CS772: Deep Learning for Natural Language Processing

NLP Applications of Attention and Transformer

Pushpak Bhattacharyya

Computer Science and Engineering Department

IIT Bombay

Week 14 of 4th April, 2022

Historical Perspective

History: RNN class of Deep Nets

- Backpropagation
  - Williams, Ronald J.; Hinton, Geoffrey E.; Rumelhart, David E. 1986. Learning representations by back-propagating errors. *Nature*
- **RNN, LSTM** (Hochreiter & Schmidhuber, 1997)**, GRU** (Cho et al., 2014)
- Iterative and Sequential

History: Success of RNN- Speech

- 2007- LSTM started to revolutionize speech recognition
- Outperformed traditional models in speech applications
- 2014- Baidu used CTC (connectionist temporal classification)-trained RNNs to break the Switchboard Hub5'00 speech recognition dataset benchmark without using any traditional speech processing methods

History: Success of RNN- Connected Handwriting

- 2009: a Connectionist Temporal Classification (CTC)-trained LSTM network

- First RNN to win pattern recognition contests
- Won several competitions in connected handwriting recognition

History: CNN and ImageNet

- CNN
  - LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation.
- ImageNet Dataset- Million images from the web; 1,000 different classes
- Spectacular results!
- Almost halving the error rates of the best competing approaches1.

History: RNN in NLP leading to Transformer Architecture

- Recurrent neural networks (RNNs) in NLP (Mikolov et al., 2010; Sutskever, et al. 2014)

- Transformer (Vaswani et al., 2017)
  - Self Attention + Pointwise fully connected layer
  - Positional encoding
  - Can capture long range dependencies

Limitations of FFNN, RNN and CNN

- **FFNN:** feedforward NNs- cannot capture sequential context
- **RNN, LSTM, GRU**- Sequential and iterative nature; take time for information to propagate (linear in the length of the sequence); cannot exploit concurrent hardware (such as GPUs)
- **CNN-** fixed-sized contexts

Latest developments: Transformers, GPT-3

- **Generative Pre-trained** Transformer 3 (GPT-3)
- Autoregressive language model
- Uses deep learning to produce human-like text
- Third-generation language prediction model in the GPT-n series (and the successor to GPT-2)
- By OpenAI, a San Francisco-based artificial intelligence research laboratory

GPT-3 cntd.

- GPT-3's full version has a capacity of 175 billion machine learning parameters.
- Before the release of GPT-3, the largest language model was Microsoft's Turing NLG, introduced in February 2020, with a capacity of 17 billion parameters or less than a tenth of GPT-3s.

GPT-3's success

- Applied extensively in NLG situations- Natural Language Generation
- Summarization
- Question Answering
- Machine Translation

Demo

- ***Plot and Scene Generation***: Team- Vishal, Prerak, Ashita, Naveen, Narjis

- ***Machine Translation***: Team- Sourabh, Shivam, Aman, Dhiren, Rohit, Vineet, Shyam, Akshay

Attention is a general and important concept in Deep learning

Given a set of VALUES ➔ select a summary of the values that is relevant to a QUERY

Each VALUE represented by a KEY ➔ the QUERY is matched to the KEY (content similarity)

Select a summary with different focus on different values ➔ Weighted average

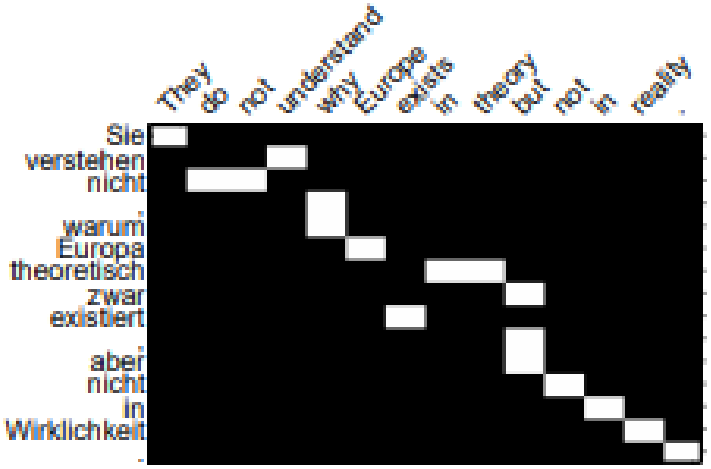
Associative memory read + selection

For MT

QUERY: decoder state

VALUE, KEY: encoder annotation vector

Benefits of Attention



- Significantly improves in NMT quality
  - Performs better on long sentences
  - Word-order is no longer a major issue for NMT
  - Used in all NMT systems
- Attention provides some interpretability
  - Attention!=Alignment
- There is more to attention

https://arxiv.org/pdf/1508.04025.pdf

A lot of interesting work with attention

- Pointer Networks
- Pointer Generator Networks
- Modeling Coverage
- Learning word-alignments

Paper walk through: Vaswani et al.

Introduction to Transformers

“Attention Is All You Need”

Akshay Batheja
Shivam Mhaskar

Guide : Prof. Pushpak Bhattacharyya

Outline

- Transformers:
  - Problem Statement
  - Motivation
  - Contributions
  - Model Architecture
  - Datasets Used
  - Results
  - Summary and Conclusion
- Demo:
  - Training a Transformer model using OpenNMT-py
  - IITB Speech–To-Speech Machine Translation System

Problem Statement

- To perform the sequence-to-sequence task of Machine Translation using only **attention** mechanism instead of recurrent and convolutional layers.

Motivation

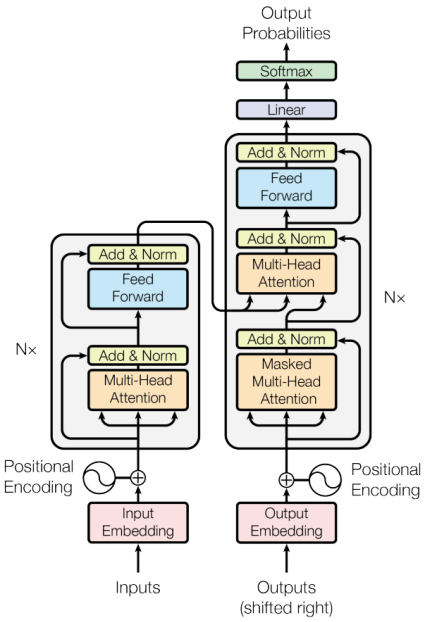
- Limitation of RNN (Recurrent Neural Network) based Encoder-Decoder Architecture:
  - RNN generates a sequence of hidden states h_t, as a function of previous hidden state h_{t-1} and the input at position t. So, to process the input at t^{th} step, the encoder or decoder has to wait for t-1 steps.
  - This sequential nature of RNN prevents it from parallelization.

Contributions

- Proposed a novel approach, **Transformer**, the first sequence transduction model based entirely on **attention**, replacing **recurrent** layers with **multi-headed self attention**.
- On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, it achieved a new state of the art.

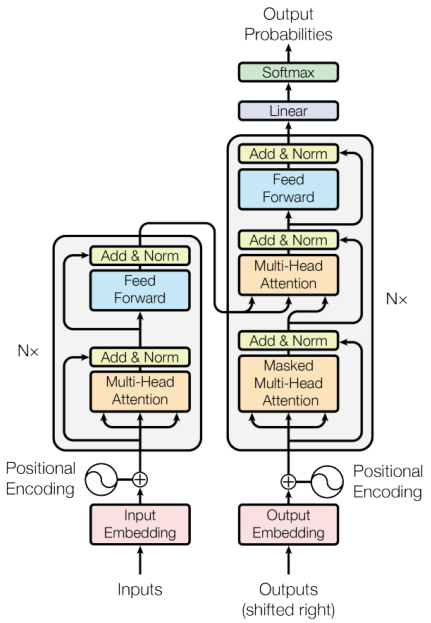
Model Architecture (1/4)

- Encoder:
  - It is composed of 6 identical layers, each layer has two sublayers.
  - First is the multi-head self attention mechanism and second is the position wise fully connected FFNN.
  - A residual connection is employed around each of the two sub-layers, followed by layer normalization.
  - So the output of each sublayer is :
    - LayerNorm(x + Sublayer(x))
  - All sublayers produce outputs of dimension $d_{model} = 512$.



Model Architecture (2/4)

- Decoder:
  - It is also composed of 6 identical layers.
  - Each layer consists of 3 sub-layers , two of which are same as that present in each encoder layer.
  - There is a third sublayer which performs multi-head attention over the output of the encoder stack.
  - Similar to encoder, Residual connections are employed around each of the sub-layers, followed by layer normalization.
  - Self attention sublayer is also modified to prevent positions from attending to subsequent positions. This is performed by masking the subsequent positions during training.
  - This ensures that the predictions for position *i* can depend only previously generated outputs.



Positional Encodings (3/4)

- Since the model has no recurrence or convolution layers, it doesn't have any information about the positions of the inputs.
- We generate positional encodings which represent the positional information of each word in the input sequence.
- These positional encoding vectors are of dimension d_{model} (same as embeddings).
- These positional encodings are added to each input embedding, which helps the model determine the position of each embedding.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Positional Encodings (4/4)

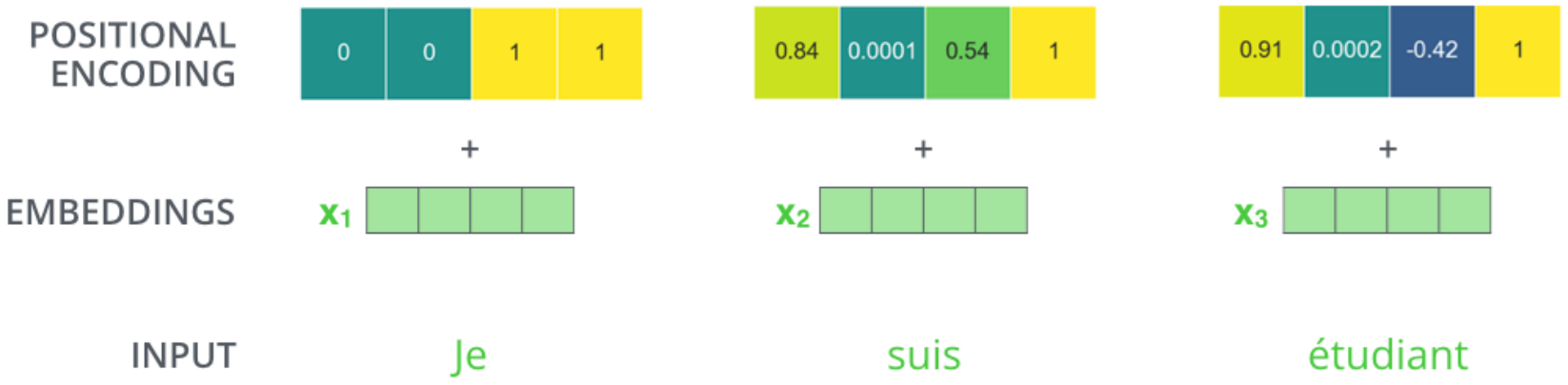


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Self Attention (1/3)

- First we create 3 vectors using the input embeddings (x).
  - Query (q)
  - Key (k)
  - Value (v)
- We obtain these vectors by matrix multiplication with the weight matrix W^Q, W^K, W^V which are the parameters of the self attention module.

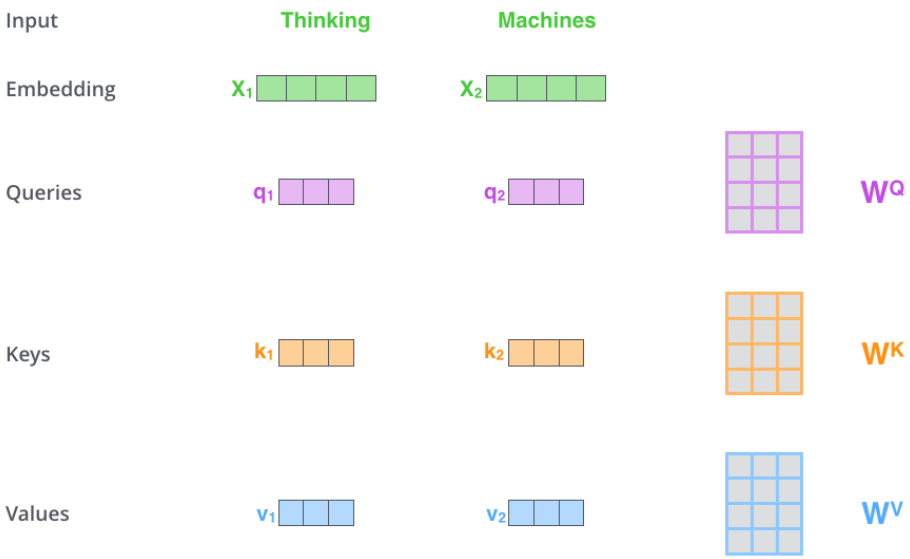


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Self Attention (2/3)

- Now we need to score each input position(word) against the current position(word) which we are processing.
- This score represents how much focus to place on each word while processing the current word.
- These scores are computed by taking the dot product of the query(q) vector of current word with the key(k) vector of each input word.

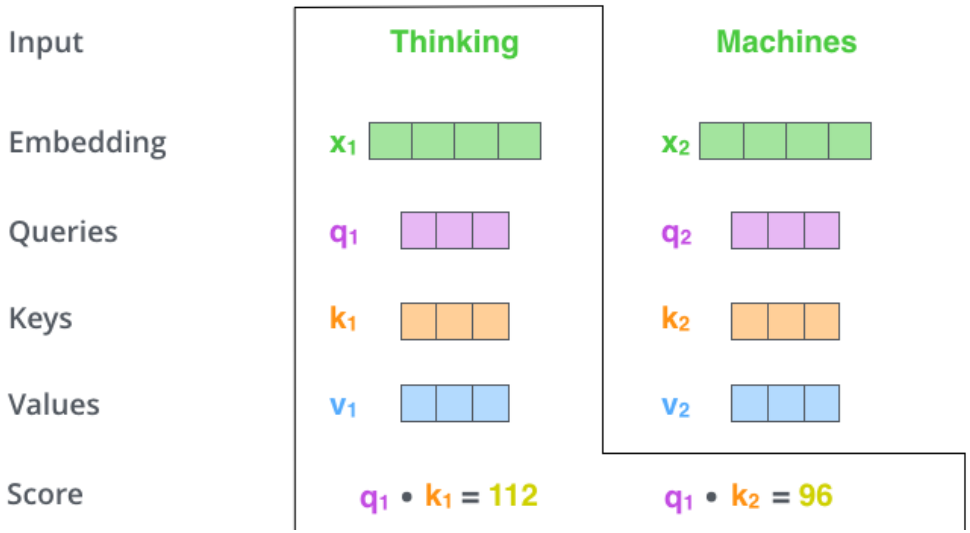


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Self Attention (3/3)

- We scale the scores by diving the scores by d_k and then we perform the softmax operation on the scores.

- We weight the value(v) vectors by multiplying the vectors with the corresponding scores of that position.
- Then we compute the weighted sum of the value(v) vectors which forms the output of self attention layer.

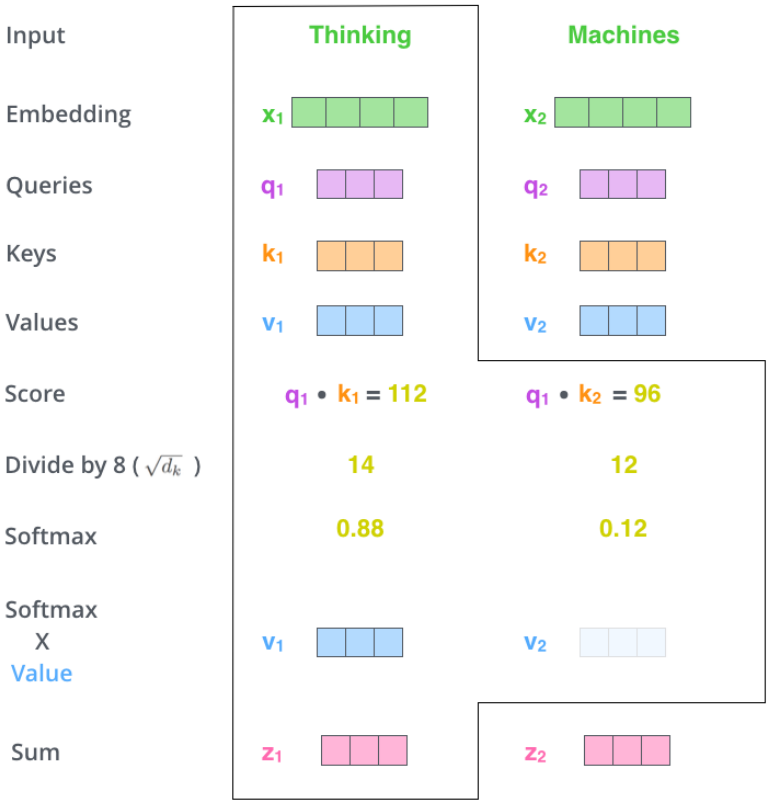


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention (1/3)

- In Multi-Head Attention we have different sets of WQ, WK, WV weight matrices for each attention head.

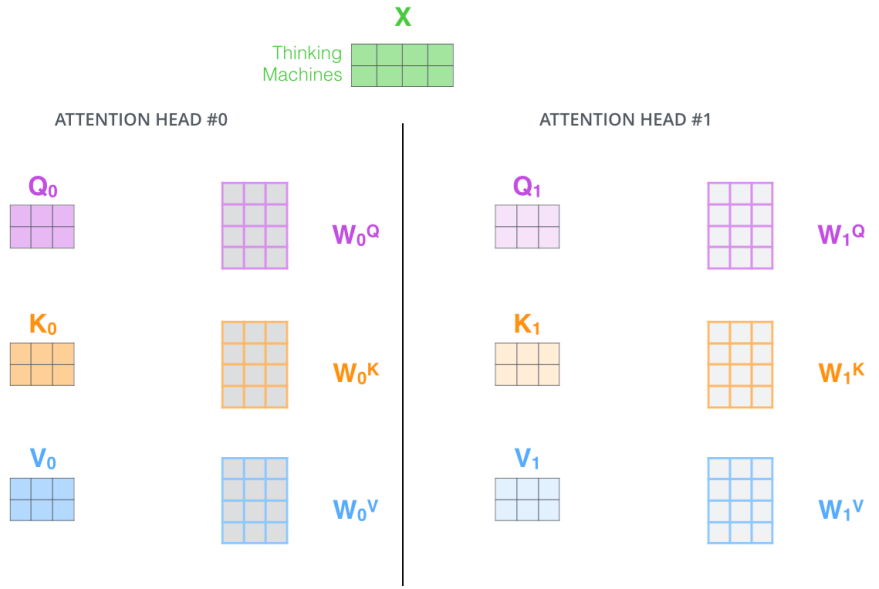


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention (2/3)

- We perform the same self-attention calculations using these different weight matrices to produce *h* output vectors

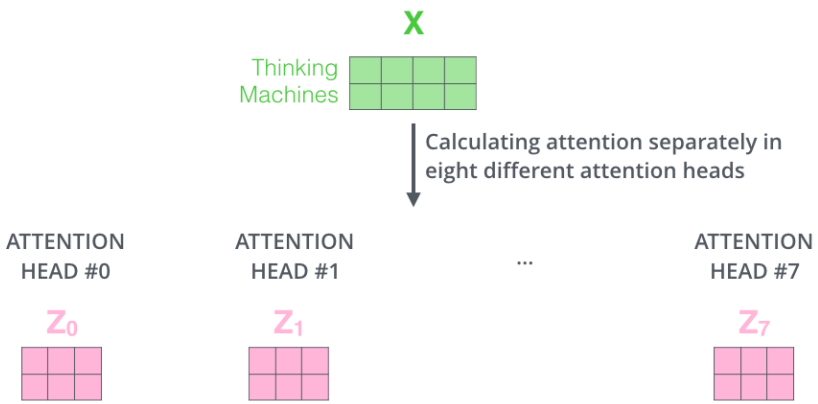
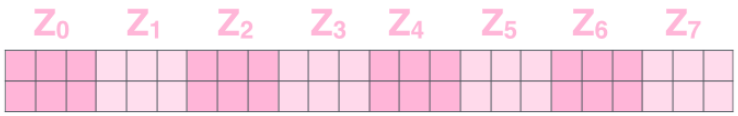


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

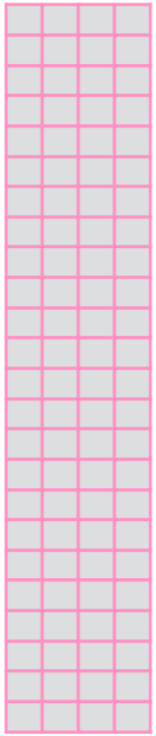
Multi Headed Attention (3/3)

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

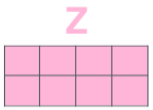
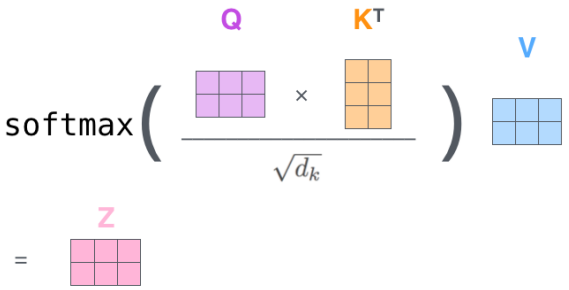


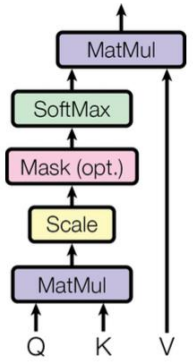
Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Attention Overview

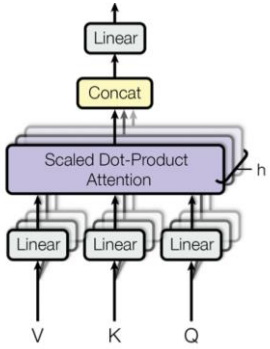
- Self Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$





$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



- Multi Headed Attention

Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Visualizing Multi-headed Attention

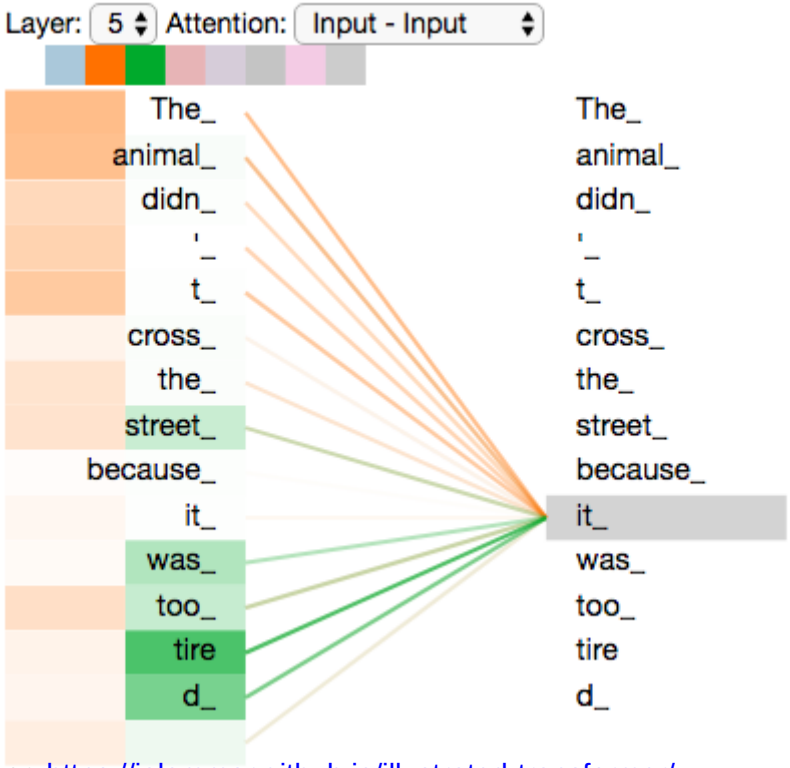


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Cross Attention

- The cross attention mechanism works in the same way as self attention mechanism.
- The only difference is that,
  - The Key and Value vector comes from the outputs of the encoder
  - The Query vector comes from the output of the decoder's self attention layer.
- So the attention scores represent

Final Linear and Softmax Layer

- We generate the output word from the decoder output vector using a linear layer and

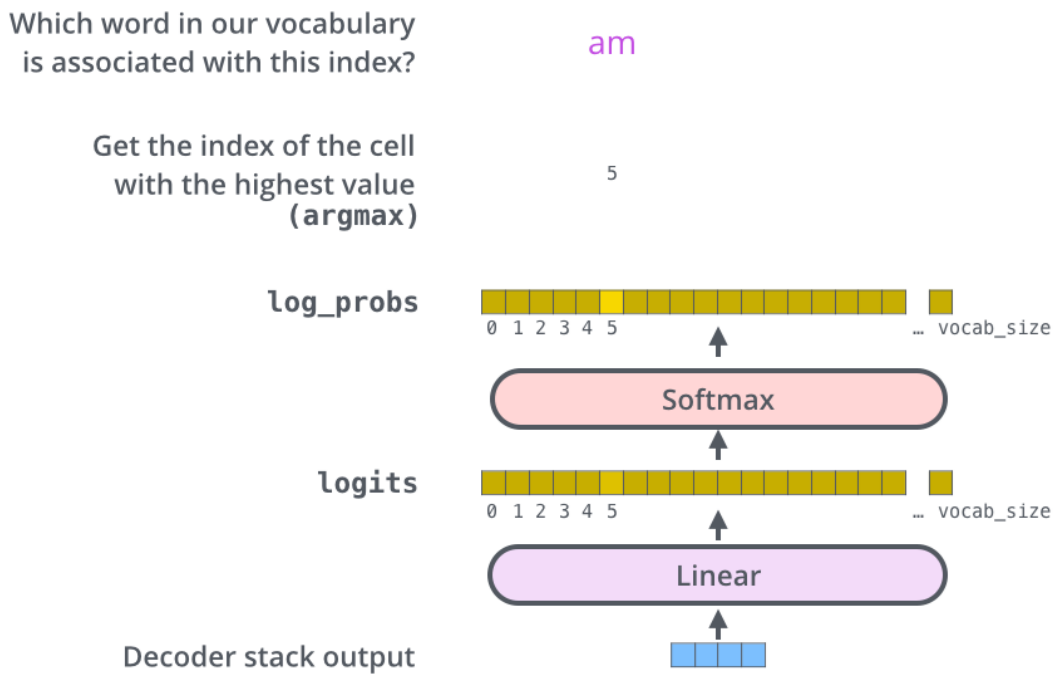


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Transformers Decoding

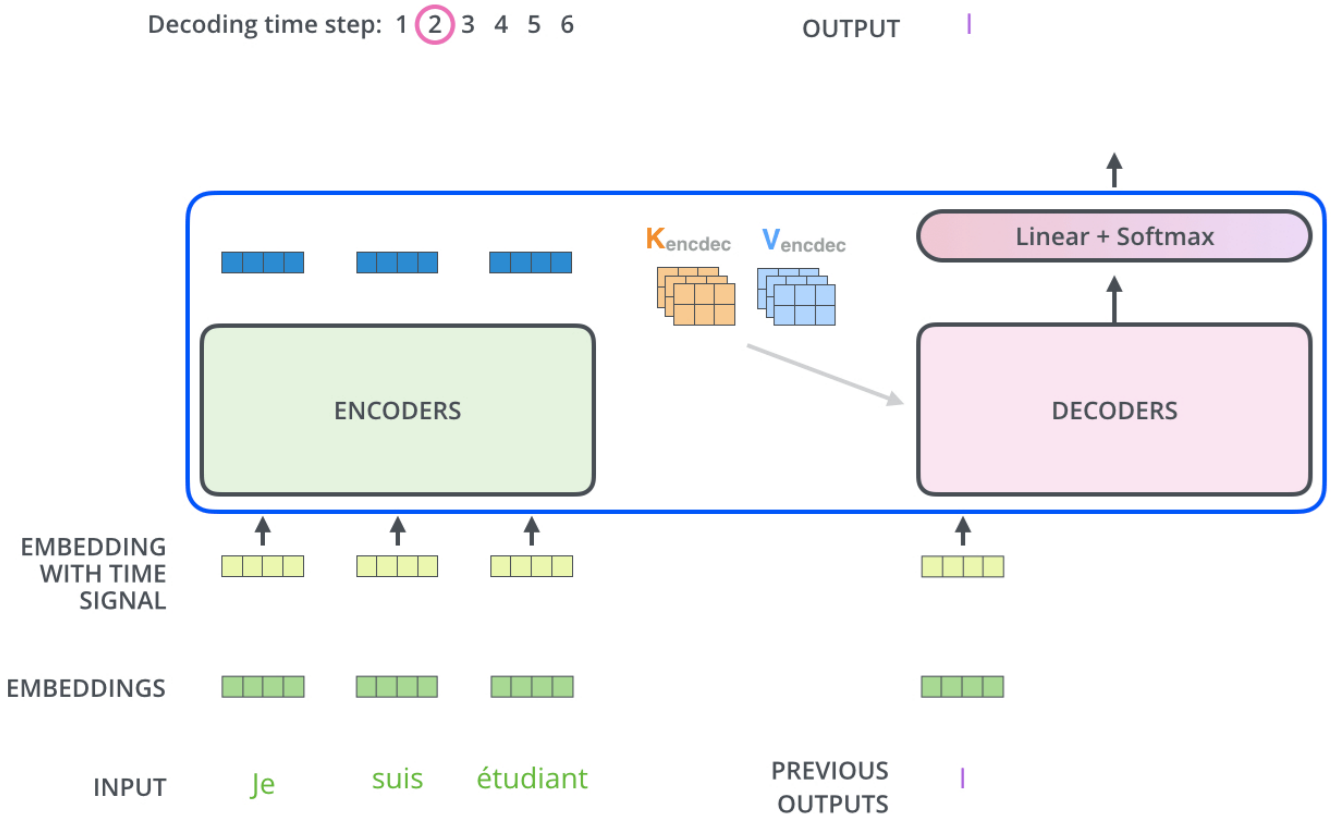


Image Source: The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/

Dataset

Corpus	# of sentence pairs
WMT’14 English-to-German	4.5M
WMT’14 English-to-French	36M

Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Summary

- Introduced Transformers, the first sequence transduction model based entirely on attention, replacing recurrent layers in enc-dec architecture with multi-head self attention.

Conclusion

- For the Task of MT, Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers.
- Transformer model achieved state-of-the-art performance on both WMT’14 English-German and English-French MT task

Important Intuitive points about Transformer (1/3)

- Properties of current and context words determine the next action. Eg: generating the next word.
- Properties: lexical, syntactic, semantic, pragmatic
- Attention varies from task to task, like sentiment classification, translation, inference etc. For different tasks the nature of attention among the words in a sentence varies.
  E.g: i love mumbai because of its dynamic nature and opportunities.
  - Here, sentiment analyzer would focus on the word “love”
  - But a QA model would focus on phrase “dynamic nature and opportunities” when asked to answer the question “why do I love mumbai?”

Intuitive points continued (2/3)

- Attention is of three kinds: self, multihead, cross.
- Self attention: attention of one part of the sentence(input/output) to another part of the same sentence.
- Cross attention: attention of one part of the output to parts of the input.
- Multi head attention: Multiple heads are introduced to capture various context and word properties (lexical, syntactic, semantic, pragmatic).

Intuitive points continued (3/3)

- Positional encoding:
  - Motivation (example of pos tagging): If we know that a word is being preceded by an adjective then most likely the word is a noun.
  - If position information is used, then context is automatically captured.
  - Eg: Instead of using “preceded by” or “followed by” relation, we can say the fifth word is a noun because the fourth word is an adjective.
  - Positional information liberates the processing from sequentiality, enabling parallelism.
  - For a task like POS, influence from nearby words is strongest, tapering of to the left and to the right (a bell-shaped curve), peaking at the current word.
  - sin and cos functions have this property which is why positional encoding uses these functions.

References

- Ashish Vaswani, Noam Shazeer, Niki Parmar,Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. “**Attention Is All You Need**”. NIPS 2017

Model Architecture (contd)

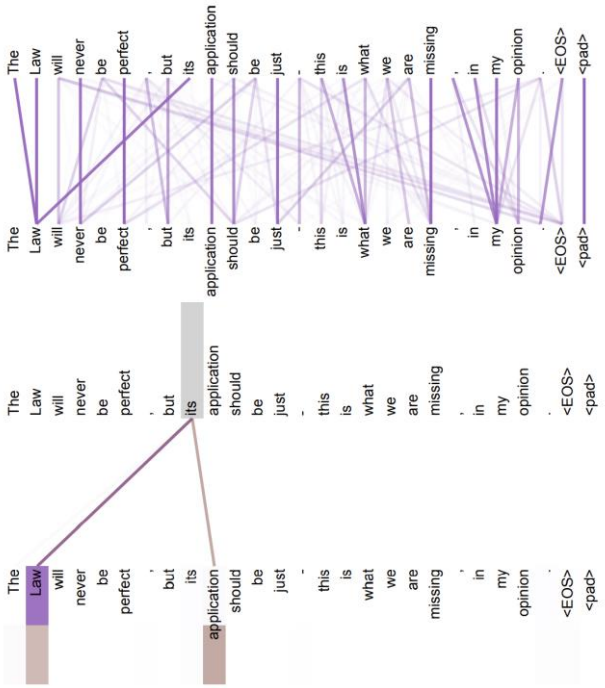
- Positional encoding:
  - Since the model contains no recurrence and convolution, it is injected with some information about the relative or absolute position of the tokens in the sequence.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Anaphora Resolution

- An example where multi-head attention is involved in Anaphora resolution.



Literature Survey

- **ByteNet** (Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time.)
- **ConvS2S** (Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning)
- Both these models use CNN to compute hidden representation for all input and output positions parallely.
- JianPeng Cheng-2016, Ankur Parikh-2016, Romain Paulus-2017, Zhouhan Lin-2017:
  - Used Self-Attention in variety of tasks like reading comprehension, abstractive summarization, textual entailment and learning task-dependent sentence representations respectively.

NLG with GPT*

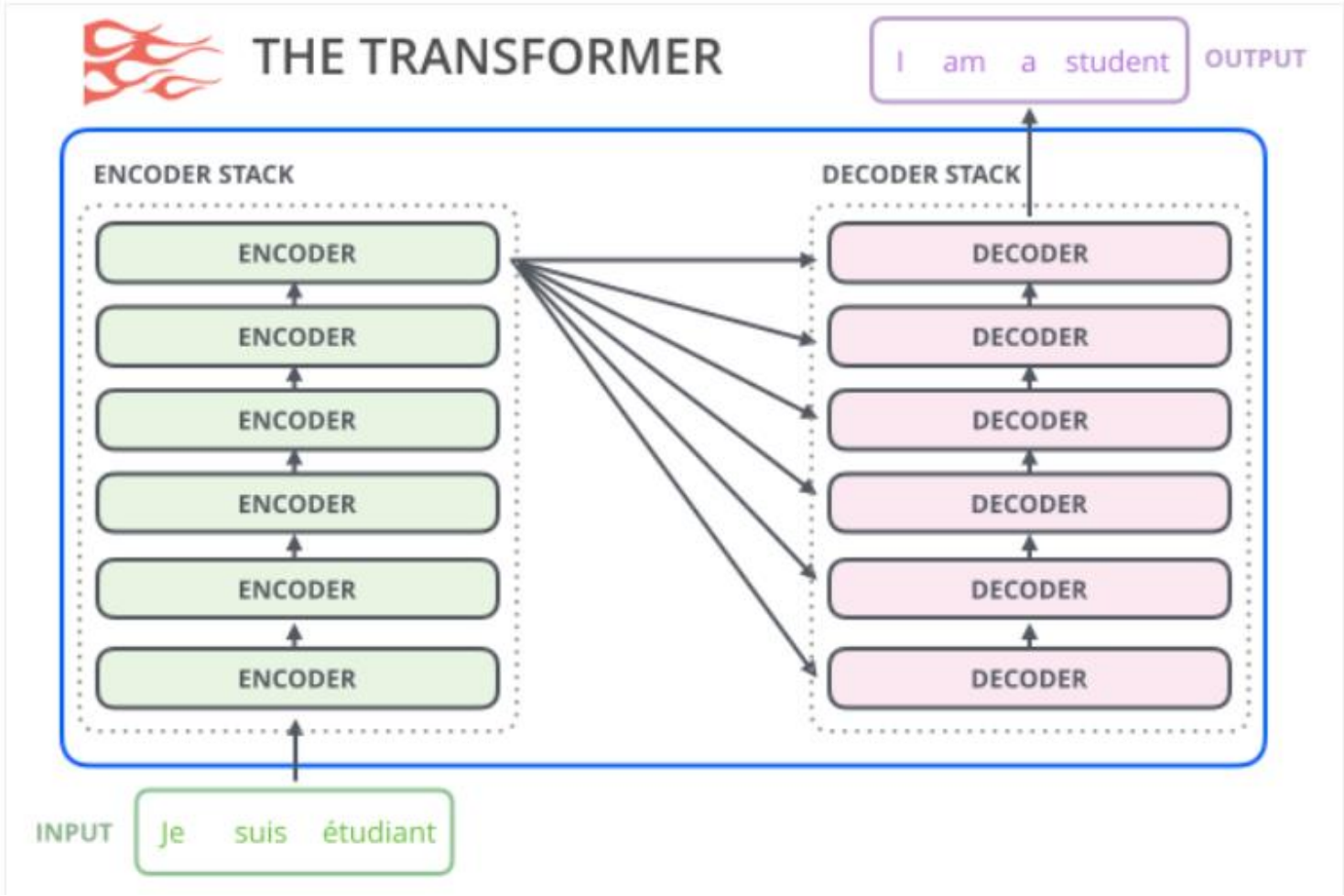
Automatic Film Plot & Script Generation

Prerak Gandhi
Vishal Pramanik
Guide: Prof. Pushpak Bhattacharyya
IIT Bombay

Architecture

Transformer

- The transformer model is made up of an encoder and decoder.
- The encoding component is a stack of 6 encoders .
- The decoding component is a stack of decoders of the same number.



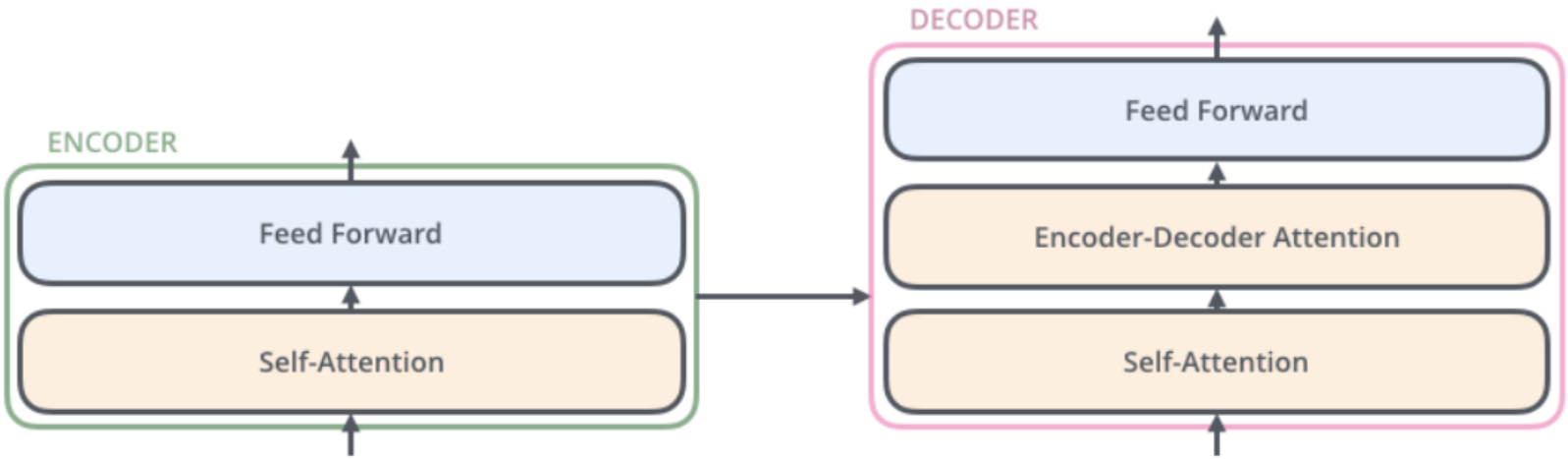
https://jalammar.github.io/illustrated-gpt2/

Encoder and Decoder of Transformer

- The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.

- The outputs of the self-attention layer are fed to a feed-forward neural network.

- The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence



The encoder decoder block of the Transformer

https://jalammar.github.io/illustrated-gpt2/

GPT 2 - 1/2

- The OpenAI GPT-2 exhibited impressive ability of writing coherent and passionate texts
- The GPT-2's architecture is a decoder-only transformer. The GPT-2 was, however, a very large, transformer-based language model trained on a massive dataset.
- Unlike the original transformer decoder, GPT-2 does not have encoder-decoder attention.

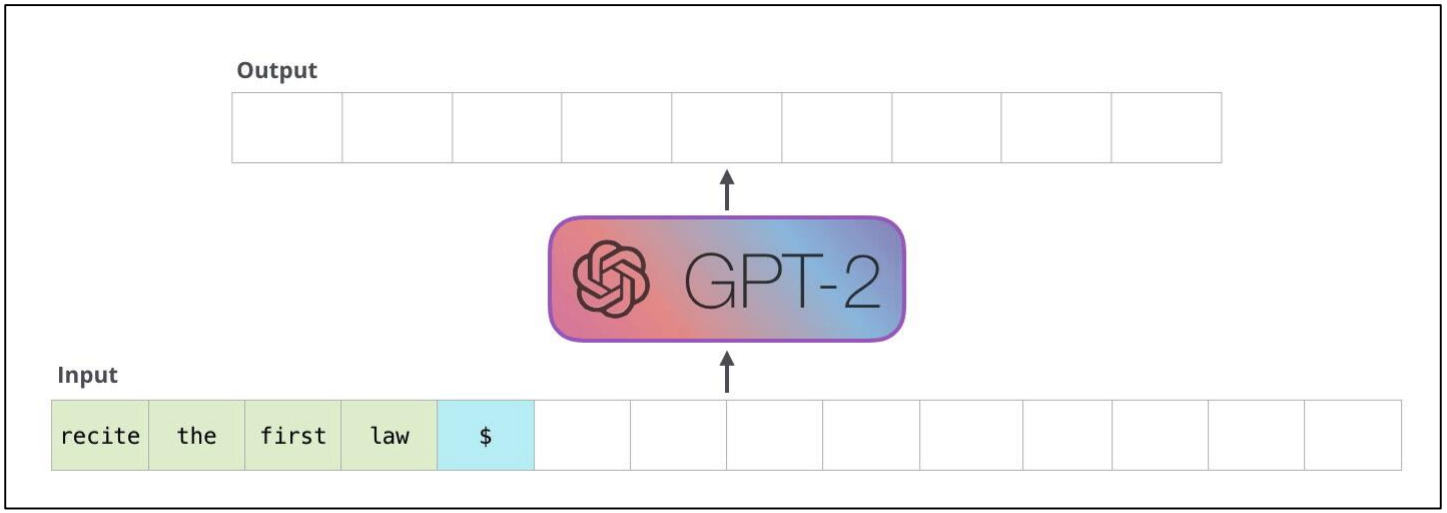
GPT 2 as a Language Model

- A language model is basically a machine learning model that is able to look at part of a sentence and predict the next word.

https://jalammar.github.io/illustrated-gpt2/

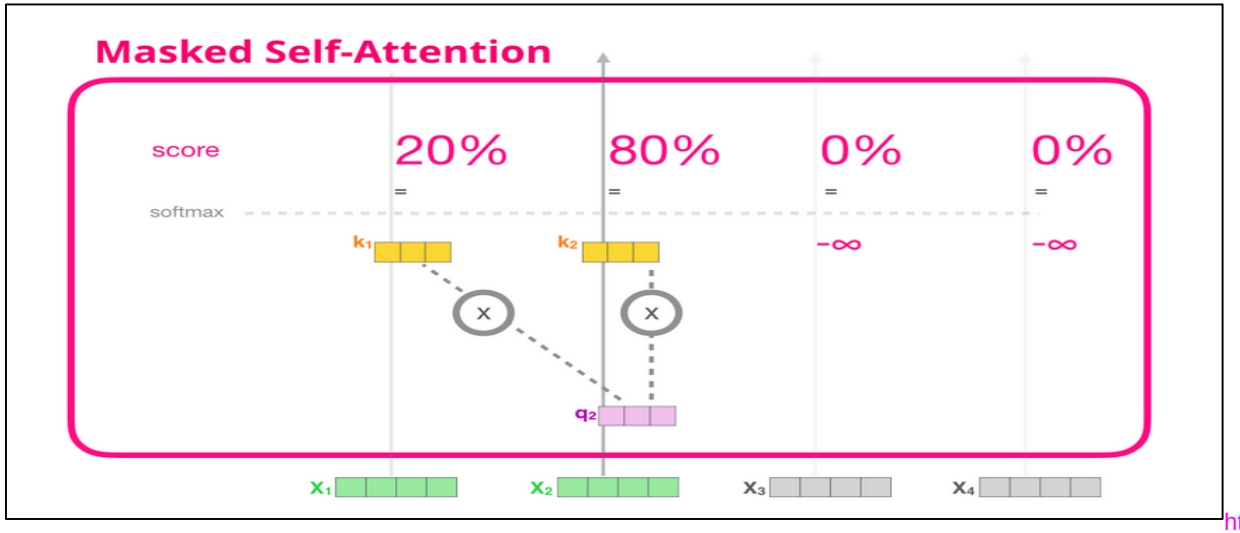
GPT2 - 1/2

- **GPT as an AutoRegressive Model-** The way GPT actually work is that after each token is produced, that token is added to the sequence of inputs. And that new sequence becomes the input to the model in its next step. This is an idea called “auto-regression”.

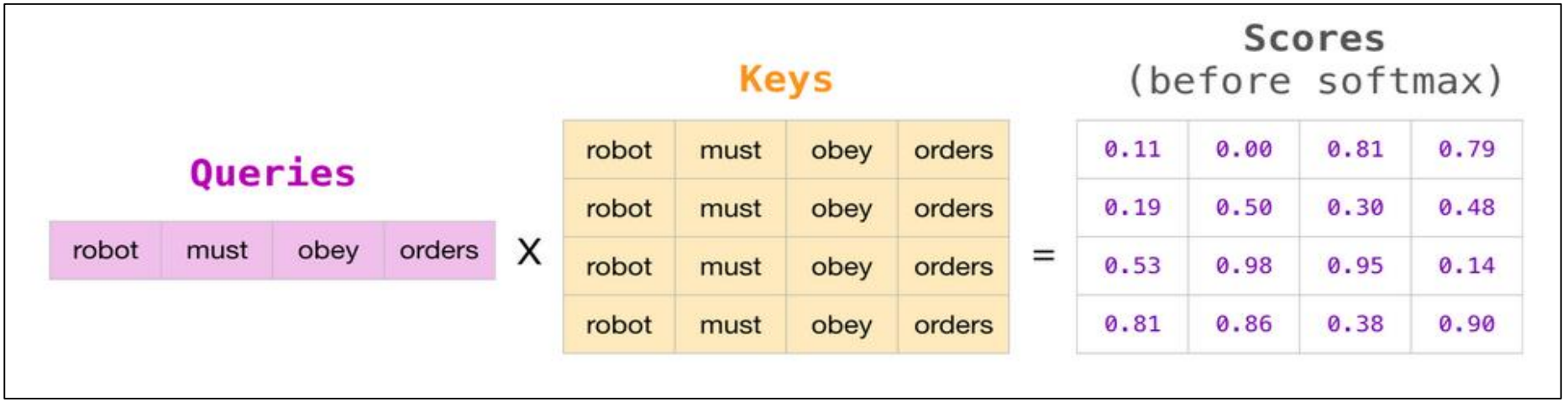


GPT2 - 2/2

- **Masked self-attention -** In the self-attention layer of GPT-2, it masks future tokens by interfering in the self-attention calculation, blocking information from tokens that are to the right of the position being calculated.



https://jalammar.github.io/illustrated-gpt2/



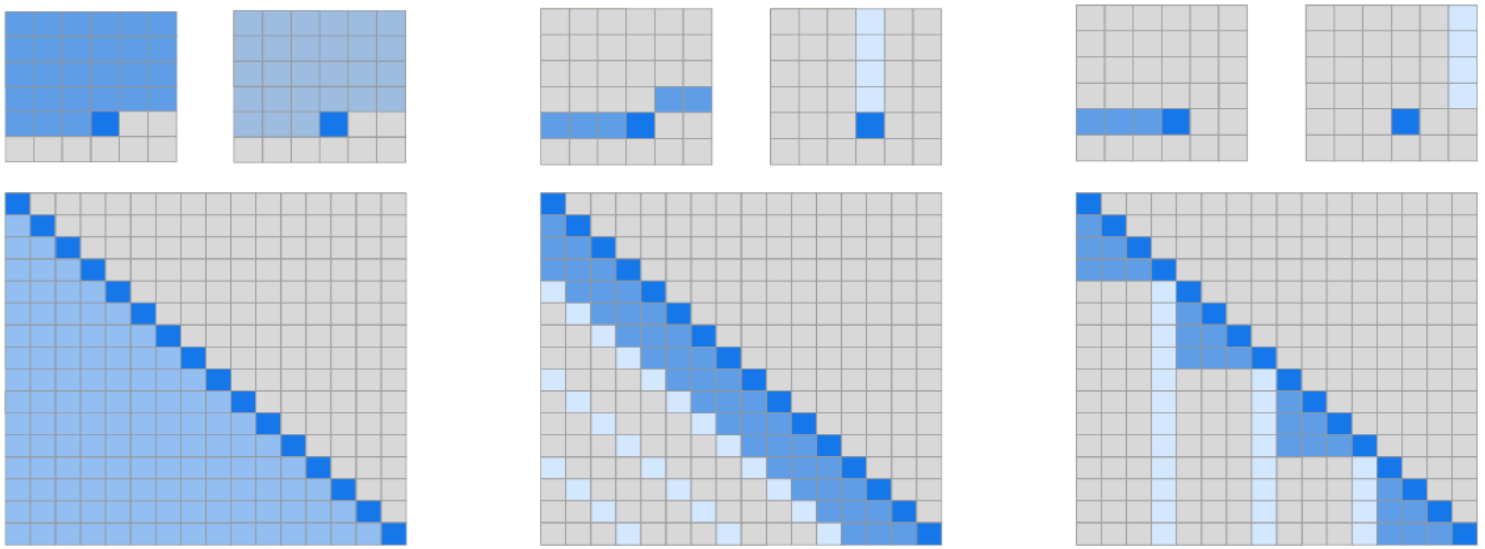
https://jalammar.github.io/illustrated-gpt2/

GPT-3

- The techniques behind GPT-3 are almost same as GPT-2.
- The main difference is that the GPT-3 model uses alternating dense and banded sparse attention pattern.
- Also, GPT-3 has more number of layers and heads to train the language model.

Sparse Attention

Sparsity in the attention calculation can lead to significantly faster computation.

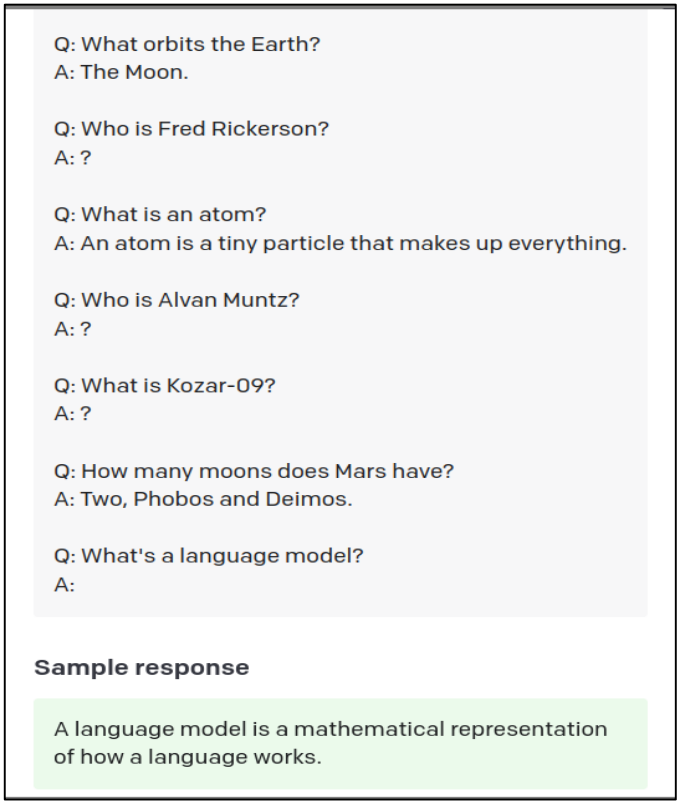


(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

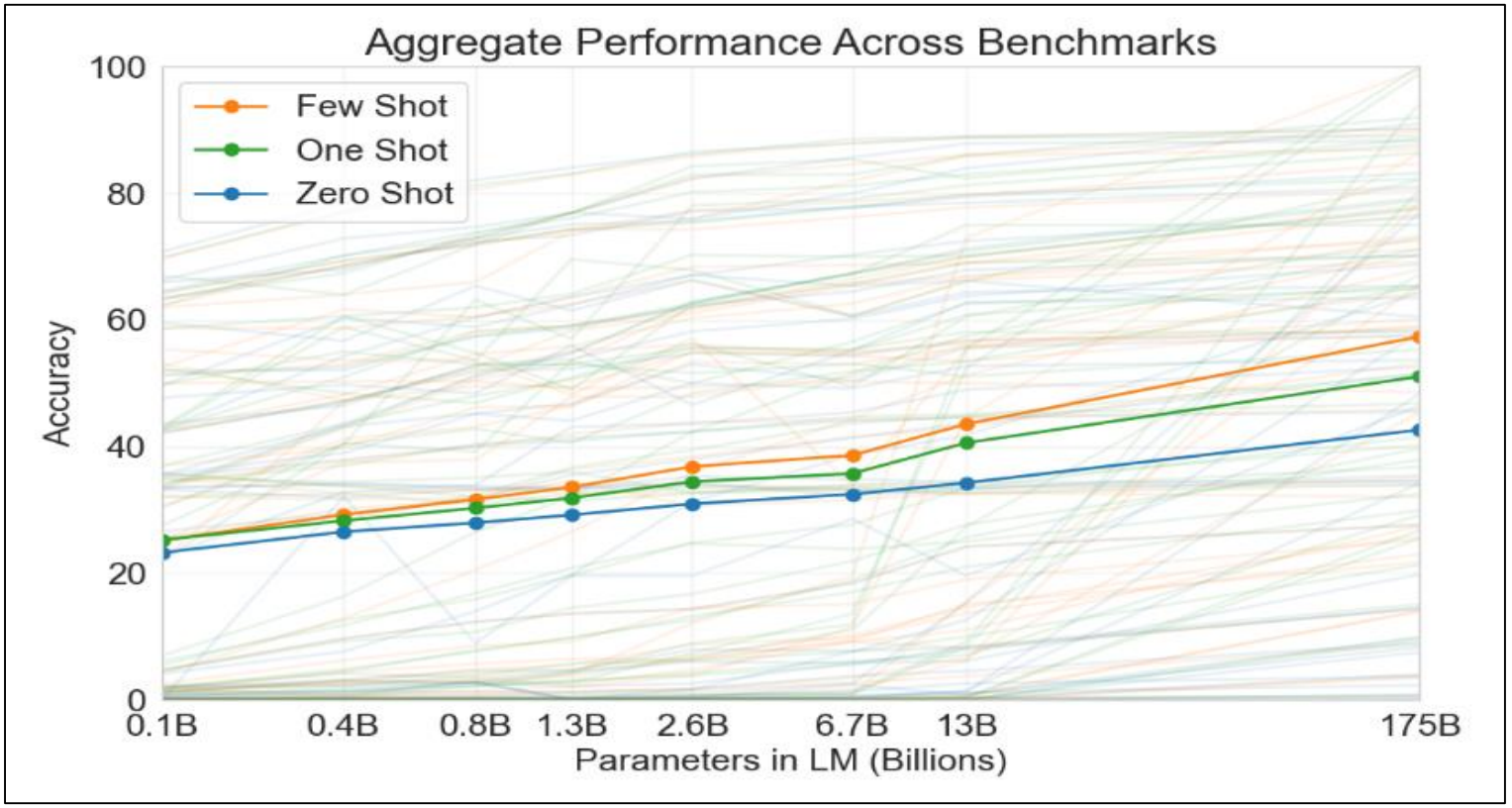
Child, Rewon & Gray, Scott & Radford, Alec & Sutskever, Ilya. (2019). Generating Long Sequences with Sparse Transformers, *arXiv preprint* arXiv:1904.10509v1

Few-shot Learning - 1/2

Few-shot learning is the problem of making predictions based on a limited number of samples.



Few-shot Learning - 2/2



Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. & others (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, .

GPT-3 Models

There are 4 pre-trained models available in GPT-3.

1. Ada (2.7B)
2. Babbage (6.7B)
3. Curie (13B)
4. Davinci (175B)

GPT-3 Dataset Preparation - 1/2

- The combined tokens in an input-output pair should not exceed 2048. (1 token ~ ¾ words)
- It is recommended to add a separator tag at the end of each datapoint for both input and output.

  Eg, Input: “Start…end\n\n###”,

  Output: “Start…end\n\n<eos>”

GPT-3 Dataset Preparation - 2/2

```
openai tools fine_tunes.prepare_data -f <LOCAL_FILE>
```

```
You can also pass files in CSV, TSV, XLSX, JSON or JSONL format to this tool and it
will help you convert it into a fine-tuning ready dataset.
```

- The data will be converted to jsonl format and stored in the same directory as your initial file.

```
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
```

GPT-3 Fine-tuning

https://beta.openai.com/docs/guides/fine-tuning

Actual work on Automatic Film Plot & Script Generation

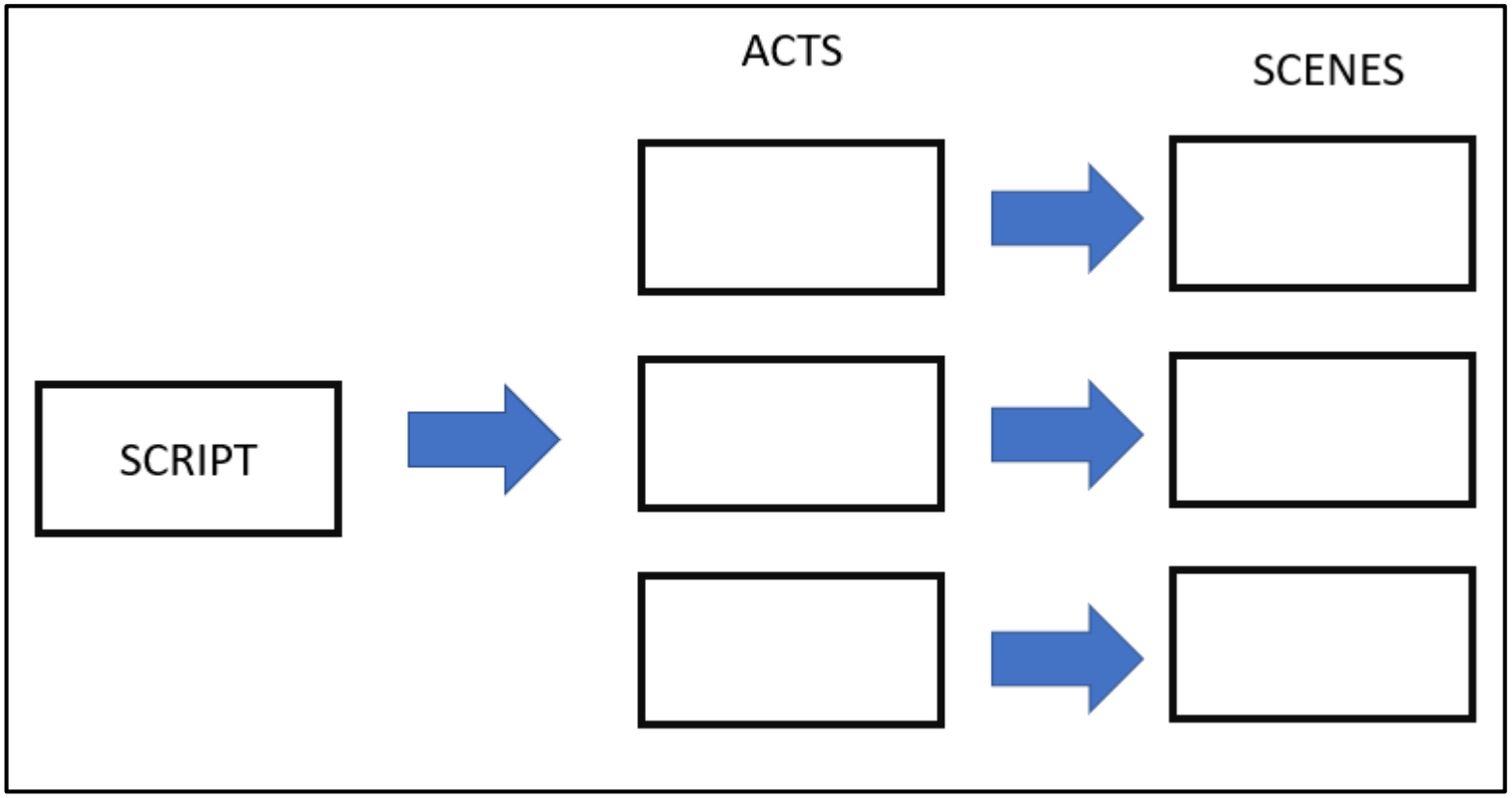
Problem Statement

Create a Writer's Assistant tool that can help the writer to generate multiple new ideas for a plot and the next scene of a screenplay.

BACKGROUND: TERMINOLOGY & DEFINITIONS

- A script is made up of 3 Acts: the set-up, conflict, and resolution.
- A standard movie script consists of 120 pages and the average screen time is 2hrs.
- Each page will be approximately 1 min of the screen time.

Hierarchical Structure of a Movie Script



ELEMENTS OF A SCENE - 1/2

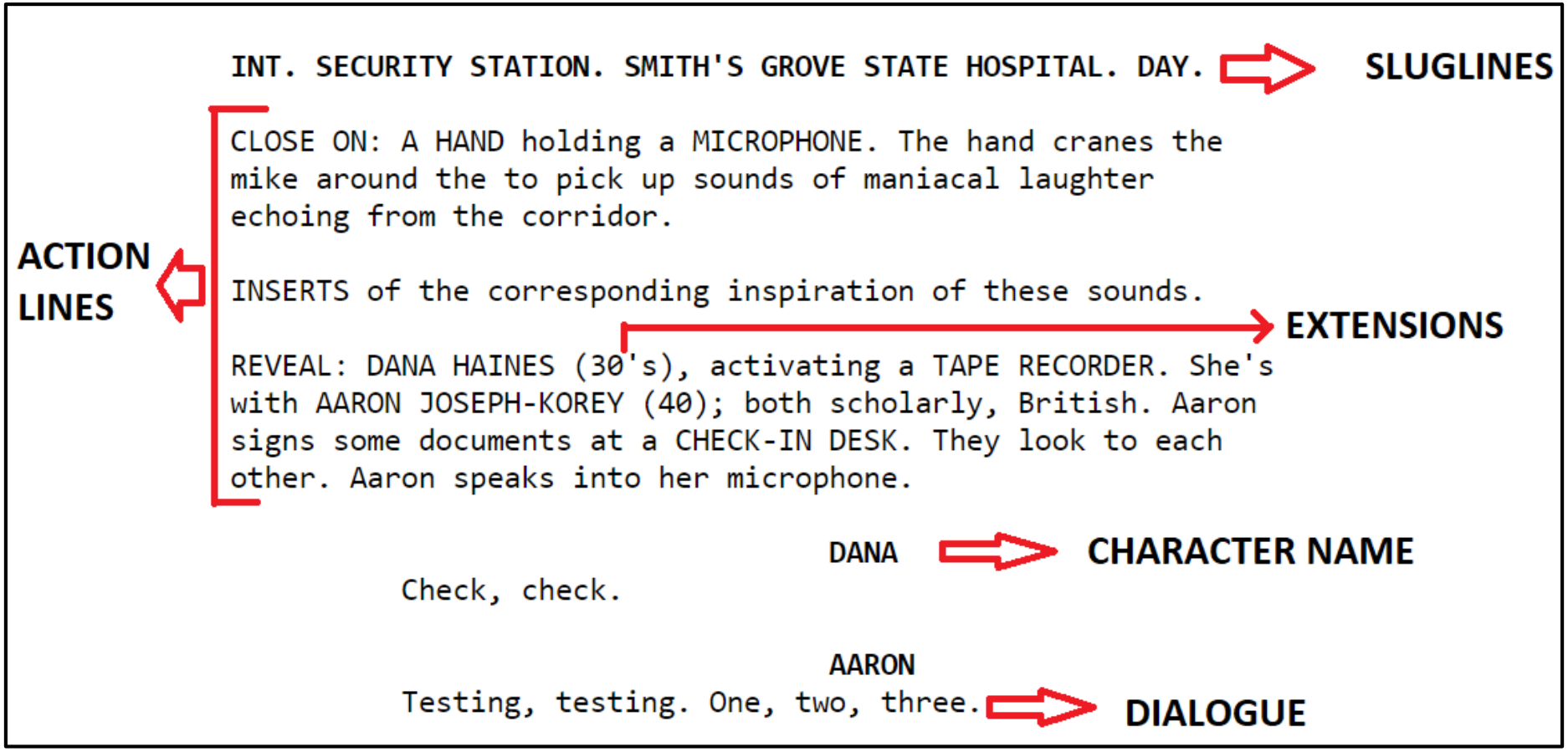
- Scene heading / sluglines - Written in capital letters and this part tells about the place and time when the scene is taking place.Eg: INT.-THE PALACE-DAY TIME
- Action lines-This comes just after the Scene Heading. The action lines tells what is happening in the scene
- Character names-The characters those are present in the scene.

ELEMENTS OF A SCENE - 2/2

- Dialogues of the character
- Extensions-occur along with the character name that provides extra information about the character or the dialogue of the character.

  Eg:V.O(Voice Over),O.S(Off Screen) etc.
- Transitions-These are keywords used to change from one scene to another.

  Eg:CUT TO:, INTERCHANGE TO:, FADE IN: etc.


INT. SECURITY STATION. SMITH'S GROVE STATE HOSPITAL. DAY.
SLUGLINES
CLOSE ON: A HAND holding a MICROPHONE. The hand cranes the mike around the to pick up sounds of maniacal laughter echoing from the corridor.
INSERTS of the corresponding inspiration of these sounds.
ACTION LINES
EXTENSIONS
REVEAL: DANA HAINES (30's), activating a TAPE RECORDER. She's with AARON JOSEPH-KOREY (40); both scholarly, British. Aaron signs some documents at a CHECK-IN DESK. They look to each other. Aaron speaks into her microphone.
DANA
CHARACTER NAME
Check, check.
AARON
Testing, testing. One, two, three.
DIALOGUE

First Scene Generation

- We collected movie scripts from IMSDb and storylines from IMDb.
- We separated the first scene by finding scene headings and keeping an upper limit of 500 words.
- Input: Storyline (~20-40 words)
- Output: Scene (~400 words)

Annotation of the scripts

We used the following the tags to help the model understand the structure of the script:

- SlugLines:<bsl> … <esl>
- Action Lines: <bal> … <eal>
- Character Name: <bcn> … <ecn>
- Dialogue: <bd> … <ed>

Plot Generation

- Input: A short prompt (< 150 words).
- Output: A movie plot (> 500 words).
- The dataset for input was collected from IMDb.
- The dataset for output was collected from Wikipedia.

Wikipedia Movie Plots

Plot example: Die Hard (1988) - 590 words

On Christmas Eve, New York City Police Department (NYPD) Detective John McClane arrives in Los Angeles, hoping to reconcile with his estranged wife, Holly, at a party held by her employer, the Nakatomi Corporation. He is driven to Nakatomi Plaza by a limo driver, Argyle, who offers to wait for McClane in the garage. While McClane changes clothes, the tower is seized by German radical Hans Gruber and his heavily armed team, including Karl and Theo. Everyone in the tower is taken hostage except for McClane, who slips away.

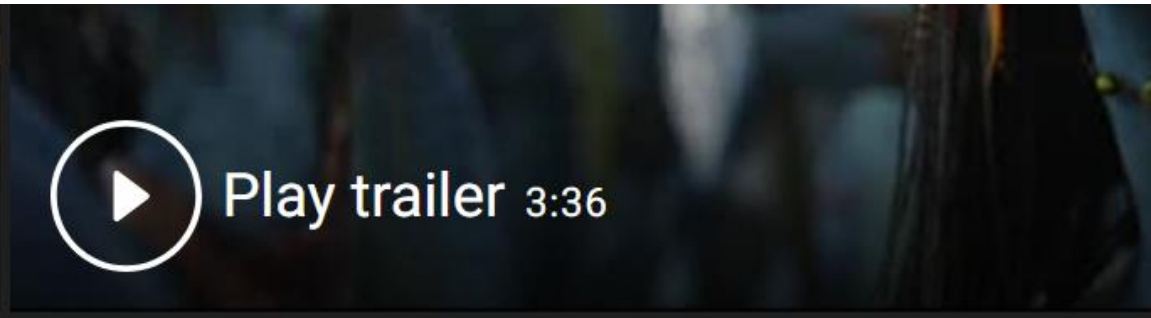
Gruber is posing as a terrorist to steal the $640 million in untraceable bearer bonds in the building's vault. He murders executive Joseph Takagi after failing to extract the access code from him, and tasks Theo with breaking into the vault. The terrorists are alerted to McClane's presence and one of them, Tony, is sent after him. McClane kills Tony and takes his weapon and radio, which he uses to contact the skeptical Los Angeles Police Department (LAPD), and Sergeant Al Powell is sent to investigate. Meanwhile, McClane kills more terrorists and recovers their bag of C-4 and detonators. Having found nothing amiss, Powell is about to leave until McClane drops a terrorist's corpse onto his car. After Powell calls for backup, a SWAT team attempts to storm the building but is assaulted by the terrorists. McClane throws some C-4 down an elevator shaft, causing an explosion that kills some of the terrorists and ends the assault.

Holly's co-worker Harry Ellis attempts to negotiate on Gruber's behalf, but when McClane refuses to surrender, Gruber kills Ellis. While checking the explosives on the roof, Gruber encounters McClane and pretends to be an escaped hostage; McClane gives Gruber a gun. Gruber attempts to shoot McClane but finds the weapon is unloaded, and is saved only by the intervention of other terrorists. McClane escapes but is injured by shattered glass and loses the detonators. Outside, Federal Bureau of Investigation (FBI) agents take control. They order the power to be shut off which, as Gruber had anticipated, disables the final vault lock so his team can collect the bonds.

The FBI agrees to Gruber's demand for a helicopter, intending to send gunship helicopters to eliminate the group. McClane realizes Gruber plans to blow the roof to kill the hostages and fake his team's deaths. Karl, enraged by the death of his brother Tony, attacks McClane and is apparently killed. Gruber sees a news report by Richard Thornburg on McClane's children and deduces that he is Holly's husband. The hostages are taken to the roof while Gruber keeps Holly with him. McClane drives the hostages from the roof just before Gruber detonates it and destroys the approaching FBI helicopters. Meanwhile, Theo retrieves a van from the parking garage but is neutralized by Argyle, who has been following events on his car radio.

A weary and battered McClane finds Holly with Gruber and his remaining henchman. McClane surrenders to Gruber and is about to be shot, but grabs his concealed service pistol taped to his back and uses his last two bullets to wound Gruber and kill his accomplice. Gruber crashes through a window but grabs onto Holly's wristwatch and makes a last-ditch attempt to kill the pair before McClane unclasps the watch and Gruber falls to his death. Outside, Karl ambushes McClane and Holly but is killed by Powell. Holly punches Thornburg when he attempts to interview McClane before Argyle crashes through the parking garage door in the limo and drives McClane and Holly away together.

IMDb Prompts



Action | Adventure | Fantasy

A paraplegic Marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.

Director James Cameron

Plot Annotation

- ACT ONE- This marks the paragraph in the plot where the characters are introduced
- ACT TWO A- This part depicts the phase where the protagonist has a love story and the premise promised in the trailer is usually shown. The audience love this part.
- ACT TWO B- In this part the protagonist face challenges and hurdles in life and reaches rock bottom.
- ACT THREE- The protagonist overcomes the hurdles and the story reaches a conclusion.

Demo

https://www.cfilt.iitb.ac.in/generator/

References

1. https://huggingface.co/blog/how-to-generate
2. https://jalammar.github.io/illustrated-gpt2/
3. https://harishgarg.com/writing/how-to-fine-tune-gpt-3-api/
4. https://pakodas.substack.com/p/finetuning-gpt3-with-openai?s=r
5. https://beta.openai.com/docs/guides/fine-tuning

NMT with Transformers using OpenNMT-py

Installing Dependencies

```
# Download Files
!pip3 install gdown -U
# Neural Machine Translation Library
!pip3 install OpenNMT-py
!git clone https://github.com/moses-smt/mosesdecoder.git
!pip3 install indic_nlp_library
!git clone https://github.com/anoopkunchukuttan/indic_nlp_library.git
!pip3 install -r indic_nlp_library/requirements.txt
!git clone https://github.com/anoopkunchukuttan/indic_nlp_resources.git
!pip3 install https://github.com/rsennrich/subword-nmt/archive/master.zip
!pip3 install sacrebleu
!pip3 install ctranslate2
!pip3 install mosestokenizer
```

Set Environment Variables

```
%%bash
export PYTHONPATH=$PYTHONPATH:/content/indic_nlp_library
export INDIC_RESOURCES_PATH=/content/indic_nlp_resources

echo $INDIC_RESOURCES_PATH
```

Download Dataset

```
!wget https://storage.googleapis.com/samanantar-public/V0.2/data/en2indic/en-mr.zip
!unzip en-mr.zip

sourceValidLen = 1000
targetValidLen = 1000
sourceTestLen = 1000
```

Train-Test-Validation Splits(1/4)

```
!mkdir data
sourceData = open("en-mr/train.en",
'r').readlines()
targetData = open("en-mr/train.mr",
'r').readlines()

sourceValid =
sourceData[0:sourceValidLen]
targetValid =
```

Train-Test-Validation Splits(2/4)

```
sourceTrain =
sourceData[sourceValidLen +
sourceTestLen:sourceValidLen +
sourceTestLen + sourceTrainLen]
targetTrain = targetData[targetValidLen
+ targetTestLen:targetValidLen +
targetTestLen + targetTrainLen]

sourceTestFile = open("data/test.en",
```

Train-Test-Validation Splits(3/4)

```
sourceValidFile = open("data/valid.en", "w+")
for line in sourceValid:
    sourceValidFile.write(line.strip("\n") + "\n")
sourceValidFile.close()

targetValidFile = open("data/valid.mr", "w+")
```

Train-Test-Validation Splits(4/4)

```
sourceTrainFile = open("data/train.en", "w+")
for line in sourceTrain:
    sourceTrainFile.write(line.strip("\n") + "\n")
sourceTrainFile.close()

targetTrainFile = open("data/train.mr", "w+")
```

Lowercase English Text

```
!/content/mosesdecoder/scripts/tokenizer/lowercase.perl < data/train.en > data/train-low.en

!/content/mosesdecoder/scripts/tokenizer/lowercase.perl < data/test.en > data/test-low.en

!/content/mosesdecoder/scripts/tokeniz
```

Tokenize English Text

```
!/content/mosesdecoder/scripts/tokenizer/tokenizer.perl < data/train-low.en > data/train-tok.en

!/content/mosesdecoder/scripts/tokenizer/tokenizer.perl < data/test-low.en > data/test-tok.en

!/content/mosesdecoder/scripts/tokeniz
```

Normalize Marathi Text

```
!python3 indic_nlp_library/indicnlp/normalize/indic_normalize.py data/train.mr data/train-norm.mr mr
```

```
!python3 indic_nlp_library/indicnlp/normalize/indic_normalize.py data/valid.mr data/valid-norm.mr mr
```

Tokenize Marathi Text

```
!python3
indic_nlp_library/indicnlp/tokenize/indic
_tokenize.py data/train-norm.mr
data/train-tok.mr mr
```

```
!python3
indic_nlp_library/indicnlp/tokenize/indic
_tokenize.py data/valid-norm.mr
data/valid-tok.mr mr
```

Byte Pair Encoding Subwordization(1/2)

```
!mkdir codes
!mkdir data/bpe

!subword-nmt learn-bpe -s 8000 --num-workers 40 < data/train-tok.en > codes/codes.en

!subword-nmt apply-bpe --num-workers 40 -c codes/codes.en < data/train-
```

Byte Pair Encoding Subwordization(2/2)

```
!subword-nmt learn-bpe --num-workers 40 -s 8000 < data/train-tok.mr > codes/codes.mr

!subword-nmt apply-bpe --num-workers 40 -c codes/codes.mr < data/train-tok.mr > data/bpe/train-bpe.mr
!subword-nmt apply-bpe --num-workers 40 -c codes/codes.mr < data/test-
```

Create Vocabulary (1/2)

```
!mkdir data/pre
vocab = """
save_data: data/pre
src_vocab: data/pre/vocab.en
tgt_vocab: data/pre/vocab.mr
overwrite: False
data:
    corpus_1:
        path_src: data/hne/train.hne.en
```

Create Vocabulary (2/2)

```
createVocabYaml =
open("vocab.yaml", 'w+')
createVocabYaml.write(vocab)
createVocabYaml.close()

!onmt_build_vocab -config vocab.yaml
-n_sample -1
```

Train NMT (1/9)

```
!mkdir checkpoints
!mkdir tensorboard
train = """
save_data: data/pre
src_vocab: data/pre/vocab.mr
tgt_vocab: data/pre/vocab.en
overwrite: False
src_seq_length: 200
```

Train NMT (2/9)

```
data:
    corpus_1:
      path_src: data/bpe/train-bpe.en
      path_tgt: data/bpe/train-bpe.mr
      transforms: [filtertoolong]
    valid:
      path_src: data/bpe/valid-bpe.en
      path_tgt: data/bpe/valid-bpe.mr
      transforms: [filtertoolong]
```

Train NMT (3/9)

```
# Training
world_size: 1
gpu_ranks: [0]
master_port: 5001
# General opts
log_file: checkpoints/log.txt
save_model: checkpoints/model
```

Train NMT (4/9)

```
tensorboard_log_dir: "tensorboard"
tensorboard: true
keep_checkpoint: 20
save_checkpoint_steps: 10000
average_decay: 0.0005
seed: 1234
report_every: 1
train_steps: 300000
valid_steps: 10000
```

Train NMT (5/9)

```
# Batching
queue_size: 10000
bucket_size: 32768
pool_factor: 8192
batch_type: "tokens"
batch_size: 4096
valid_batch_size: 16
batch_size_multiple: 1
```

Train NMT (6/9)

```
# Optimization
model_dtype: "fp16"
optim: "adam"
learning_rate: 2
warmup_steps: 8000
decay_method: "noam"
adam_beta1: 0.9
adam_beta2: 0.998
max_grad_norm: 0
label_smoothing: 0.1
param_init: 0.0
param_init_glorot: "true"
normalization: "tokens"
```

Train NMT (7/9)

```
# Model
encoder_type: transformer
decoder_type: transformer
enc_layers: 3
dec_layers: 3
heads: 4
rnn_size: 256
dec_rnn_size: 256
```

Train NMT (8/9)

```
word_vec_size: 256
transformer_ff: 1024
dropout_steps: [0]
dropout: [0.1]
attention_dropout: [0.1]
share_decoder_embeddings: "true"
position_encoding: "true"
"""

trainYaml = open("train.yaml",'w+')
```

Train NMT (9/9)

```
! CUDA_VISIBLE_DEVICES=0
onmt_train -config train.yaml
```

Inference: Download Test Set

```
url =
'https://drive.google.com/drive/folders/1
hy80UglVd7dUA_osGbl50BF4NWbp2
msd?usp=sharing'
gdown.download_folder(url,
quiet=True)

url =
'https://drive.google.com/drive/folders/1
```

Inference: Prepare Data

```
!mkdir test

!/content/mosesdecoder/scripts/tokenizer/lowercase.perl < covid-19-testset/test.en > covid-19-testset/test-low.en

!/content/mosesdecoder/scripts/tokenizer/tokenizer.perl < covid-19-testset/test-
```

Inference: Translate

```
! CUDA_VISIBLE_DEVICES=0 onmt_translate \
        -gpu 0 \
        -batch_size 4096 -batch_type tokens \
        -beam_size 5 \
        -model checkpoints/model.pt \
        -src covid-19-testset/test-bpe.en \
        -output test/test.mr \
        -replace_unk
! sed -r -i 's/(@@ )|(@@ ?$)//g' test/test.mr
! sed -r -i 's/ &amp;apos;//g' test/test.mr
!python3 indic_nlp_library/indicnlp/tokenize/indic_detokenize.py
test/test.mr test/test-detok.mr mr
```

Compute BLEU Score

```
import sacrebleu
hypothesis = open("test/test-detok.mr", 'r').readlines()
reference = open("covid-19-testset/test.mr", 'r').readlines()
hypothesisSentences, referenceSentences = [], []
for i in range(len(hypothesis)):
    hypothesisSentence = hypothesis[i].strip("\n")
    referenceSentence = reference[i].strip("\n")
    hypothesisSentences.append(hypothesisSentence)
    referenceSentences.append(referenceSentence)
bleu = sacrebleu.corpus_bleu(hypothesisSentences,
[referenceSentences])
print(bleu.score)
```

Deploy Model

```
import ctranslate2
model = "checkpoints/model.pt"
converter =
ctranslate2.converters.OpenNMTPyCo
nverter(model)
!mkdir model_deploy
output = "model_deploy"
converter.convert(output,
    force=True
```

Inference (1/4)

```
from mosestokenizer import
MosesSentenceSplitter,
MosesTokenizer

from indicnlp.tokenize import
sentence_tokenize, indic_tokenize
from indicnlp.normalize.indic_normalize
import IndicNormalizerFactory
```

Inference (2/4)

```
## Tokenize
englishTokenizer =
MosesTokenizer('en')
## BPE
englishBpeCodes =
codecs.open("codes/codes.en",
encoding='utf-8')
englishBpeEncoder =
BPE(englishBpeCodes)
```

Inference (3/4)

```
# Lowercase
sourceSentence =
sourceSentence.lower()

# Tokenize
sourceSentence = '
'.join(englishTokenizer(sourceSentence
))
```

Inference (4/4)

```
# Translate
targetSentence =
translator.translate_batch([sourceSente
nce], beam_size=5,
max_batch_size=16)

# Remove BPE
targetSentence = ('
'.join(targetSentence[0].hypotheses[0])
```

Thank You