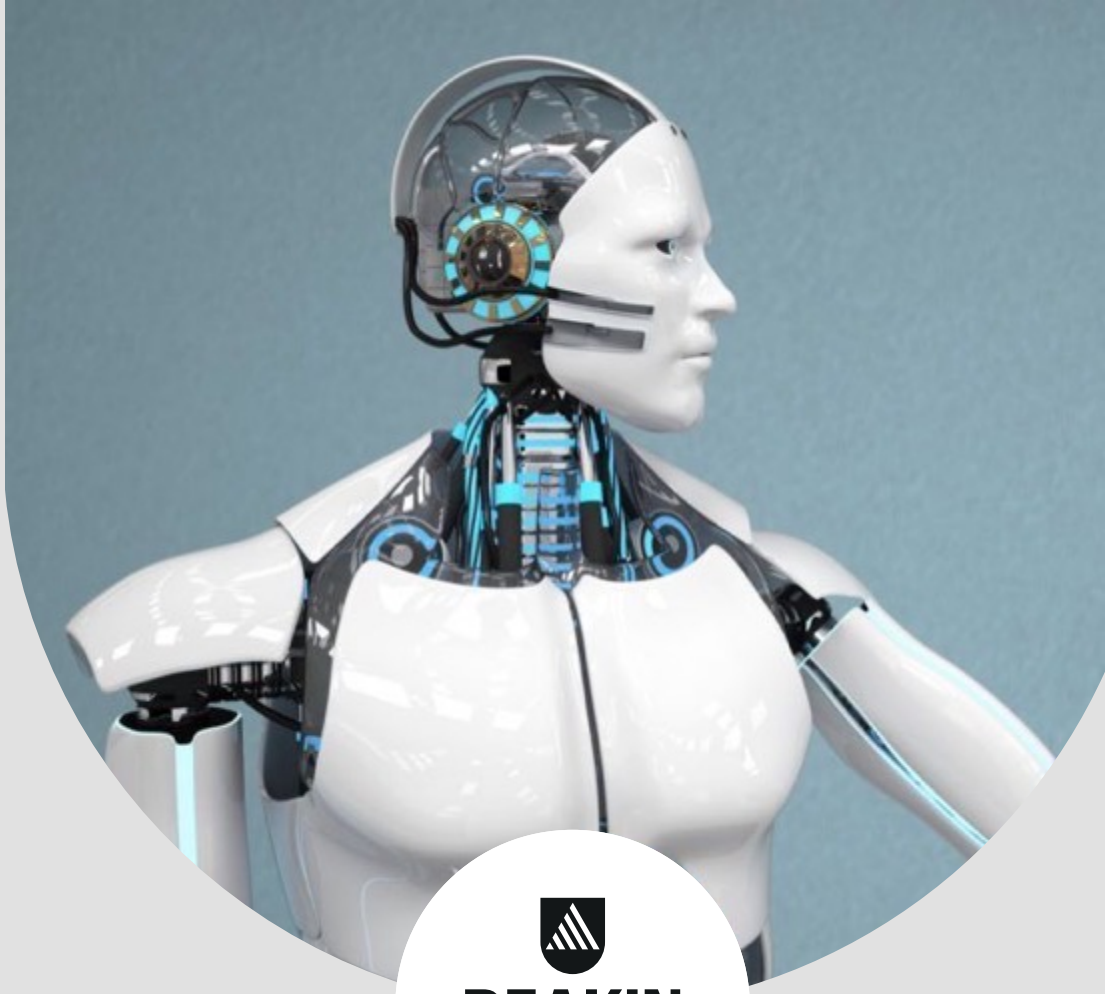
SIT330-770: Natural Language Processing

Week 10 - Transformers and Pretrained LMs

Dr. Mohamed Reda Bouadjenek

School of Information Technology, Faculty of Sci Eng & Built Env

reda.bouadjenek@deakin.edu.au



DEAKIN UNIVERSITY

SIT330-770: Natural Language Processing

Week 10.1 – Introduction to Transformers

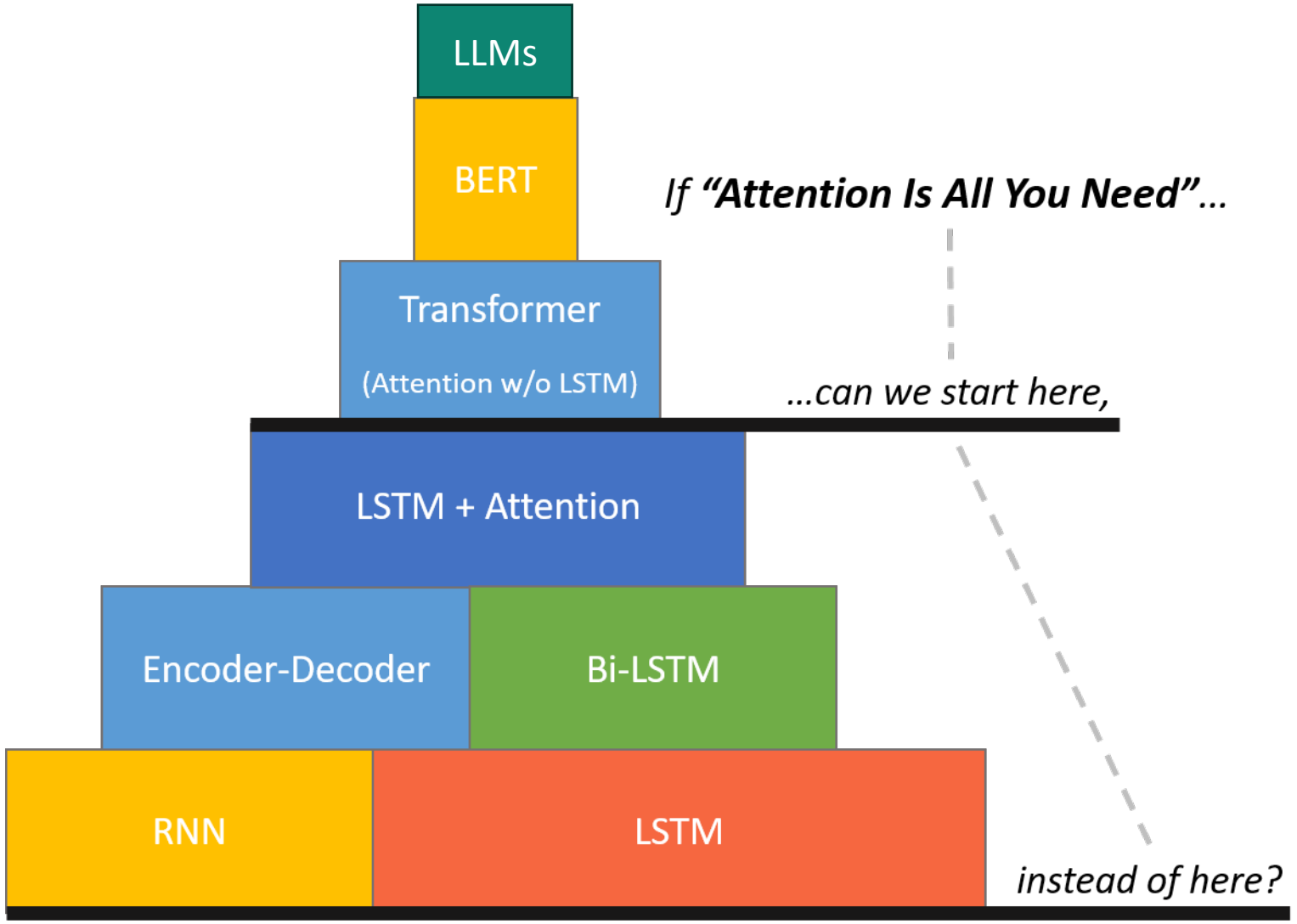
Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

The LLMs Mountain!

By Chris McCormick

(Bidirectional Encoder Representations from Transformers)



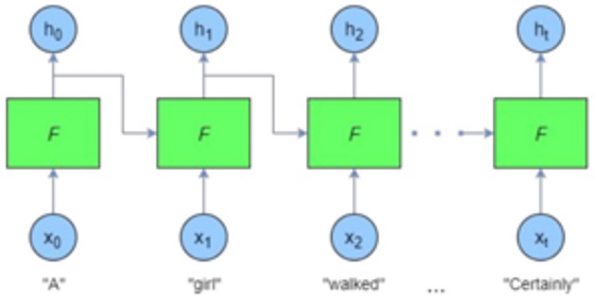
Deep Learning Models for NLP so far...

Neural Networks → RNN → Encoder-Decoder → Attention Models

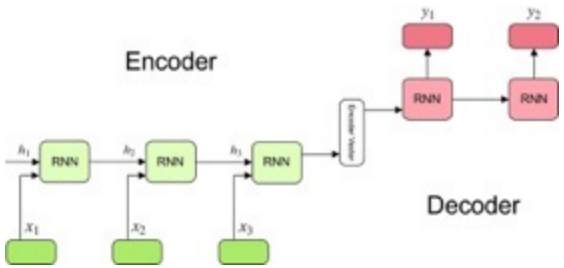
Word to representation (word2vec)

- Layered structure
- True targets vs output predictions
- Weights and loss functions
- Optimizers

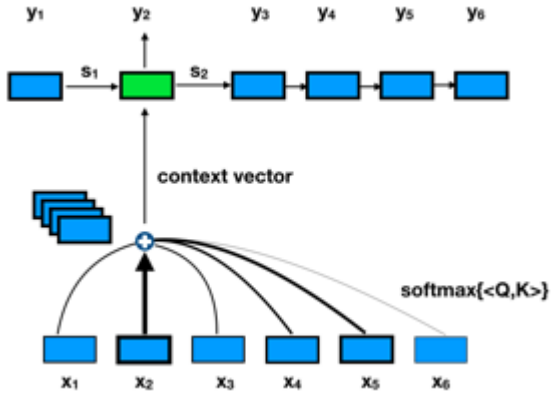
Language Generation



Translation



Translation



Problem/Motivation

- Encoder-decoder models have largely used RNN and LSTM, but the computation is sequential
  - RNN experiences vanishing gradient
  - Both are slow to train, even with factorization and conditional computation
- ConvS2S uses CNN to compute representations in parallel
  - Computation scales linearly with distance between positions
- **Transformer** provides constant computation and uses Multi-headed Attention to negate reduced effectiveness

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

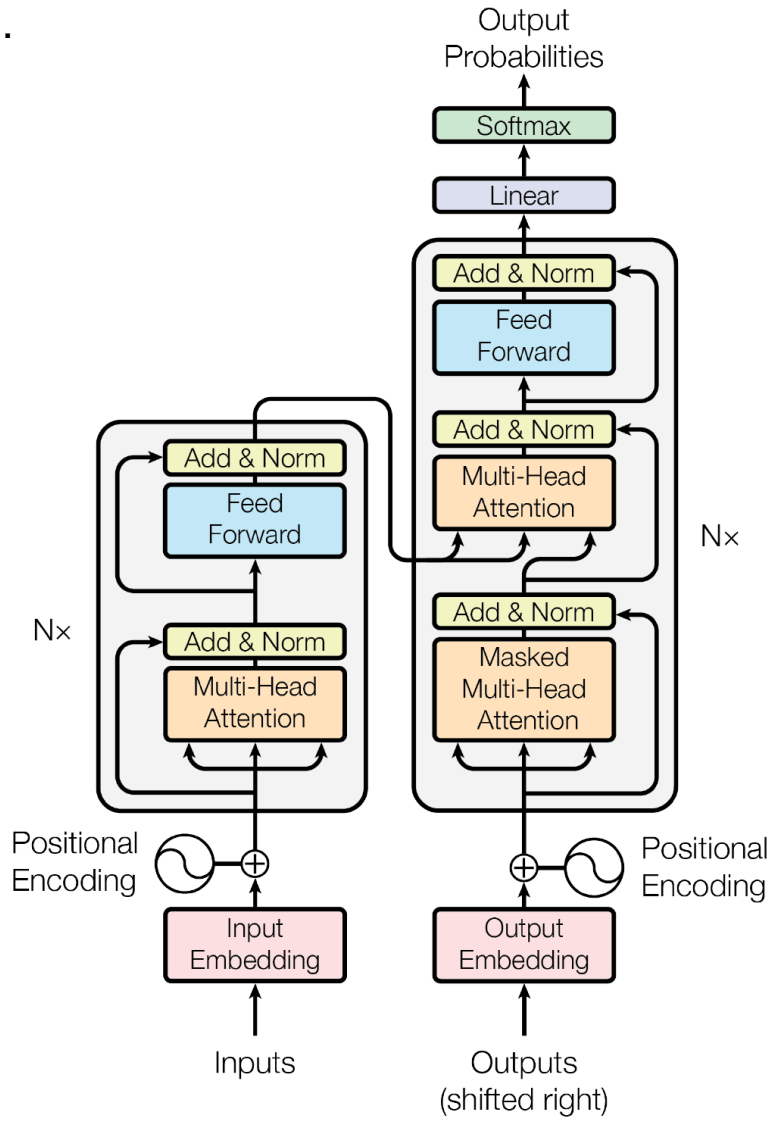
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or

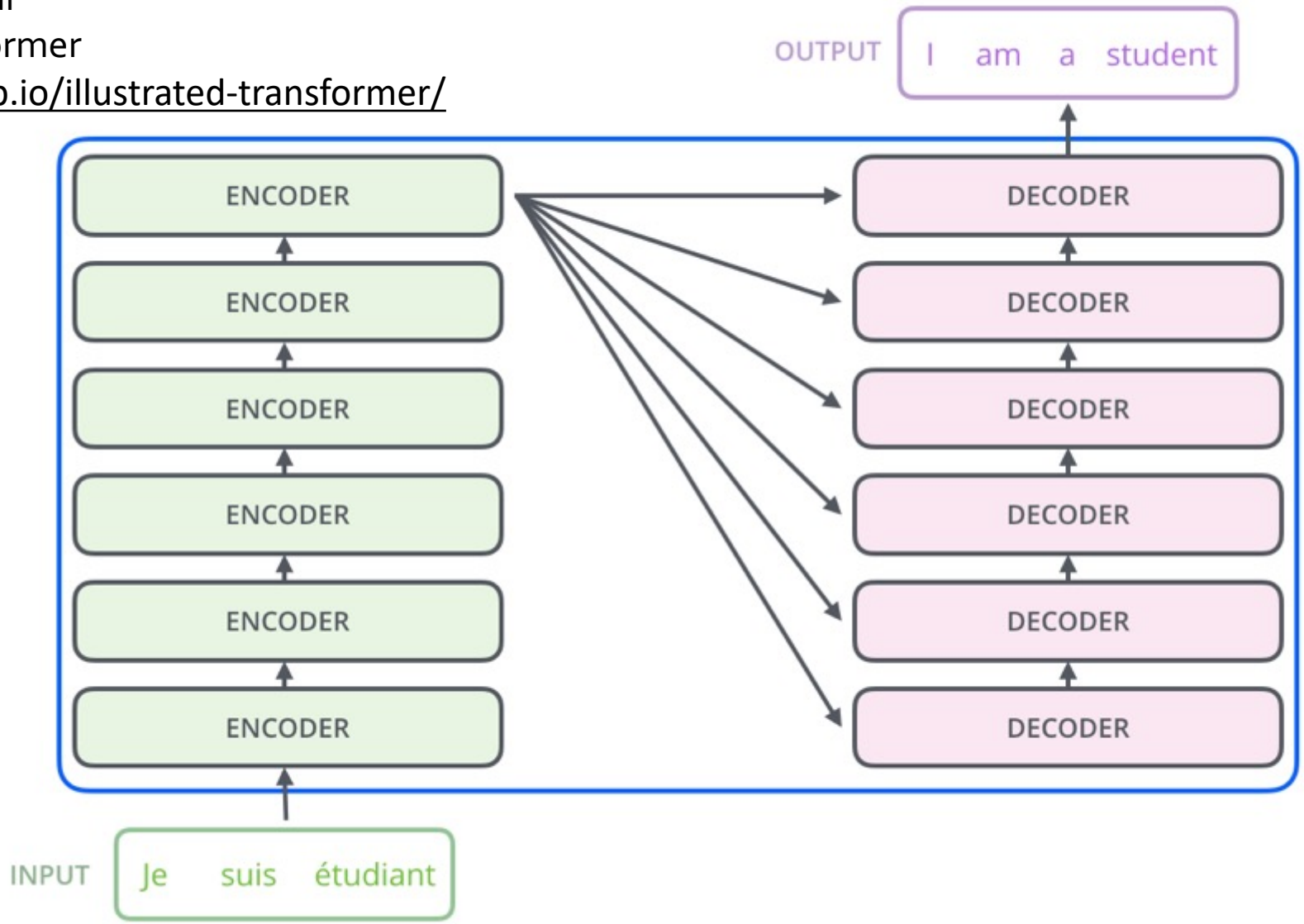
Encoder-Decoder

Ashish Vaswani et. Attention Is All You Need. NIPS 2017.



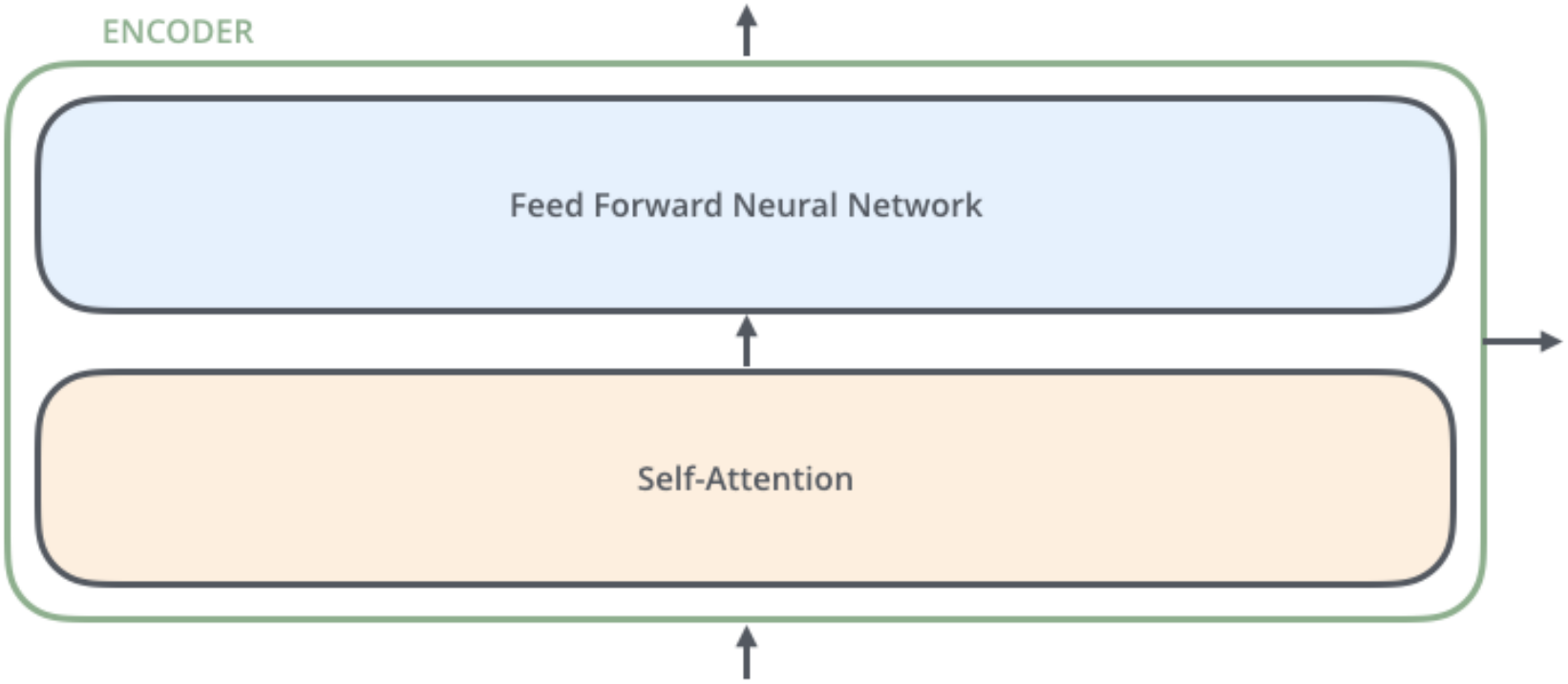
Encoder-Decoder

Figure by: Jay Alammar
The Illustrated Transformer
http://jalammar.github.io/illustrated-transformer/



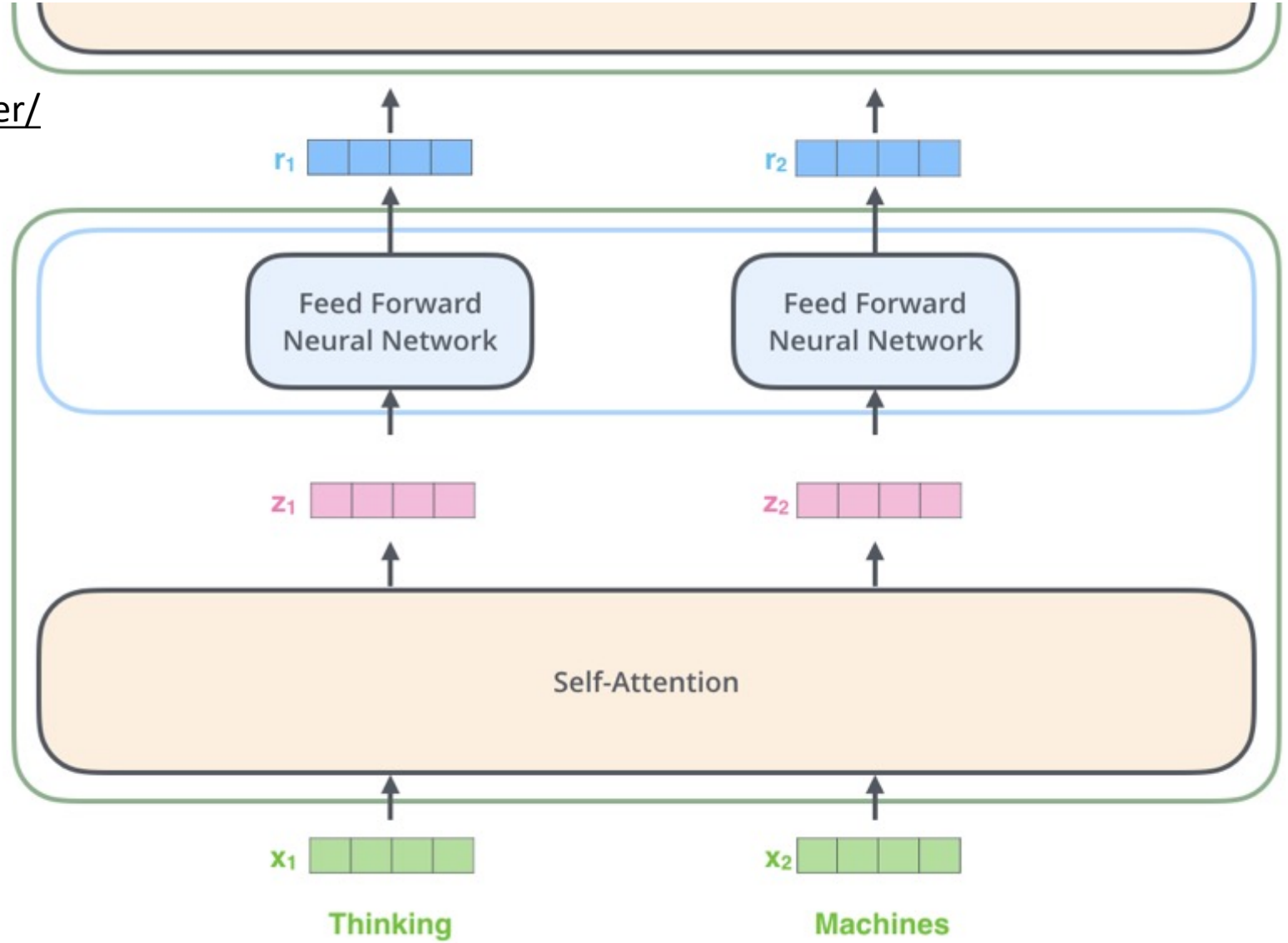
An Encoder Block: same structure, different parameters

Figure by: Jay Alammar
The Illustrated Transformer
http://jalammar.github.io/illustrated-transformer/



Encoder

Figure by: Jay Alammar
The Illustrated Transformer
http://jalammar.github.io/illustrated-transformer/



Note: The FFNN is independent for each word.
Hence can be parallelized.

Transformers vs. LSTM

- Like the LSTMs, transformers can handle distant information
- But unlike LSTMs, transformers are not based on recurrent connections
- Transformers are made up of stacks of transformer blocks, each of which is a multilayer network made by combining:
  - simple linear layers
  - feedforward networks
  - self-attention layers
- **Self-attention** allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or

SIT330-770: Natural Language Processing

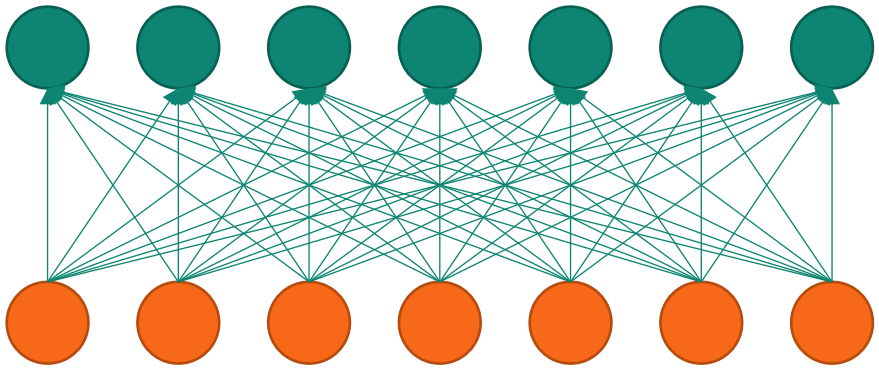
Week 10.2 – Self-Attention Mechanism

Dr. Mohamed Reda Bouadjenek

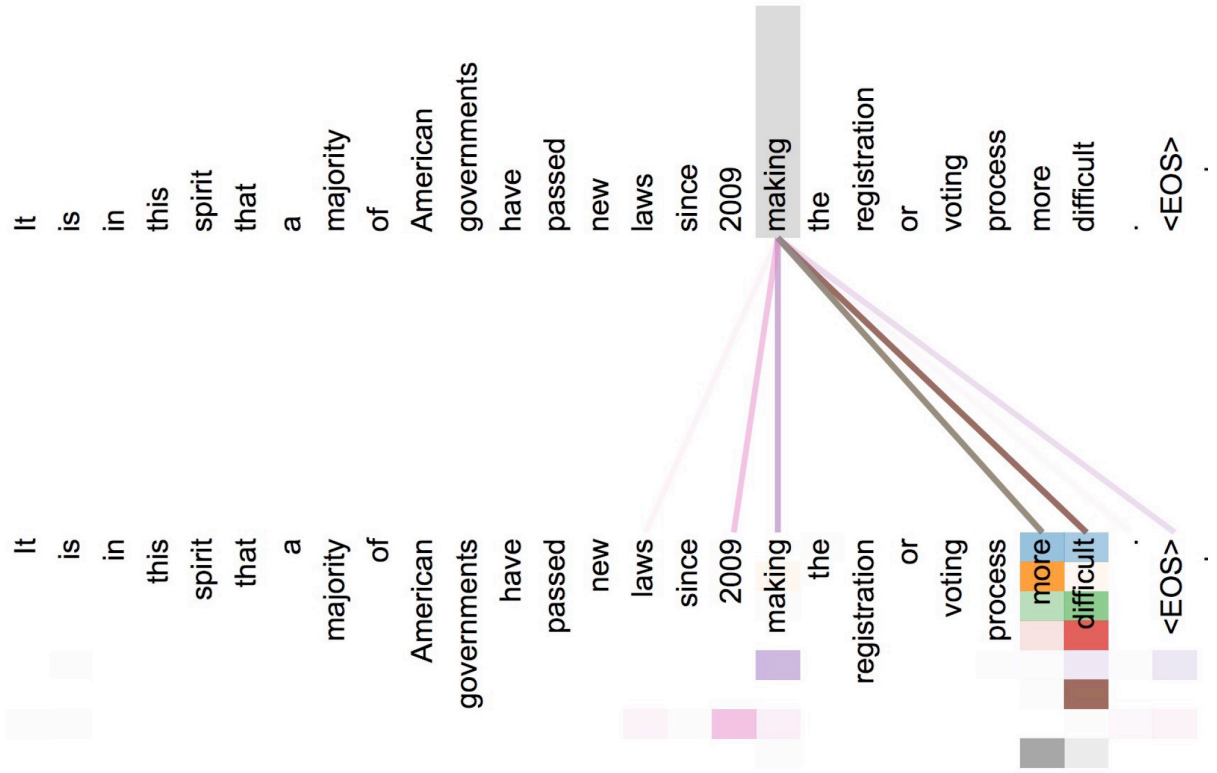
School of Information Technology,
Faculty of Sci Eng & Built Env

Self-Attention Mechanism

- All-to-all comparison.
  - Each layer is $O(N^2)$ for sequence of length N - **self attention**.
- Every output is a weighted sum of every input.
  - The weighting is a function to learn.



Relevance scores from each input to output



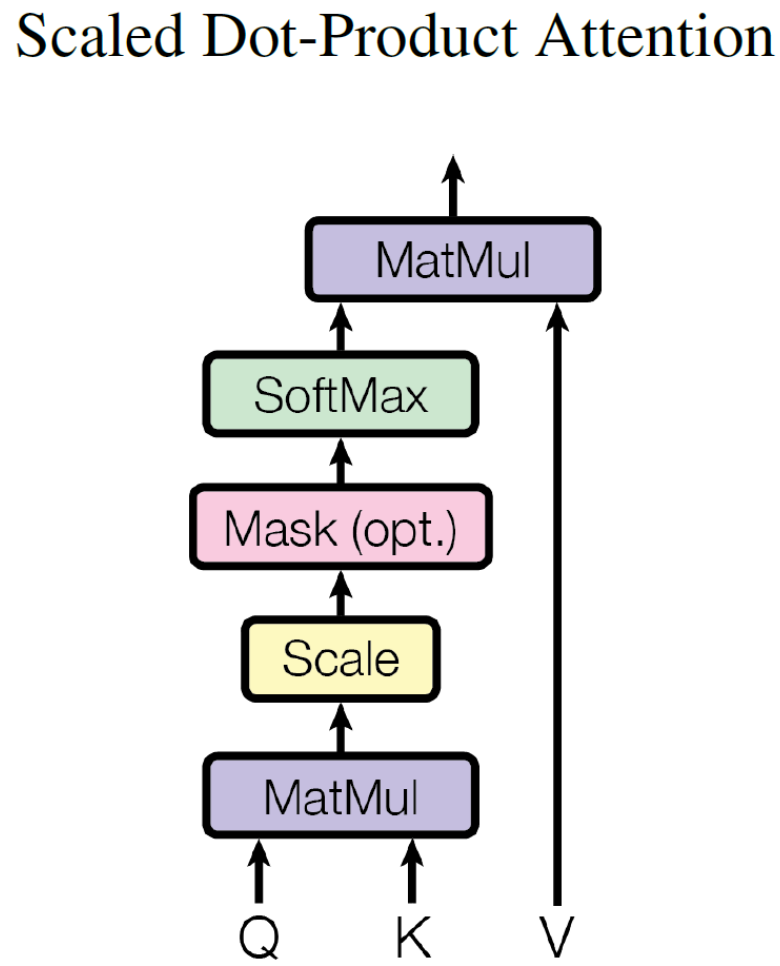
query[17] # “making”

key[24] # “difficult”

Relevance[17,24]=query[17] * key[24]
relevance of difficult to making

Attention

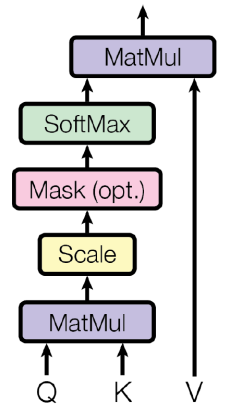
- Q: Query (output token)
- K: Key (input token)
- Relevance = softmax($\frac{Q \times K}{\sqrt{d_k}}$)
  - d_k is the dimension of Q or K
- V: Value (input token)
- Out = Relevance x V



Attention in "pseudo-code"

```
def attention(self, X_in:List[Tensor]):
    # For every token transform previous layer's out
    for i in range(self.sequence_lenght):
        query[i] = self.Q * X_in[i]
        key[i]   = self.K * X_in[i]
        value[i] = self.V * X_in[i]
```

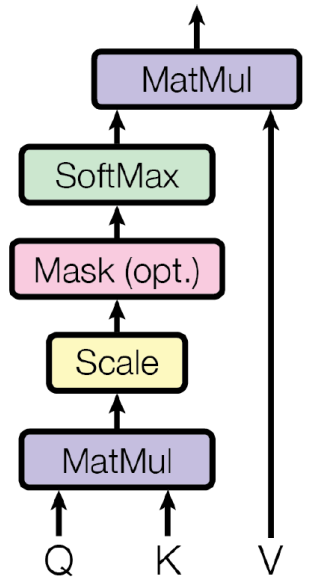
Scaled Dot-Product Attention



```
def attention(self, X_in:List[Tensor]):
        # For every token transform previous layer's out
        for i in range(self.sequence_lenght):
                query[i] = self.Q * X_in[i]
                key[i]   = self.K * X_in[i]
                value[i] = self.V * X_in[i]

        # Compute output values, one at a time
        for i in range(self.sequence_lenght):
                this_query = query[i]
                # how relevance is each input to this output?
                for j in rang(self.sequence_lenght):
                        relevance[j] = this_query * key[j]
                # normalize relevance score to sum to 1
                relevance = scaled_softmax(relevance)
                # compute a weighted sum of values
                out[i]   = 0 # out[i] is a vector
                 for j in rang(self.sequence_lenght):
                        out[i] += relevance[j] * value[j]
        return out
```

Scaled Dot-Product Attention



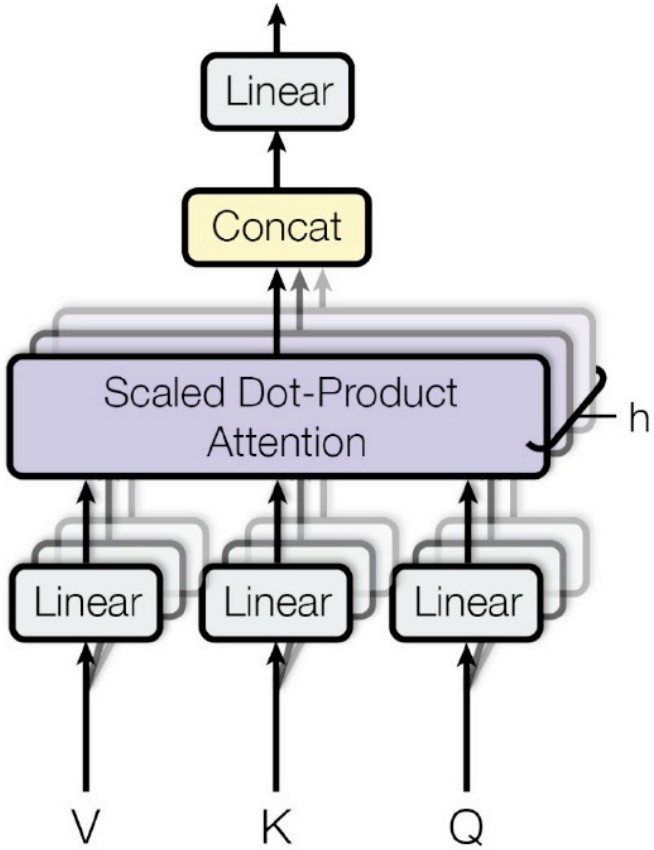
Multi-Headed Attention (i)

- Clever, important innovation.
  - Not that hard.
- Just do that same thing 8 times with different Q,K,V matrices.
- Let the network learn 8 different semantic meanings of attention.
  - E.g., One grammar, one for vocabulary, one for conjugation, etc.
  - Very flexible mechanism for sequence processing.

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q ; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K ; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V \quad (10.17)$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (10.18)$$

$$\mathbf{A} = \text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O \quad (10.19)$$



Multi-Headed Attention (ii)

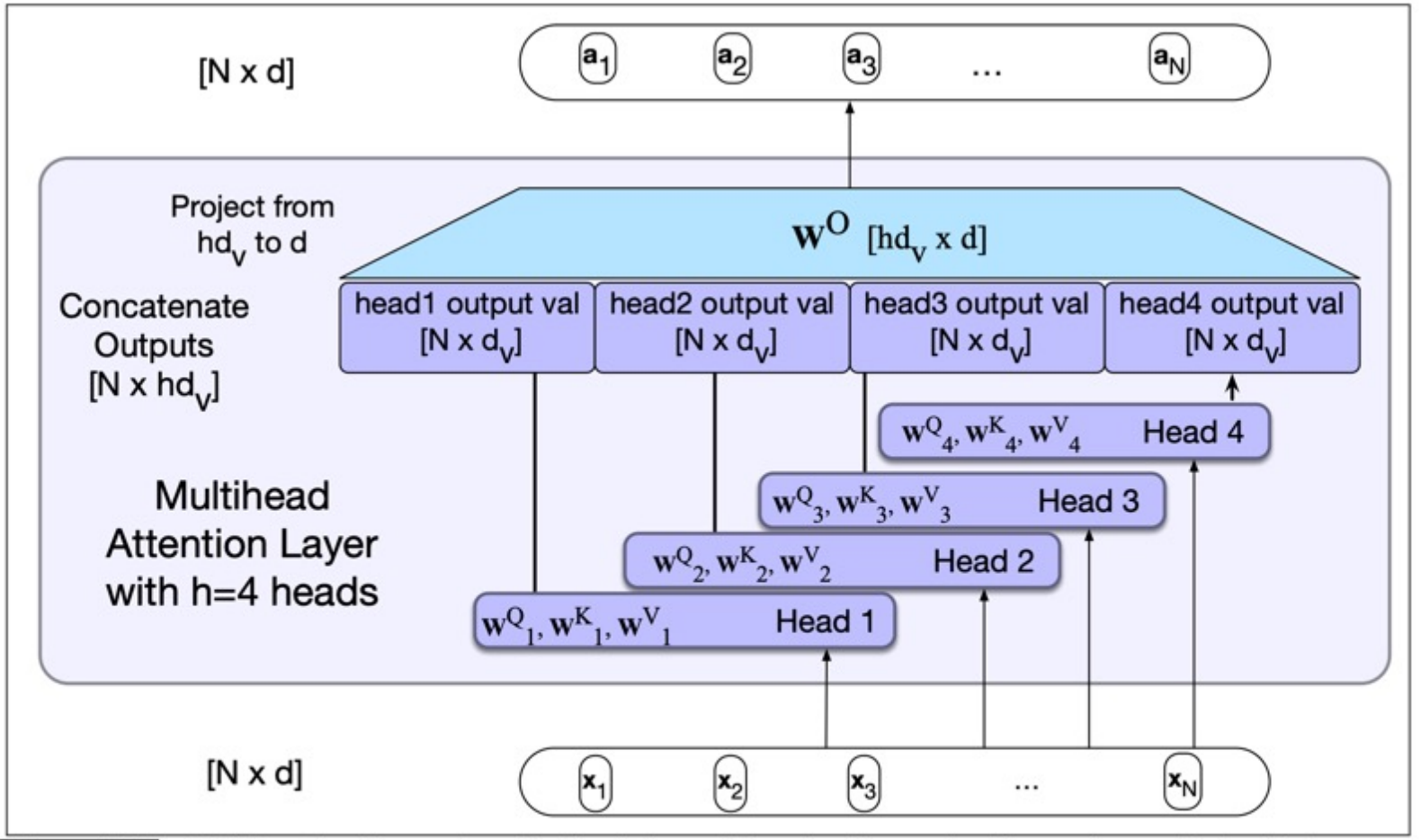
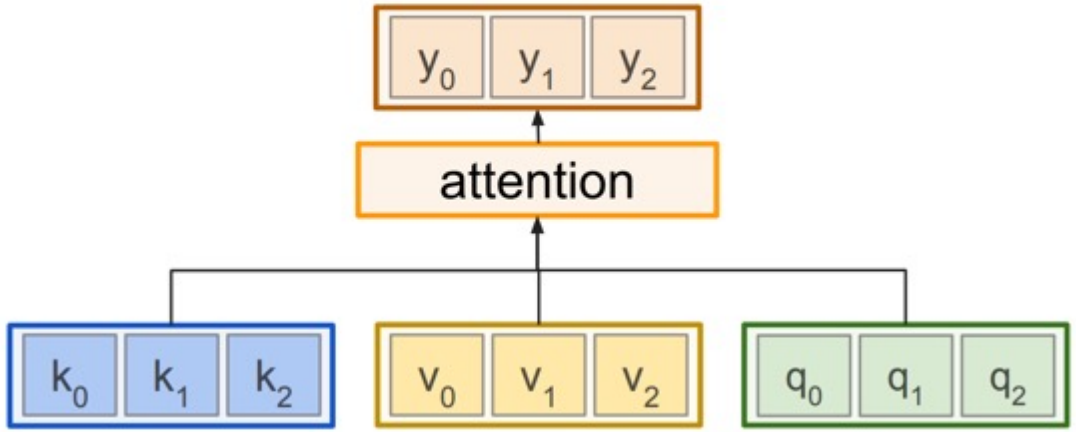
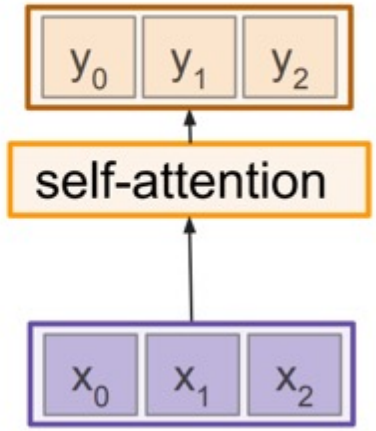


Figure 10.5 Multihead self-attention: Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected to d, thus producing an output of the same size as the input so the attention can be followed by layer norm and feedforward and layers can be stacked.

General attention versus self-attention





SIT330-770: Natural Language Processing

Week 10.3 – The Encoder Transformer Block

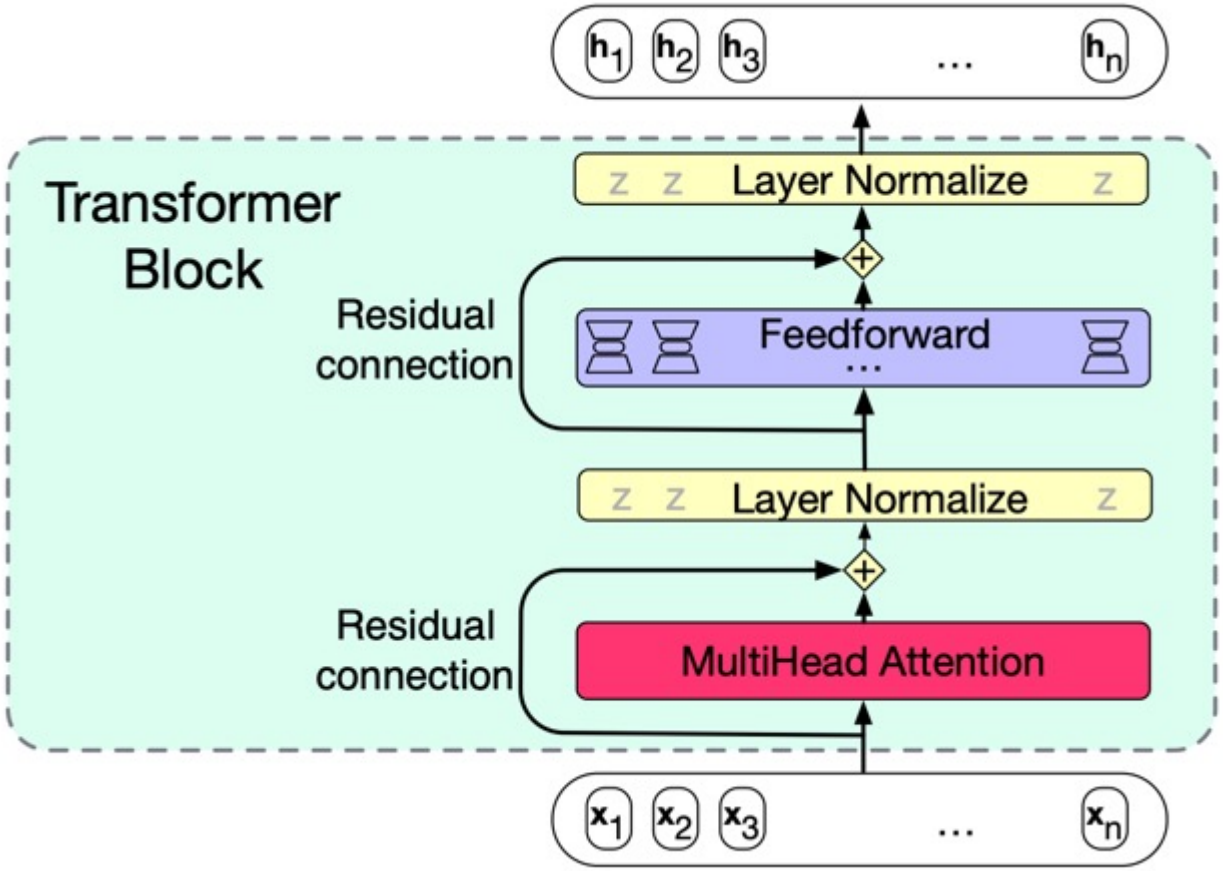
Week 10.3 – The Encoder Transformer Block

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

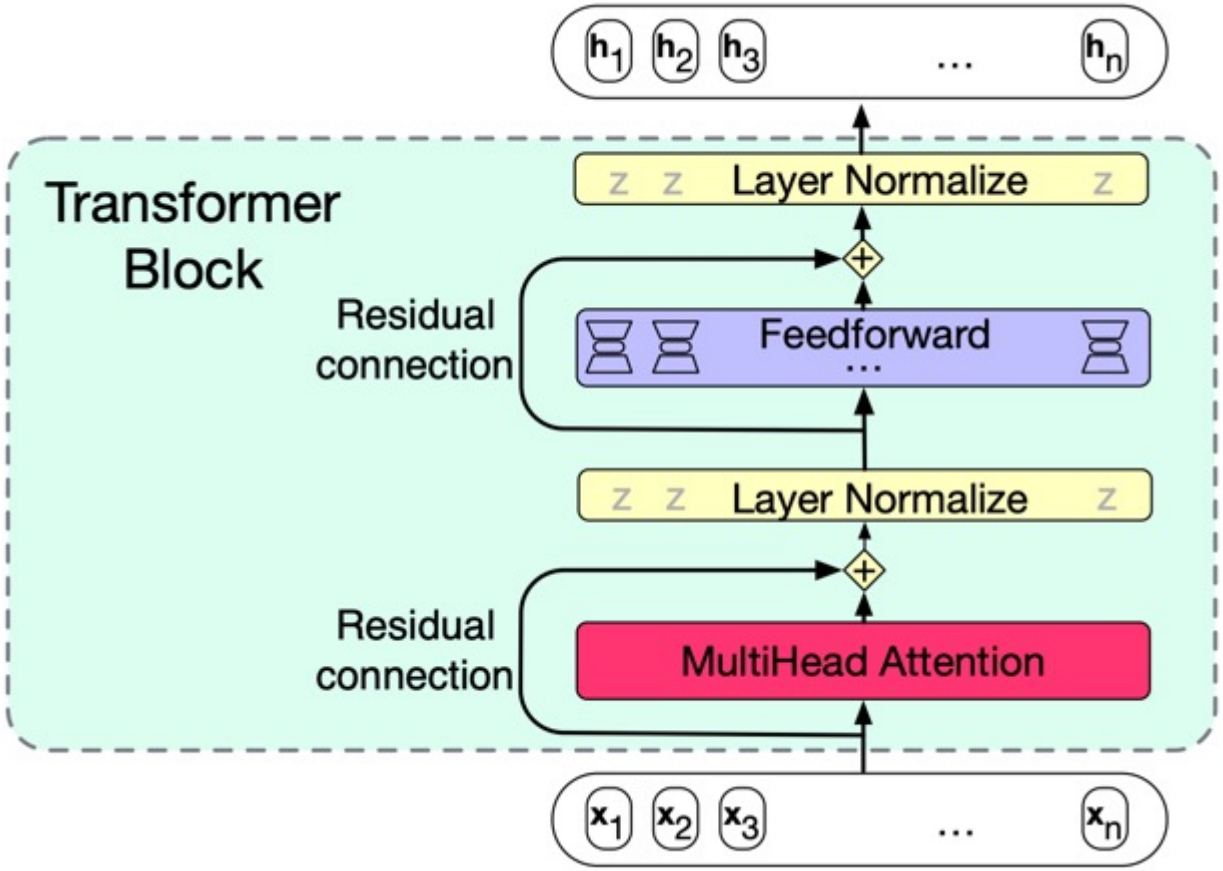
The Encoder Transformer Block

- The **Encoder Transformer block** includes four kinds of layers:
  - A self-attention layer
  - A feedforward layer
  - Residual connections
  - Normalizing layers

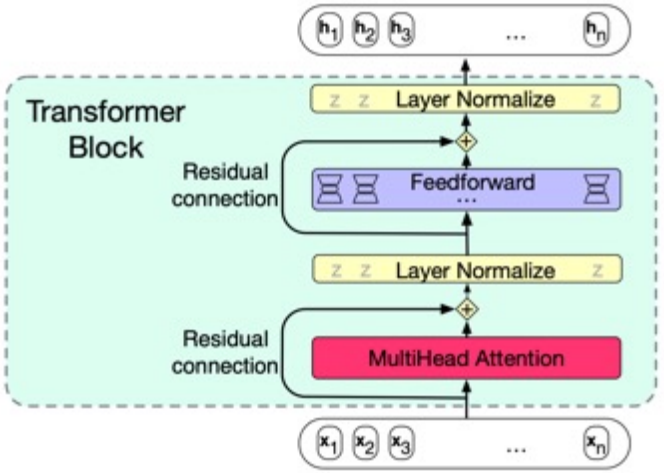


The Encoder Transformer Block: Residual connection

- Residual connections enable information to bypass intermediate layers, facilitating improved learning and direct access to lower layer information.
- In Transformers, residual connections involve adding a layer's input vector to its output vector before forwarding it.



The Encoder Transformer Block: Layer Normalization



- **Layer Normalization Process:**
  - Calculates mean (μ) and standard deviation (σ) over vector elements for normalization.
  - Normalizes vector components by subtracting mean and dividing by standard deviation, yielding a new vector with zero mean and unit standard deviation.

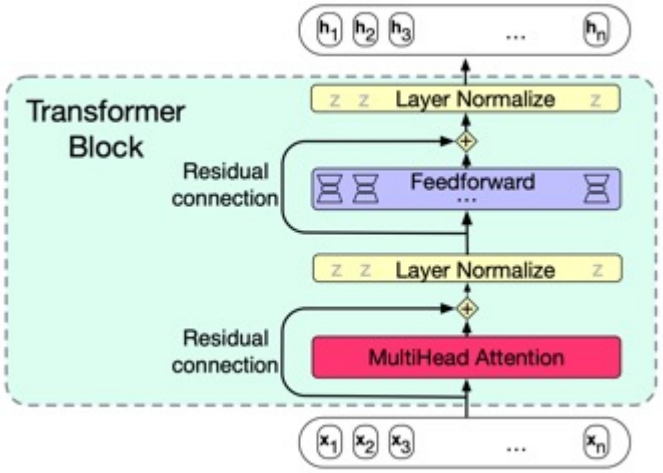
$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2} \qquad \mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i \qquad \hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

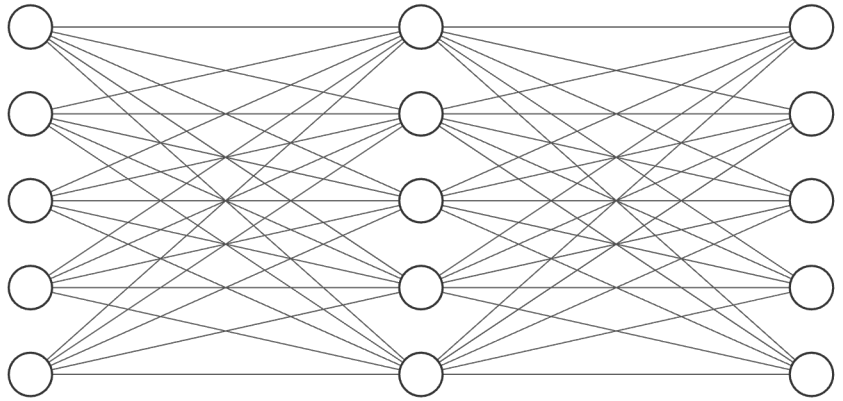
- **Learnable Parameters:**
  - Introduces two learnable parameters, γ (gain) and β (offset), in standard layer normalization implementation.

$$\text{LayerNorm} = \gamma\,\hat{\mathbf{x}} + \beta$$

The Encoder Transformer Block: feedforward layer

- The feedforward layer consists of N position-wise networks, each positioned independently.
  - Each network is a fully-connected 2-layer neural network, comprising one hidden layer and two weight matrices.
- While the weights remain consistent across positions, the parameters differ from layer to layer.
- Unlike attention mechanisms, the feedforward networks operate independently at each position, enabling parallel computation.

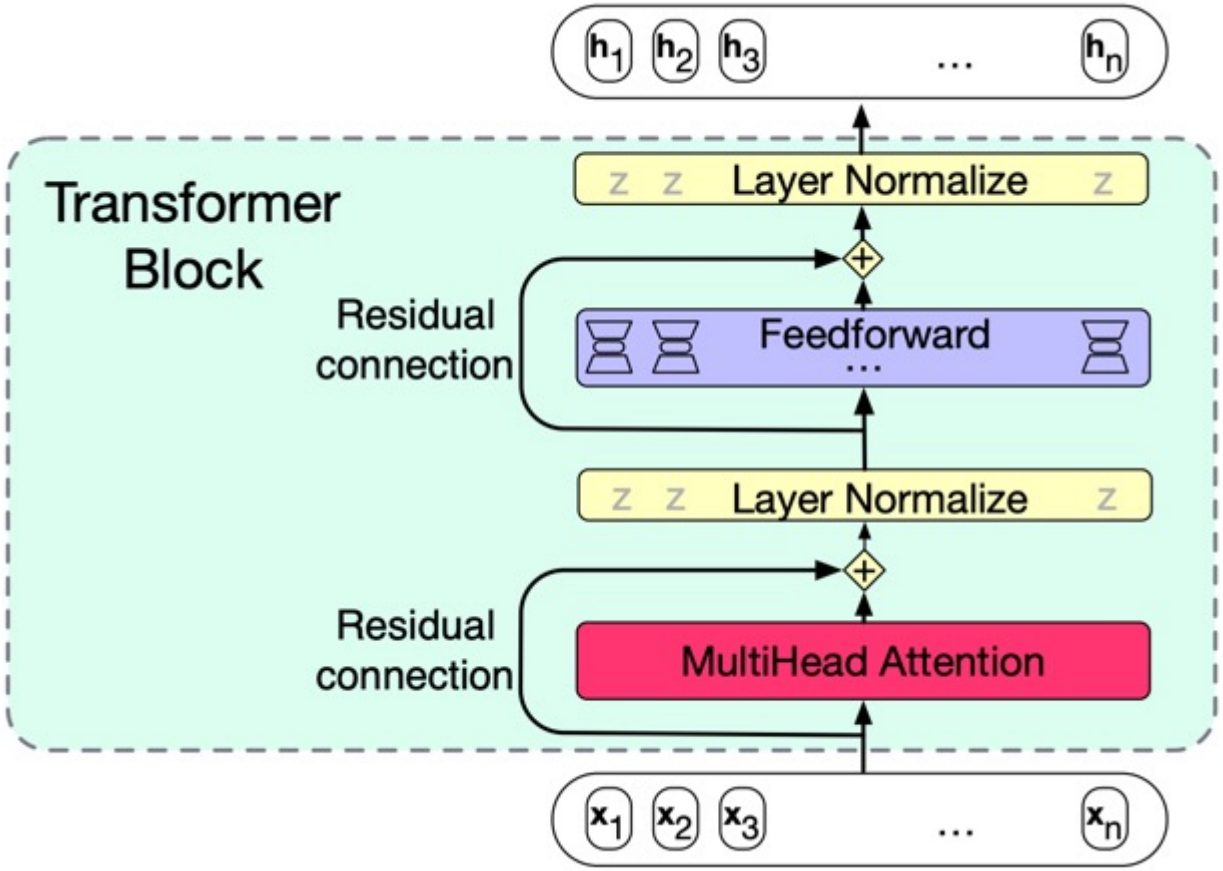




The Encoder Transformer Block: Putting it all together

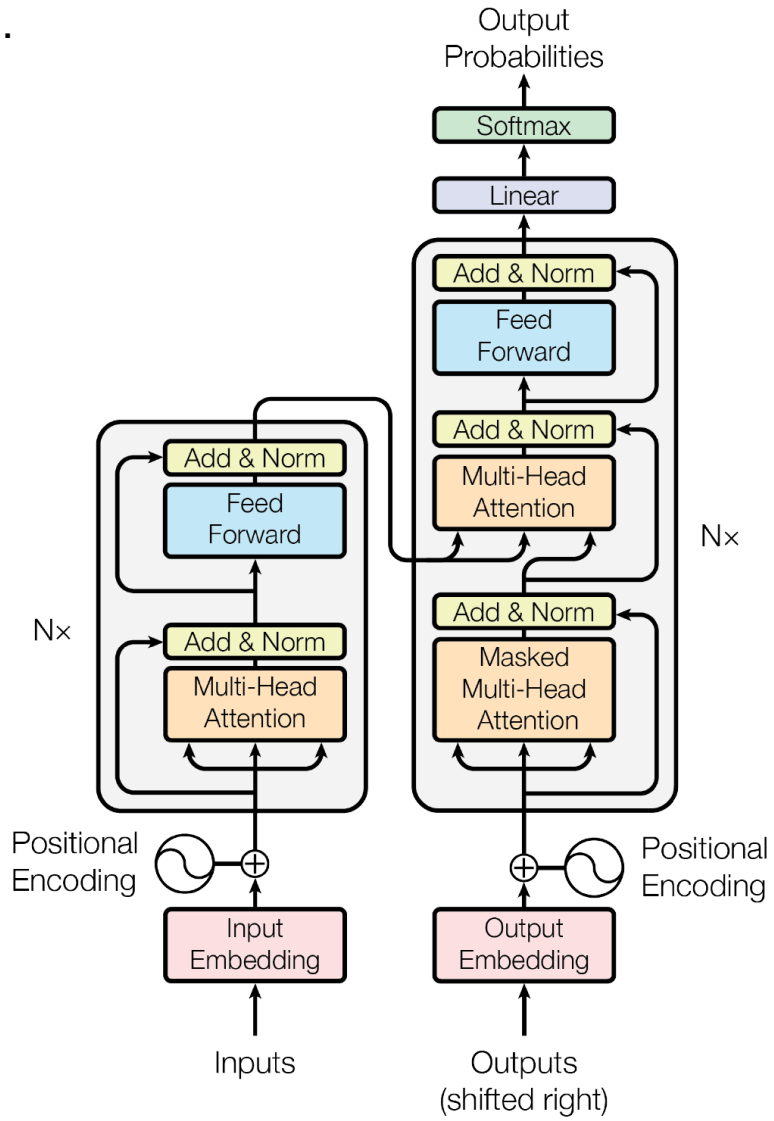
- The **Transformer block** includes four kinds of layers:
  - A self-attention layer
  - Residual connections
  - Normalizing layers
  - A feedforward layer

$$
\begin{aligned}
\mathbf{T}^1 &= \text{SelfAttention}(\mathbf{X}) \\
\mathbf{T}^2 &= \mathbf{X} + \mathbf{T}^1 \\
\mathbf{T}_3 &= \text{LayerNorm}(\mathbf{T}^2) \\
\mathbf{T}^4 &= \text{FFN}(\mathbf{T}^3) \\
\mathbf{T}^5 &= \mathbf{T}^4 + \mathbf{T}^3 \\
\mathbf{H} &= \text{LayerNorm}(\mathbf{T}^5)
\end{aligned}
$$



Encoder-Decoder Transformers

Ashish Vaswani et. Attention Is All You Need. NIPS 2017.



SIT330-770: Natural Language Processing

Week 10.4 – The Input: Embeddings for Tokens

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

The input X

- **Source of Input X**:
  - Originates from a sequence of N tokens, where N represents the number of tokens in the context.
  - Matrix X has dimensions $[N \times d]$ where d is the dimension of each embedding.
- **Composition of Embeddings**:
  - The transformer model computes two separate embeddings:
    - **Input Token Embedding**: Specific to each token in the sequence.
    - **Input Positional Embedding**: Reflects the position of each token within the sequence.

Input Token Embedding

- Each token's initial representation is a vector of dimension d.
- These embeddings evolve through transformer layers, incorporating contextual nuances.
- Embedding Storage:
  - Stored in matrix E with shape [$|V| \times d$], where $|V|$ is the vocabulary size.
- Token Conversion and Indexing:
  - Tokens first converted to vocabulary indices using techniques like BPE or SentencePiece (discussed in Week 3).
  - Example: "thanks for all" converts to indices [5, 4000, 10532, 2224].
  - These indices are used to fetch the corresponding embeddings from E.

One-Hot Vector

- One-Hot Vector Representation:
  - Alternative method using one-hot vectors of size $|V|$.
  - Example: Word "thanks" represented as a vector where only the 5th position is 1, all others are 0.

```
[0 0 0 0 1 0 0 ... 0 0 0 0]
 1 2 3 4 5 6 7 ...  ... |V|
```

- Multiplying by a one-hot vector that has only one non-zero element xi = 1 simply selects out the relevant row vector for word i, resulting in the embedding for word i,

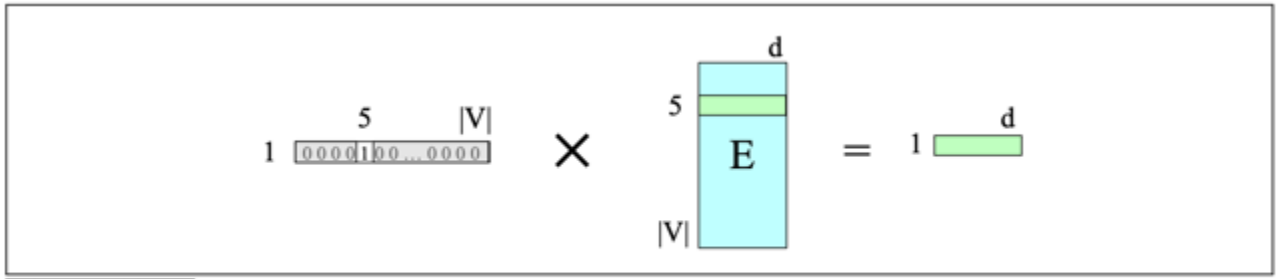


Figure 10.10 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.

One-Hot Vector

- We can extend this idea to represent the entire token sequence as a matrix of onehot vectors, one for each of the N positions in the transformer's context window,

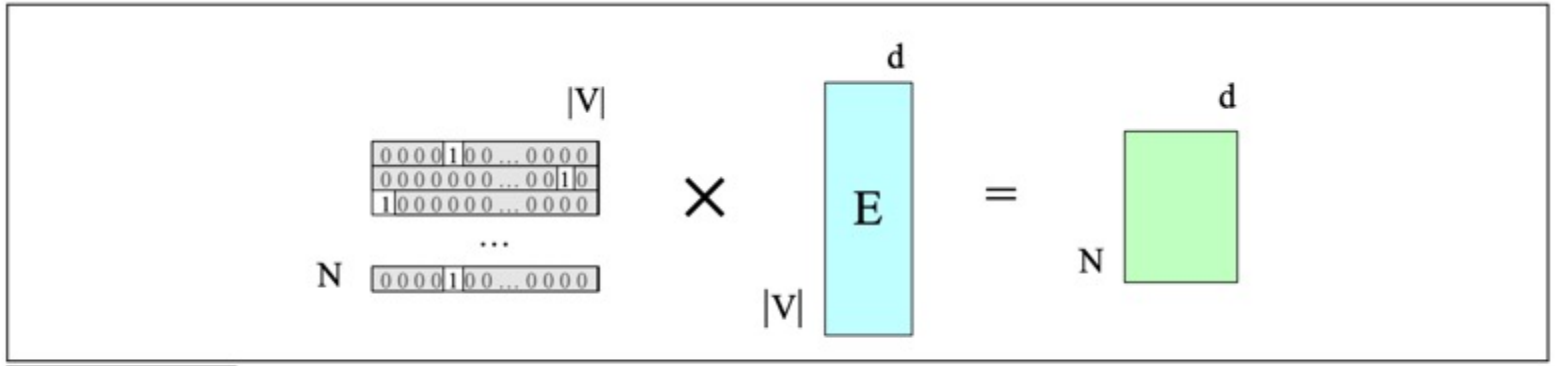


Figure 10.11 Selecting the embedding matrix for the input sequence of token ids W by multiplying a one-hot matrix corresponding to W by the embedding matrix **E**.

SIT330-770: Natural Language Processing

Week 10.5 – The Input: Embeddings for Positions

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

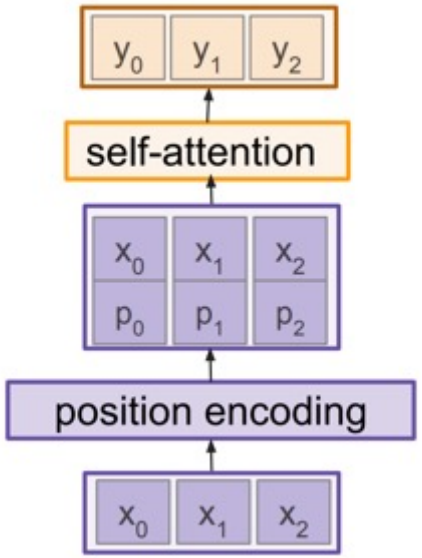
Input Positional Embedding

- How does a transformer model the position of each token in the input sequence?
  - With RNNs, information about the order of the inputs was built into the structure of the model, Not with Transformers
- Solution: **Positional**
  - Embeddings Modify the input embeddings by combining them with positional embeddings specific to each position in an input sequence

Positional Embedding

- Positional encoding assigns a unique representation to each position within a sequence to describe the location or order of entities.
- Limitations of Using Single Numbers for Position:
  - Using index values alone (e.g., sequence position numbers) is problematic for several reasons:
    - Large indices for long sequences can grow unmanageably large.
    - Normalizing indices between 0 and 1 creates inconsistencies across sequences of different lengths.
- Transformers' Approach to Positional Encoding:
  - Instead of single numbers, transformers map each position to a unique vector.
  - This results in a matrix where each row represents an object in the sequence combined with its positional information.

Positional encoding



- Concatenate/add special positional encoding p_j to each input vector x_j
- We use a function pos: N →Rd to process the position j of the vector into a d-dimensional vector
- So, pj = pos(j)

- Desiderata of pos(.) :
  1. It should output a unique encoding for each time-step (word's position in a sentence)
  2. Distance between any two time-steps should be consistent across sentences with different lengths.
  3. Our model should generalize to longer sentences without any efforts. Its values should be bounded.
  4. It must be deterministic.

Positional Encoding Layer in Transformers

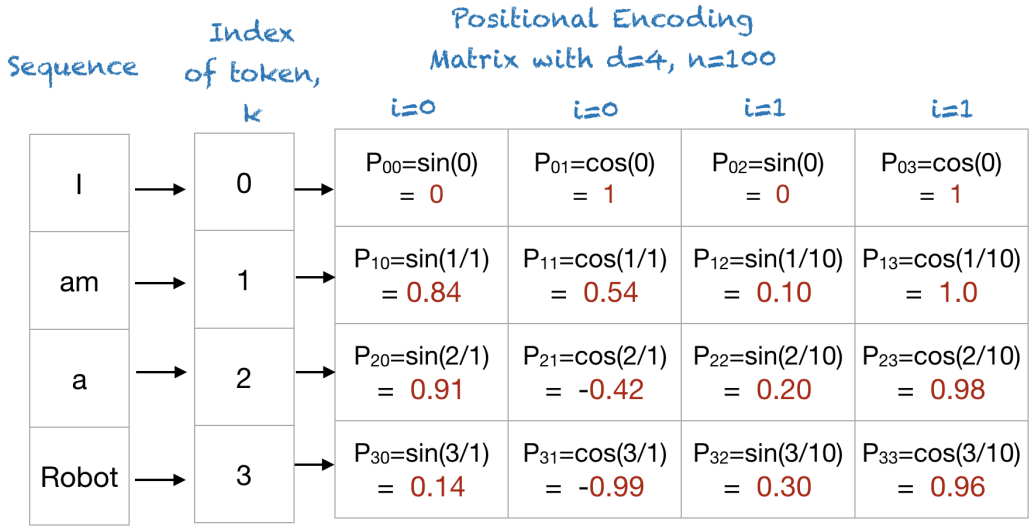
- The positional encoding is given by sine and cosine functions of va ues:

$$P(k,2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k,2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

  - k: Position of an object in the input sequence,
  - d: Dimension of the output embedding space
  - n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

- **Example:**
  - "I am a robot," with n=100 and d=4



Positional Encoding Matrix for the sequence 'I am a robot'

Coding the Positional Encoding Matrix from Scratch

```python
import numpy as np
import matplotlib.pyplot as plt

def getPositionEncoding (seq_len, d, n=10000):
        P = np.zeros((seq_len, d))
        for k in range(seq_len):
                for i in np.arange(int(d/2)):
                        denominator = np.power(n, 2*i/d)
                        P[k, 2*i] = np.sin(k/denominator)
                        P[k, 2*i+1] = np.cos(k/denominator)
    return P

P = getPositionEncoding(seq_len=4, d=4, n=100)
print(P)
```

```
[[ 0.          1.          0.          1.        ]
 [ 0.84147098  0.54030231  0.09983342  0.99500417]
 [ 0.90929743 -0.41614684  0.19866933  0.98006658]
 [ 0.14112001 -0.9899925   0.29552021  0.95533649]]
```

Positional Embeddings

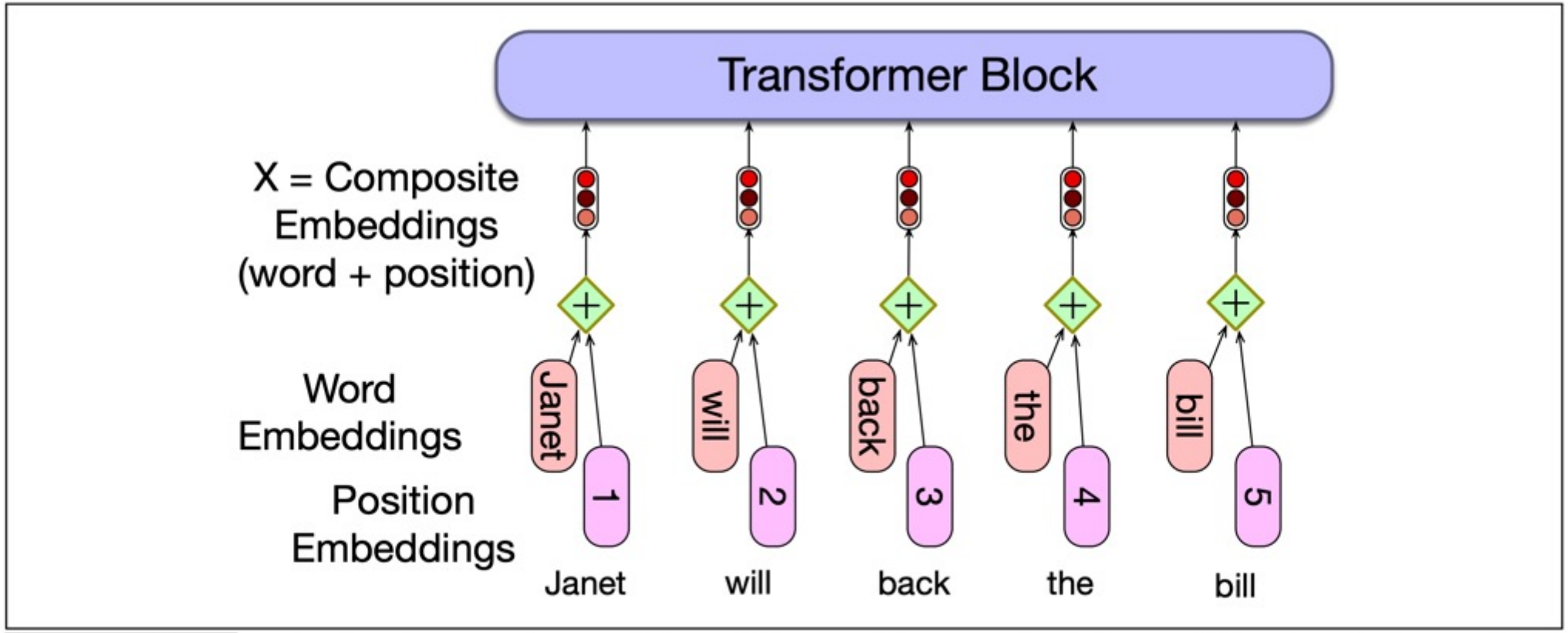


Figure 10.12 A simple way to model position: add an embedding of the absolute position to the token embedding to produce a new embedding of the same dimenionality.

SIT330-770: Natural Language Processing

Week 10.6 – The Task Specific Head

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

The Language Modeling Head

- The Language Modeling Head is an additional neural circuitry integrated with the basic transformer architecture.
- It enables specific tasks such as language modeling by enhancing the transformer's capabilities.
- Given a context of words, a Language Model assigns a probability to each possible next word
  - Calculating the probability of the next word “fish” given the context “Thanks for all the”: *P(fish|Thanks for all the)*

The Language Modeling Head

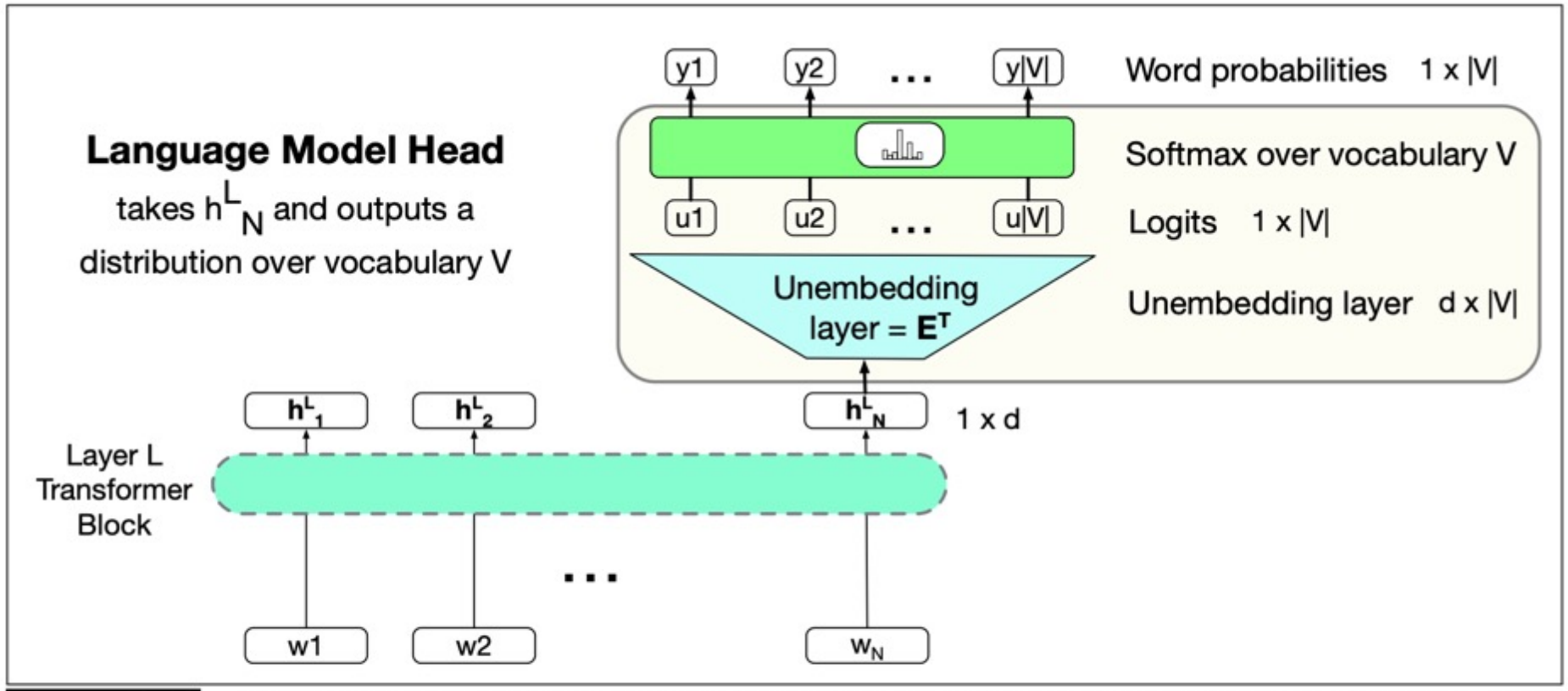
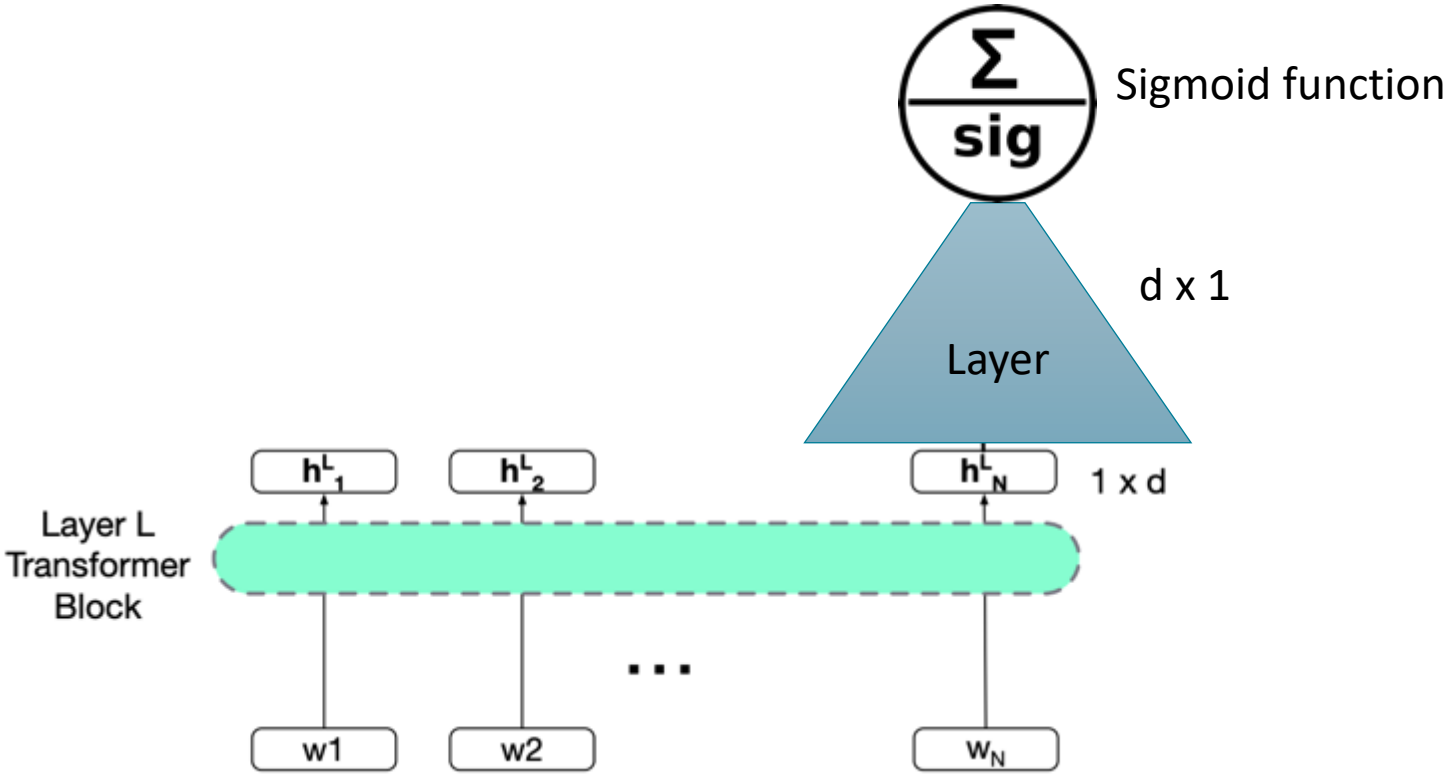


Figure 10.13 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer ($\mathbf{h}^L_N$) to a probability distribution over words in the vocabulary V.

The Classification Head



SIT330-770: Natural Language Processing

Week 10.7 – BERT: Bidirectional Encoder Representations from Transformers

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Bidirectional Transformer Encoders

- The focus of bidirectional encoders is on computing contextualized representations of the tokens in an input sequence that are generally useful across a range of downstream applications

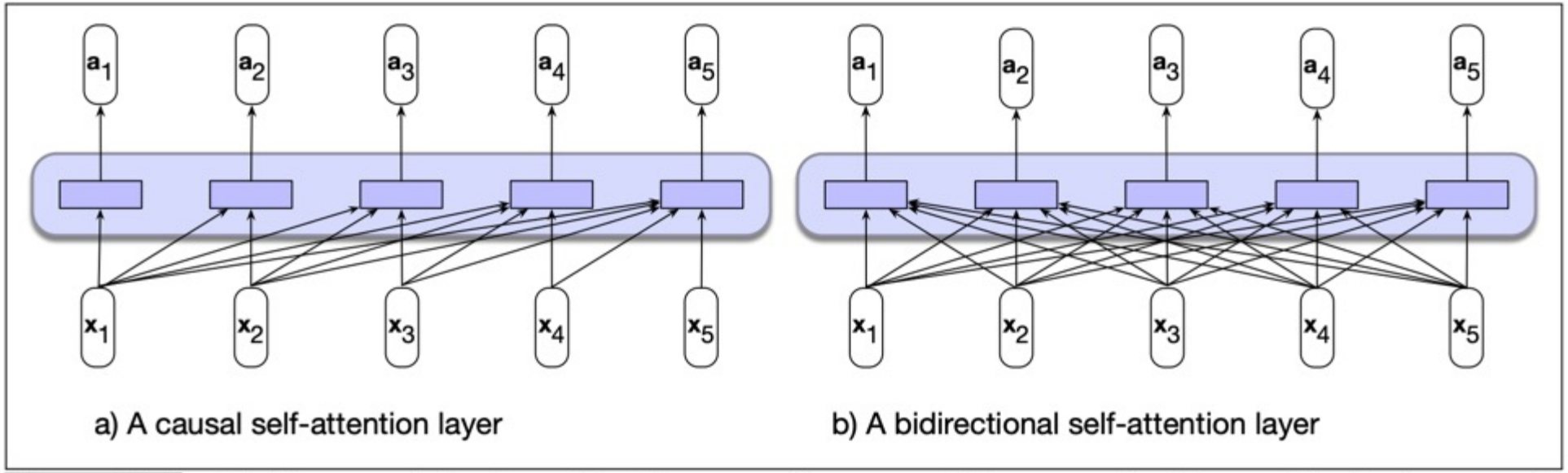
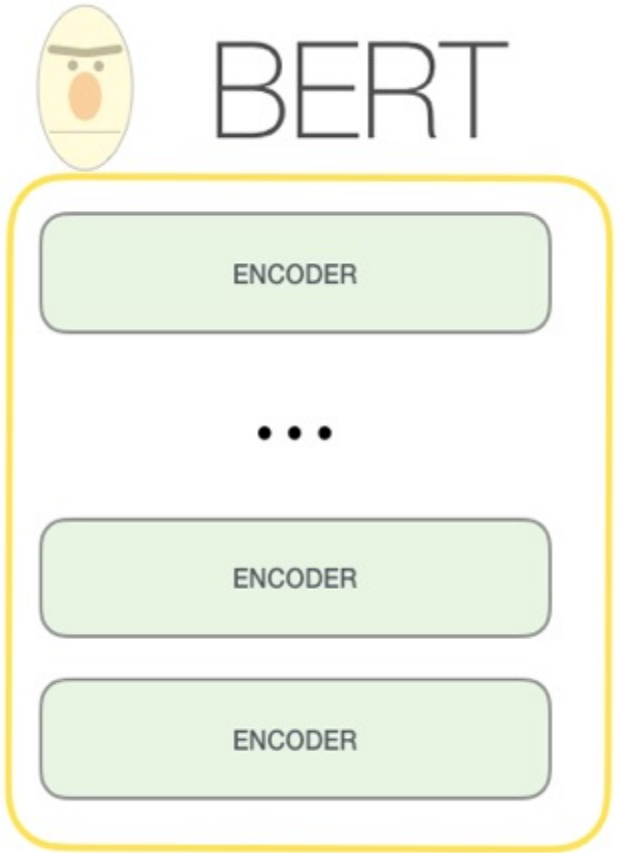


Figure 11.1 (a) The causal, backward looking, transformer model we saw in Chapter 10. Each output is computed independently of the others using only information seen earlier in the context. (b) Information flow in a bidirectional self-attention model. In processing each element of the sequence, the model attends to all inputs, both before and after the current one.

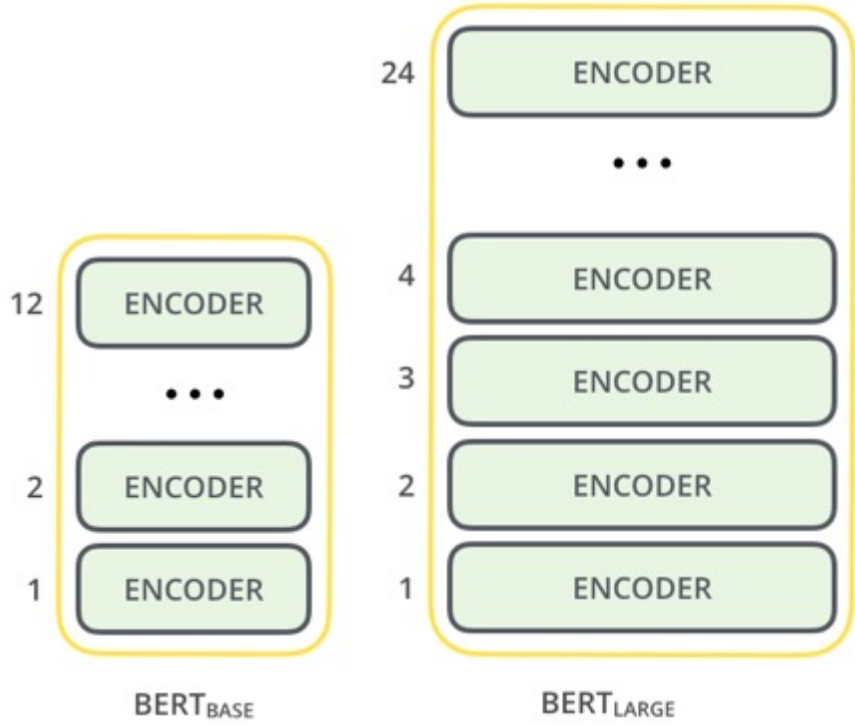
BERT architecture

- Transformers is an **Encoder-Decoder** architecture
  - A transformer uses Encoder stack to model input, and uses Decoder stack to model output (using input information from encoder side).
- BERT
  - If we are only interested in training a language model for the input for some other tasks, then we do not need the Decoder of the transformer, that gives us BERT.

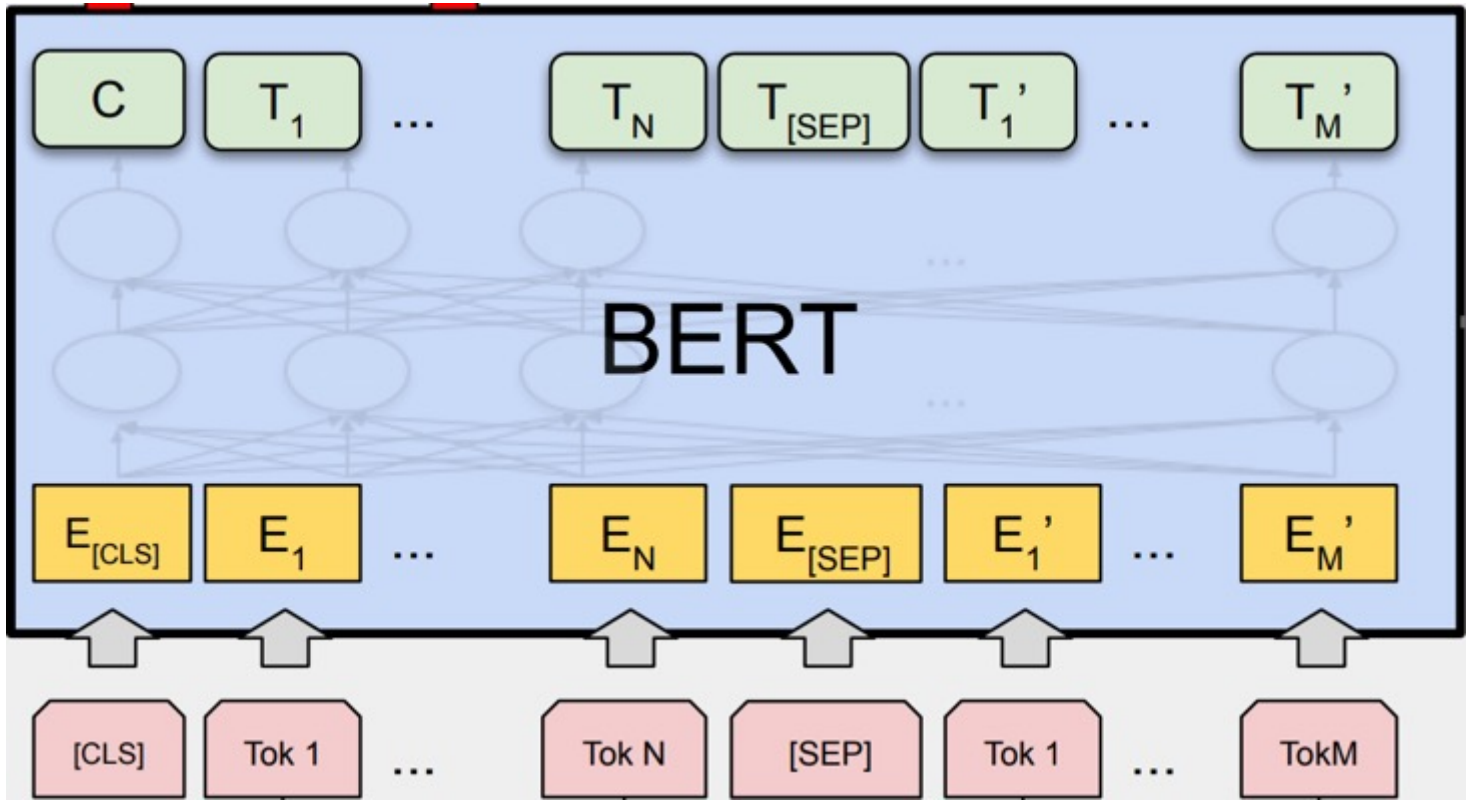


Model architecture

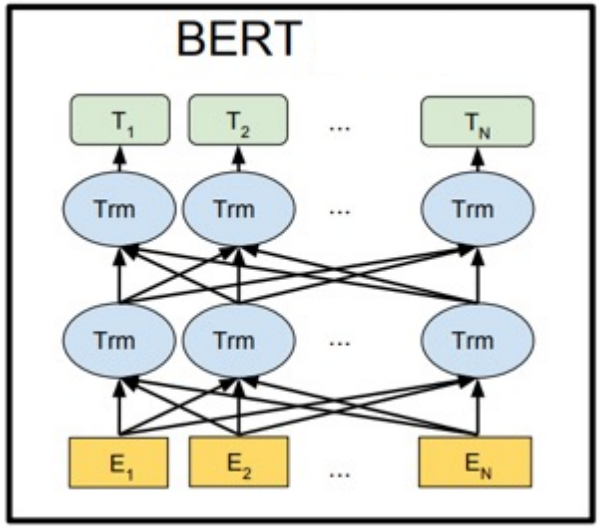
- Two models with different sizes were investigated
  - $BERT_{BASE}$: L=12, H=768, A=12, Total Parameters=110M
  - $BERT_{LARGE}$: L=24, H=1024, A=16, Total Parameters=340M
    - L: number of layers (Transformer blocks), H: the hidden size, A: the number of self-attention heads.
    - Cased and uncased versions.
    - Multiple-languages.
  - Also available: $BERT_{Tiny}$, $BERT_{Mini}$, $BERT_{Small}$, $BERT_{Medium}$

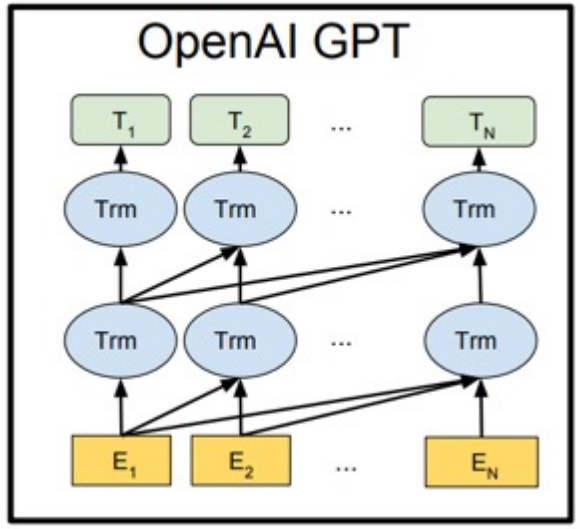


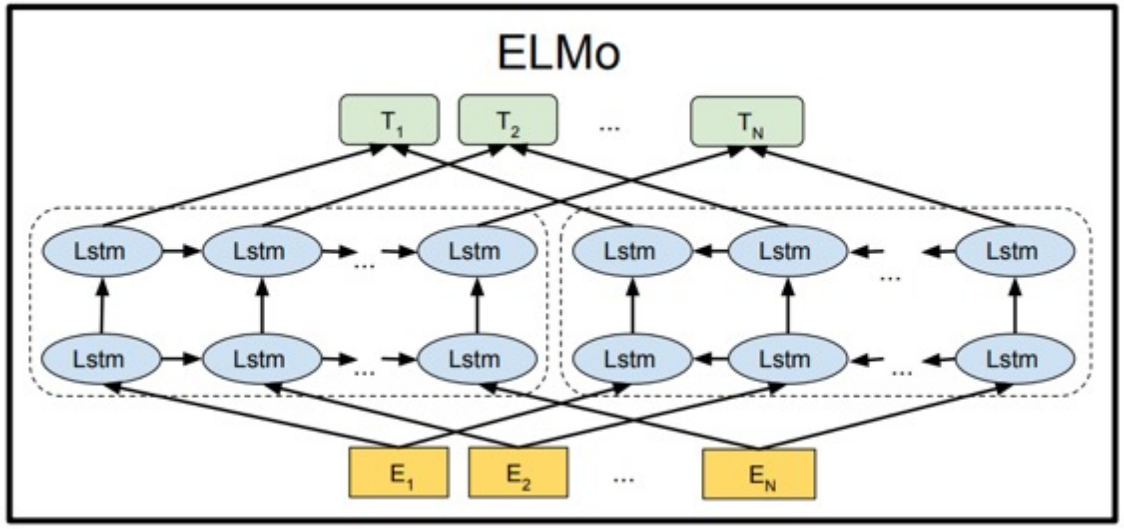
BERT in Action



Differences in pre-training model architectures: BERT, OpenAI GPT, and ELMo

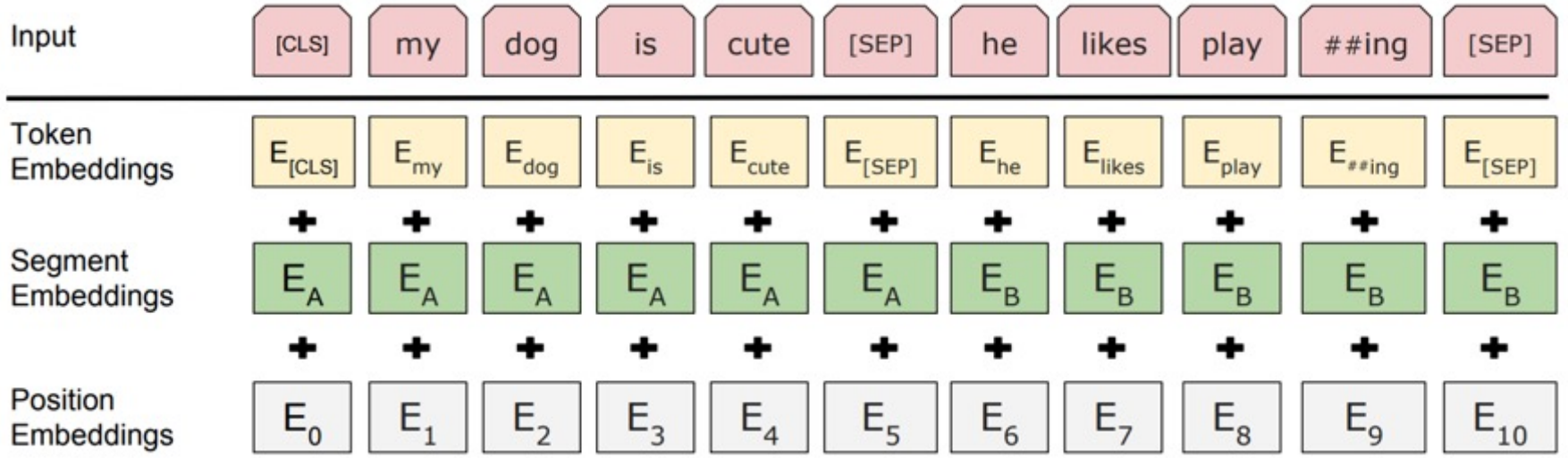






- GPT is built using transformer decoder blocks. BERT, on the other hand, uses transformer encoder blocks.
- GPT is auto-regressive in nature. BERT is not.
  - In losing auto-regression, BERT gained the ability to incorporate the context on both sides of a word to gain better results.

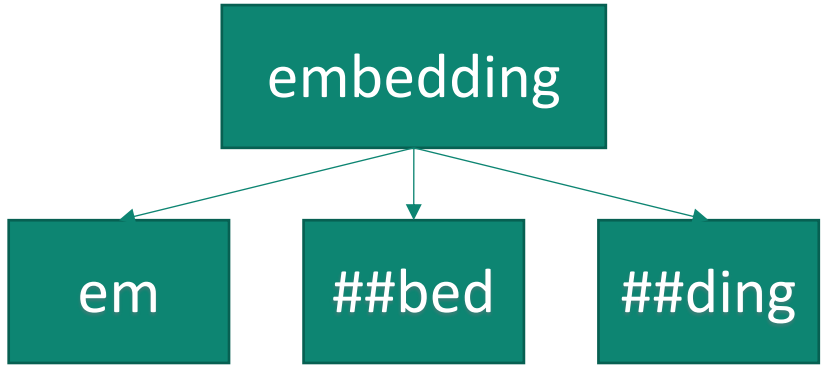
Input Representation

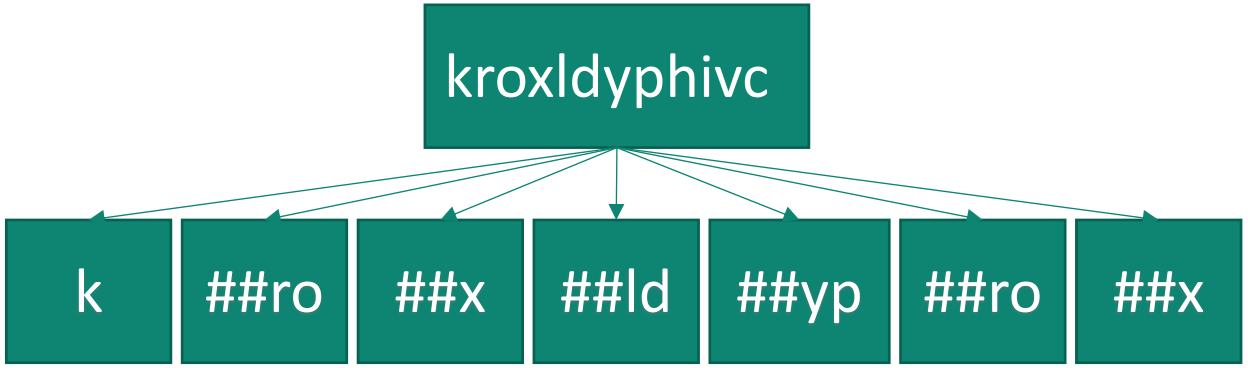


- **Token Embeddings:** Use pretrained WordPiece embeddings.
- **Segment Embeddings (Optional):** Added sentence embedding to every tokens of each sentence.
- **Position Embeddings:** Use learned Position Embeddings.
- Use **[CLS]** for the classification tasks.
- Separate sentences by using a special token **[SEP]**.

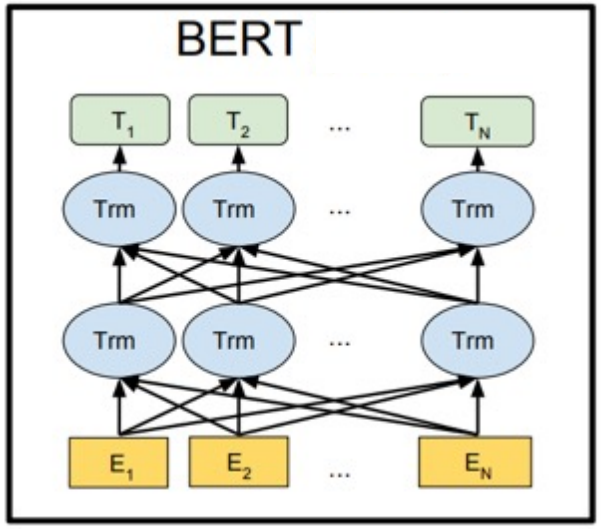
BERT's Vocabulary

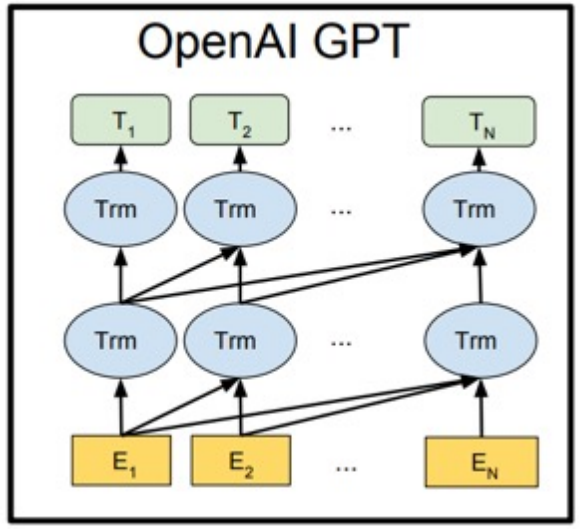
- BERT is pre-trained ➔ Vocabulary is fixed
  - The vocabulary contains **30,522** tokens.
  - How to deal with unknown words?
- Break down **unknown words** into **subwords**:
  - Using the **wordpiece** model.
- A subword exists for every character.
  - 2 types of subwords
    - All subwords start with "##"…
    - Except for the first subword in a word

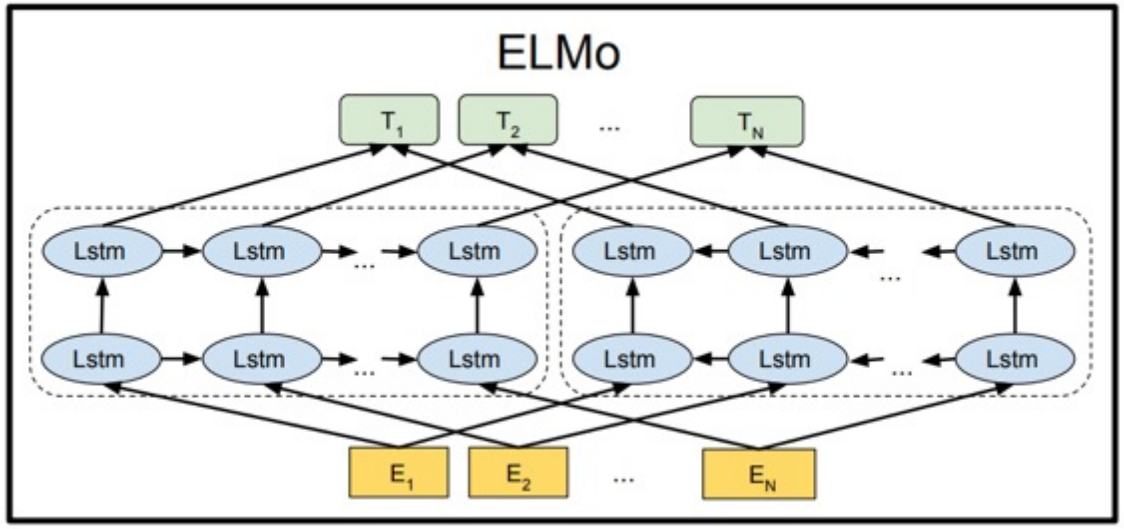




Differences in pre-training model architectures: BERT, OpenAI GPT, and ELMo







- GPT is built using transformer decoder blocks. BERT, on the other hand, uses transformer encoder blocks.
- GPT is auto-regressive in nature. BERT is not.
  - In losing auto-regression, BERT gained the ability to incorporate the context on both sides of a word to gain better results.

SIT330-770: Natural Language Processing

Week 10.8 – BERT pre-training

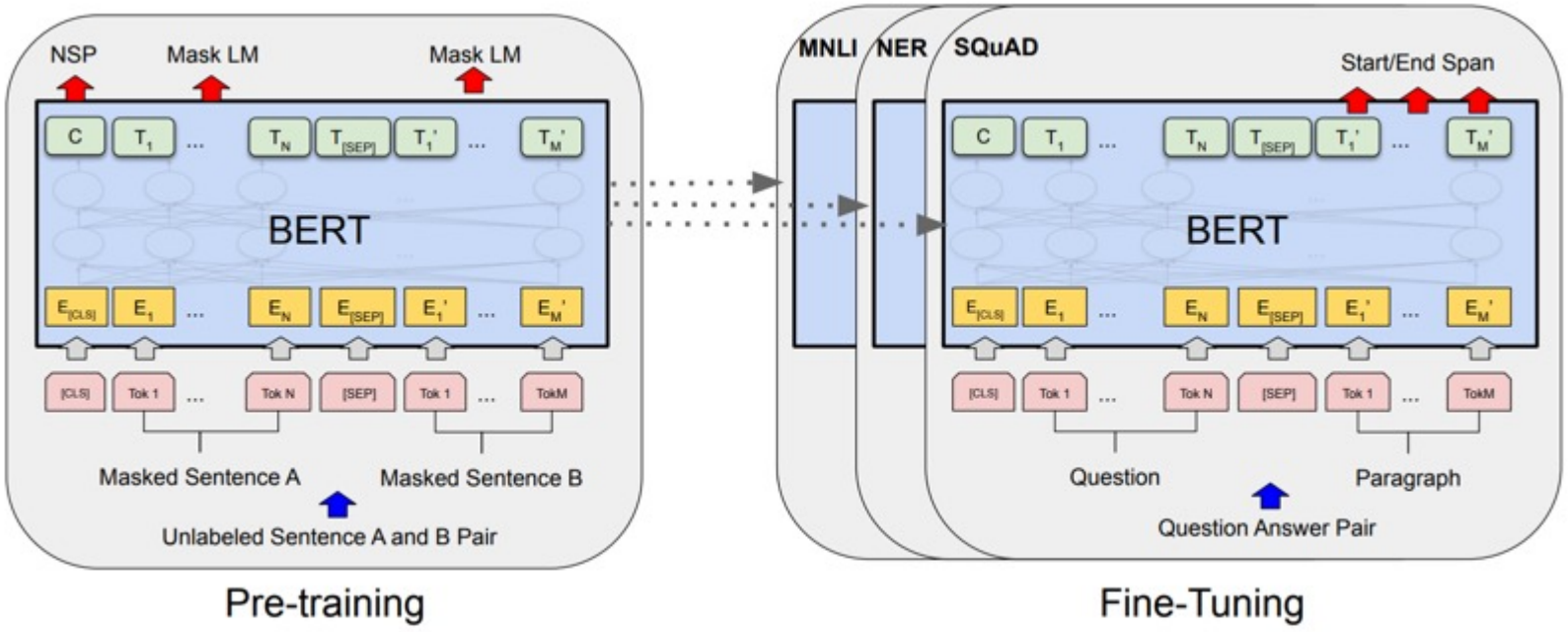
Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Pre-trained Models

- Pre-trained models are ML models that have been previously trained on a large dataset, typically on a general task
- These models can significantly reduce the computational cost and time required to develop ML applications by leveraging learned features and weights.
  - **Speeds Up Development**
    - By using models that are already trained, developers can focus on fine-tuning rather than starting from scratch.
  - **Improves Performance**
    - Pre-trained models often bring high levels of accuracy and efficiency, especially on complex tasks that require learning from large amounts of data.
  - **Resource Efficiency**
    - Saves resources by reducing the need for extensive training data and computational power.

Pre-training and Fine-tuning



- The same architectures are used in both pre-training and fine-tuning
- [CLS] is a special symbol added in front of every input example, and [SEP]is a special separator token
- **Pre-training**: two tasks are considered
  - **Masked LM**: mask some percentage of the input tokens at random, and then predict those masked tokens Mask 15% of all WordPiece tokens in each sequence at random
  - **Next Sentence Prediction**: understanding the relationship between two sentence (50% of positive pairs) Used BooksCorpus (800M words) and Wikipedia (2,500M words)

Pre-training
Task#1: Masked Language Model (MLM)

- The original approach to training bidirectional encoders is called **Masked Language Modeling (MLM)**
  - MLM uses unannotated text from a large corpus
  - Model is presented with a series of sentences from the training corpus, where a random
- sample of tokens from each training sequence is selected for use in the learning task. Once chosen, a token is used in one of three ways:
  - It is replaced with the unique vocabulary token [MASK]
  - It is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities
  - It is left unchanged
- In BERT, 15% of the input tokens in a training sequence are sampled for learning
  - Of these, 80% are replaced with [MASK], 10% are replaced with randomly selected tokens, and the remaining 10% are left unchanged

Pre-training
Task#1: Masked Language Model (MLM)

- The MLM training objective is to predict the original inputs for each of the masked
- tokens using a bidirectional encoder of the kind described in the last section
  - The **cross-entropy** loss from these predictions drives the training process for all the parameters in the model
  - Note that all the input tokens play a role in the self-attention process, but **only the sampled tokens** are used for learning
- Input:
  - Original input sequence is first tokenized using a subword model
  - The sampled items which drive the learning process are chosen from among the set of tokenized inputs
  - Word embeddings for all the tokens in the input are retrieved from the word embedding matrix and then combined with positional embeddings to form the input to the transformer

Pre-training
Task#1: Masked Language Model (MLM)

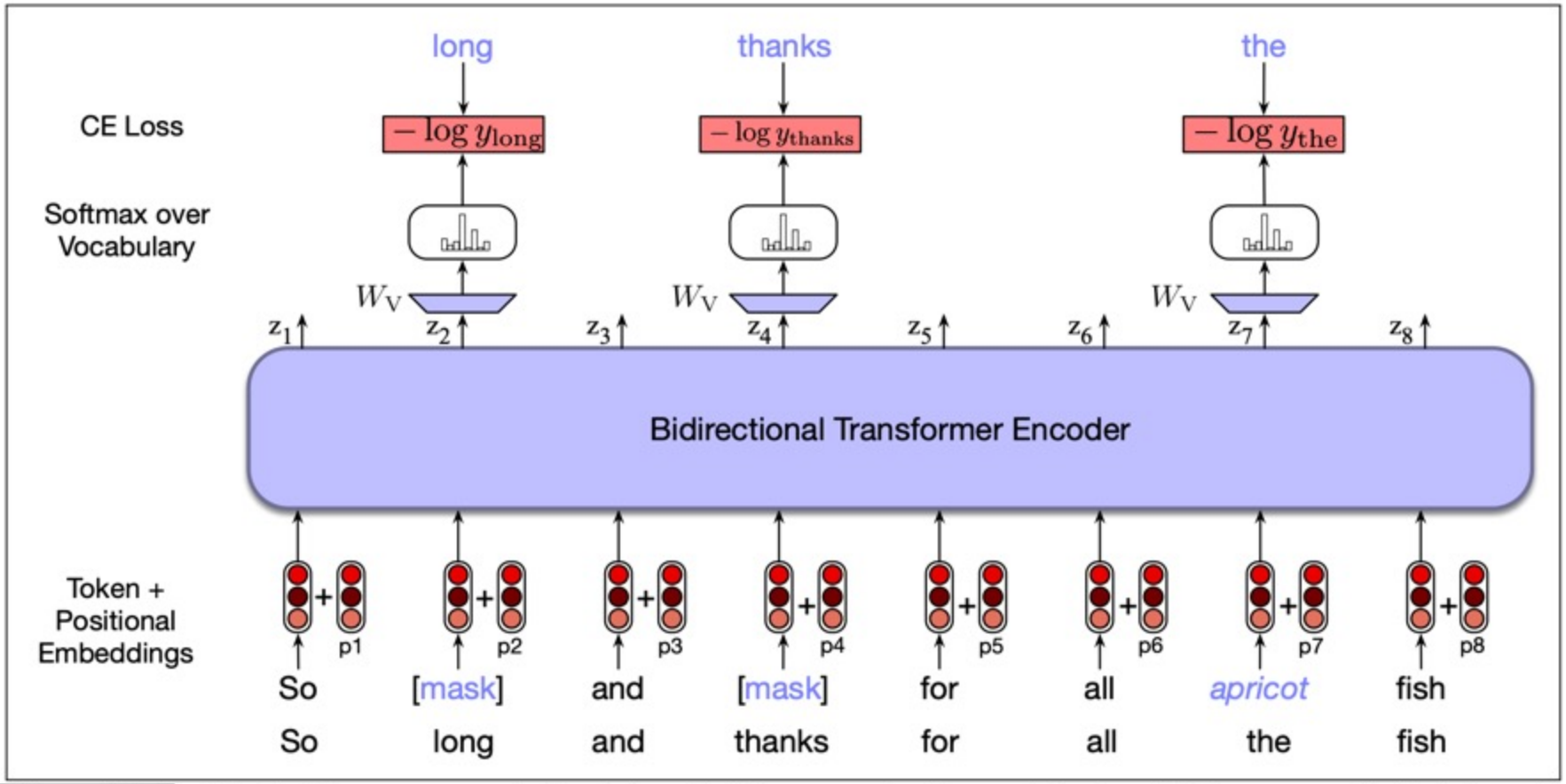


Figure 11.4 Masked language model training. In this example, three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word. The probabilities assigned by the model to these three items are used as the training loss. The other 5 words don't play a role in training loss. (In this and subsequent figures we display the input as words rather than subword tokens; the reader should keep in mind that BERT and similar models actually use subword tokens instead.)

Pre-training
Task#2: Next Sentence Prediction (NSP)

- An important class of applications involves determining the relationship between pairs of sentences
  - paraphrase detection (detecting if two sentences have similar meanings)
  - entailment (detecting if the meanings of two sentences entail or contradict each other)
  - discourse coherence (deciding if two neighboring sentences form a coherent discourse)
- To capture the kind of knowledge required for applications such as these, BERT introduced a second learning objective called Next Sentence Prediction (NSP)
- **Training:** The model is presented with pairs of sentences and is asked to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentences

Pre-training
Task#2: Next Sentence Prediction (NSP)

- In BERT, 50% of the training pairs consisted of positive pairs, and in the other 50% the second sentence of a pair was randomly selected from elsewhere in the corpus
  - The NSP loss is based on how well the model can distinguish true pairs from random pairs
- BERT introduces two new tokens to the input representation
  - After tokenizing the input with the subword model, the token **[CLS]** is prepended to the input sentence pair, and the token **[SEP]** is placed between the sentences and after the final token of the second sentence
  - During training, the output vector from the final layer associated with the **[CLS]** token represents the next sentence prediction

Pre-training
Task#2: Next Sentence Prediction (NSP)

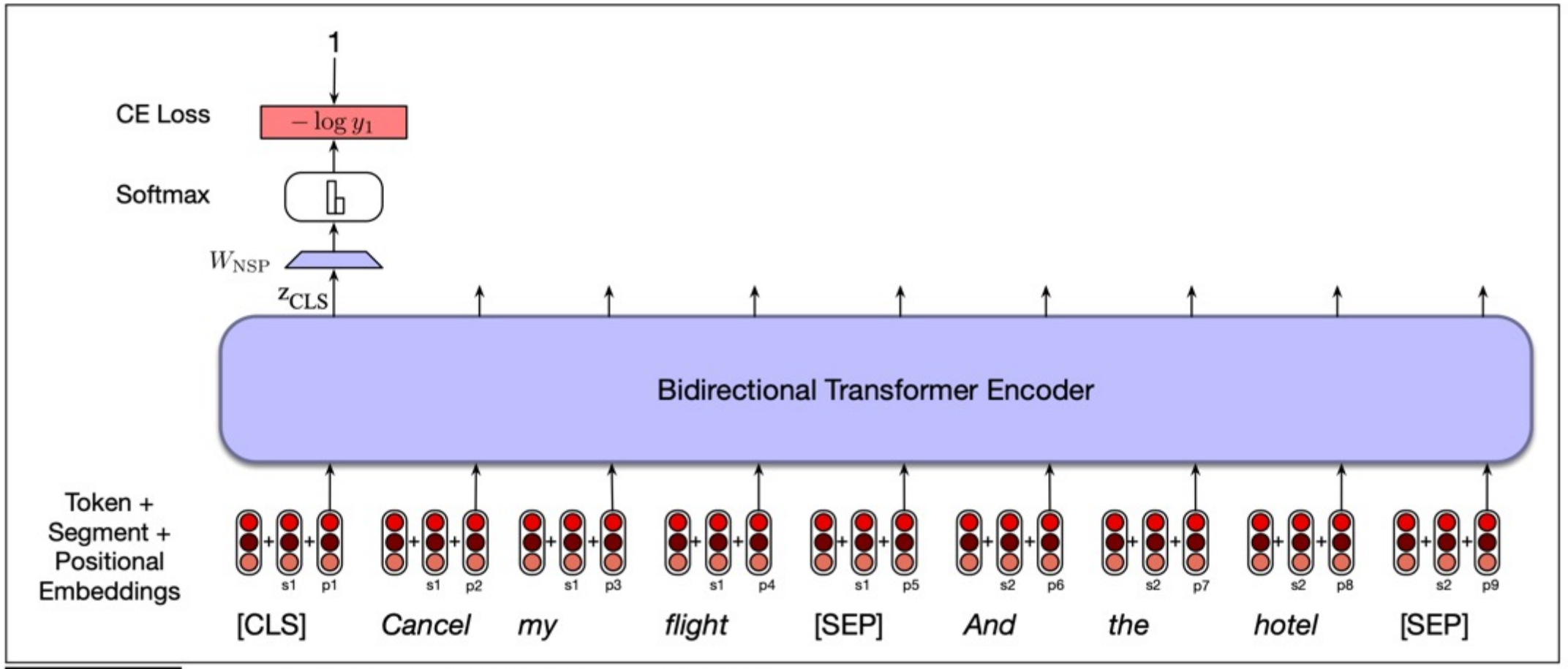


Figure 11.5 An example of the NSP loss calculation.

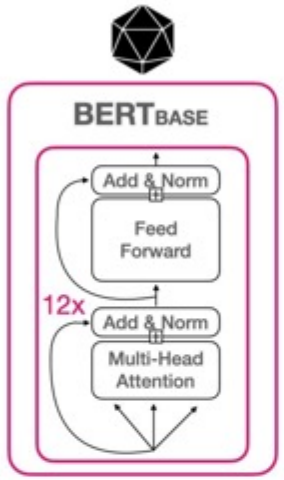
Pre-training procedure

- Training data: BooksCorpus (800M words) + English Wikipedia (2.5B words).
- To generate each training input sequences: sample two spans of text (A and B) from the corpus.
  - The combined length is ≤ 500 tokens.
  - 50% B is the actual next sentence that follows A and 50% of the time it is a random sentence from the corpus.
- The training loss is the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood.

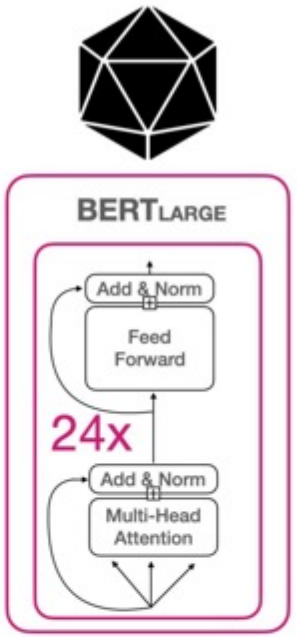
Under the Hood

ML Architecture Parts	Definition
Parameters	Number of learnable variables/values available for the model
Transformer Layers	Number of Transformer blocks. A transformer block transforms a sequence of word representations to a sequence of contextualized words (numbered representations)
Hidden Size	Layers of mathematical functions, located between the input and output, that assign weights (to words) to produce a desired result
Attention Heads	The size of a Transformer block
Processing	Type of processing unit used to train the model
Length of Training	Time it took to train the model

BERT Size & Architecture



110M Parameters



340M Parameters

	Transformer Layers	Hidden Size	Attention Heads	Parameters	Processing	Length of Training
BERTbase	12	768	12	110M	4 TPUs	4d
BERTlarge	24	1024	16	340M	16 TPUs	4d

SIT330-770: Natural Language Processing

Week 10.9 – BERT fine-tuning

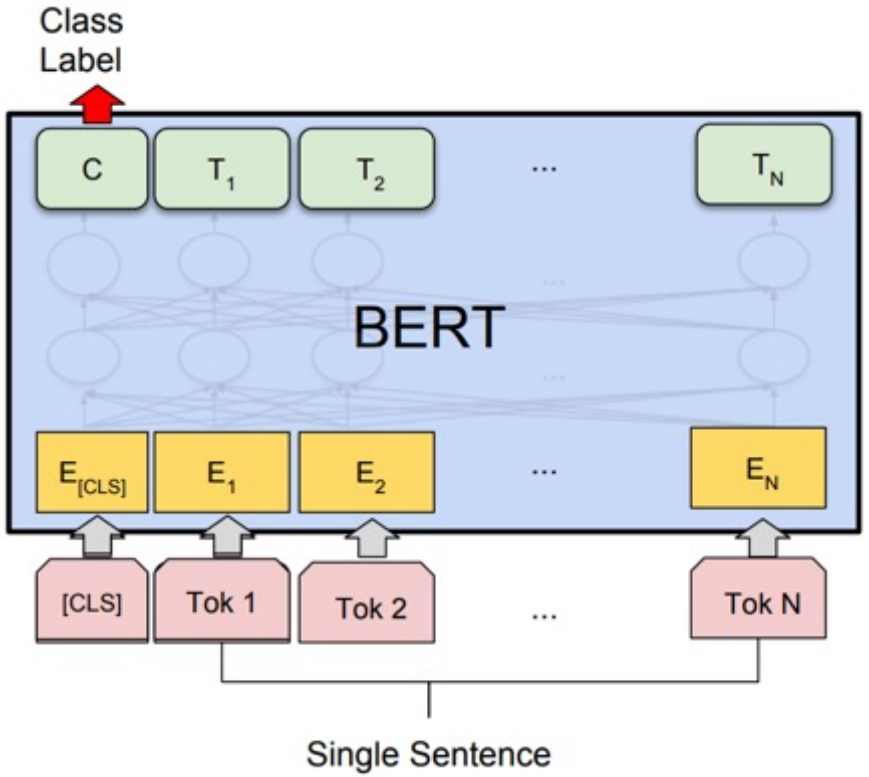
Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Fine-tuning procedure 1: Classification

- For sequence-level classification task
  - Obtain the representation of the input sequence by using the final hidden state (hidden state at the position of the special token [CLS]) $C \in R^H$
  - Just add a classification layer and use **softmax** to calculate label probabilities. Parameters $W \in R^{K \times H}$

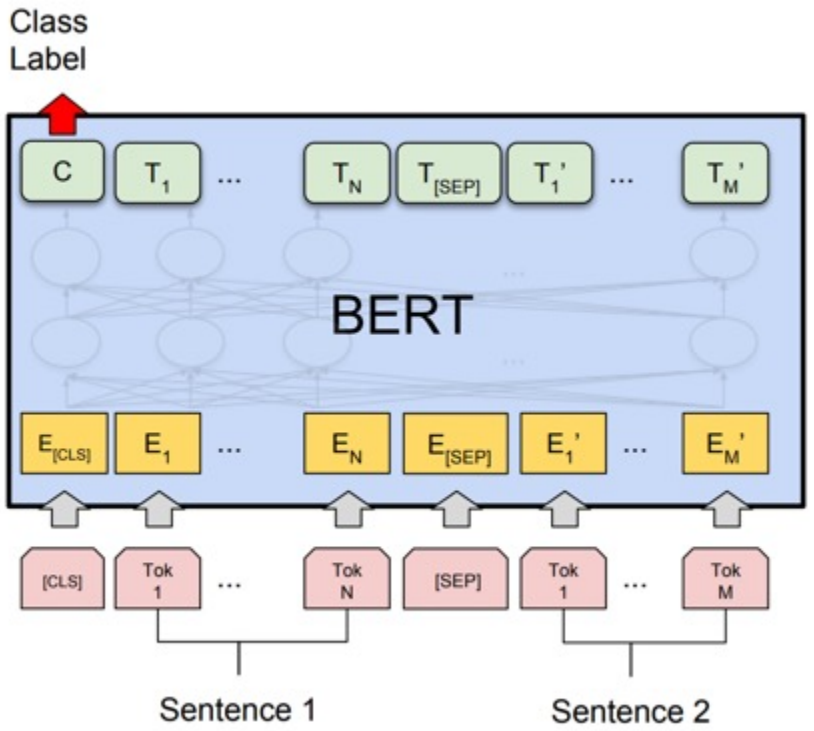
$$P = \textbf{softmax}(CW^T)$$



Fine-tuning procedure 2: Sentence Pair Classification

- For sentence pair classification task
  - Sentences are separated with a a special token **[SEP]**
  - Obtain the representation of the input sequence by using the final hidden state (hidden state at the position of the special token **[CLS]**) $C \in R^H$
  - Just add a classification layer and use **softmax** to calculate label probabilities. Parameters $W \in R^{K \times H}$

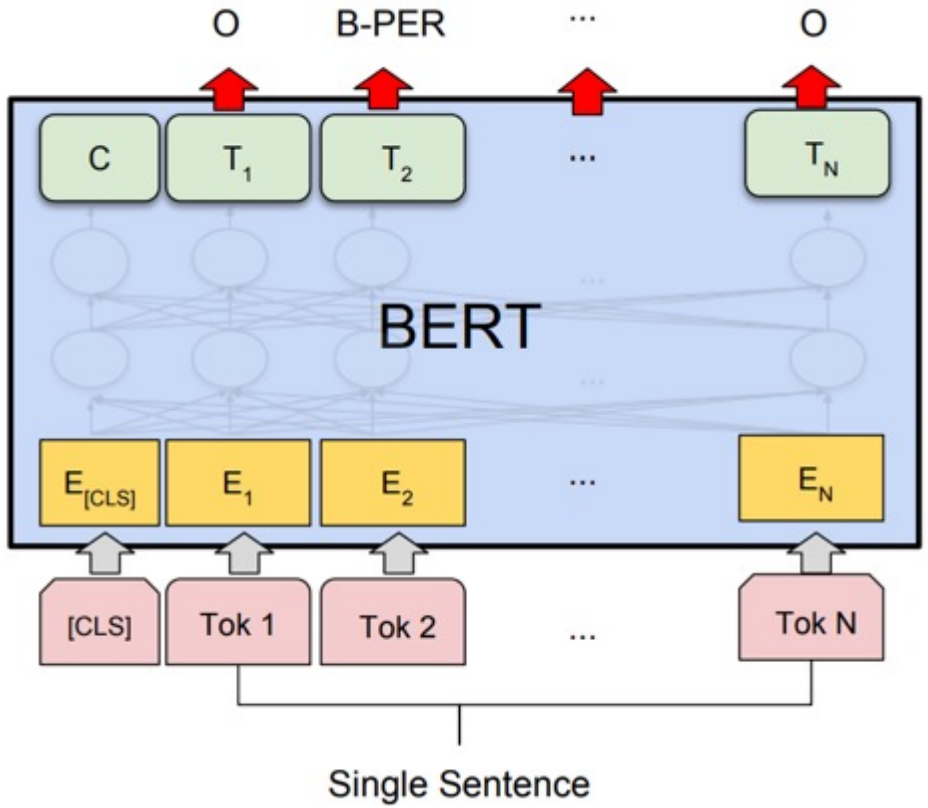
$$\mathrm{P} = \textbf{softmax}(CW^T)$$



Fine-tuning procedure 3: Named Entity Recognition

- Feed the final hidden representation $\boldsymbol{T_i} \in \boldsymbol{R^H}$ for each token $\boldsymbol{i}$ into a classification layer for the tagset.
- To make the task compatible with WordPiece tokenization
  - Predict the tag for the first sub-token of a word
  - No prediction is made for X

Jim	Hen	##son	was	a	puppet	#er
I-PER	I-PER	X	O	O	O	X



Fine-tuning procedure 4: Query Answering 1/2

- **Input Question:**

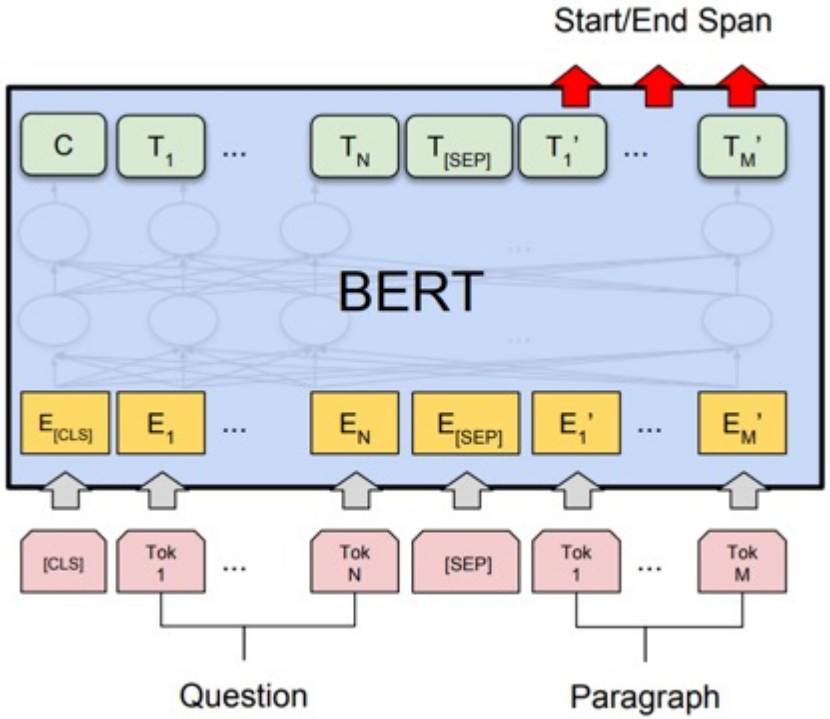
Where do water droplets collide with ice crystals to form precipitation?

- **Input Paragraph:**

…. Precipitation forms as smaller dropletscoalesce via collision with other rain dropsor ice crystals within a cloud. …

- **Output Answer:**

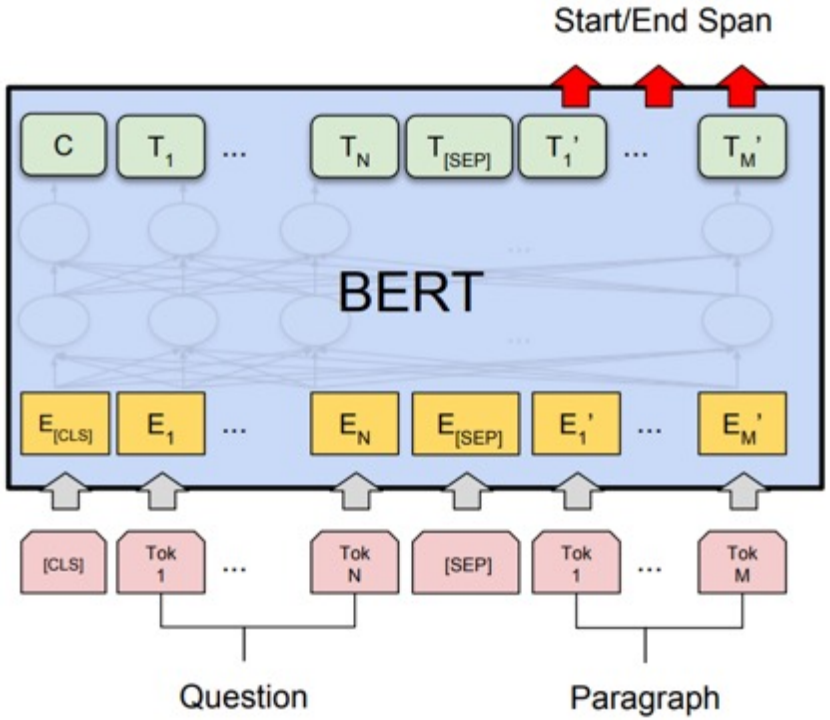
within a cloud



Fine-tuning procedure 4: Query Answering 2/2

- Represent the input question and paragraph as a single packed sequence.
  - The question uses the **A** embedding and the paragraph uses the **B** embedding.
- New parameters to be learned in fine-tuning are start vector $S \in \boldsymbol{R^H}$ and end vector $\boldsymbol{E} \in \boldsymbol{R^H}$.
- Calculate the probability of word & being the start of the answer span:

  $\boldsymbol{P_{Start} = Softmax(ST^T)}$ and $\boldsymbol{P_{end} = Softmax(ET^T)}$
- The training objective is the log-likelihood the correct and end positions.



SIT330-770: Natural Language Processing

Week 10.10 – BERT Performance

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Experiments

- GLUE (General Language Understanding Evaluation) benchmark
  - Distribute canonical Train, Dev and Test splits
  - Labels for Test set are not provided
- Datasets in GLUE:
  - MNLI: Multi-Genre Natural Language Inference
  - QQP: Quora Question Pairs
  - QNLI: Question Natural Language Inference
  - SST-2: Stanford Sentiment Treebank
  - CoLA: The corpus of Linguistic Acceptability
  - STS-B: The Semantic Textual Similarity Benchmark
  - MRPC: Microsoft Research Paraphrase Corpus
  - RTE: Recognizing Textual Entailment
  - WNLI: Winograd NLI

GLUE Results

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

SQuAD: The Stanford Question Answering Dataset

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.

SWAG

- The Situations with Adversarial Generations (SWAG)

  On stage, a woman takes a seat at the piano.

  She …

  a) sits on a bench as her sister plays with the doll.

  b) smiles with someone as the music plays.

  c) is in the crowd, watching the dancers.

  d) nervously sets her fingers on the keys.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table 4: SWAG Dev and Test accuracies. †Human performance is measured with 100 samples, as reported in the SWAG paper.

- The only task-specific parameters is a vector $V \in \boldsymbol{R^H}$
- The probability distribution is the **softmax** over the four choices

Conclusions

- Unsupervised pre-training (pre-training language model) is increasingly adopted in many NLP tasks.
  - Google Search is applying BERT models for search queries for over 70 languages.
- Major contribution of the paper is to propose a deep bidirectional architecture from Transformer.
  - Advance state-of-the-art for many important NLP tasks.
- **Cannot do everything in NLP!**

SIT330-770: Natural Language Processing

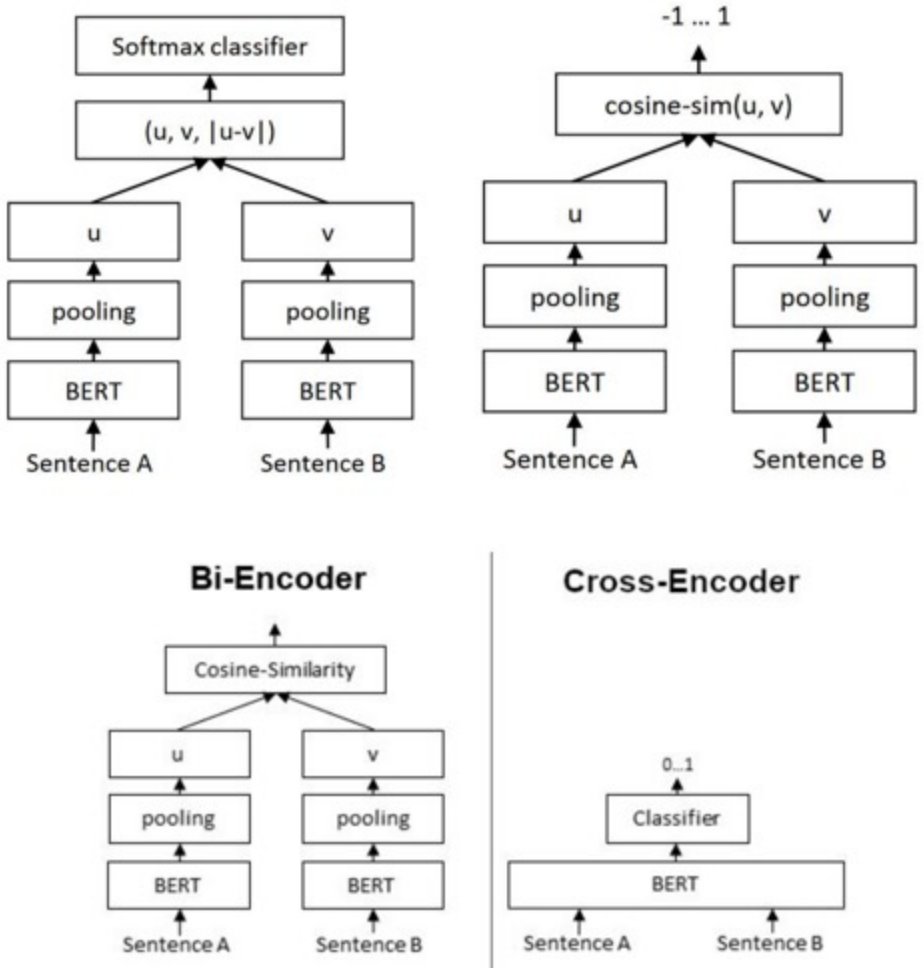
Week 10.11 – Other Models Based on Transformers

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

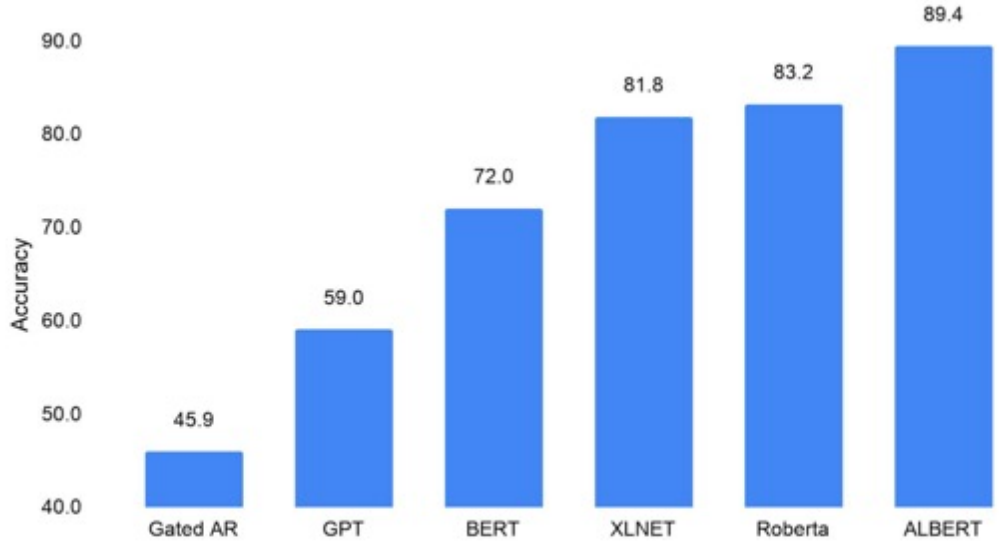
Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

- A modification of the pretrained BERT network that use Siamese network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity
  - https://www.sbert.net/



BERT-based Models

- RoBERTa: Robustly Optimized BERT Approach
  - Facebook AI research team
  - They used 160 GB of text instead of the 16 GB dataset originally used to train BERT
  - Increased the number of iterations from 100K to 300K and then further to 500K
  - Dynamically changing the masking pattern applied to the training data
  - Removing the next sequence prediction objective from the training procedure
- ALBERT, XLNET, ColBERT

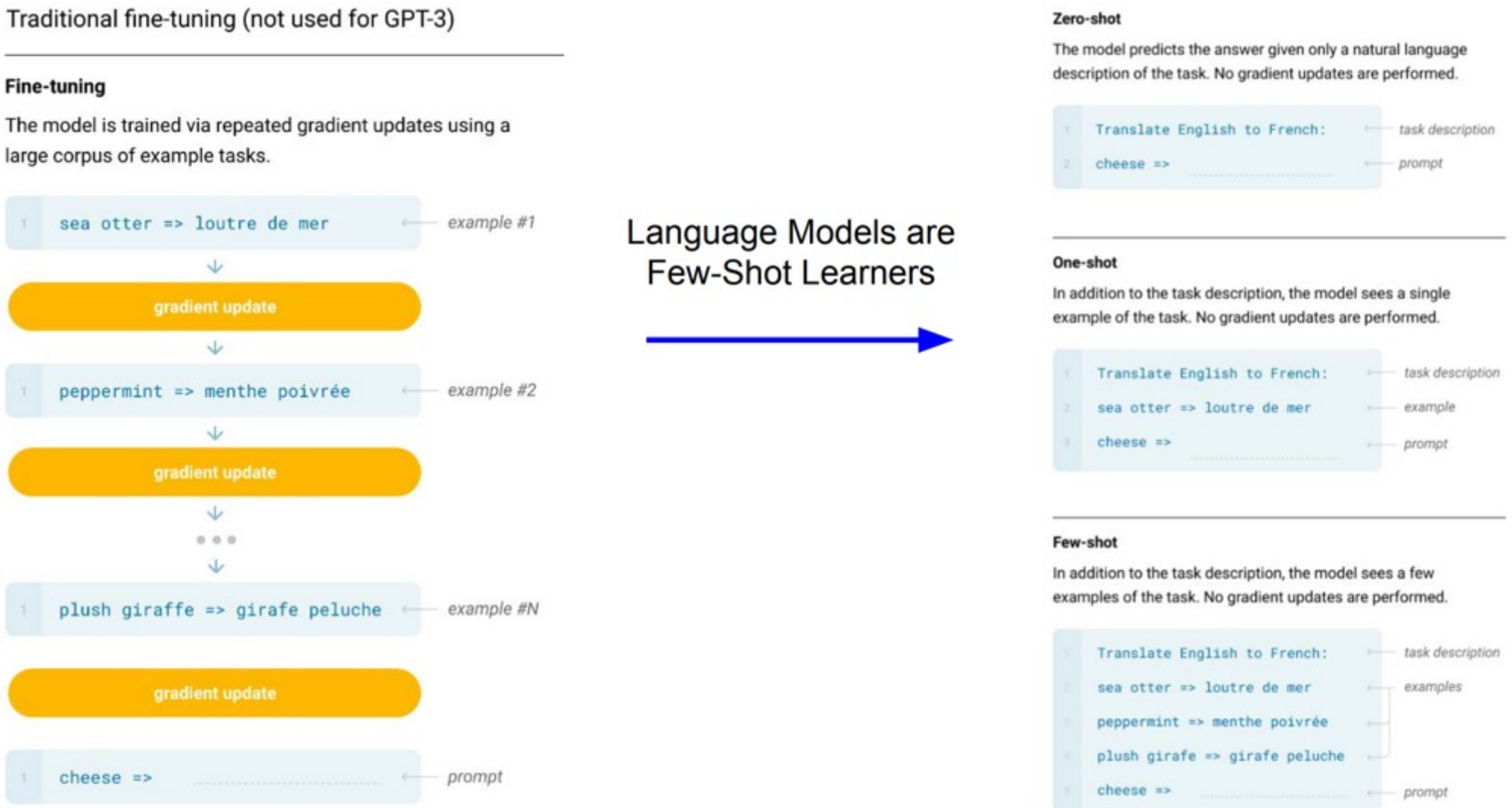


Machine performance on the RACE challenge (SAT-like reading comprehension)

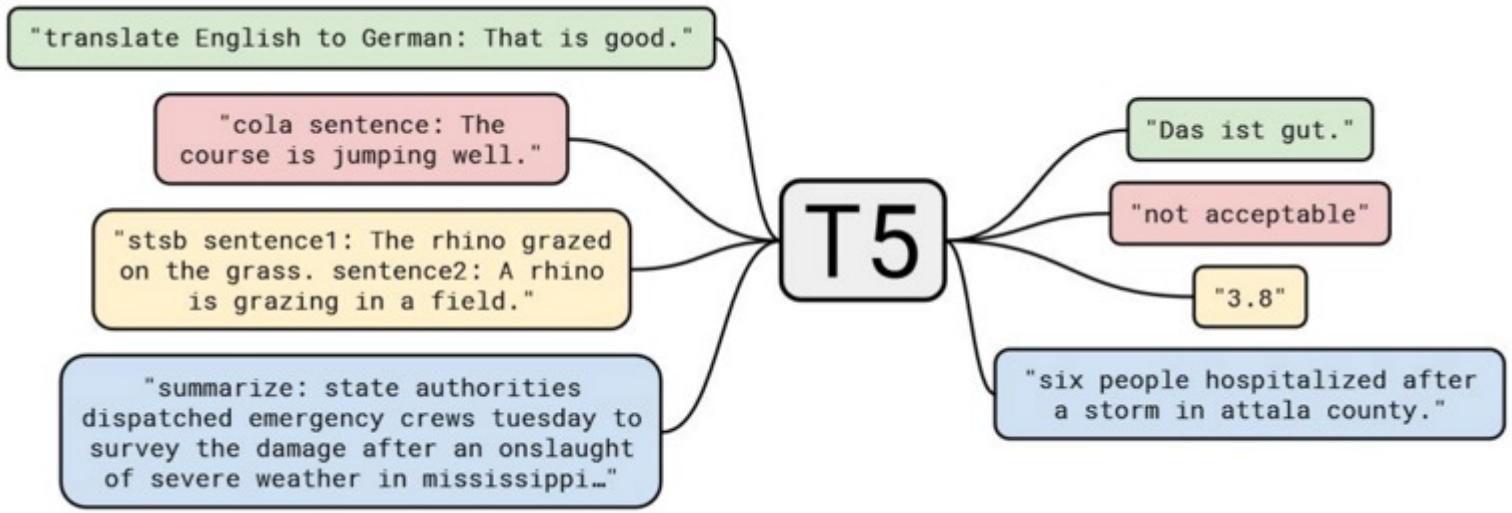
Generative Pre-trained Transformer (GPT)

- **GPT** (Generative Pre-trained Transformer) is a series of language generation models developed by OpenAI. These models are based on the Transformer architecture (2018)
- **GPT-2** (Generative Pre-trained Transformer 2) was the second model in the GPT series, released in 2019. It was trained on a large corpus of internet text and was designed for language generation tasks such as question answering, and text summarization
- **GPT-3** (Generative Pre-trained Transformer 3) Released in 2020, with over 175 billion parameters, and was trained on a much larger and diverse dataset, including web pages, books, and scientific articles
- **GPT-4** (Generative Pre-trained Transformer 4) Released in 2023, a multimodal model which can accept image and text inputs and produce text outputs

Zero-shot, One-shot and Few-shot, Contrasted with Traditional Fine-tuning



Text-to-Text Transfer Transformer (T5)



- A diagram of the text-to-text framework (T5)
- Every task, including translation, question answering, and classification, is cast as feeding T5 model text as input and training it to generate some target text

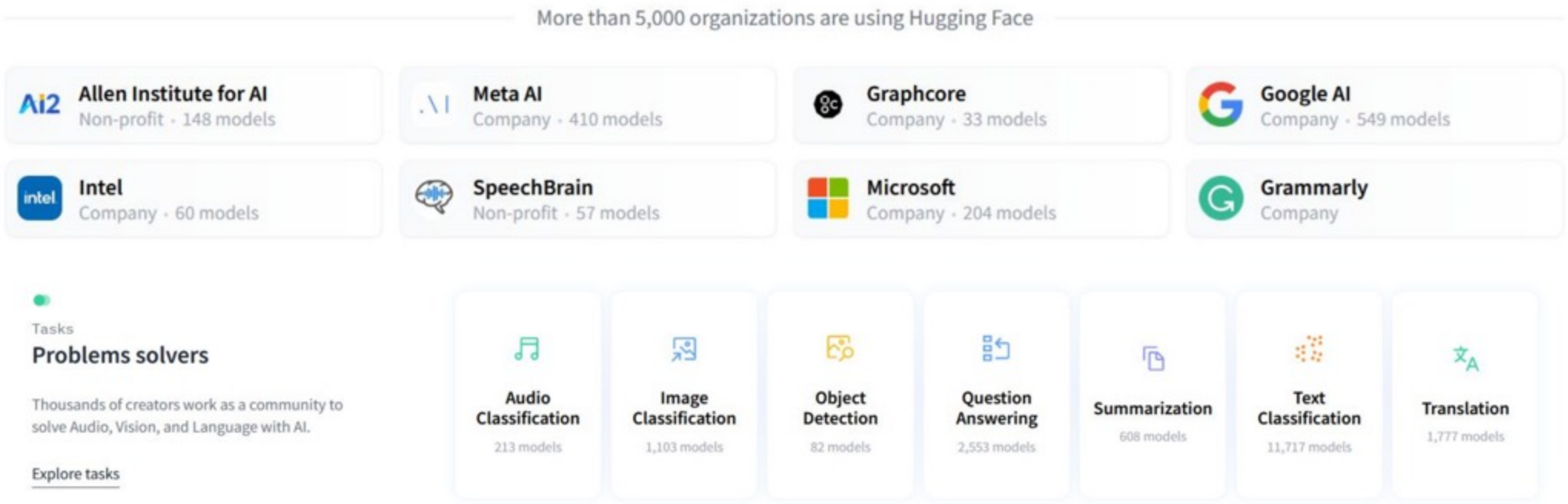
SIT330-770: Natural Language Processing

Week 10.12 – HuggingFace

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

HuggingFace



- HuggingFace has a course on NLP: https://huggingface.co/course/chaptero/1?fw=pt 35

https://huggingface.co/

Documentations and Tutorials

- There is a wide range of examples provided, here are some useful links:
  - Notebooks:
    - https://huggingface.co/transformers/v3.0.2/notebooks.html
    - https://github.com/huggingface/transformers/tree/main/notebooks
  - Models:
    - https://huggingface.co/models
  - Course:
    - https://huggingface.co/course/chapter0/1?fw=pt

Getting Started

- Use pip for the installation, which is the package manager for Python. In notebooks, you can run system commands by preceding them with the ! character, so you can install the 🤗 Transformers library as follows:

```
!pip install transformers
```

- You can make sure the package was correctly installed by importing it within your Python runtime:

```
import transformers
```

Pipelines

- The pipelines are a great and easy way to use models for inference
- These pipelines are objects that abstract most of the complex code from the library, offering a simple API dedicated to several tasks, including Named Entity Recognition, Masked Language Modeling, Sentiment Analysis, Feature Extraction and Question Answering

Example (1): Sentiment Analysis

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace course my whole life.")
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b
Using a pipeline without specifying a model name and revision in production is not recommended.
```


```
[{'label': 'POSITIVE', 'score': 0.9598049521446228}]
```

```
classifier(
    ["I've been waiting for a HuggingFace course my whole life.",
     "I hate this so much!"]
)
```

```
[{'label': 'POSITIVE', 'score': 0.9598049521446228},
 {'label': 'NEGATIVE', 'score': 0.9994558691978455}]
```

Example (2): Question Answering

```
from transformers import pipeline

question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in Brooklyn",
)
```

{'score': 0.6949767470359802, 'start': 33, 'end': 45, 'answer': 'Hugging Face'}

```
from transformers import pipeline

oracle = pipeline(model="deepset/roberta-base-squad2")
oracle(question="Where do I live?",
       context="My name is Wolfgang and I live in Berlin")
```

{'score': 0.9190717935562134, 'start': 34, 'end': 40, 'answer': 'Berlin'}


WHAT HAVE
YOU LEARNED?

Summary

- Today we learned about:
  - Transformers
  - Attention is All you need
  - BERT