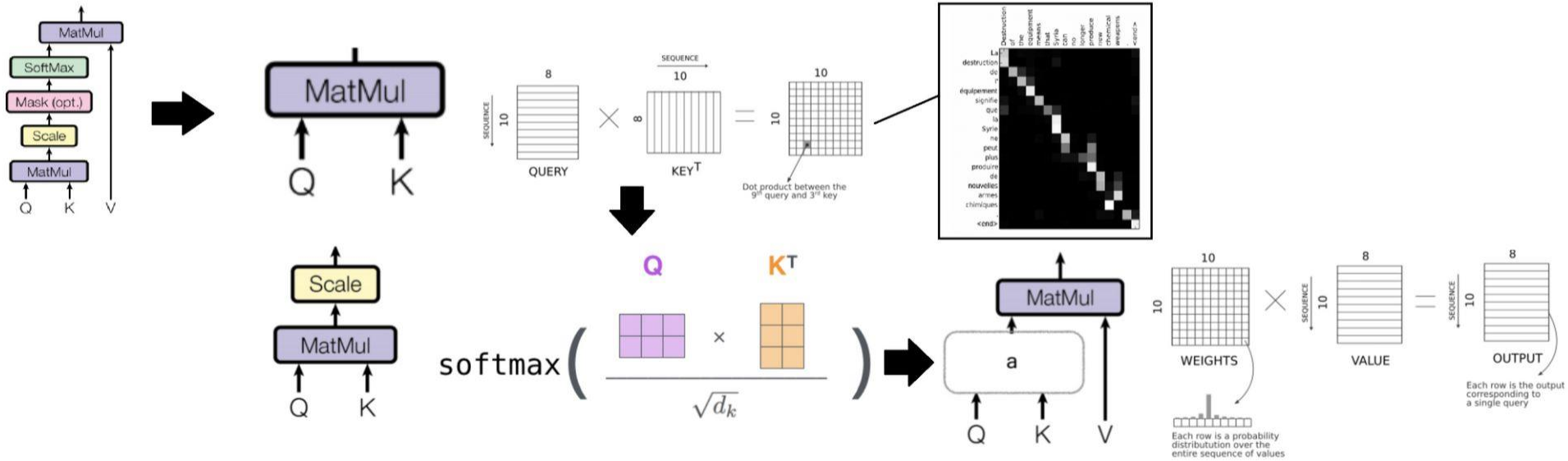
- Embeddings are generated for each word
- Query, Key and Value tensors are trained over epochs
  - Query comes from target sequence
  - Key & Value comes from source sequence
  - Initially K, V can be same
- Query and Key representations (embeddings) are multiplied (dot-product) to give dynamic alignment (attention) weights
  - Can be visualized as a heatmap
- These attention context weights are multipled (dot-product) with the Value representation (embeddings) to give outputs for input Query
- This is then scaled using softmax

Self-Attention: A Detailed Overview - Tensor Ops

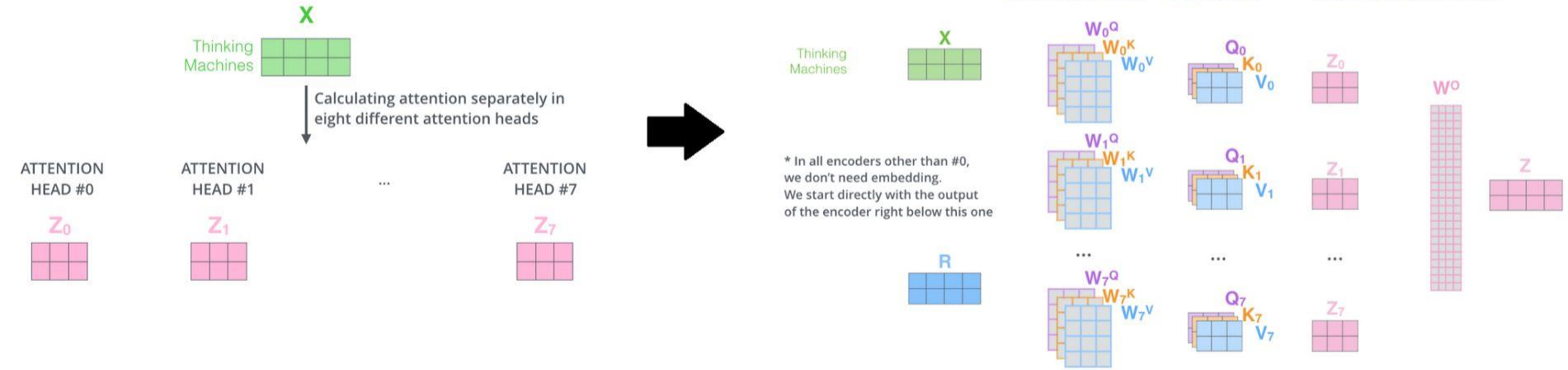


JAY ALAMMAR, 2018

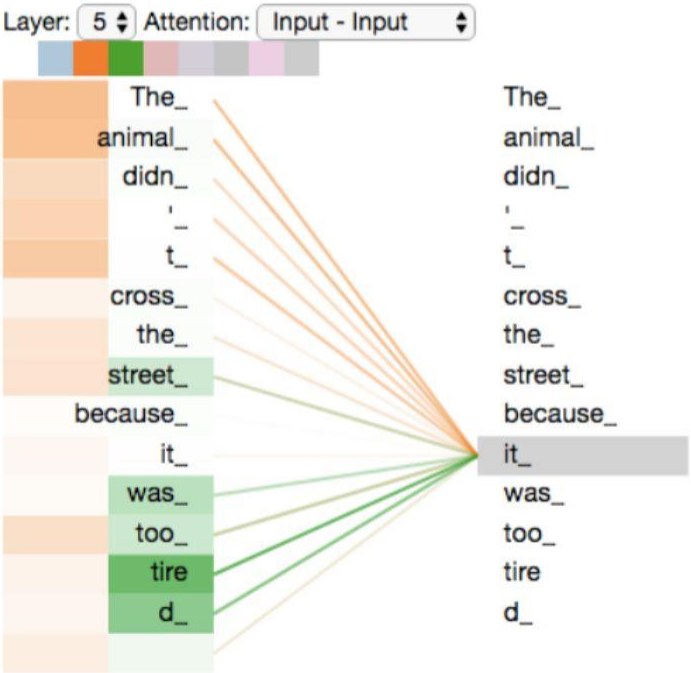
Self-Attention: A Detailed Overview - Tensor Ops



Multi-Headed Attention

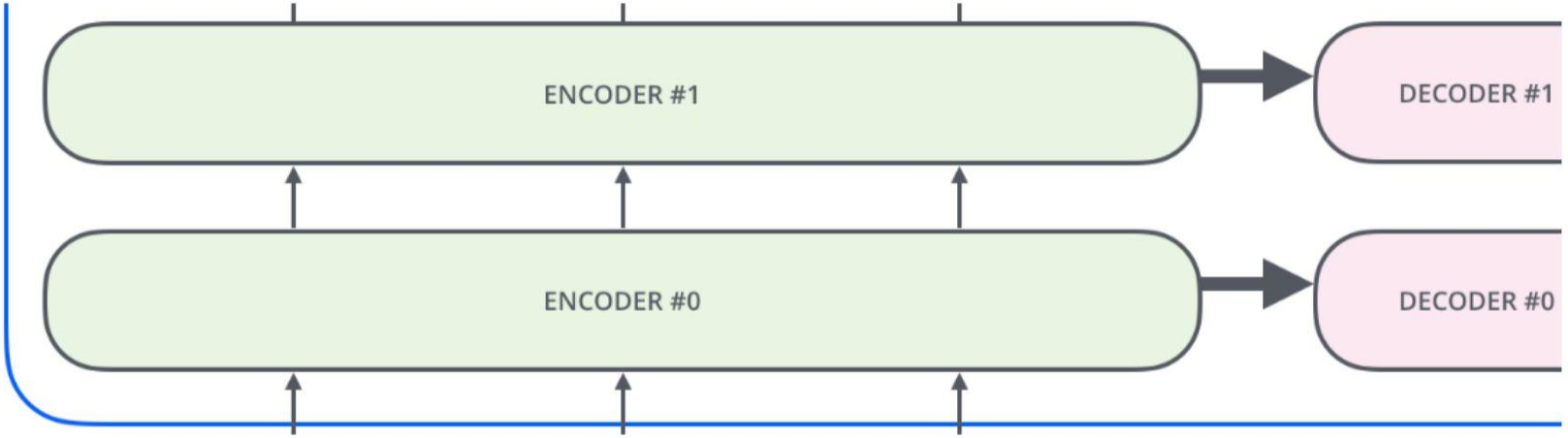


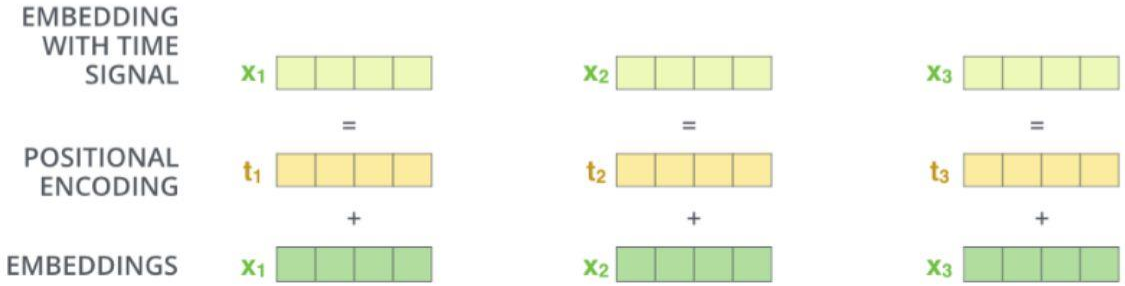
Multi-Headed Attention - Encode different aspects



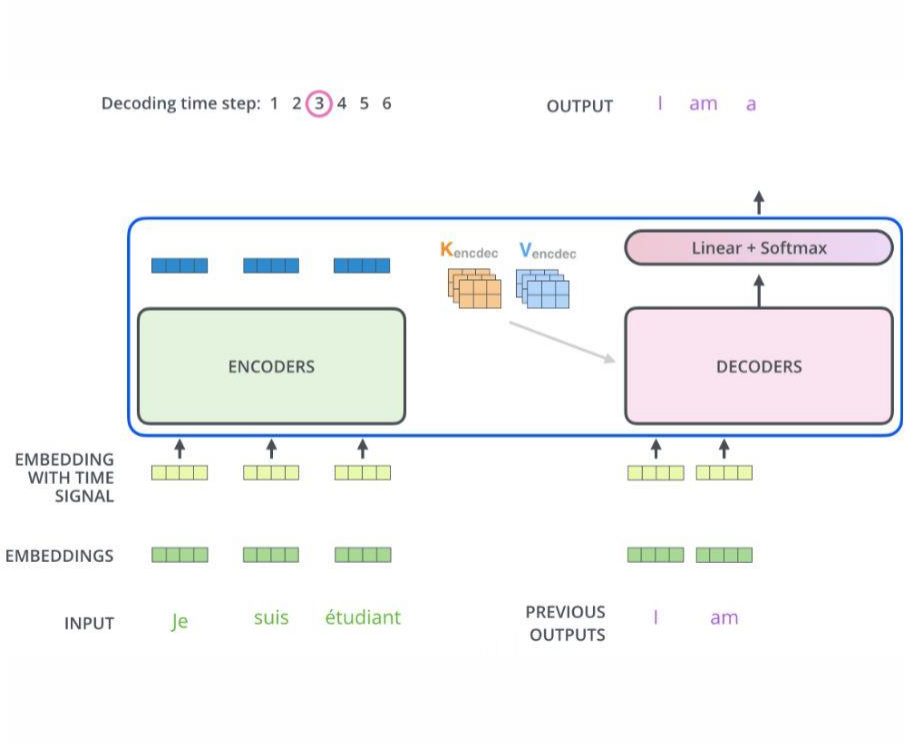
As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

Encoding Sequential Representations in Transformers



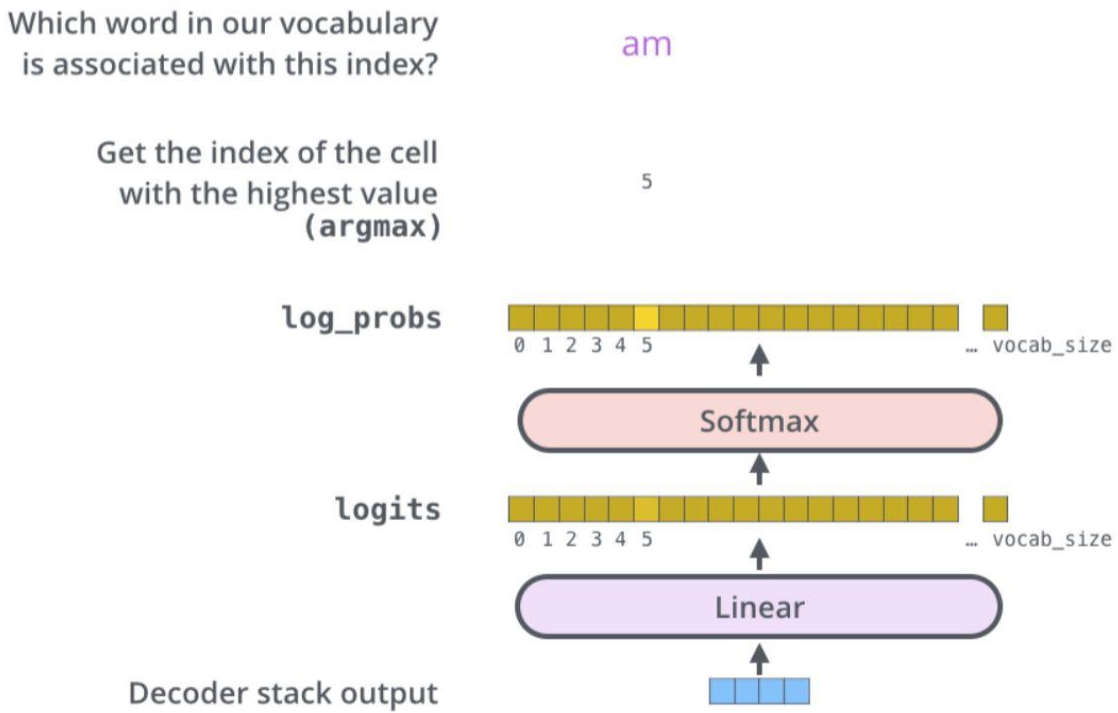


Transformer Model - Decoder Flow



- The output of the top encoder is transformed into a set of attention vectors K and V
- These are used by each decoder in its “encoder-decoder attention” layer
  - Helps the decoder focus on appropriate places in the input sequence
  - Works just like multiheaded self-attention
  - Creates its Queries matrix from the layer below it
  - Takes the Keys and Values matrix from the output of the encoder stack
- The output of each step is fed to the bottom decoder in the next time step
- The decoders bubble up their decoding results just like the encoders did
  - We embed and add positional encoding as we did before
- In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence
  - This is done by masking future positions

Final Layers in the Transformer Model




Transformer Model Architectures

Transformer Model Types

1 Autoregressive Models

- Correspond to the decoder of the original transformer model
- Mask is used on top of the full sentence so that the attention heads can only see previous words
- Pretrained on the classic language modeling task: guess the next token
- Example - GPT family

2 Autoencoding Models

- They correspond to the encoder of the original transformer model
- They get access to the full inputs without any mask
- Pretrained by corrupting the input tokens in some way and trying to reconstruct the original sentence (Masked Language Modeling)
- Example - BERT family

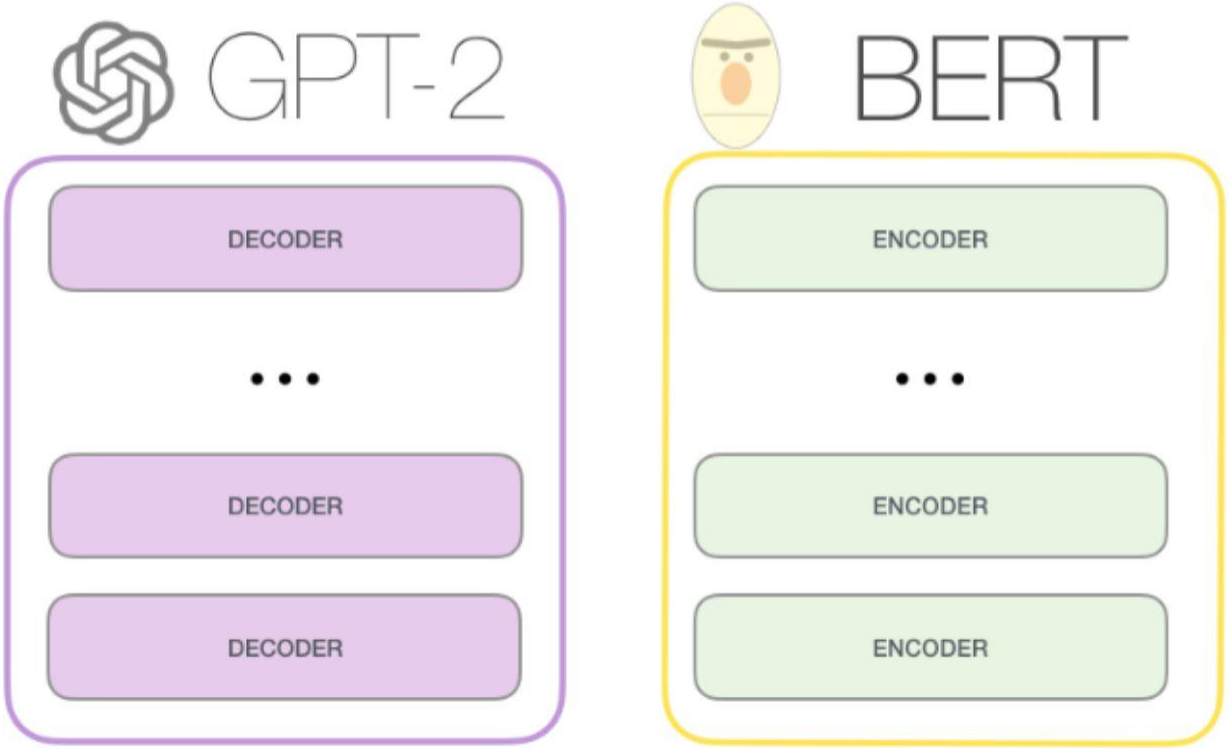
3 Sequence to Sequence Models

- Uses both the encoder and the decoder of the original transformer
- Most popular applications are translation, summarization and question answering
- Example - BART, T5

4 Multi-modal Models

- Mixes data of different modalities e.g text & images

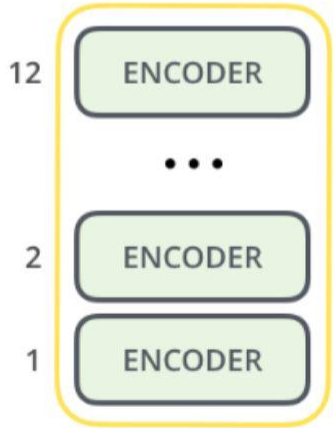
Popular Transformer Models



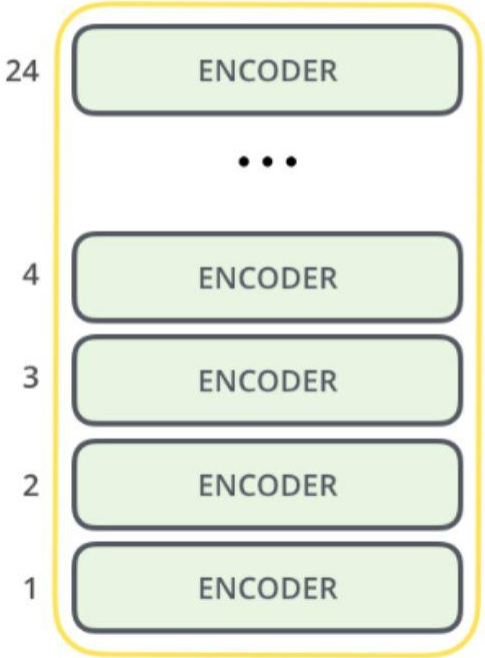
JAY ALAMMAR, 2019


Understanding BERT & DistilBERT

BERT - Trained Transformer Encoder Stack



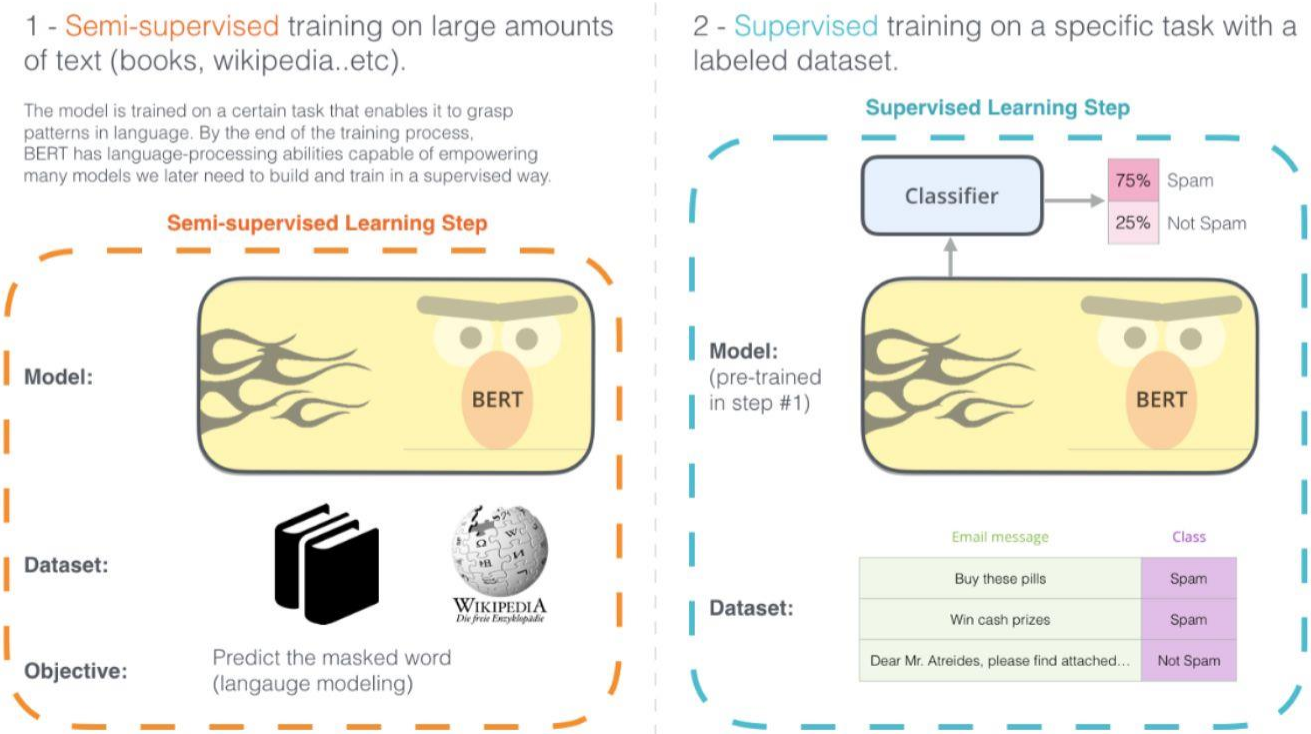
$BERT_{BASE}$



$BERT_{LARGE}$

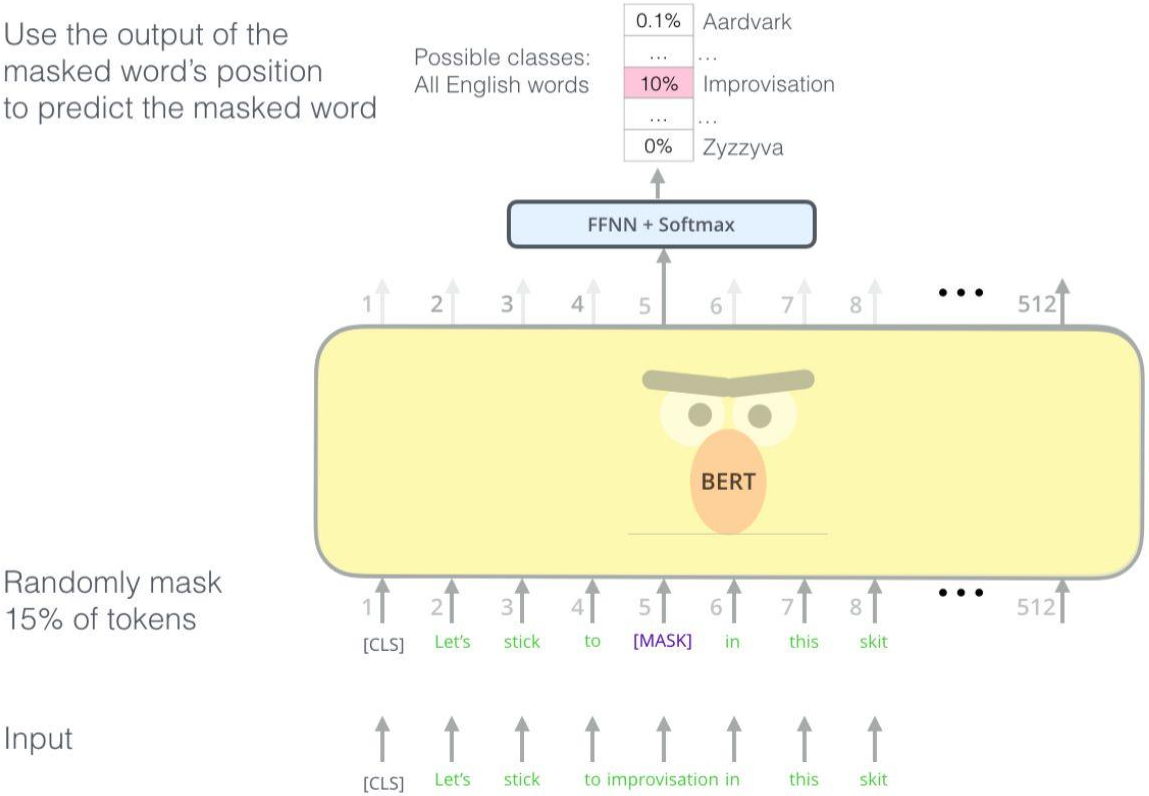
JAY ALAMMAR, 2018

BERT Training Workflow



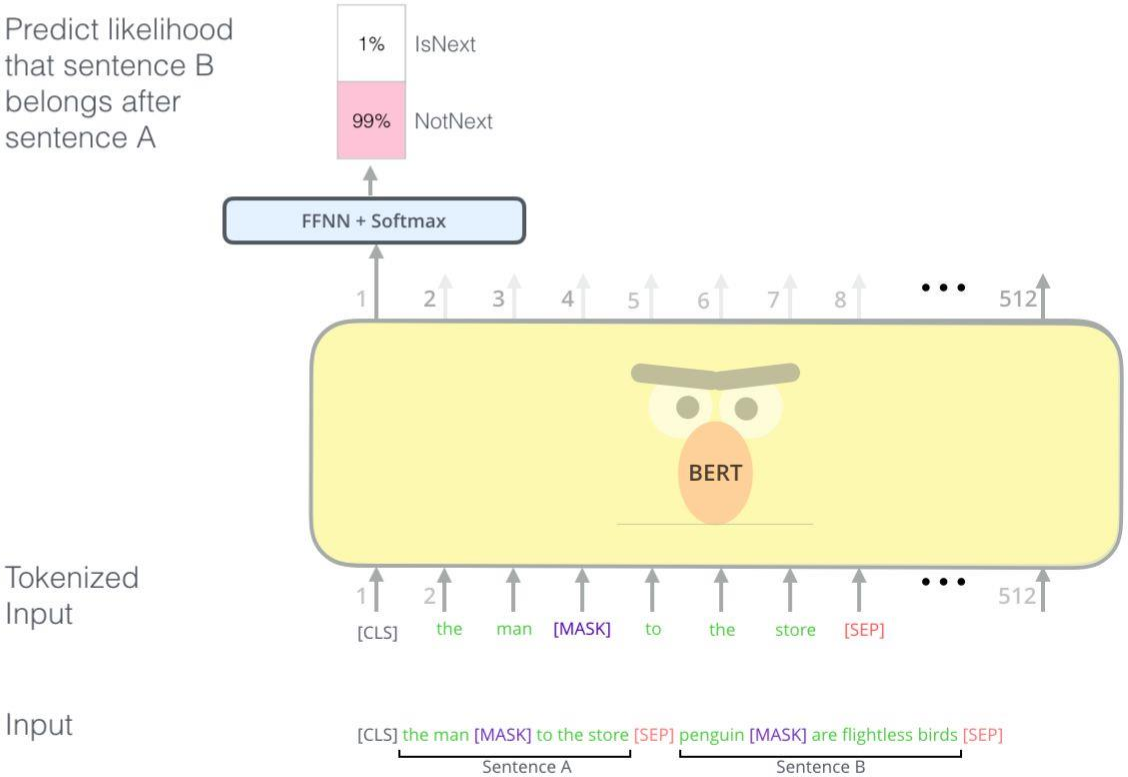
JAY ALAMMAR, 2018

BERT Training Workflow - Masked Language Modeling



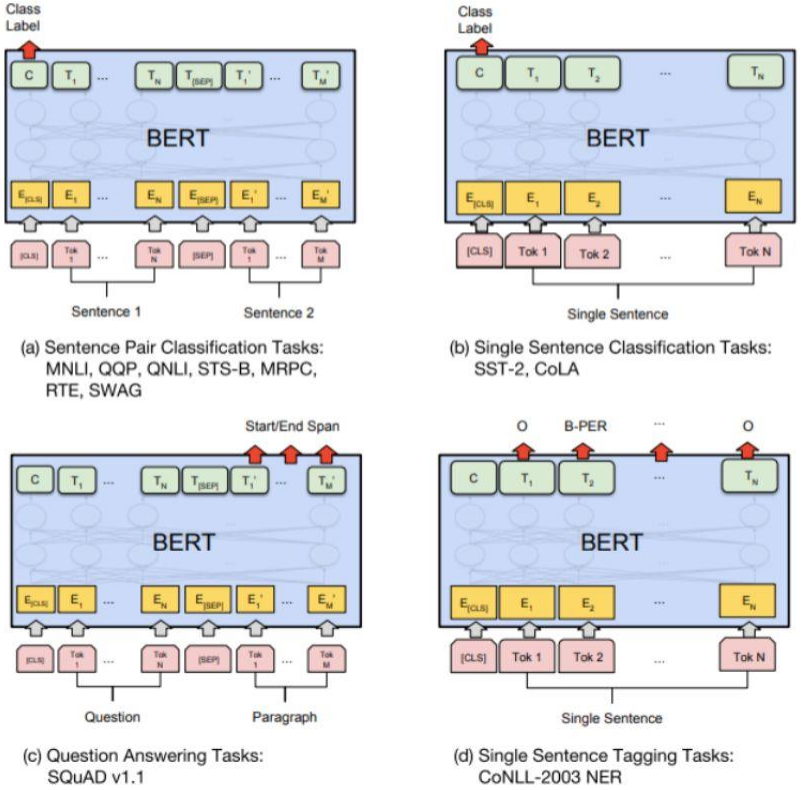
JAY ALAMMAR, 2018

BERT Training Workflow - Next Sentence Prediction



JAY ALAMMAR, 2018

Fine-tuning BERT - Multi-Task NLP



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

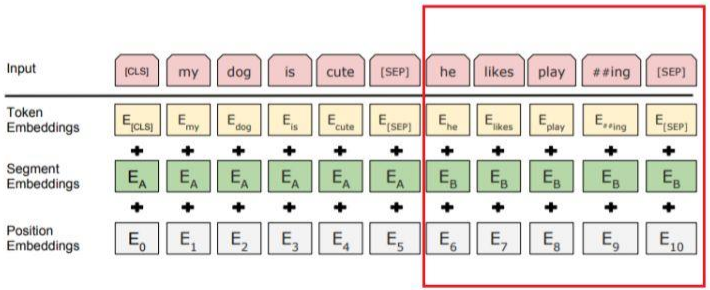
(b) Single Sentence Classification Tasks: SST-2, CoLA

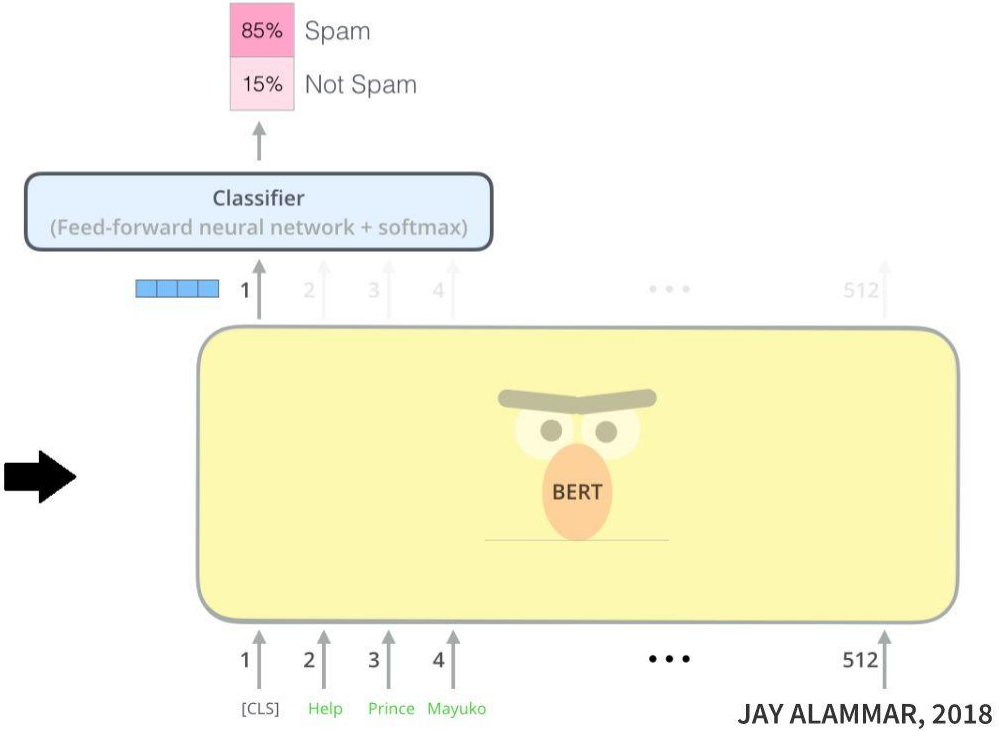
(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

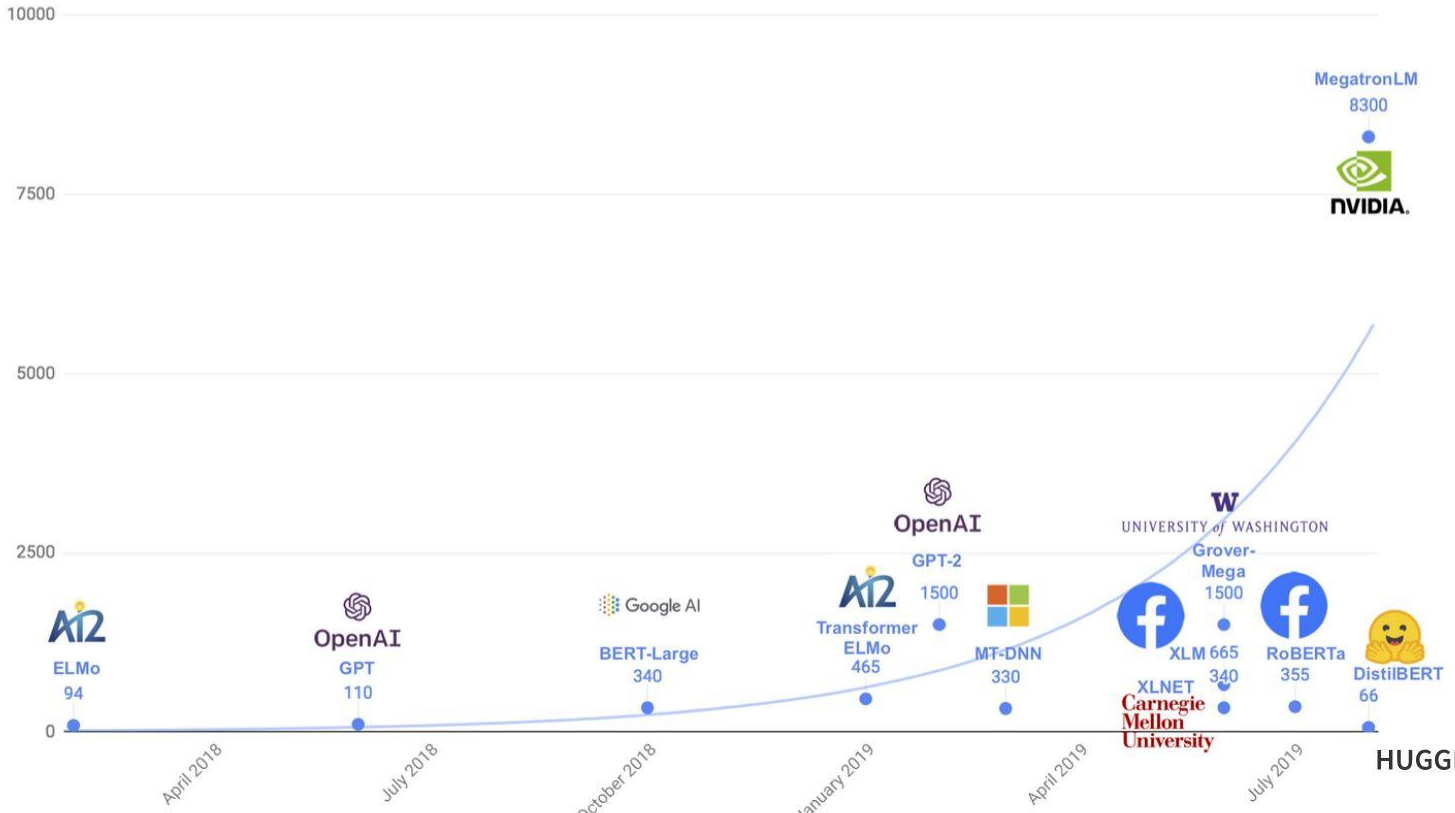
JAY ALAMMAR, 2018

Fine-tuning BERT - Text Classification





Challenges with Massive Size of Pre-trained Models



HUGGINGFACE, 2019

DistilBERT - Compressing BERT

- Compression technique known as knowledge distillation is used
- Teacher-Student Training is used - training student to mimic output distribution of teacher model (BERT)

  Trained with cross-entropy on soft targets (probabilities from teacher model)
- HuggingFace uses KL-divergence loss to train DistilBERT
- Has almost halved the number of parameters from BERT and retains 95% of performance

https://medium.com/huggingface/distilbert-8cf3380435b5

References

- **Research Papers**
  - https://github.com/dipanjanS/transformers_nlp_essentials/tree/master/papers
- **Visuals & Content**
  - https://jalammar.github.io/illustrated-transformer/
  - https://medium.com/@b.terryjack/deep-learning-the-transformer-9ae5e9c5a190
  - https://www.analyticsvidhya.com (understanding-transformers-nlp-state-of-the-art-models)
  - NLP Using Transformer Architectures - A. Géron
  - Medium \ Towards Data Science
  - https://ruder.io/state-of-transfer-learning-in-nlp/

Stay in Touch!

LinkedIn

https://www.linkedin.com/in/dipanzan

GitHub

https://github.com/dipanjanS

Medium

https://medium.com/@dipanzan.sarkar