KSBi-BIML 2023

Bioinformatics & Machine Learning(BIML)
Workshop for Life Scientists, Data Scientists,
and Bioinformatians

생물정보학&머신러닝 워크샵(오프라인)

Pre-trained Models for Transfer Learning

전민지 _ 고려대학교

SBi 한국생명정보학회
Korean Society for Bioinformatics

본 강의 자료는 한국생명정보학회가 주관하는 BIML 2023 워크샵 오프라인 수업을 목적으로 제작된 것으로 해당 목적 이외의 다른 용도로 사용할 수 없음을 분명하게 알립니다.

이를 다른 사람과 공유하거나 복제, 배포, 전송할 수 없으며 만약 이러한 사항을 위반할 경우 발생하는 **모든 법적 책임은 전적으로 불법 행위자 본인에게 있음을 경고**합니다.

KSBi-BIML 2023

Bioinformatics & Machine Learning (BIML)
Workshop for Life Scientists, Data Scientists, and Bioinformatians

안녕하십니까?

한국생명정보학회가 개최하는 동계 교육 워크샵인 BIML-2023에 여러분을 초대합니다. 생명정보학 분야의 연구자들에게 최신 동향의 데이터 분석기술을 이론과 실습을 겸비해 전달하고자 도입한 전문 교육 프로그램인 BIML 워크샵은 2015년에 시작하여 올해로 9차를 맞이하게 되었습니다. 지난 2년간은 심각한 코로나 대유행으로 인해 아쉽게도 모든 강의가 온라인으로 진행되어 현장 강의에서만 가능한 강의자와 수강생 사이에 다양한 소통의 기회가 없음에 대한 아쉬움이 있었습니다. 다행히도 최근 사회적 거리두기 완화로 현장 강의가 가능해져 올해는 현장 강의를 재개함으로써 온라인과 현장 강의의 장점을 모두 갖춘 프로그램을 구성할 수 있게 되었습니다.

BIML 워크샵은 전통적으로 크게 인공지능과 생명정보분석 두 개의 분야로 구성되었습니다. 올해 AI 분야에서는 최근 생명정보 분석에서도 응용이 확대되고 있는 다양한 심층학습(Deep learning) 기법들에 대한 현장 강의가 진행될 예정이며, 관련하여 심층학습을 이용한 단백질구조예측, 유전체 분석, 신약개발에 대한 이론과 실습 강의가 함께 제공될 예정입니다. 또한 싱글셀오믹스 분석과 메타유전체분석 현장 강의는 많은 연구자의 연구 수월성 확보에 큰 도움을 줄 것으로 기대하고 있습니다. 이외에 다양한 생명정보학 분야에 대하여 30개 이상의 온라인 강좌가 개설되어 제공되며 온라인 강의의 한계를 극복하기 위해서 실시간 Q&A 세션 또한 마련했습니다. 특히 BIML은 각 분야 국내 최고 전문가들의 강의로 구성되어 해당 분야의 기초부터 최신 연구 동향까지 포함하는 수준 높은 내용의 강의가 될 것입니다.

이번 BIML-2023을 준비하기까지 너무나 많은 수고를 해주신 BIML-2023 운영위원회의 남진우, 우현구, 백대현, 정성원, 정인경, 장혜식, 박종은 교수님과 KOBIC 이병욱 박사님께 커다란 감사를 드립니다. 마지막으로 부족한 시간에도 불구하고 강의 부탁을 흔쾌히 허락하시고 훌륭한 현장 강의와 온라인 강의를 준비하시는데 노고를 아끼지 않으신 모든 연사분께 깊은 감사를 드립니다.

2023년 2월

한국생명정보학회장 이 인 석

강의 시간표

DAY1 (2.6 월)

시간	강 의 서울대 자연과학대학 26동B101호	강사	강 의 서울대 자연과학대학 26동B102호	강사
09:00-09:20 (20)	등록			
09:20-09:30 (10)	개회사/공지사항전달			
09:30-10:50 (80)	**Best practice for single-cell data analysis**	**박종은 교수**	**Introduction to ML & DNN (이론)**	**이상근 교수**
10:50-11:00 (10)	휴식			
11:00-12:10 (70)	**Practice1: Scanpy basic workflow**	**김우석 김성룡 조교**	**CNN (이론)**	**이상근 교수**
12:10-13:40 (90)	점심 (KOBIC 세미나)			
13:40-15:10 (90)	**Public data, batch correction, cell annotation**	**박종은 교수**	**RNN, GAN, XAI (이론)**	**이상근 교수**
15:10-15:20 (10)	휴식			
15:20-16:50 (90)	**Practice2: Advanced single-cell analysis**	**김우석 김성룡 조교**	**AI 모델 구조 정의, 학습 알고리즘 적용, 성능 평가, 시각화 방법 (Tensorflow 실습)**	**이정현 한성민 조교**

DAY2 (2.7 화)

시간	강 의 서울대 자연과학대학 26동B101호	강사	강 의 서울대 자연과학대학 26동B102호	강사
09:00-09:20 (20)	등록			
09:20-09:30 (10)	공지사항전달			
09:30-10:50 (80)	**Introduction to protein structure prediction - Homology modeling - Coevolution-guided modeling Early AI-based approaches**	**백민경 교수**	**Pre-trained Models for Transfer Learning (이론)**	**전민지 교수**
10:50-11:00 (10)	휴식			
11:00-12:10 (70)	**단백질 구조 예측 실습 - MSA generation, template search - homology modeling contact prediction & modeling**	**백민경 교수**	**Pre-trained Models for Transfer Learning (실습)**	**정민수 조교**
12:10-13:40 (90)	점심			
13:40-15:10 (90)	**AI-based protein structure prediction - AlphaFold/RoseTTAFold Applications to PPI prediction & protein design**	**백민경 교수**	**Deep learning in Bioinformatics**	**노미나 교수**
15:10-15:20 (10)	휴식			
15:20-16:50 (90)	**단백질 구조 예측 실습 II AlphaFold, RoseTTAFold 실습 및 응용**	**백민경 교수**	**Deep learning model을 이용한 실습**	**곽호진 박예솔 조교**

DAY3 (2.8 수)

시간	강 의 서울대 자연과학대학 26동B101호	강사	강 의 서울대 자연과학대학 26동B102호	강사
09:00-09:20 (20)	등록			
09:20-09:30 (10)	공지사항전달			
09:30-10:50 (80)	**화학정보학 기초(Cheminformatics) 약물특성 및 약물다움(druglikeness) Molecular Notations & Descriptors AI 신약개발을 위한 Databases AI 신약개발을 위한 Programming 기초**	**김동섭 교수**	**마이크로바이옴 기본 이론**	**이선재 교수**
10:50-11:00 (10)	휴식			
11:00-12:10 (70)	**Google Colab에 RDKit 설치 화합물 정보 읽기 실습 Bioactivity database 검색 및 정보 읽기 실습 Molecular descriptor (fingerprint) 생성 및 similarity 계산 실습**	**문채영 나민주 조교**	**16S rRNA amplicon seq. - DADA2**	**서영창 조준우 조교**
12:10-13:40 (90)	점심 (KOBIC 세미나)			
13:40-15:10 (90)	**AI 신약개발을 위한 기계학습법 기초 QSAR 모델링 기초 AI 신약개발을 위한 딥러닝 모델 Virtual screening (ligand-based, structure-based) 및 de novo design**	**김동섭 교수**	**최신 메타지놈 분석 기법의 현황**	**이선재 교수**
15:10-15:20 (10)	휴식			
15:20-16:50 (90)	**QSAR modeling 전체 과정 실습 화합물의 Bioactivity 예측 모델 개발 Virtual screening 과정을 통한 신약후보물질 발굴 실습**	**문채영 나민주 조교**	**Shotgun metagenome 분석 (Linux)**	**서영창 조준우 조교**

강의개요

Pre-trained Models for Transfer Learning

성능이 좋은 딥러닝 모델을 만들기 위해서는 수많은 데이터를 확보해야 한다. 하지만 데이터를 확보하는 것이 쉽지 않을 수 있으며 특히 생명의료 분야에서는 데이터를 생성하는 데에 많은 비용과 시간이 든다는 어려움이 존재한다. 이를 해결하기 위해 Transfer learning (전이학습)을 이용할 수 있다. Transfer learning이란, 어떤 태스크를 위해 학습된 모델 (Pre-trained model)을 다른 태스크에 이용하는 것을 의미한다. Transfer learning을 적용하기 위해서는 다양한 Pre-trained model을 이해하는 것이 중요하다.

본 강의에서는 Transfer learning에 활용할 수 있는 대표적인 Pre-trained model을 설명한다. LeNet, AlexNet, VGG-16, ResNet, GoogLeNet 등 CNN 모델과, BERT, GPT 등 Transformer 기반의 language model에 대해 핵심 내용을 배우고, 이를 실제 데이터에 적용하는 역량을 갖추는 것을 목표로 한다.

강의는 다음의 내용을 포함한다:

- Transfer learning 개요
- Pre-trained model 종류
- 실습

* 교육생준비물:
  노트북 (메모리 8GB 이상, 디스크 여유공간 30GB 이상)

* 강의 난이도: 중급

* 강의: 전민지 교수 (고려대학교 의과대학)

Curriculum Vitae

Speaker Name: Minji Jeon, Ph.D.

▶ **Personal Info**

Name	Minji Jeon
Title	Assistant Professor
Affiliation	Korea University

▶ **Contact Information**

Address	161, Jeongneung-ro, Seongbuk-gu, Seoul, 02708
Email	mjjeon@korea.ac.kr
Phone Number	010-2354-7084

Research Interest

AI-driven drug discovery, machine learning, bioinformatics

Educational Experience

2012	B.S. in Computer Science, Korea University, Korea
2014	M.S. in Interdisciplinary Graduate Program in Bioinformatics, Korea University, Korea
2018	Ph.D. in Computer Science, Korea University, Korea

Professional Experience

2018-2019	Research Professor, Korea University, Korea
2020-2022	Postdoctoral Fellow, Icahn School of Medicine at Mount Sinai, USA
2022-	Assistant Professor, Korea University, Korea

Selected Publications (5 maximum)

1. Zhaoping Xiong†, **Minji Jeon**†, Robert J Allaway†, Jaewoo Kang, Donghyeon Park, Jinhyuk Lee, Hwisang Jeon, Miyoung Ko, Hualiang Jiang, Minyue Zheng, Aik Choon Tan, Xindi Guo, The Multi-Targeting Drug DREAM Challenge Community, Kristen K Dang, Alex Tropsha, Chana Hecht, Tirtha K. Das, Heather A. Carlson, Ruben Abagyan, Justin Guinney, Avner Schlessinger*, Ross Cagan* "Crowdsourced identification of multi-target kinase inhibitors for RET- and TAU-based disease: the Multi-Targeting Drug DREAM Challenge" PLoS computational biology 17.9 (2021): e1009302.
2. **Minji Jeon**†, Kathleen M. Jagodnik†, Eryk Kropiwnicki, Daniel J. Stein, Avi Ma'ayan* "Prioritizing Pain-Associated Targets with Machine Learning" Biochemistry 60.18 (2021): 1430-1446.
3. **Minji Jeon**, Donghyeon Park, Jinhyuk Lee, Hwisang Jeon, Miyoung Ko, Sunkyu Kim, Yonghwa Choi, Aik-Choon Tan, Jaewoo Kang* "ReSimNet: Drug Response Similarity Prediction using Siamewe Neural Networks" Bioinformatics 35.24 (2019): 5249-5256.

4. Michael Patrick Menden, Dennis Wang, Yuanfang Guan, Michael Mason, Bence Szalai, Krishna C Bulusu, Thomas Yu, Jaewoo Kang, **Minji Jeon**, Russ Wolfinger, Tin Nguyen, Mikhail Zaslavskiy, AstraZeneca-Sanger Drug Combination DREAM Consortium, In Sock Jang, Zara Ghazoui, Mehmet Eren Ahsen, Robert Vogel, Elias Chaibub Neto, Thea Norman, Eric KY Tang, Mathew J Garnett, Giovanni Di Veroli, Stephen Fawell, Gustavo Stolovitzky, Justin Guinney, Jonathan R Dry, Julio Saez-Rodriguez*, "Community assessment to advance computational prediction of cancer drug combinations in a pharmacogenomic screen" Nature Communications, 10.1 (2019): 2674.
5. **Minji Jeon**, Sunkyu Kim, Sungjoon Park, Heewon Lee, Jaewoo Kang* "In silico drug combination discovery for personalized cancer therapy" BMC systems biology, 2018, 12.2: 16.

KSBi-BIML 2023

Pre-trained Models for Transfer Learning

고려대학교 전민지

목차

- Introduction
- CNN models
- Transformer models

What is Transfer Learning?

- 일상 생활에서 전이 학습
  - 피아노를 칠 줄 아는 사람은 못 치는 사람보다 바이올린을 빨리 배움
  - C언어에 익숙한 학생은 python을 금방 배움

  → 두 영역의 공통 지식을 공유하기 때문

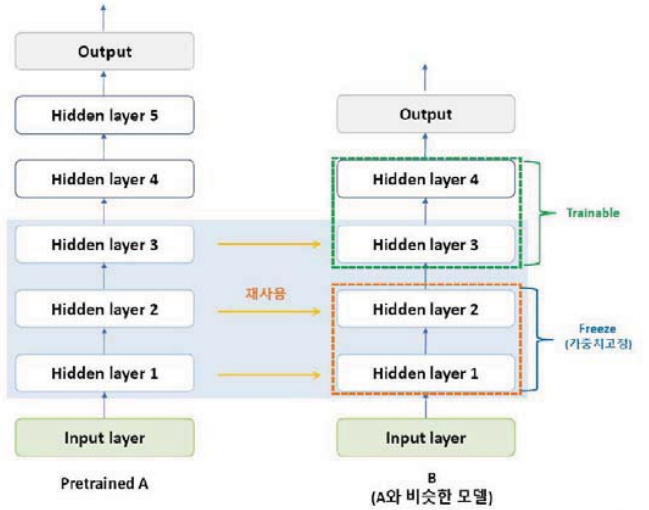
- 머신러닝에서 전이 학습
  - 어떤 도메인의 데이터로 학습한 모델을 다른 도메인의 데이터에 활용하여 성능 향상을 꾀하는 기법

What is Transfer Learning?

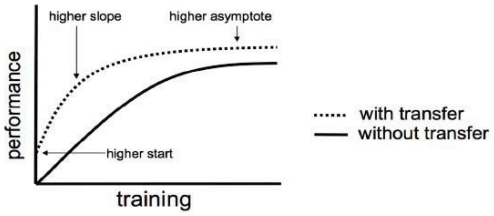
◦ 전이학습의 필요성
  - 학습 모델의 품질을 향상시키기 위해서는 많은 양의 데이터를 이용하여 학습해야 함
  - 많은 학습 데이터를 확보하려면 많은 비용과 시간이 소모되기 때문에, 이러한 데이터 부족 문제를 해결하기 위해 전이학습이 등장함

◦ 전이학습 개념
  - 특정 분야에서 풍부한 데이터를 이용해 학습된 pre-trained model을 가져와 데이터가 부족한 환경에 맞춰 재사용하는 머신러닝 학습 기법
  - 해결하고자 하는 문제의 정답이 소수만 존재하고, 해결하고자 하는 비슷한 문제의 정답은 다수 존재하는 경우 사용하는 지도학습
  - 예: 자연 영상으로 학습한 신경망을 새나 개의 종을 분류하거나 자율주행에서 차선과 보행자 인식에 사용

What is Transfer Learning?



〈자료〉 Hands-On Machine Learning with Scikit-Learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2017, Chapter 11. Training Deep Neural Nets 참조 재구성



〈자료〉 Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, Chapter 11. Transfer Learning, p.243.

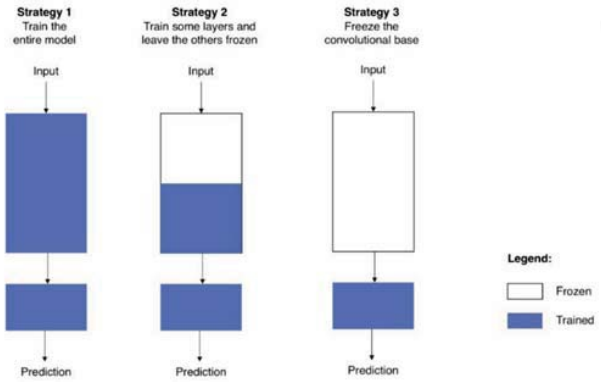
Pre-trained Models

표 6-1 텐서플로가 제공하는 예비 학습된 컨볼루션 신경망 모델(출처: https://keras.io/applications)

모델 이름	파일 크기	1순위 정확률	5순위 정확률	매개변수 개수	층의 개수(깊이)
Xception	88MB	0.790	0.945	22,910,480	126
VGG16	528MB	0.713	0.901	138,357,544	23
VGG19	549MB	0.713	0.900	143,667,240	26
ResNet50	98MB	0.749	0.921	25,636,712	–
ResNet101	171MB	0.764	0.928	44,707,176	–
ResNet152	232MB	0.766	0.931	60,419,944	–
ResNet50V2	98MB	0.760	0.930	25,613,800	–
ResNet101V2	171MB	0.772	0.938	44,675,560	–
ResNet152V2	232MB	0.780	0.942	60,380,648	–
InceptionV3	92MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215MB	0.803	0.953	55,873,736	572
MobileNet	16MB	0.704	0.895	4,253,864	88
MobileNetV2	14MB	0.713	0.901	3,538,984	88
DenseNet121	33MB	0.750	0.923	8,062,504	121
DenseNet169	57MB	0.762	0.932	14,307,880	169
DenseNet201	80MB	0.773	0.936	20,242,984	201
NASNetMobile	23MB	0.744	0.919	5,326,716	–
NASNetLarge	343MB	0.825	0.960	88,949,818	–

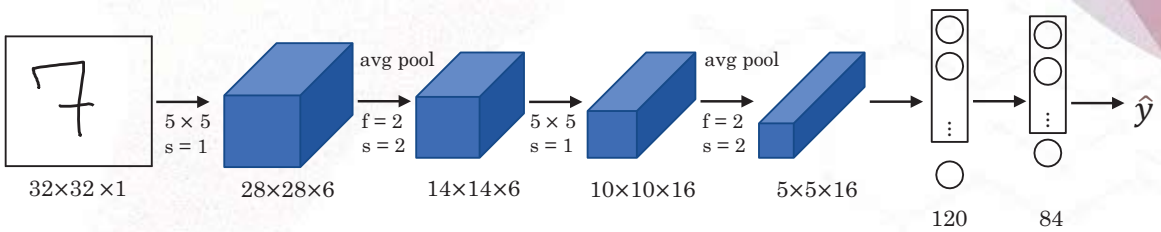
Transfer Learning Strategies

- 사전학습 모델 사용 방법 3가지
  1. 전체 모델을 새로 학습
  2. Convolutional base 일부분은 고정하고, 나머지 계층과 classifier를 새로학습
  3. Convolutional base 전체는 고정하고, classifier만 새로 학습



CNN
Classic Networks

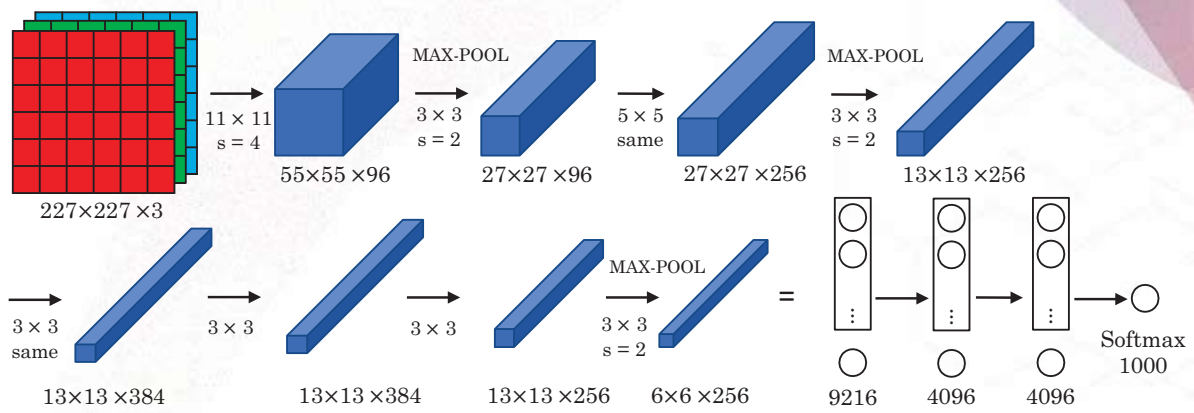
LeNet - 5



- # of parameters: 6k
- average pool
- activation function: sigmoid or tanh
- $n_H, n_W \downarrow$
- $n_C \uparrow$

[LeCun et al., 1998. Gradient-based learning applied to document recognition]

AlexNet



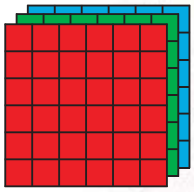
- Similar to Lenet but much bigger
- ~60M parameters
- ReLU as activation function

[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

VGG - 16

- ~138M parameters
- fixed filter size
- VGG-19

CONV = 3×3 filter, s = 1, same
MAX-POOL = 2×2 , s = 2



224×224 ×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

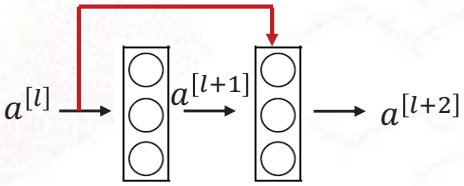
→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] ×3 → 28×28 ×512 → POOL → 14×14×512

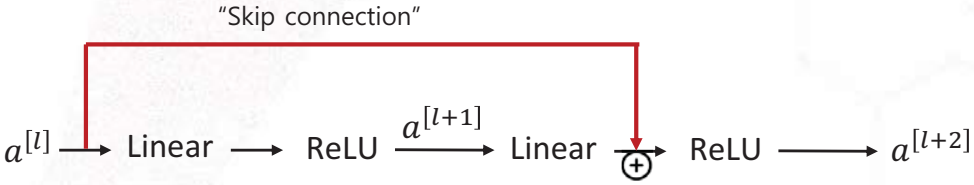
→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

CNN
ResNet

Residual block





$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$

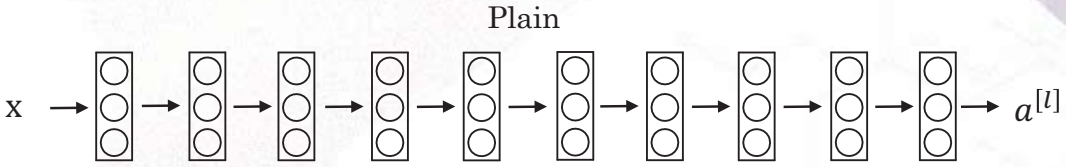
$$a^{[l+1]} = g(z^{[l+1]})$$

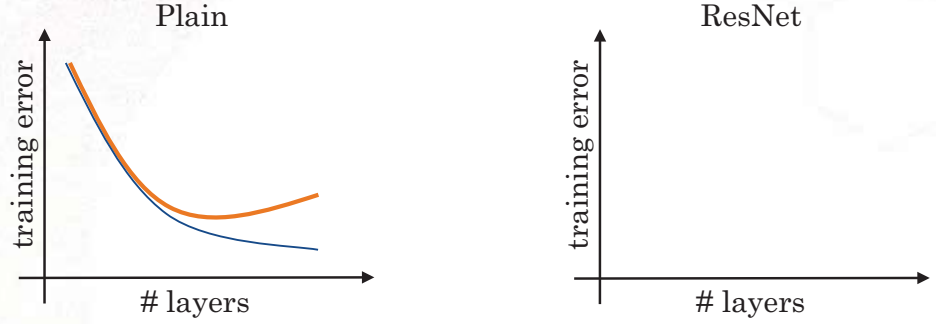
$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

[He et al., 2015. Deep residual networks for image recognition]

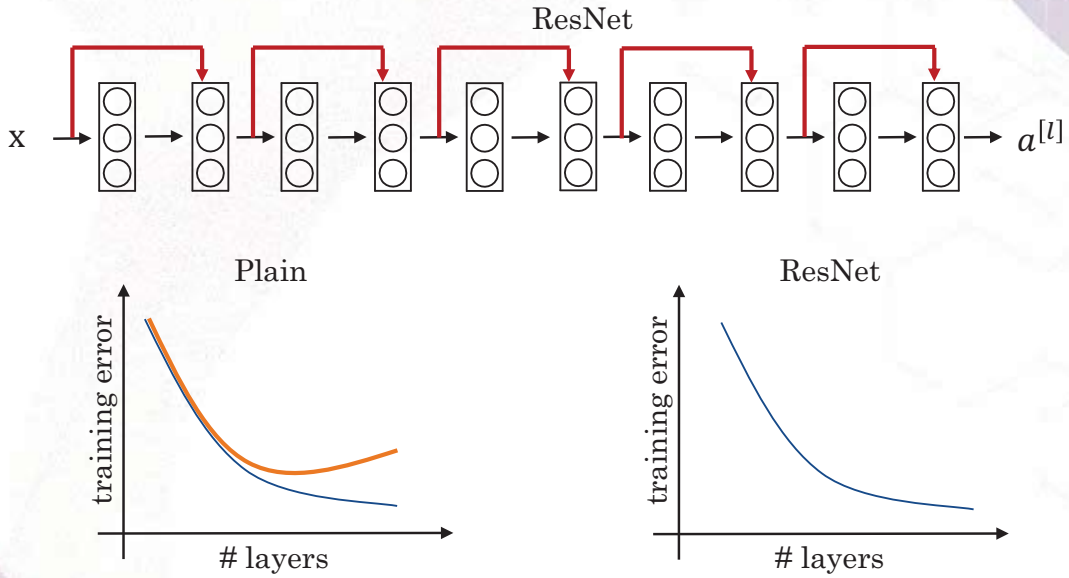
Residual Network





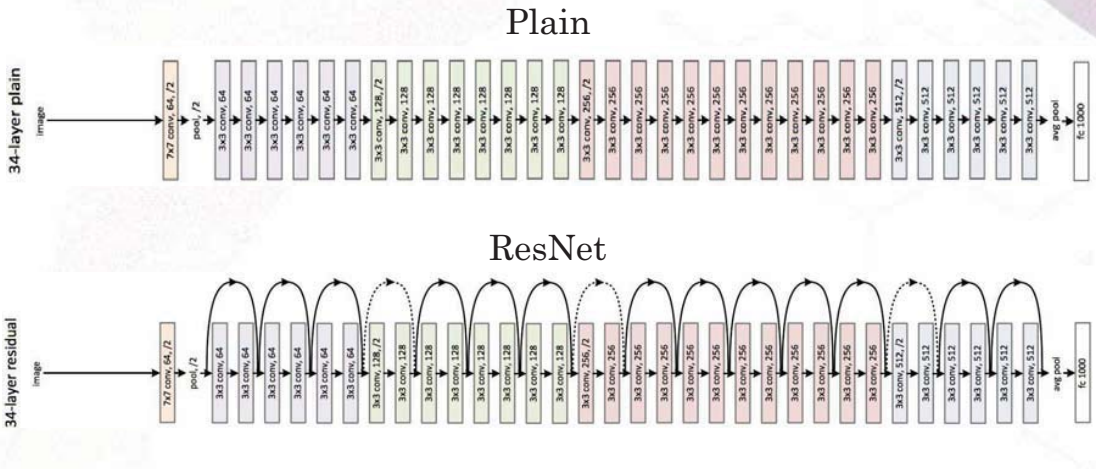
[He et al., 2015. Deep residual networks for image recognition]

Residual Network



[He et al., 2015. Deep residual networks for image recognition]

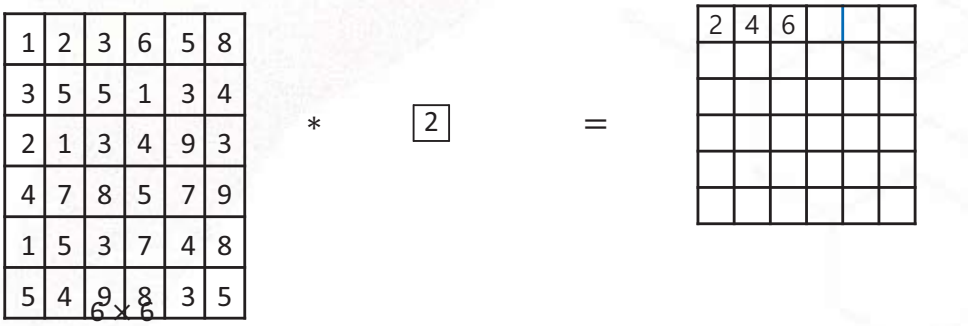
ResNet

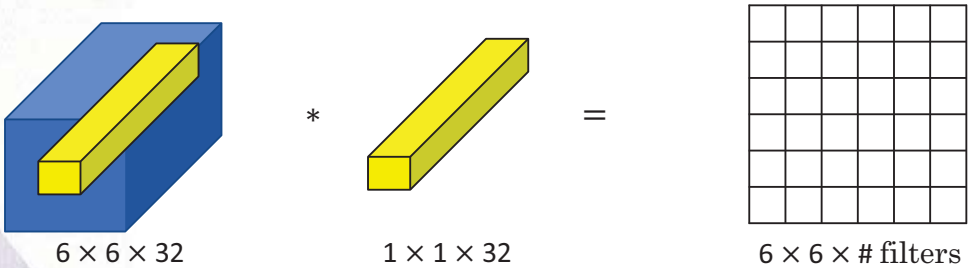


[He et al., 2015. Deep residual networks for image recognition]

CNN
Inception Network

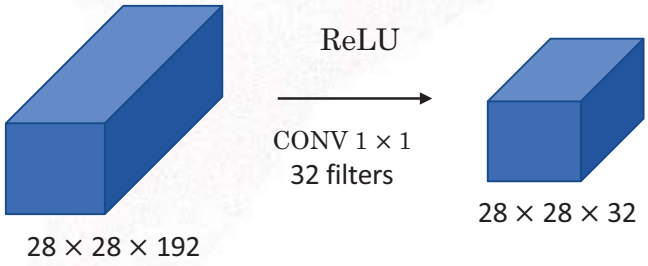
Why does a 1 × 1 convolution do?





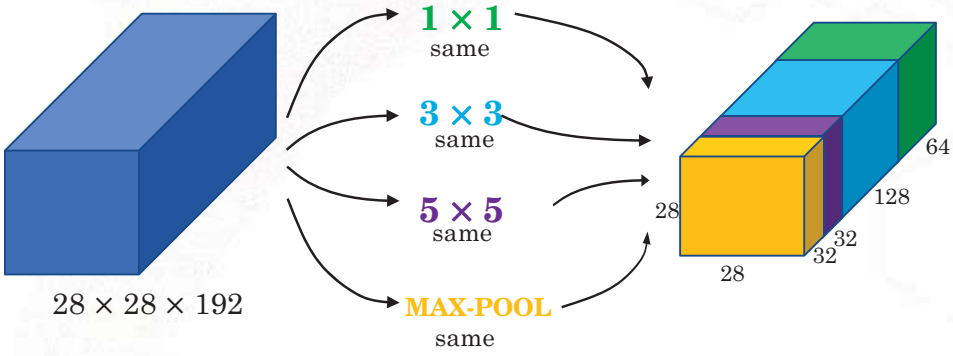
[Lin et al., 2013. Network in network]

Using 1×1 convolutions



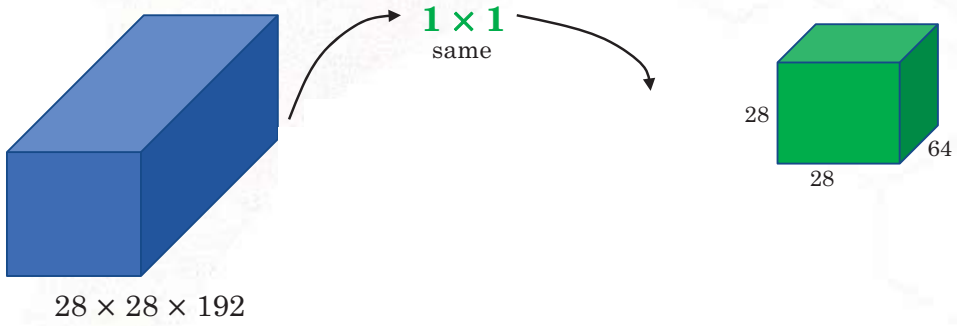
[Lin et al., 2013. Network in network]

Motivation for inception network



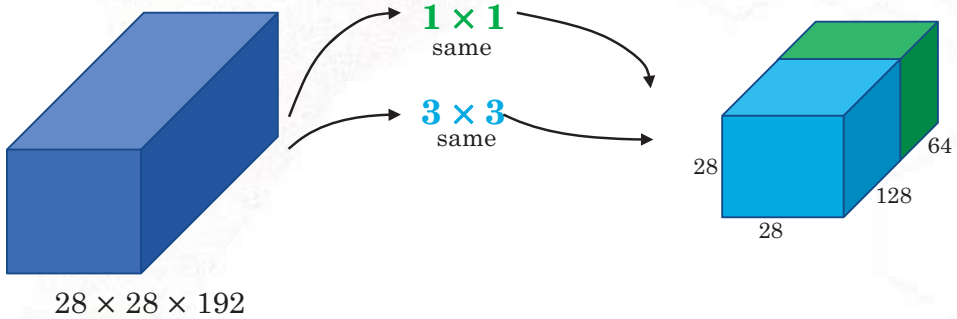
[Szegedy et al. 2014. Going deeper with convolutions]

Motivation for inception network



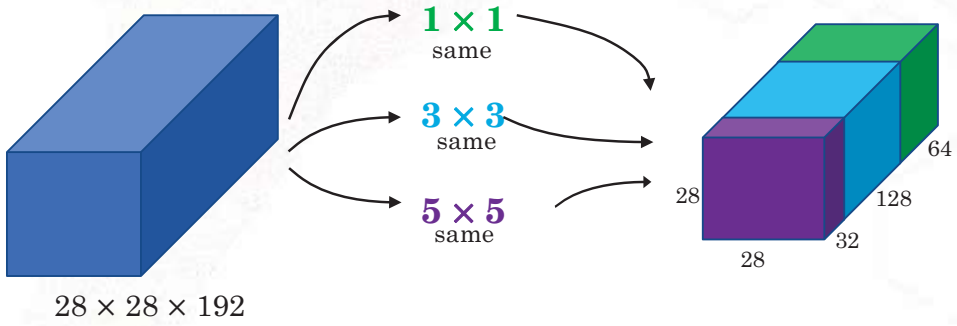
[Szegedy et al. 2014. Going deeper with convolutions]

Motivation for inception network



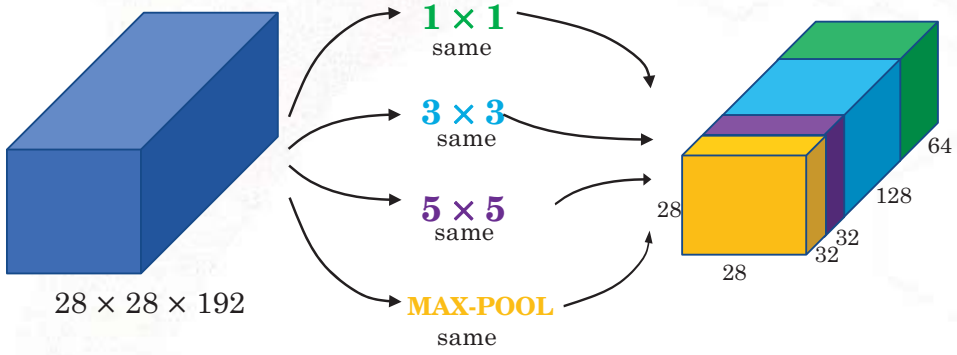
[Szegedy et al. 2014. Going deeper with convolutions]

Motivation for inception network



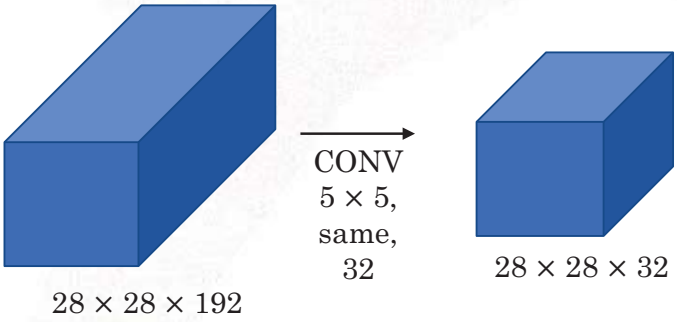
[Szegedy et al. 2014. Going deeper with convolutions]

Motivation for inception network



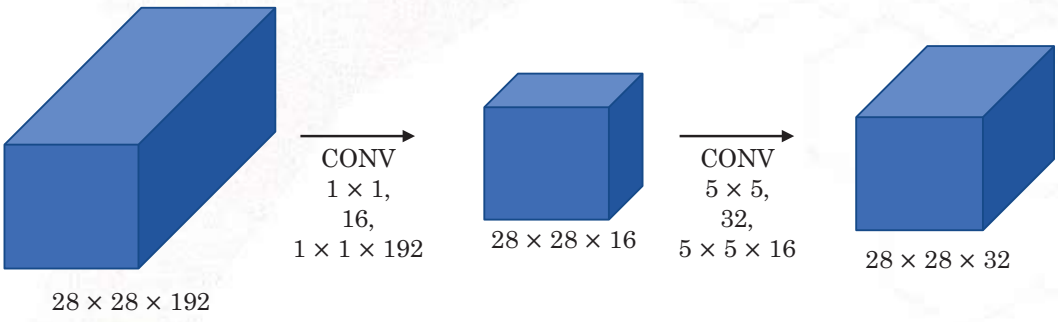
[Szegedy et al. 2014. Going deeper with convolutions]

The problem of computational cost



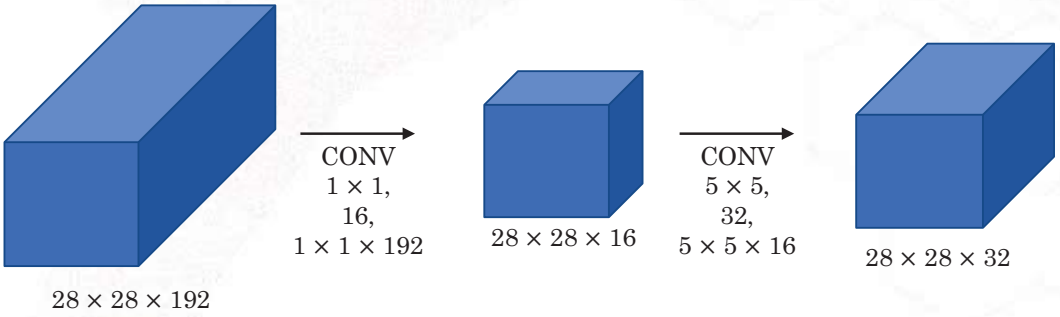
filter size: 5 x 5 x 192
32 filters
Computational cost = 28 x 28 x 32 x 5 x 5 x 192 = 120M

Using 1×1 convolution



filter size: 1 x 1 x 192
16 filters
Computational cost = 28 x 28 x 16 x 1 x 1 x 192 = 2.4M

Using 1×1 convolution

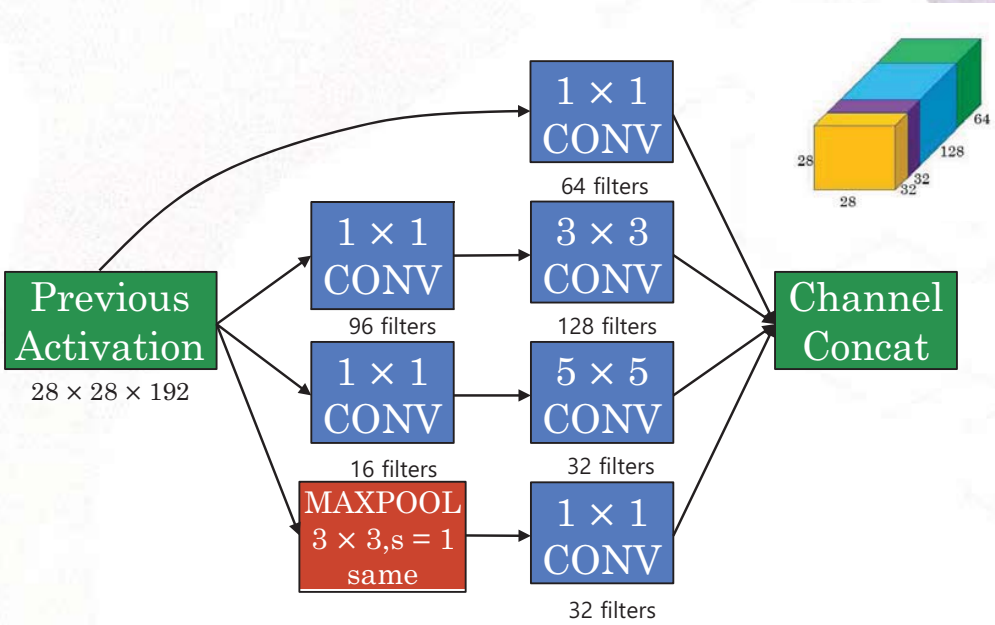


filter size: 1 x 1 x 192
16 filters
Computational cost = 28 x 28 x 16 x 1 x 1 x 192 = 2.4M

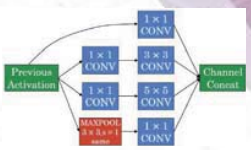
12.4M

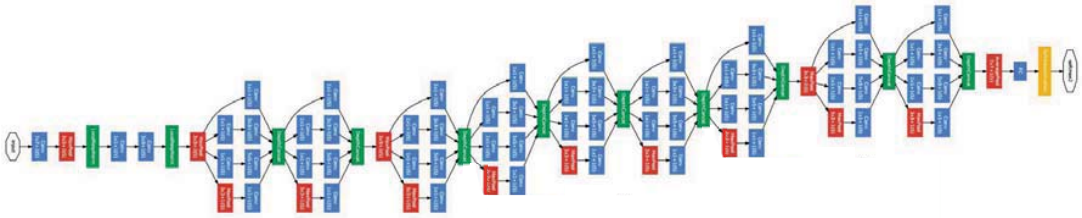
filter size: 5 x 5 x 16
32 filters
Computational cost = 28 x 28 x 32 x 5 x 5 x 16 = 10.0M

Inception module



Inception network





[Szegedy et al., 2014, Going Deeper with Convolutions]

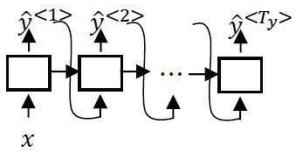
Andrew Ng

Transformer

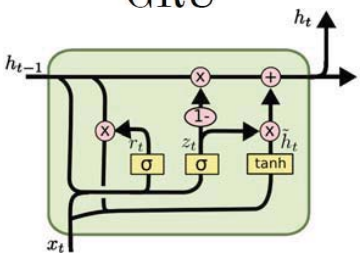
Transformers Motivation

Increased complexity, sequential

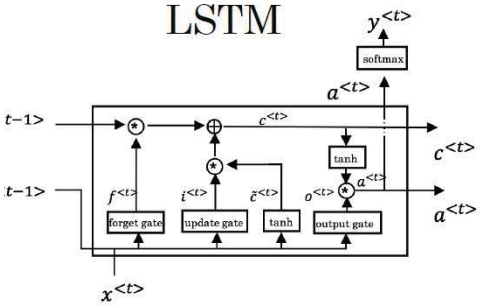
RNN



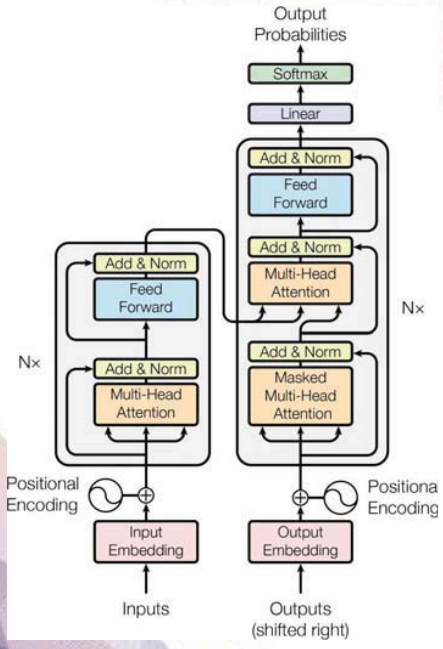
GRU



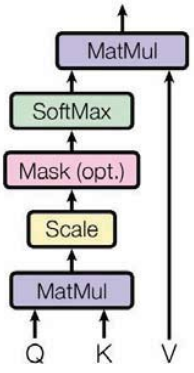
LSTM



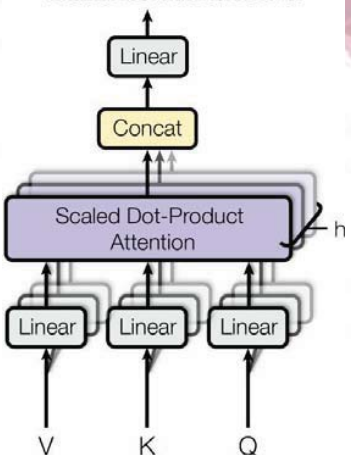
Transformers Intuition



Scaled Dot-Product Attention



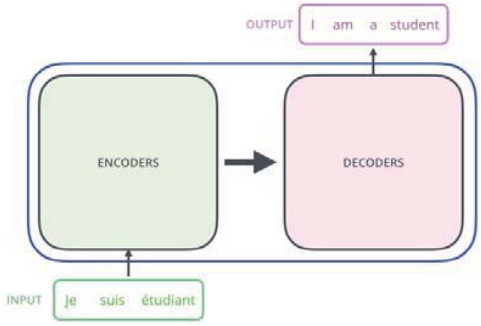
Multi-Head Attention



Transformers Intuition

- Transformer (Vaswani et al., 2017)
  - A model that uses attention to boost the speed with which these models can be trained and easy to parallelize
  - A high level look:

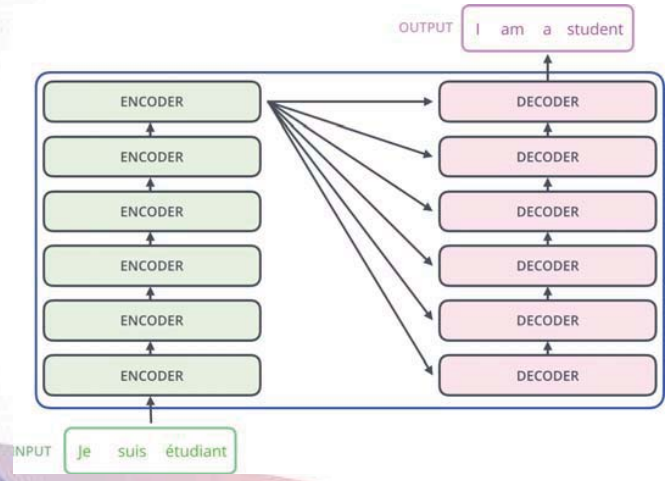
  - Inside the transformer, there are an encoding component and a decoding components and connections between them



Transformers Intuition

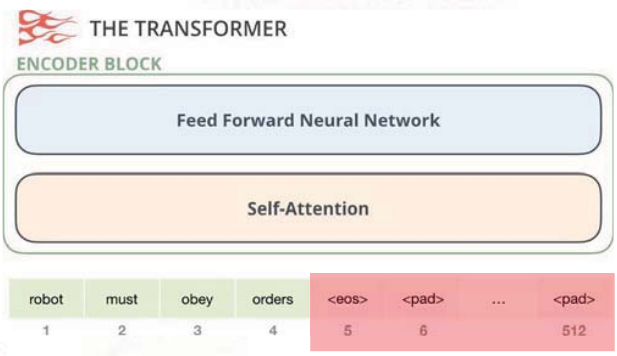
- The encoding component is a stack of encoders
- The decoding component is a stack of decoders of the same number

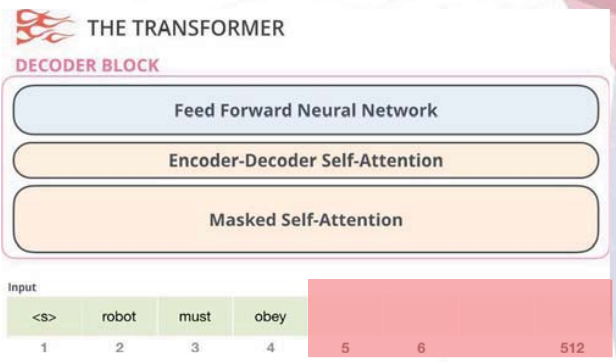
Note: The original paper stacks six of them on top of each other, but there is nothing magical about the number six



Transformers Intuition

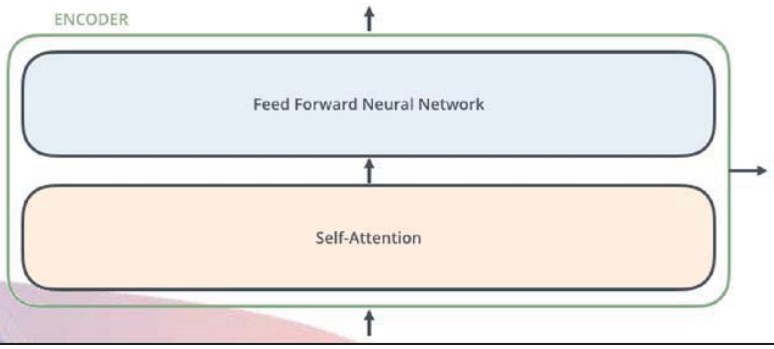
- Encoding block vs. Decoding block = Unmasked vs. Masked





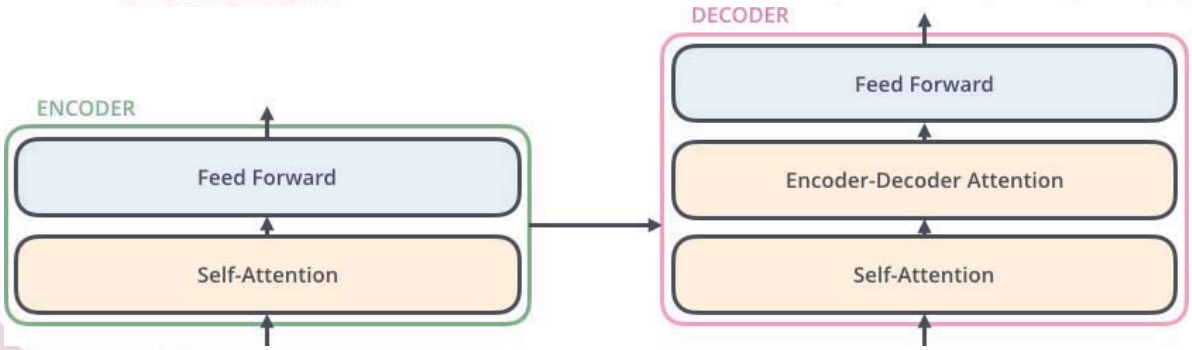
Transformers Intuition

- **The encoder** are all identical in structure (they ***do not*** share the weights), each of which is broken down into two sub-layers
  - **The encoder's input 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 output of the self-attention layer are fed to a feed-forward neural network
  - The exact same feed-forward network is independently applied to each position



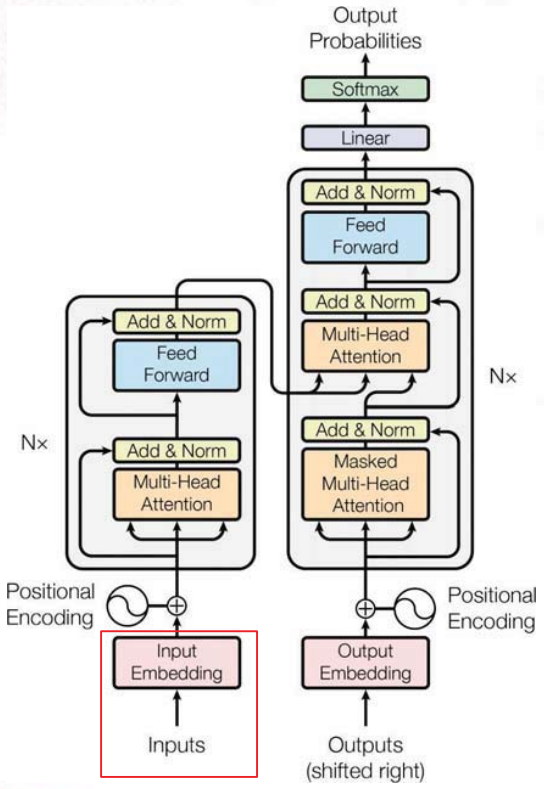
Transformers Intuition

- **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



Transformer Details: Input Embeddings

- **1. Input Embeddings**



Transformer Details: Input Embeddings

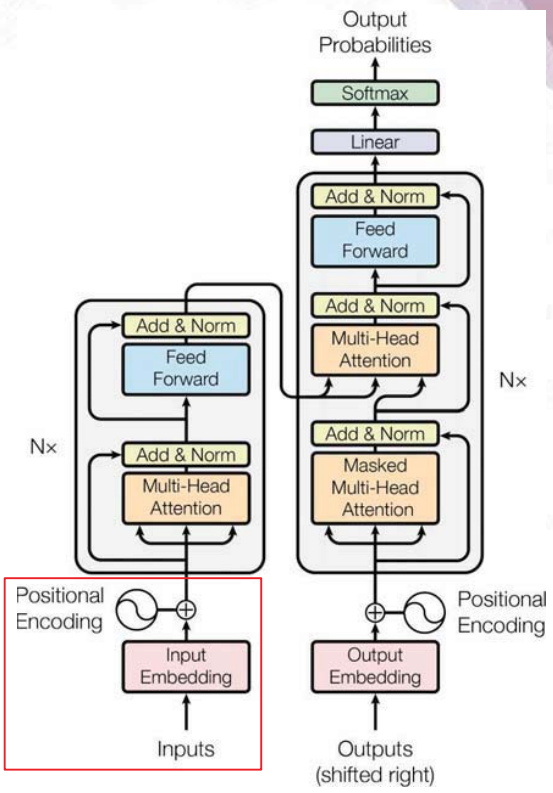
- Let's begin by turning **each input word into a vector** using an embedding algorithm

Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

  - The embedding only happens in the bottom-most encoder
  - The abstraction that is common to all the encoders is that they rec eive a list of vectors each of the size 512
  - In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that is directly below
  - The size of this list is a hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset

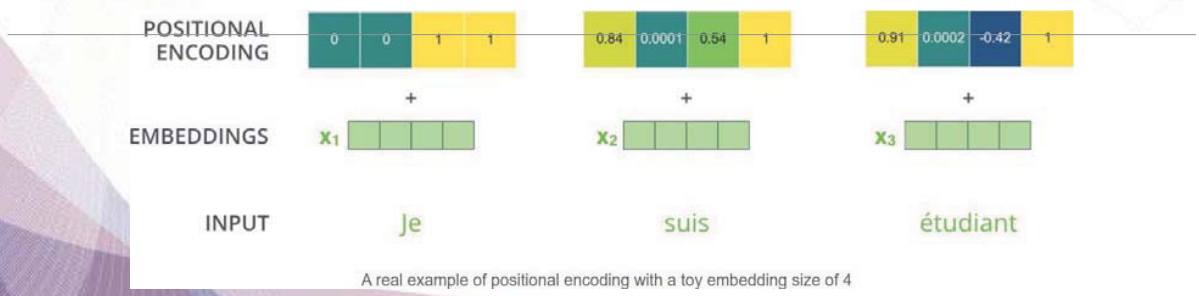
Transformer Details: Positional Encoding

- **2. Positional Encoding**



Transformer Details: Positional Encoding

- Positional encoding
  - A way to account for the order of the words in the input sequence
  - A vector added to each input embedding
  - Provides meaningful distances between the embedding vectors once they are projected into Q/K/V vectors and during dot-product attention



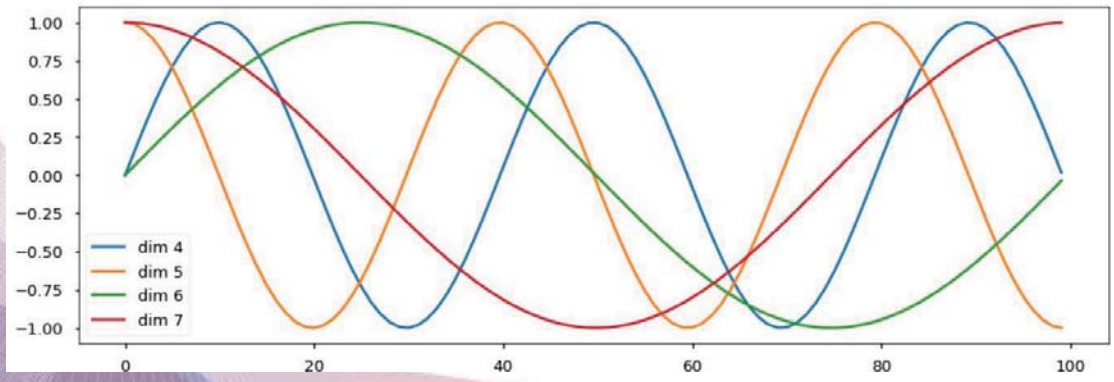
A real example of positional encoding with a toy embedding size of 4

Transformer Details: Positional Encoding

- Two properties that a good positional encoding scheme should have
  - The norm of encoding vector is the same for all positions
  - The further the two positions, the larger the distance

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

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



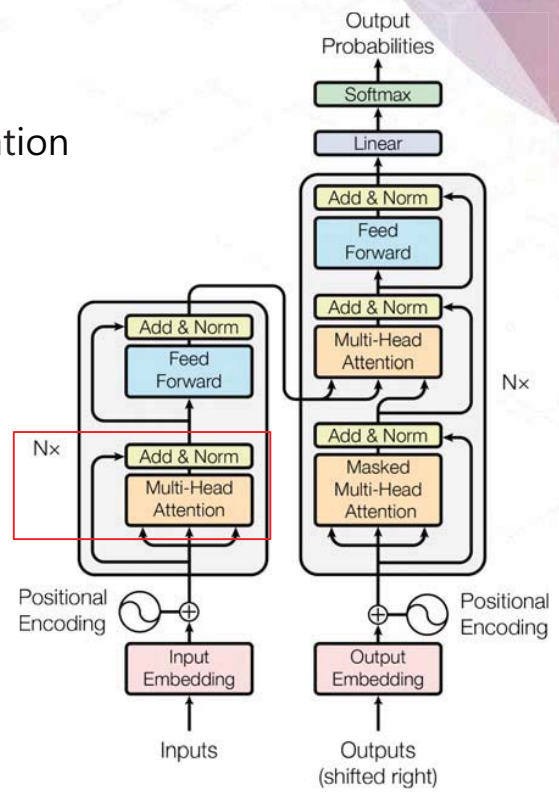
Transformer Details: Positional Encoding

- A Simple Example (n = 10, dim = 10)
  - Distances between two positional encoding vectors

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

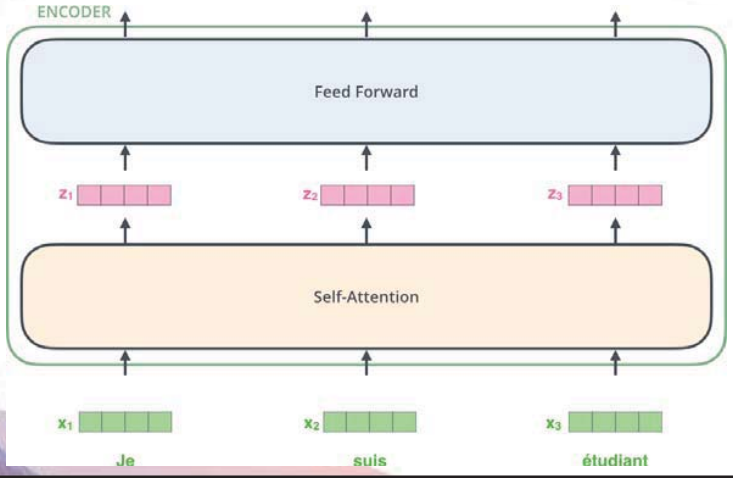
Transformer Details: Self-Attention

- Multi-Head Attention
- Residual connection & Normalization



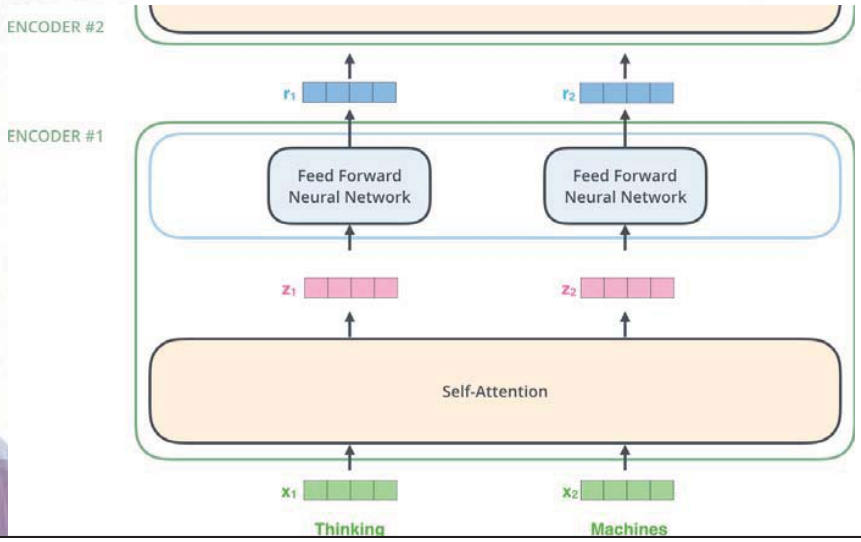
Transformer Details: Self-Attention

- After embedding the words, each of them flows through each of the two layers of the encoder
  - Word in each position flows through its own path in the encoder
    - There are dependencies between these paths in the self-attention layer
    - The feed-forward layer does not have those dependencies (parallelization becomes possible)



Transformer Details: Self-Attention

- Encoding procedure
  - An encoder receives a list of vectors as input
  - It processes this list by passing these vectors into a 'self-attention' layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder



Transformer Details: Self-Attention

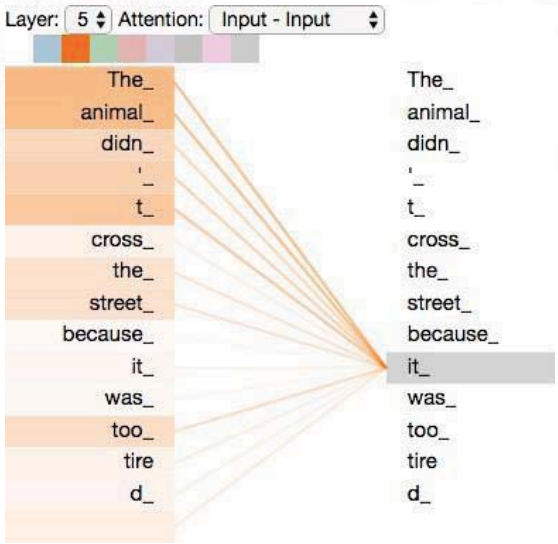
- Self-Attention at a High Level
  - Input sentence to translate:

    The animal didn't cross the street because **it** was too tired

  - What does "**it**" refer to? street or animal?
  - Simple question to a human but not as simple to an algorithm
  - Self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word
  - Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing

Transformer Details: Self-Attention

- Self-Attention example

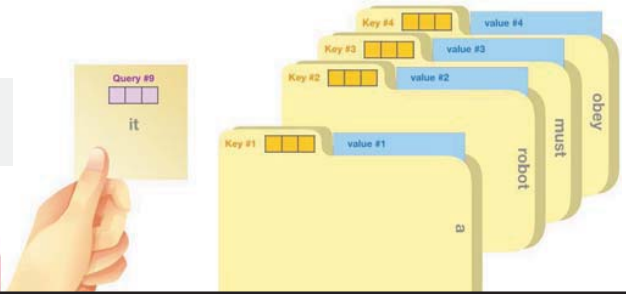


https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

Transformer Details: Self-Attention

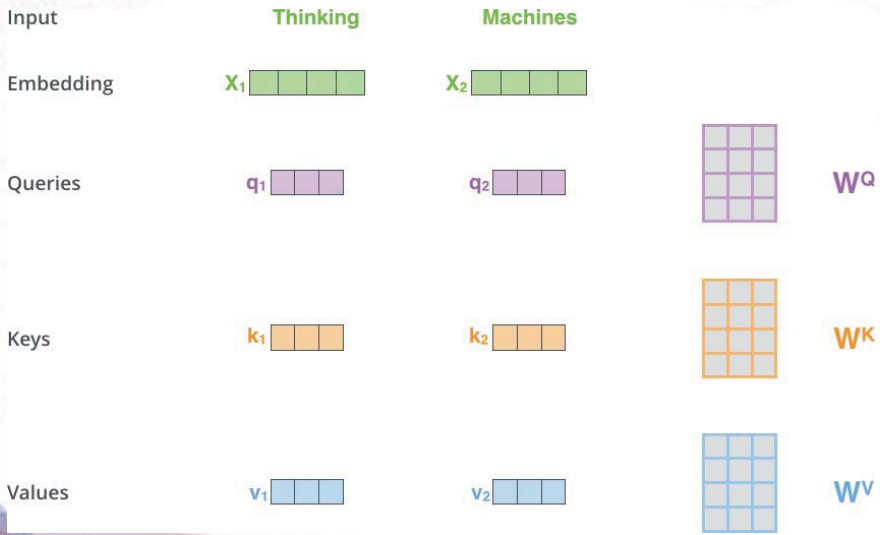
- **Step 1: Create three vectors from each of the encoder's input vectors**
  - **Query**: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
  - **Key**: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
  - **Value**: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

"A robot must obey the orders given it by human beings"



Transformer Details: Self-Attention

- **Step 1: Create three vectors from each of the encoder's input vectors**
  - These vectors (Q/K/V) are created by multiplying the embedding by three matrices (W^Q, W^K, W^V) that we trained during the training process

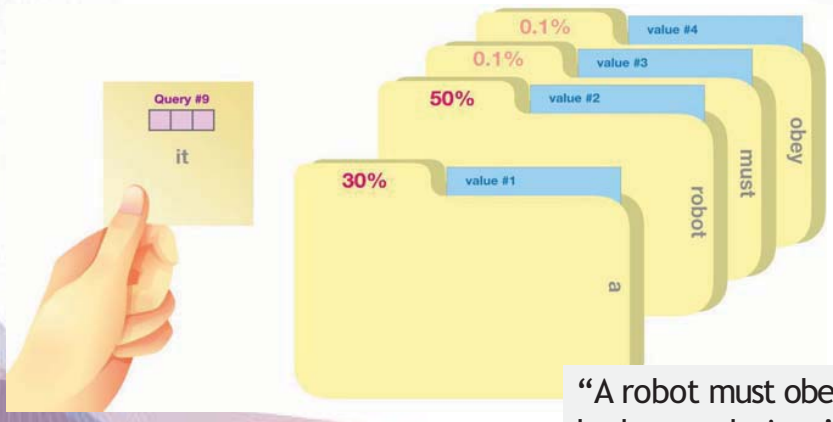


Transformer Details: Self-Attention

- **Step 1: Create three vectors from each of the encoder's input vectors**
  - Note) These new vectors are smaller in dimension that the embedding vector
  - Q, K, and V are 64-dim. while embedding and encoder input/output vectors are 512-dim.
  - They do not have to be smaller but it is an architecture choice to make the computation of multi-headed attention (mostly) constant

Transformer Details: Self-Attention

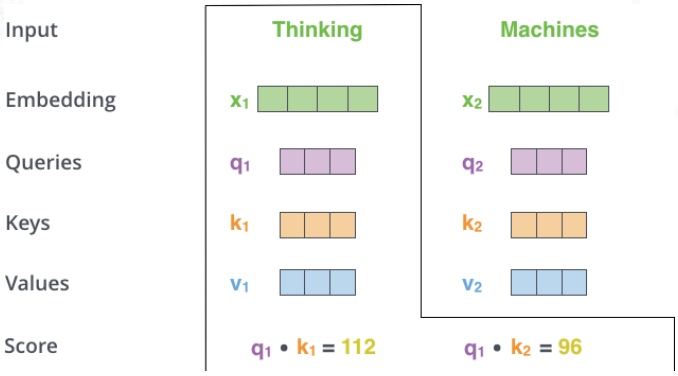
- **Step 2: Calculate self-attention to get a score**
  - We need to score each word of the input sentence against this word ("it")
  - The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position
  - Multiplying the query vector by each key vector produces a score for each folder
  - (technically: dot product followed by softmax)



"A robot must obey the orders given **it** by human beings"

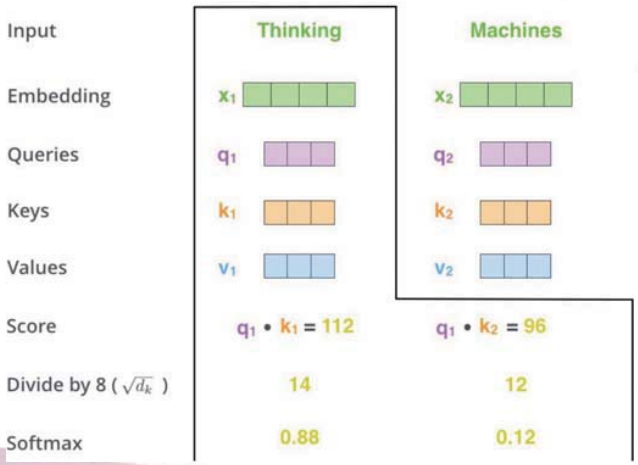
Transformer Details: Self-Attention

- **Step 2: Calculate self-attention to get a score**
  - The score is calculated by taking the dot product of the query vector with the key vector of the respective word we are scoring



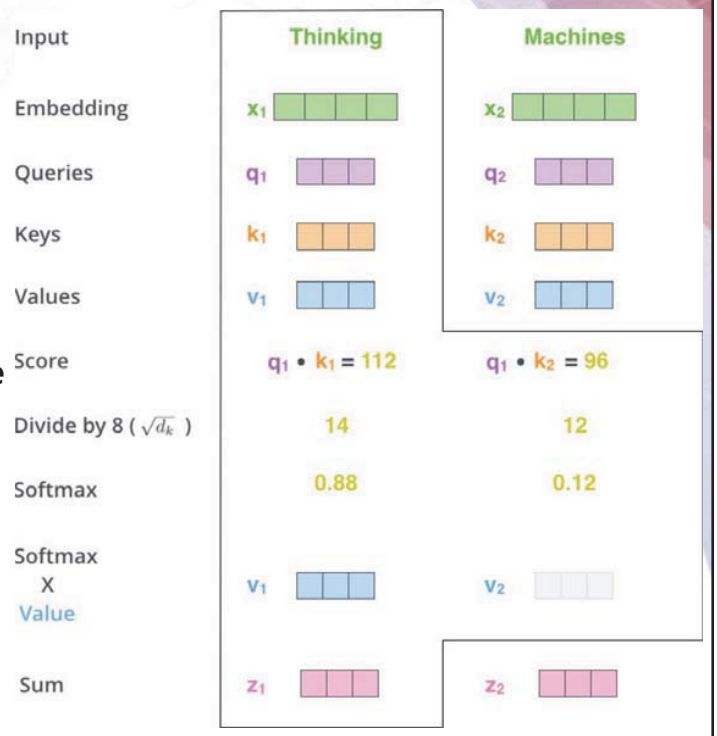
Transformer Details: Self-Attention

- **Step 3: Divide the score by $\sqrt{d_k}$ (= 8 in the original paper since d_k = 64)**
  - This leads to having more stable gradients
- **Step 4: Pass the result through a softmax operation**
  - The softmax score determines how much each word will be expressed at this position



Transformer Details: Self-Attention

- **Step 5: Multiply each value vector by the softmax score**
  - to keep intact the values of the words we want to focus on
  - drown-out irrelevant words
- **Step 6: Sum up the weighted value vector which produces the output of the self-attention layer at this position**



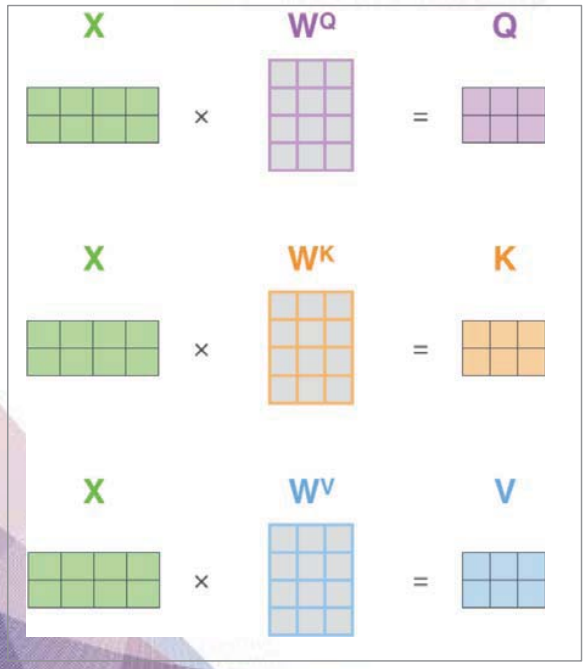
Transformer Details: Self-Attention

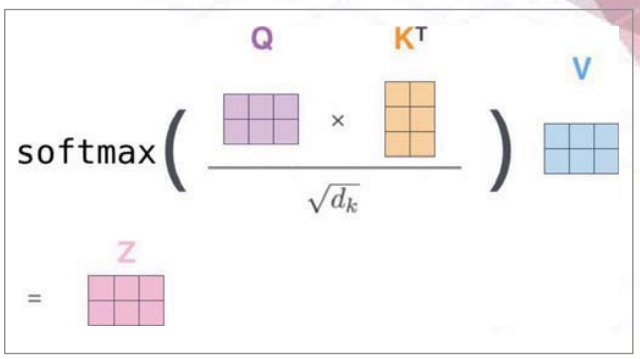
- **Step 5: Multiply each value vector by the softmax score**
- **Step 6: Sum up the weighted value vector which produces the output of the self- attention layer at this position**

Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

Transformer Details: Self-Attention

- Matrix calculation of self-attention

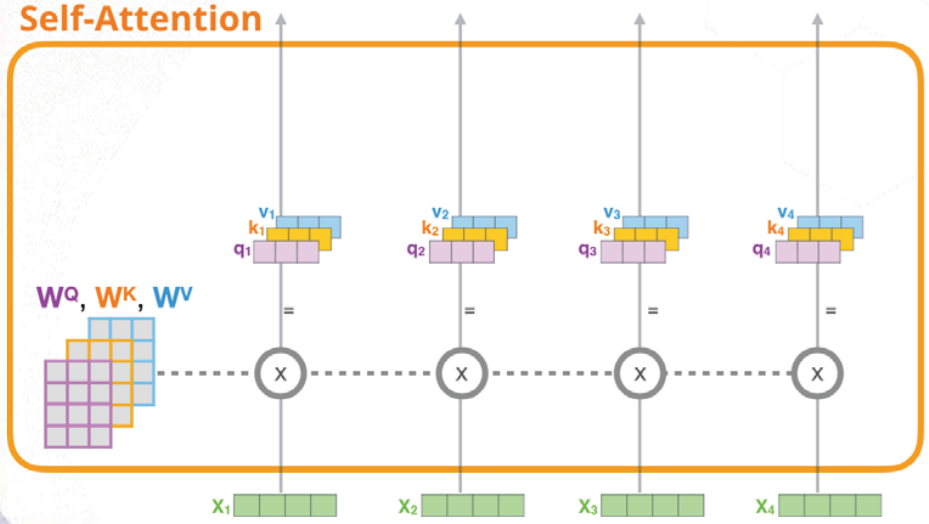




Transformer Details: Self-Attention

- Another illustration of Self-Attention

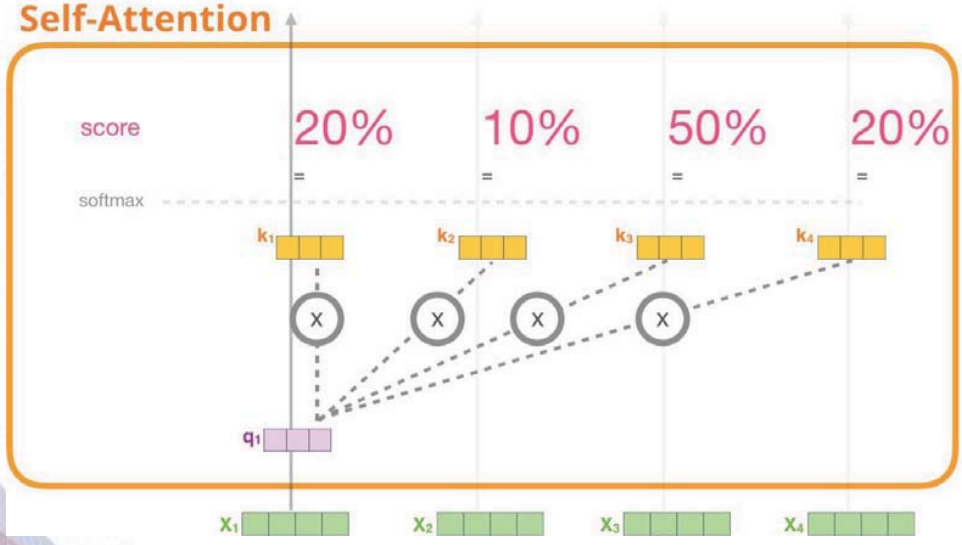
1) For each input token, create a query vector, a key vector, and a value vector by multiplying by weight Matrices W^Q, W^K, W^V



Transformer Details: Self-Attention

- Another illustration of Self-Attention

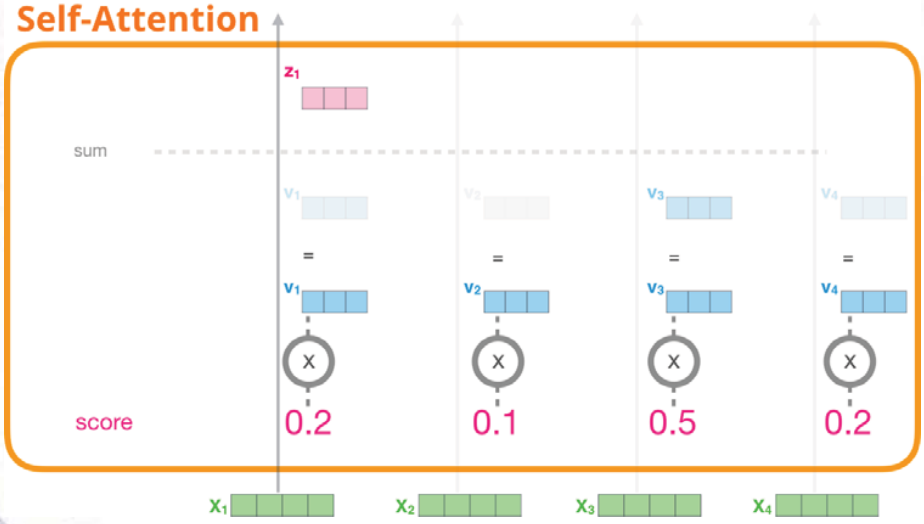
2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match



Transformer Details: Self-Attention

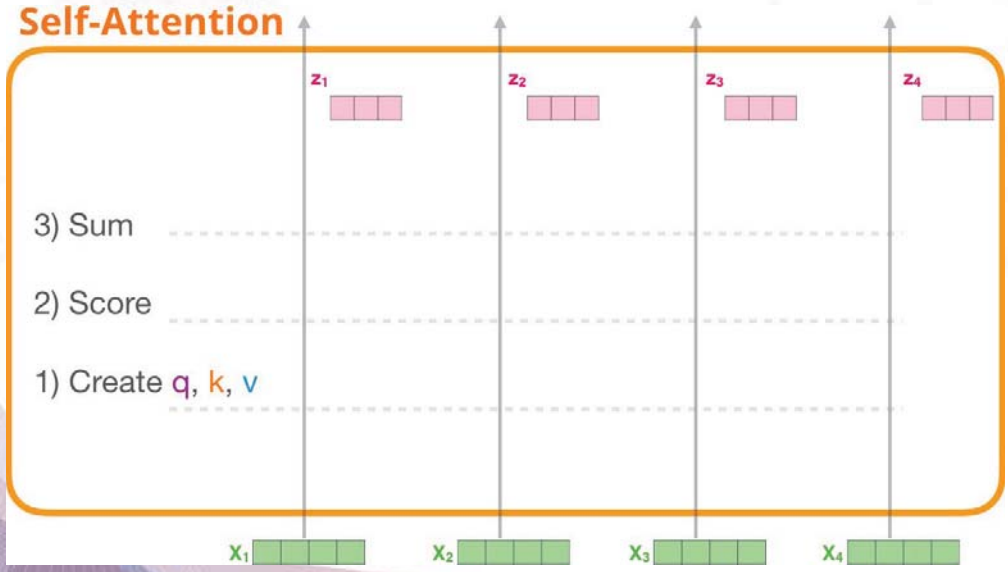
- Another illustration of Self-Attention

3) Multiply the value vectors by the scores, then sum up



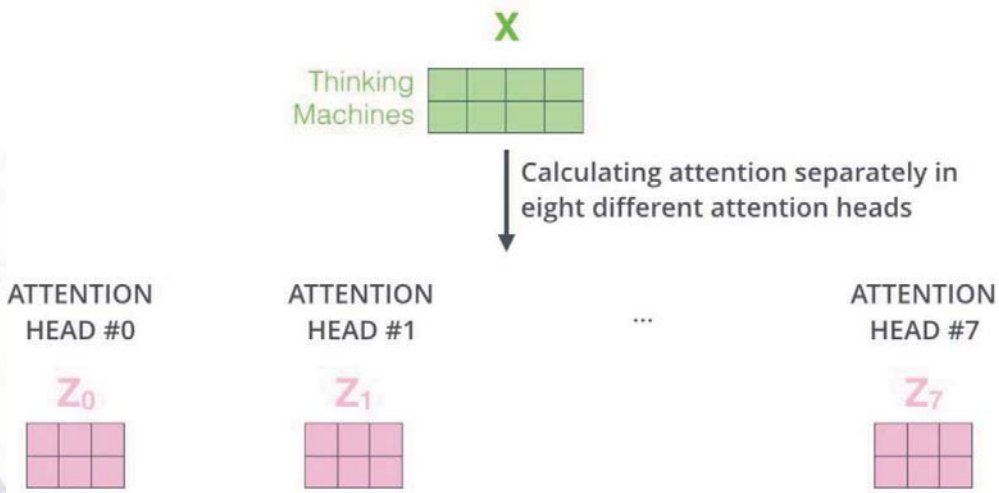
Transformer Details: Self-Attention

- Another illustration of Self-Attention
  - Do the same operation for each path to end up with a vector representing each token containing appropriate context of that token



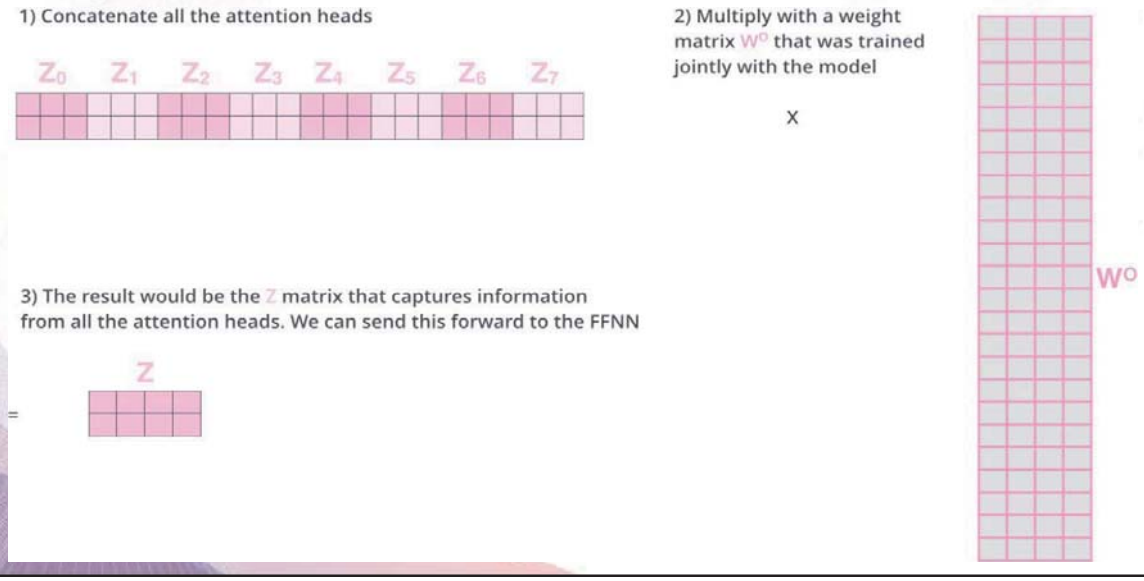
Transformer Details: Multi-headed-Attention

- Multi-headed attention
  - Expand the model's ability to focus on different positions



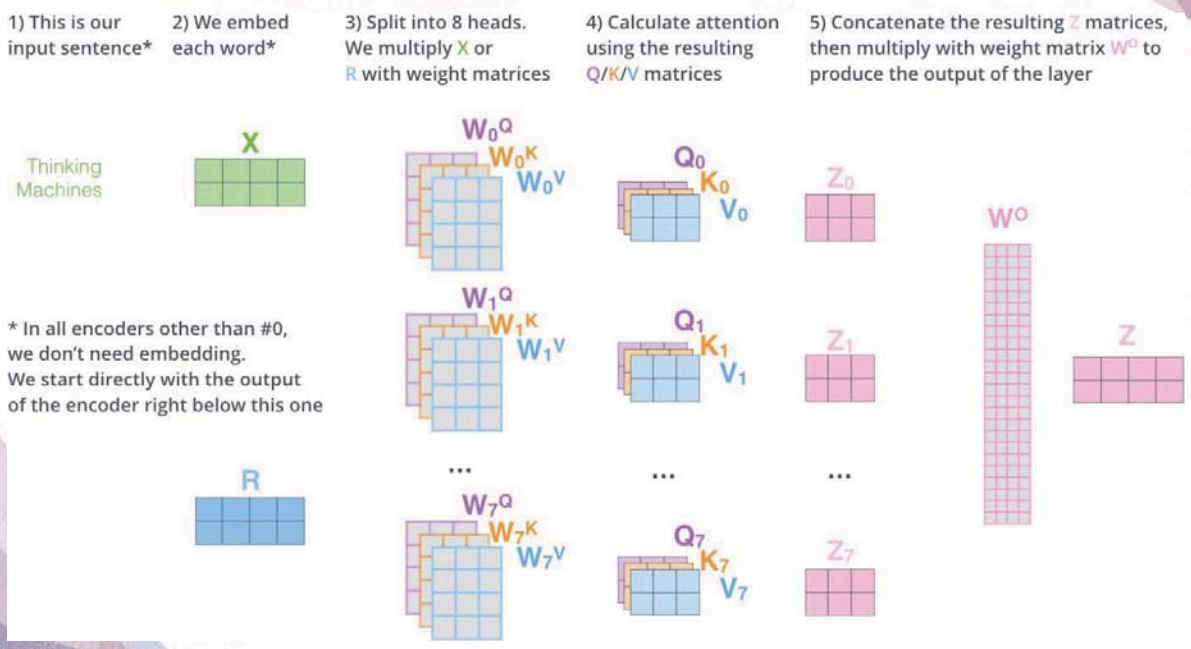
Transformer Details: Multi-headed-Attention

- Multi-headed attention
  - Attention heads are concatenated and multiplied by an additional weight matrix to be used as an input of feed-forward neural network



Transformer Details: Multi-headed-Attention

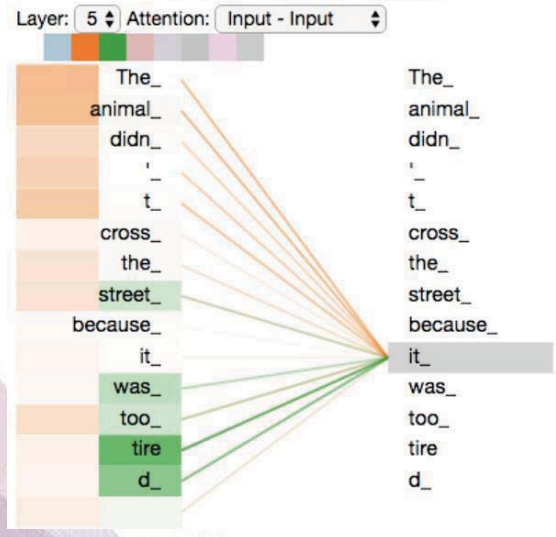
- Multi-headed attention



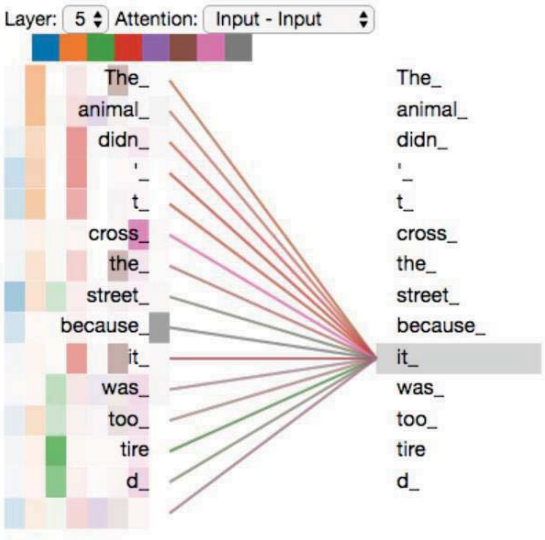
Transformer Details: Multi-headed-Attention

- Multi-headed attention

Attention with two heads

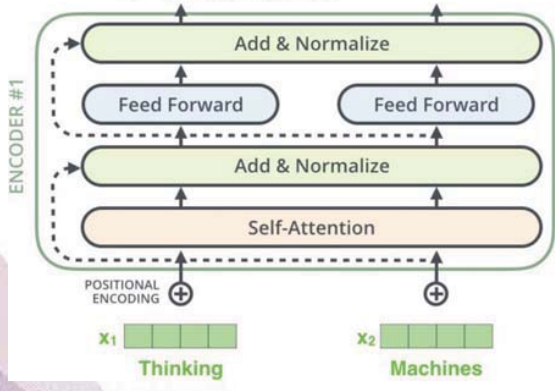


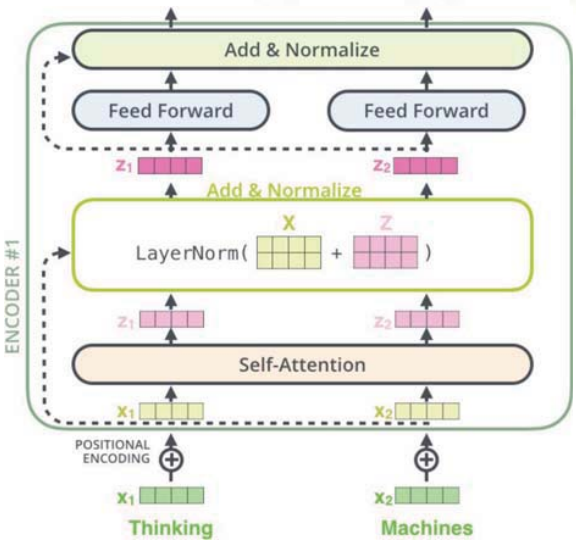
Attention with eight heads



Transformer Details: Residual Connection

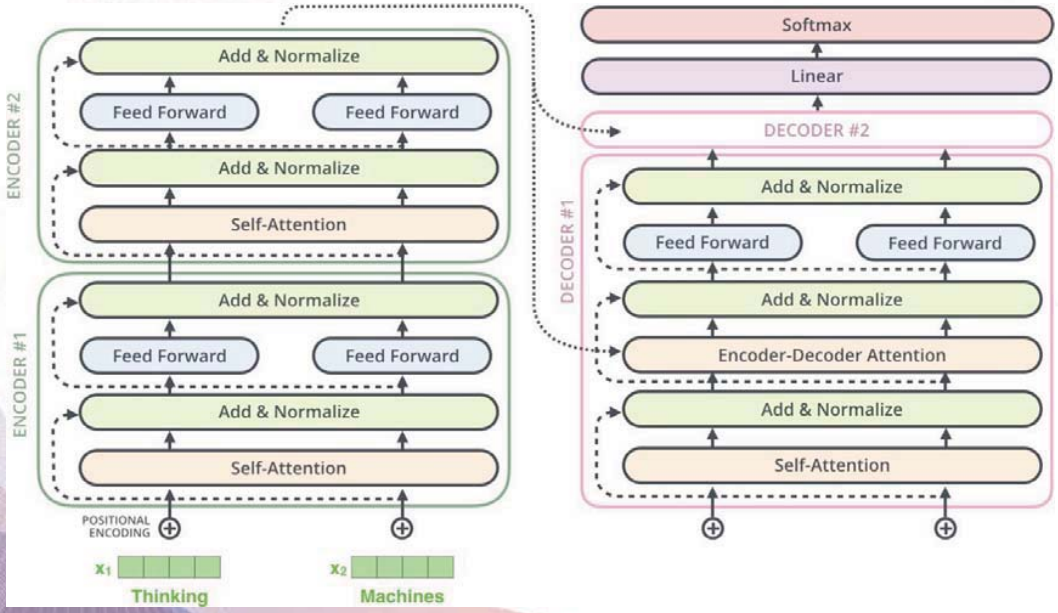
- Residual connection
  - Each sub-layer (self-attention, FFNN) in each encoder has a residual connection around it followed by a layer-normalization step





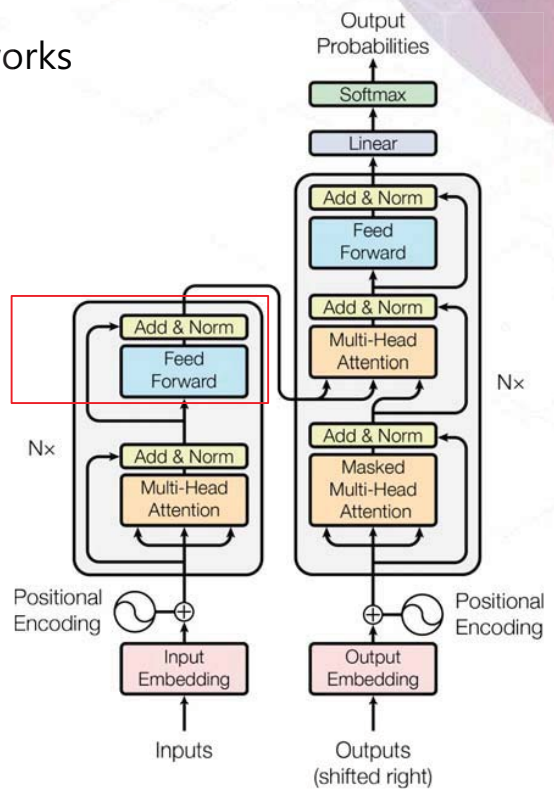
Transformer Details: Residual Connection

- Residual connection
  - This goes for the sub-layers of the decoder as well
  - Ex: 2 stacked encoders and decoders



Transformer Details: Feed Forward

- Position-wise Feed-Forward Networks

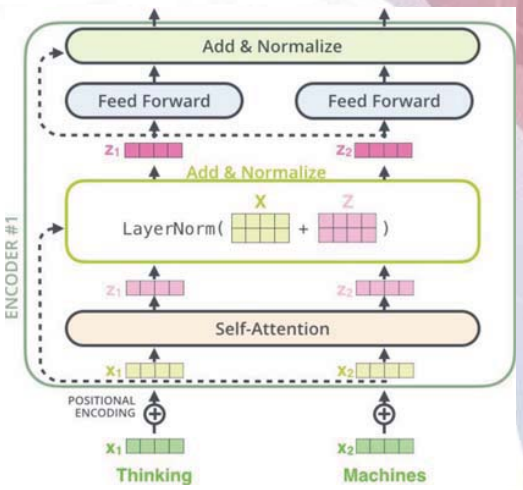


Transformer Details: Feed Forward

- Position-wise Feed-Forward Networks
  - Fully connected feed-forward network
  - Applied to each position separately and identically

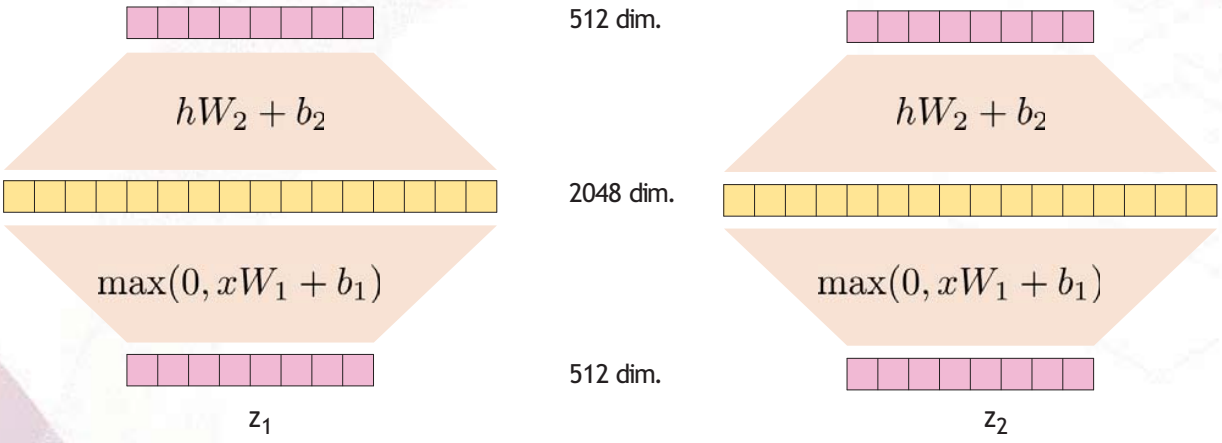
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

  - The linear transformations are the same across different positions
  - They use different parameters from layer to layer



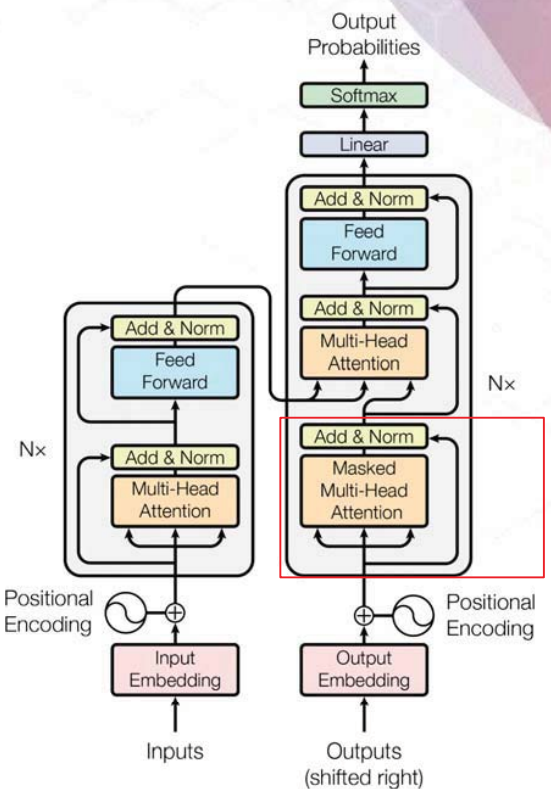
Transformer Details: Feed Forward

- Position-wise Feed-Forward Networks



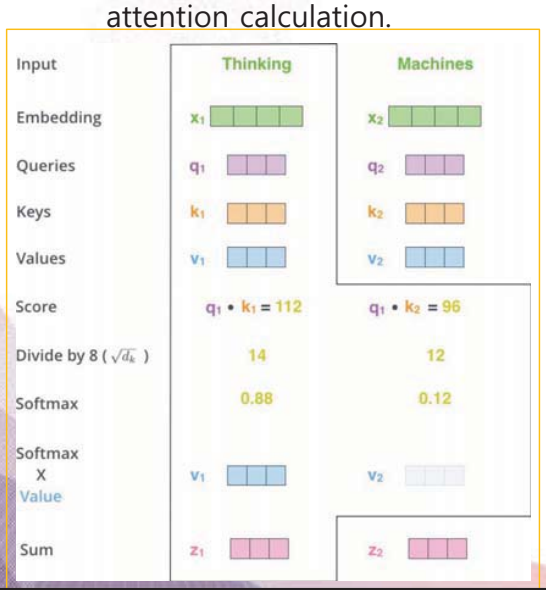
Transformer Details: Masked Multi-Head Attention

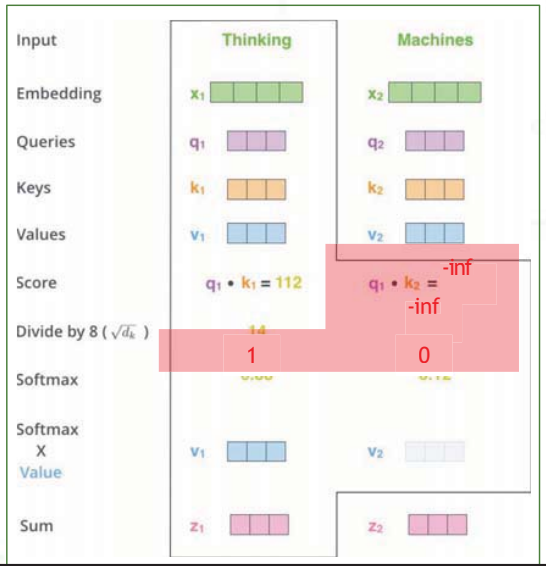
- Masked Multi-Head Attention



Transformer Details: Masked Multi-Head Attention

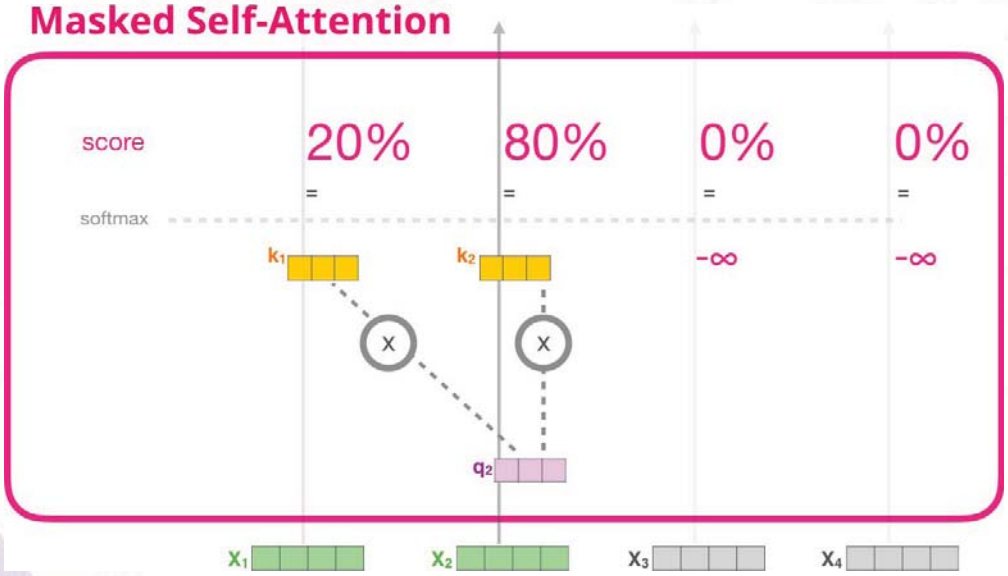
- Masked Multi-head Attention
  - Self attention layers in the decoder is only allowed to attend to earlier positions in the output sequence, which is done by masking future positions (setting them to –inf) before the softmax step in the self attention calculation.





Transformer Details: Masked Multi-Head Attention

Masked Self-Attention



Transformer Details: Masked Multi-Head Attention

- Masked Multi-head Attention

Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask →

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

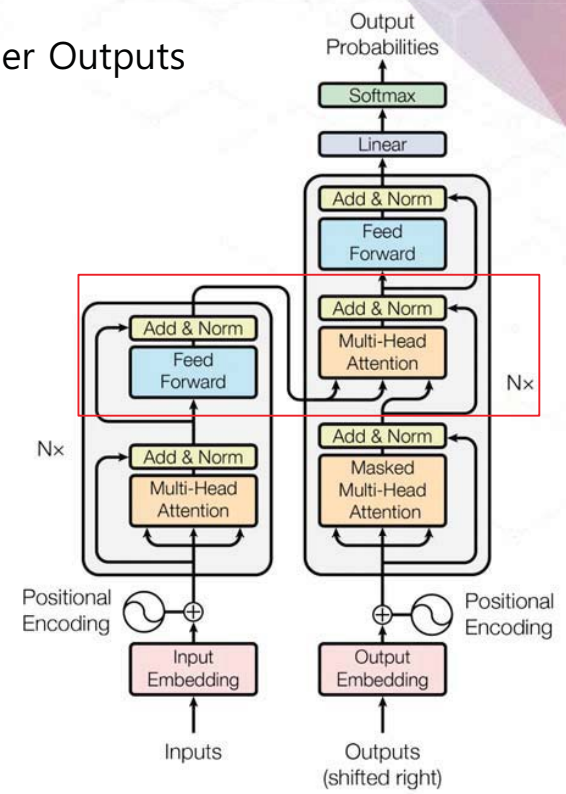
Softmax (along rows) →

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

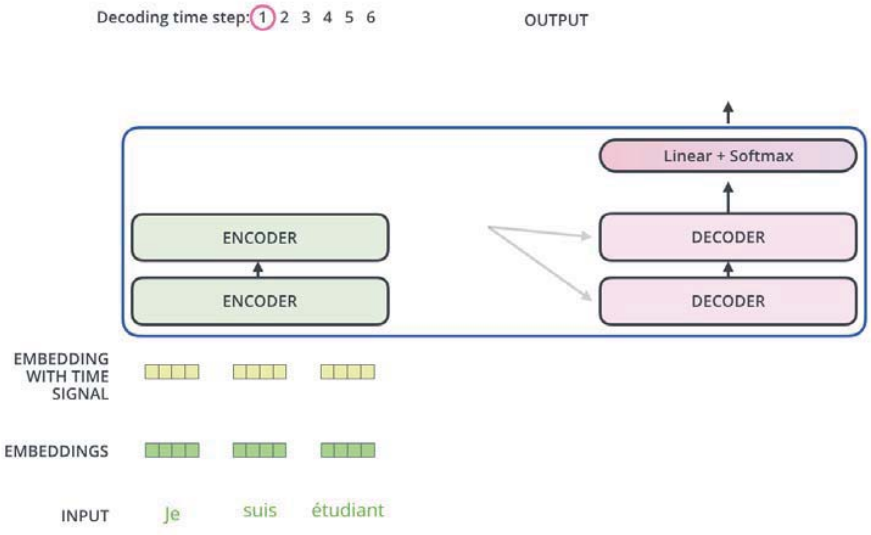
Transformer Details: Encoder-Decoder Multi-Head Attention

- Multi-Head Attention with Encoder Outputs



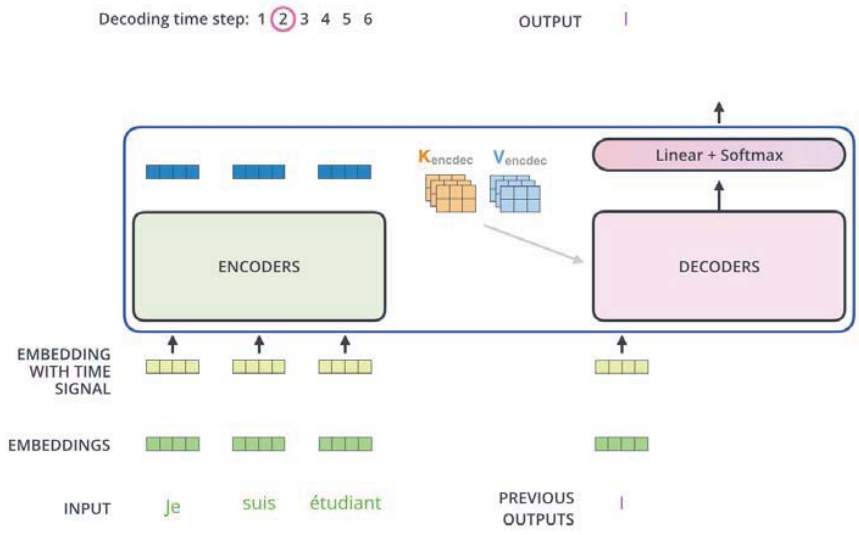
Transformer Details: Encoder-Decoder Multi-Head Attention

- The Decoder Side



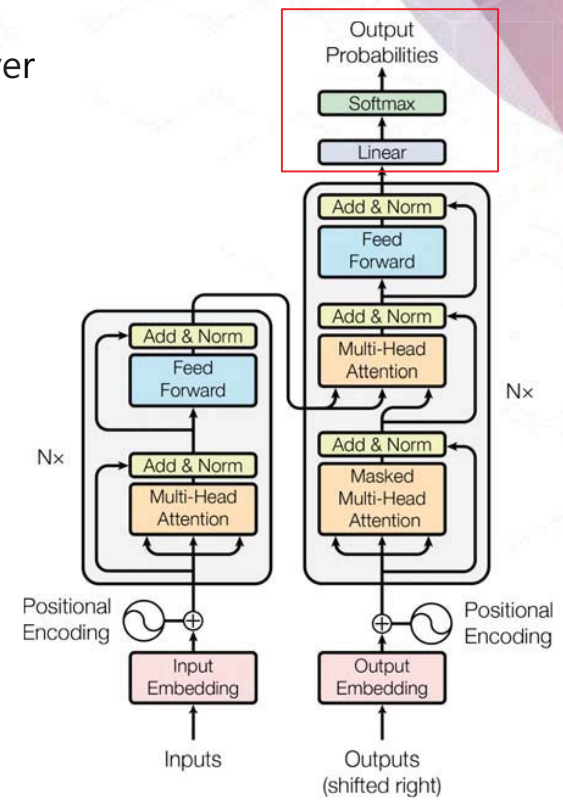
Transformer Details: Encoder-Decoder Multi-Head Attention

- The Decoder Side



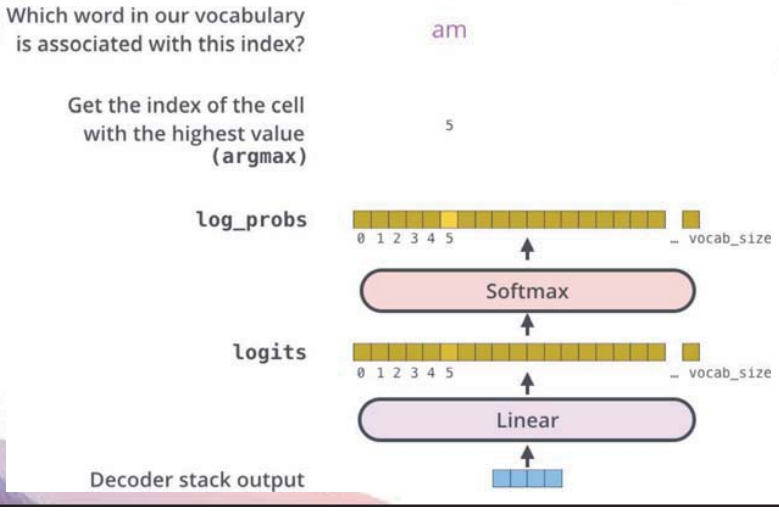
Transformer Details: Final Layer

- The Final Linear and Softmax Layer



Transformer Details: Final Layer

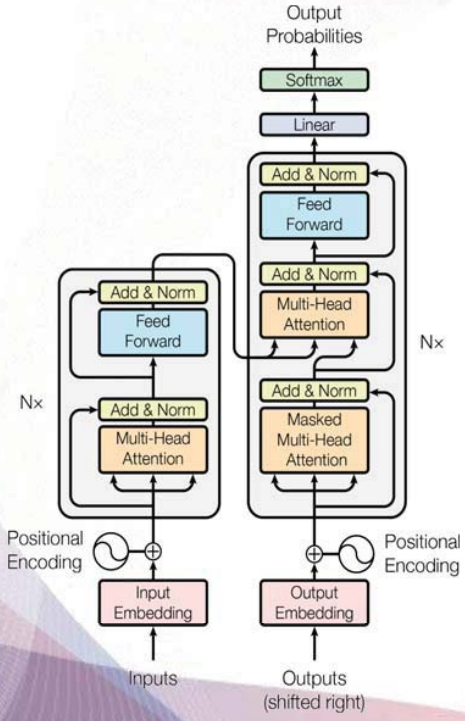
- The Final Linear and Softmax Layer
  - Linear layer: a simple fully connected neural network that projects the vector produced by the stack of decoders into a much larger vector called a logits vector
  - Softmax layer: turns those scores into probability
  - The cell with the highest probability is chosen, the word associated with it is produced as the output of this time step

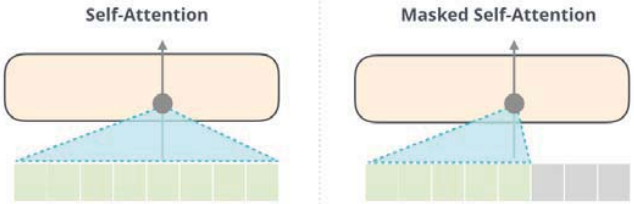


Transformer-based Pretrained Language Models

GPT: Generative Pre-Training of a Language Model

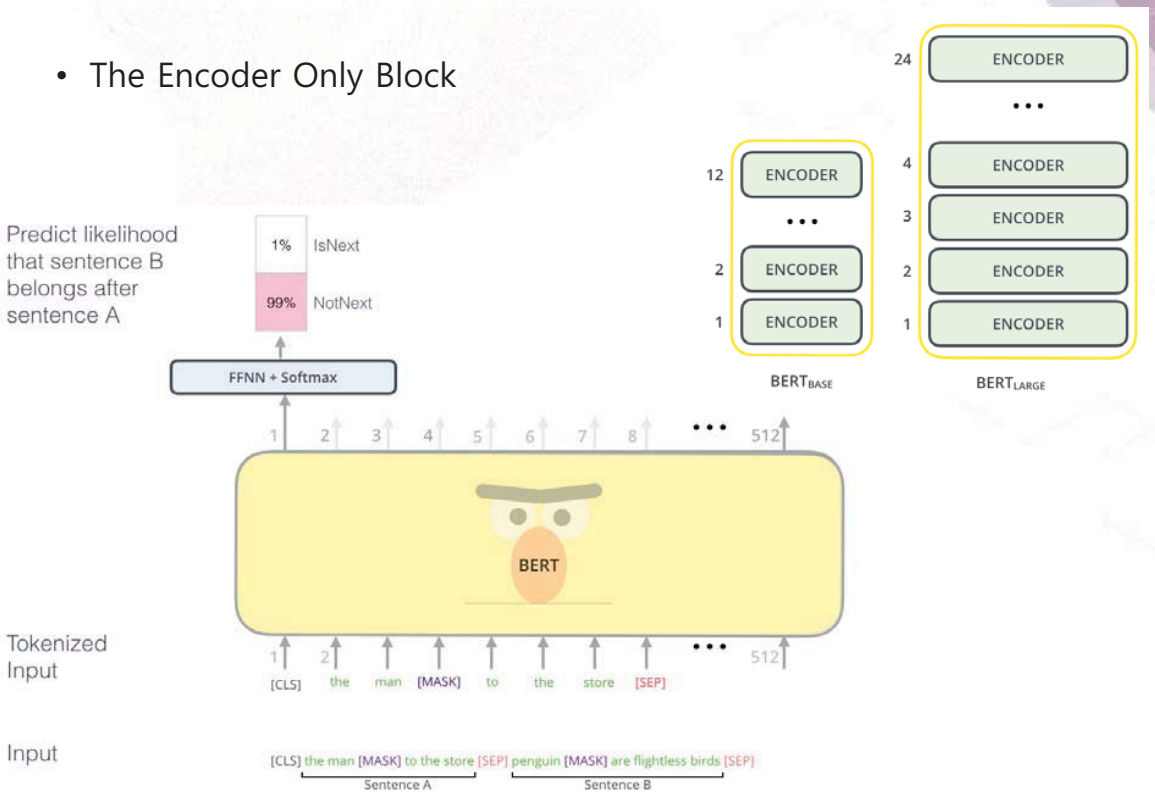
- The Decoder-Only Block



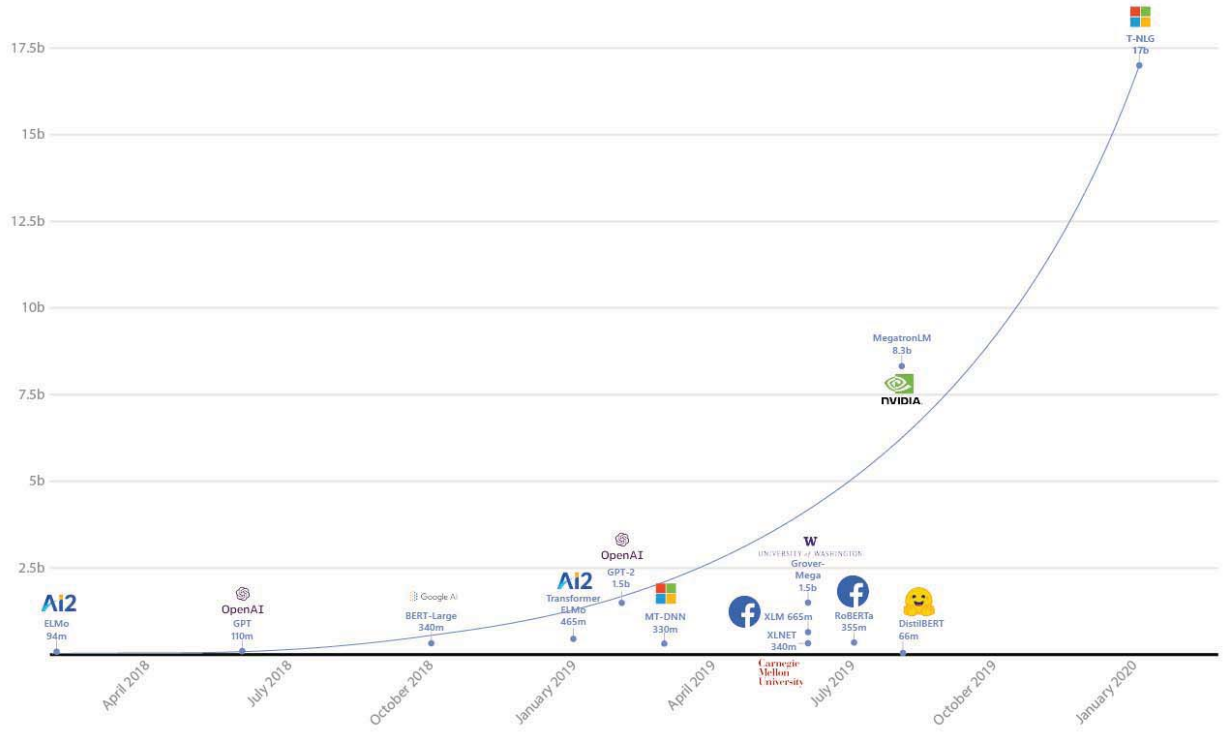


BERT: Bidirectional Encoder Representations from Transformer

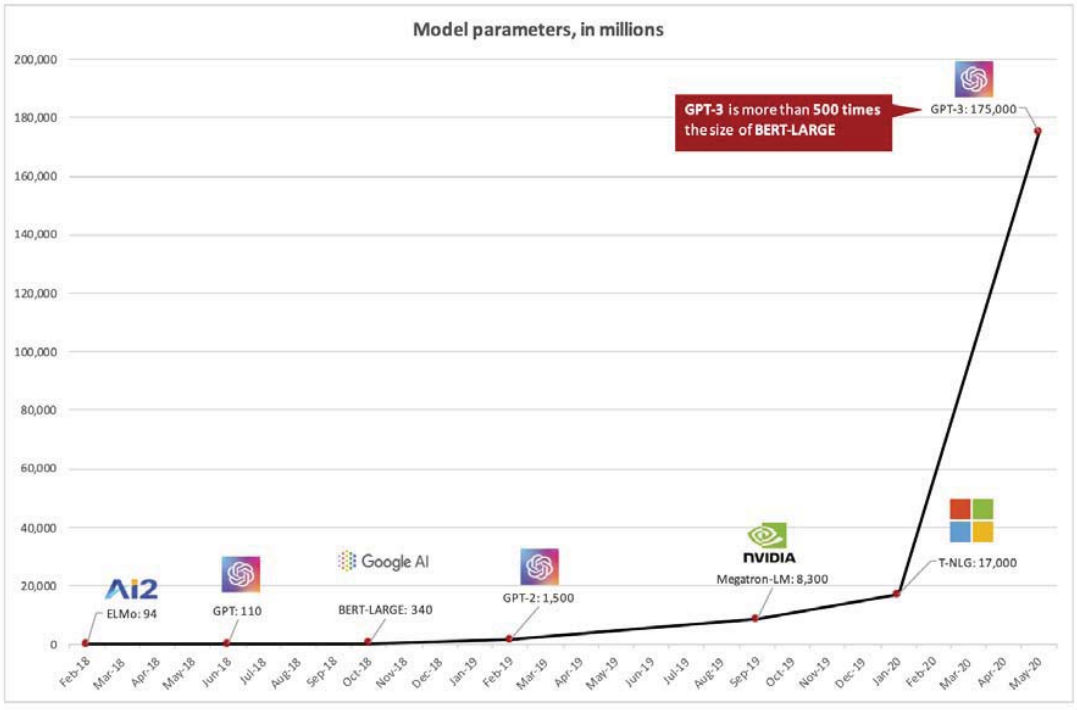
- The Encoder Only Block



Trends in Language Model Size



Trends in Language Model Size



References

- LeCun et al., 1998. Gradient-based learning applied to document recognition
- Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks
- Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition
- He et al., 2015. Deep residual networks for image recognition
- Lin et al., 2013. Network in network
- Vaswani, Ashish, et al. Attention is all you need." Advances in neural information processing systems 30 (2017).
- Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
- 파이썬으로 만드는 인공지능 (한빛미디어)
- Coursera, Convolutional neural Networks, DeepLearning.AI
- Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/