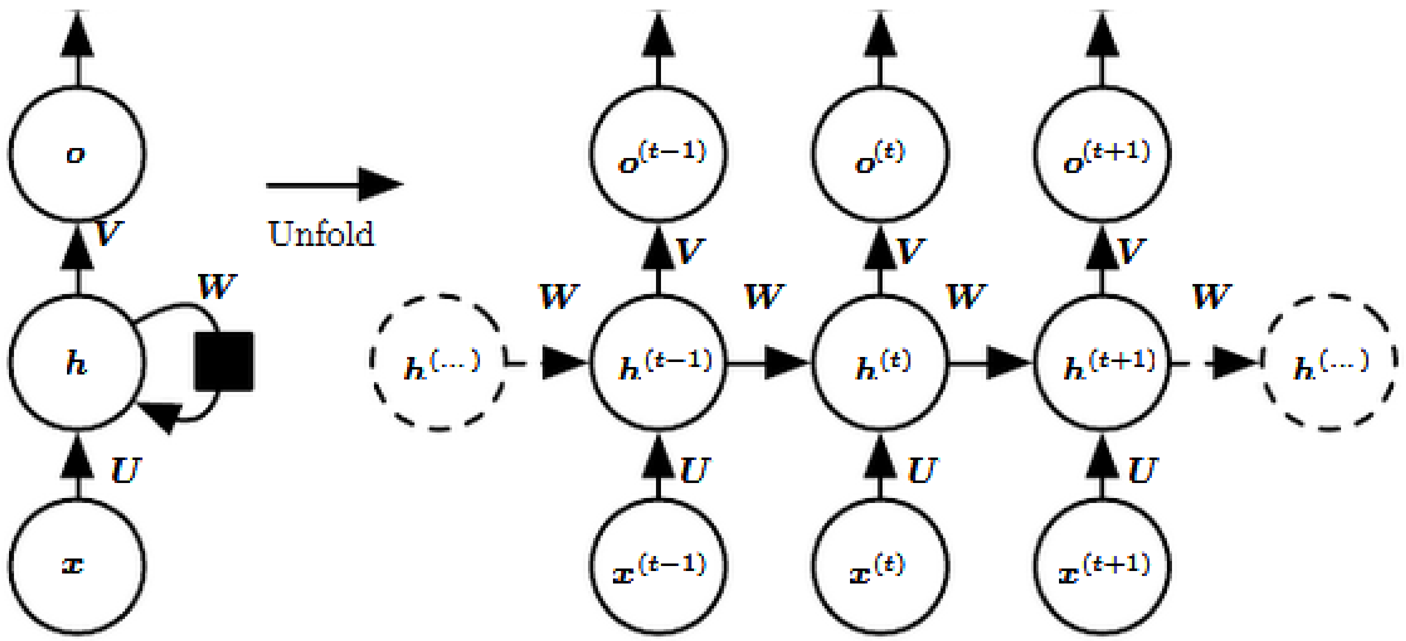
Lecture 4, Part 2
More Recent NN Models

I-Hsin Chung

Hao Yu

RNN (topology)

- "Recurrent Neural Networks (RNNs), Implementing an RNN from scratch in Python", Javaid Nabi, 20189
- https://www.deeplearningbook.org/contents/rnn.html



Input: ***x(t)*** : e.g. a word in a sentence (1-hot vector of a word corresponding to its dictionary position)

Hidden state: ***h(t)*** : represents a hidden state at time t (or position t of an input sentence/sequence). h(t) is calculated from input and the state of its predecessor: ***h(t) = f(U x(t) + W h(t−1))*** , where ***f*** : a non-linear transformation, e.g. tanh, ReLU, sigmoid, CDF.

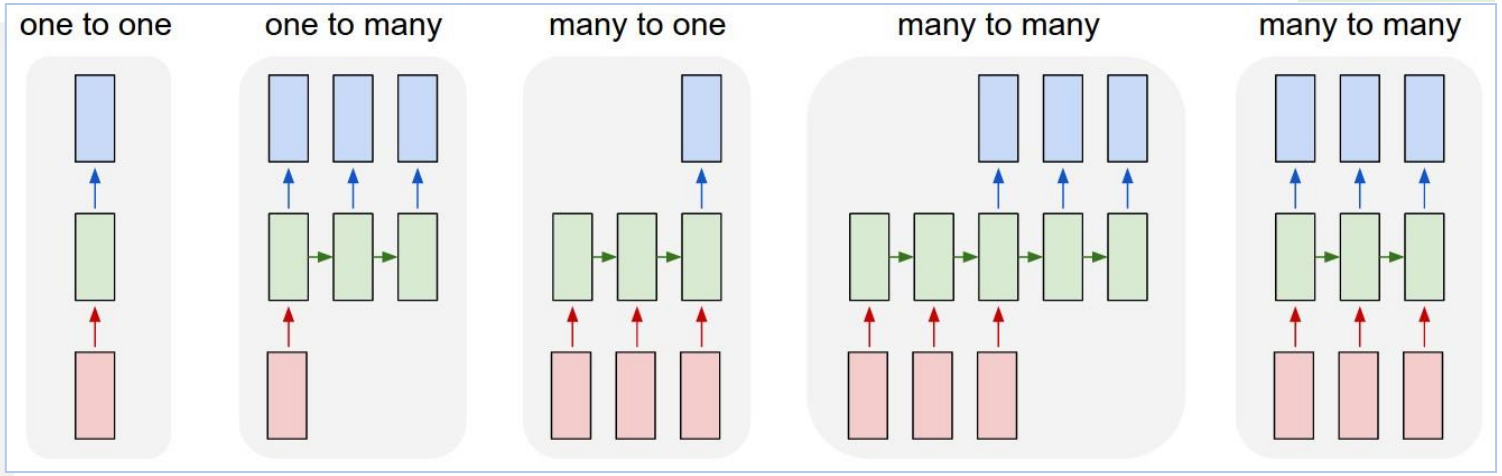
Weights: The RNN has edges parameterized by a weight matrices: ***U, V, W***. The weights from different layers are different.

Output: ***o(t)*** : think of translation, summarization.

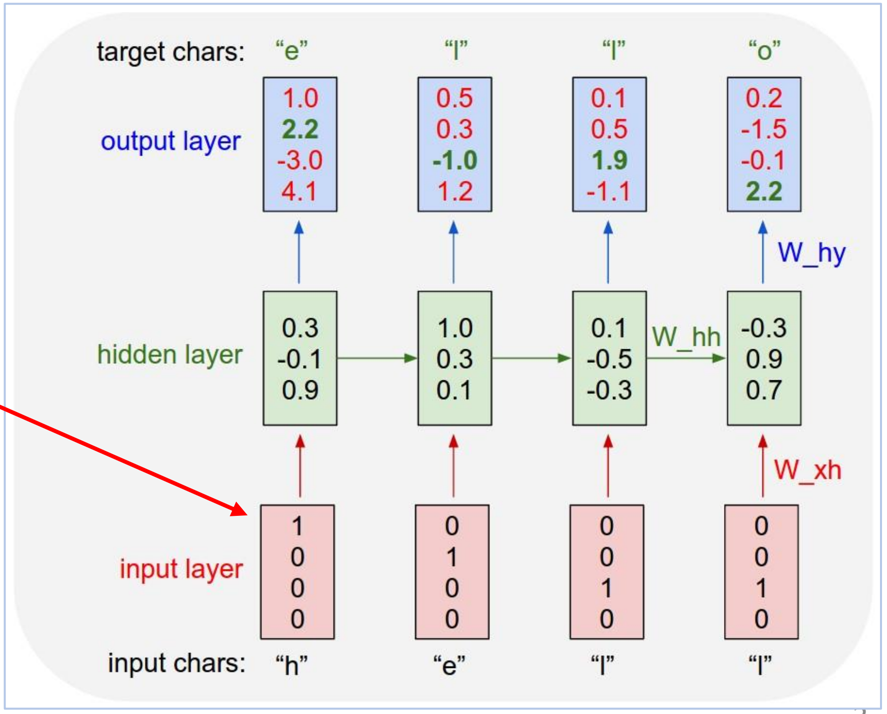
Recursive NN (use cases)

From [1,2]

- Language Use Cases:
  - "RNNs are designed to take sequences of text as inputs or return sequences of text as outputs, or both." [1]
  - Translation [1]
  - Summarization, Writing [2]
- Time Series, e.g. Stock Price prediction [3,4]
- …



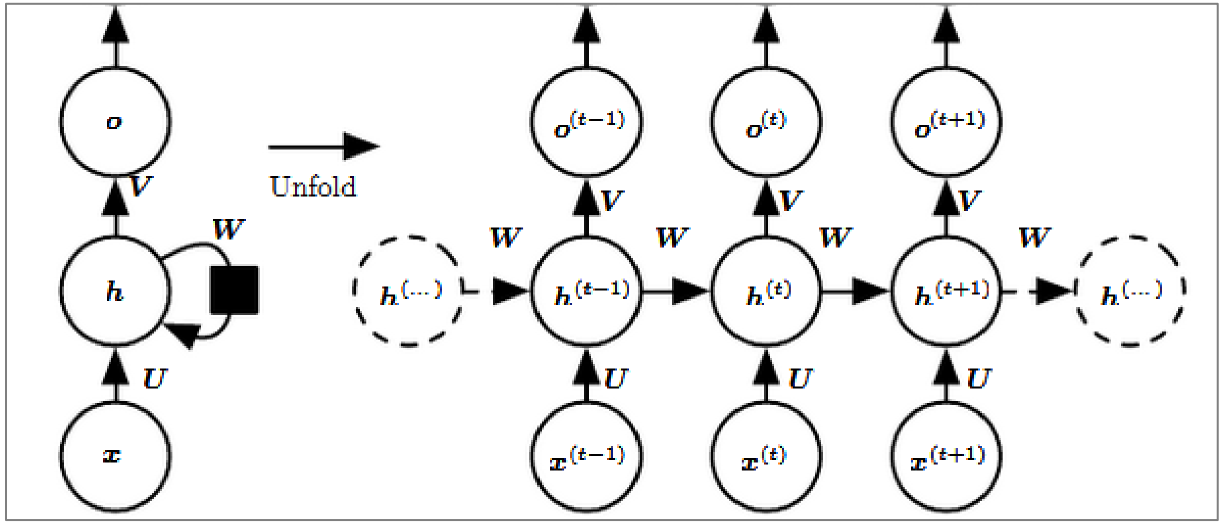
1-hot encoding of a dictionary of size 4



1. "Language Translation with RNNs, Build a recurrent neural network that translates English to French", Thomas Tracey, 2019
2. "The Unreasonable Effectiveness of Recurrent Neural Networks", Andrej Karpathy, 2015
3. "Stock Market Prediction Using LSTM Recurrent Neural Network", Adil MOGHAR, Mhamed HAMICHE, 2020
4. "Share Price Prediction using RNN and LSTM", Rishi Rajak, 2021

RNN (formulation)

- [“Recurrent Neural Networks (RNNs), Implementing an RNN from scratch in Python”, Javaid Nabi, 20189]
- [https://www.deeplearningbook.org/contents/rnn.html]



$$
\begin{aligned}
a^{(t)} &= b + W h^{(t-1)} + U x^{(t)} \\
h^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= c + V h^{(t)} \\
\hat{y}^{(t)} &= \mathrm{softmax}(o^{(t)})
\end{aligned}
$$

e.g. 1-hot encoding of a dictionary of size 8000

e.g. prob-distribution across all words

$$
\begin{aligned}
x_t &\in \mathbb{R}^{8000} \\
o_t &\in \mathbb{R}^{8000} \\
h_t &\in \mathbb{R}^{100} \\
U &\in \mathbb{R}^{100 \times 8000} \\
V &\in \mathbb{R}^{8000 \times 100} \\
W &\in \mathbb{R}^{100 \times 100}
\end{aligned}
$$

$$
\begin{aligned}
a^{(t)} &= b + W h^{(t-1)} + U x^{(t)} \\
h^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= c + V h^{(t)} \\
\hat{y}^{(t)} &= \mathrm{softmax}(o^{(t)})
\end{aligned}
$$

Why?

100*100*2 + 100*8000*2 + 100*2

(e^(x) — e^(-x))/(e^(x) + e^(-x)) → 100*5

8000*100*2 + 100

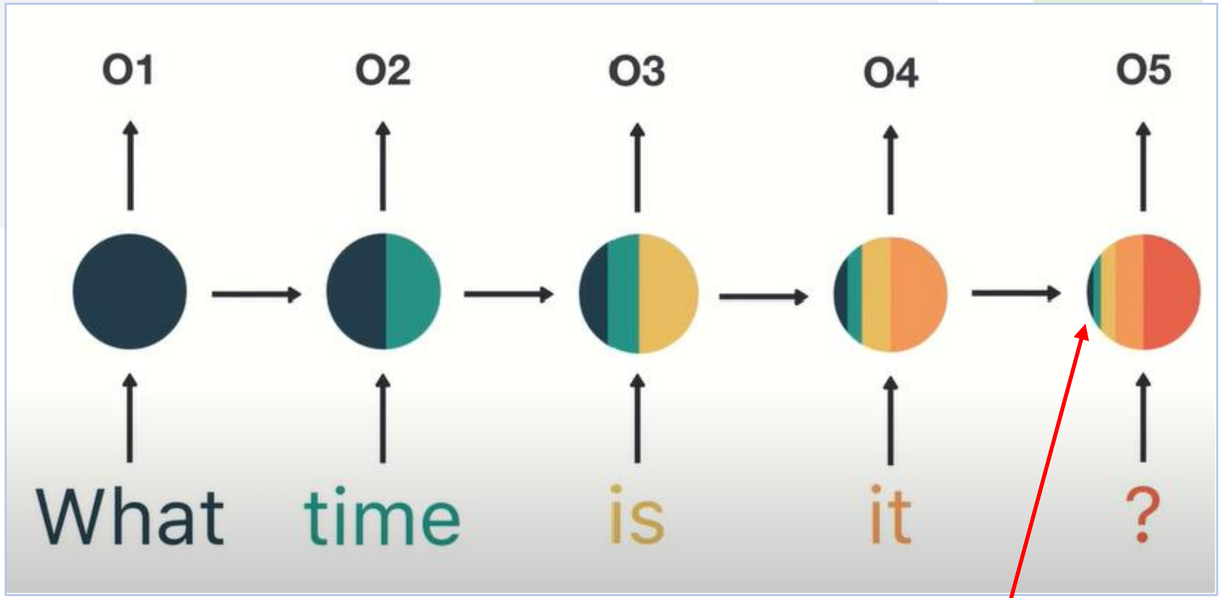
100*3

[Tanh Activation Function Time Complexity, Abhishek Chatterjee, 2018]

Recursive NN (complains)

- Information, influence vanishing across far-apart units *[1]*
- Causes *[2]* :
  - Vanishing Gradient Problem
  - Exploding Gradient Problem

From [1]

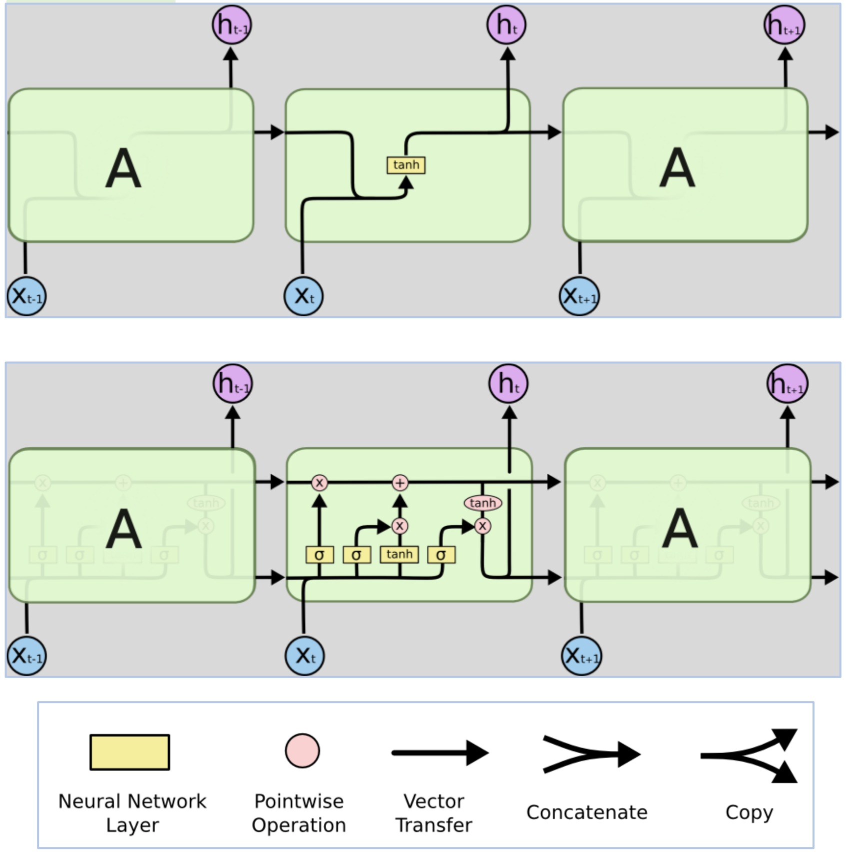


Weight associated with clue from the word “time” is becoming small.

1. Illustrated Guide to Recurrent Neural Networks: Understanding the Intuition, Michael Phi, 2018
2. ” Let’s Understand The Problems with Recurrent Neural Networks”, Siddharth M, 2021

LSTM-RNN (Long Short Term Memory)

From [1]



- Long and impressive timeline of development 1991-2020 [2].
- Many normalized and trainable (weights) gates (layers) to mitigate vanishing gradient issue.
- Forget gate to selective preserve long term influence

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}
$$

operator $\odot$ denotes the Hadamard product (element-wise product)

1. Understanding LSTM Networks, Christopher Olah, 2015
2. https://en.wikipedia.org/wiki/Long_short-term_memory
3. A Deep Dive into LSTM's Trainable Parameters, Gundluru Chadrasekhar, 2020

Time complexity: $4 * 2 * (m*n + n*n + n)$, m: len(x_t); n: len(h_t)

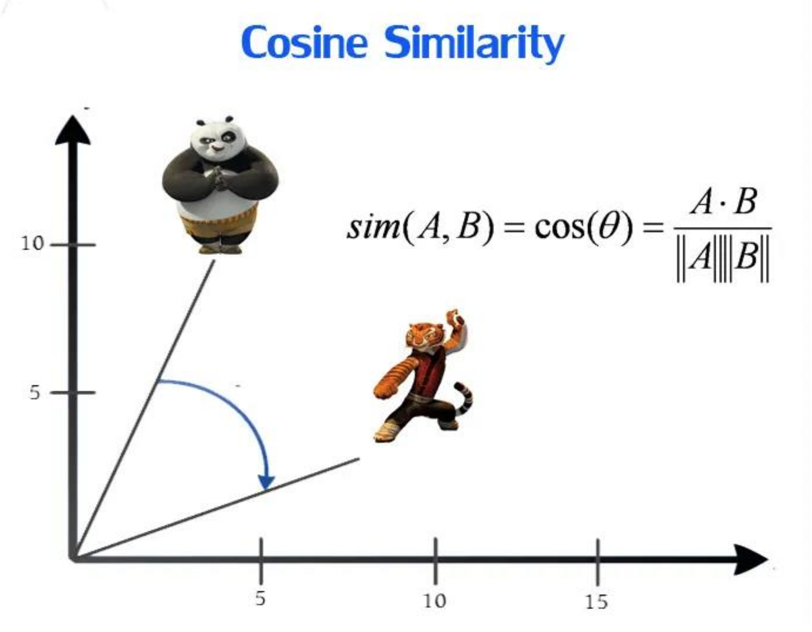
Space complexity: $4 * (m*n + n*n + n)$ floats for weights

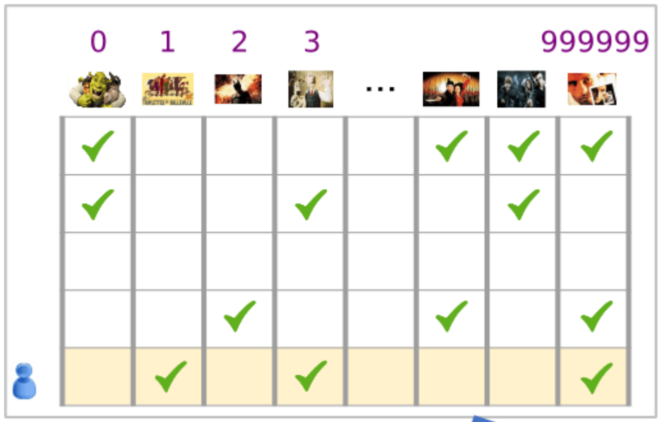
Embeddings

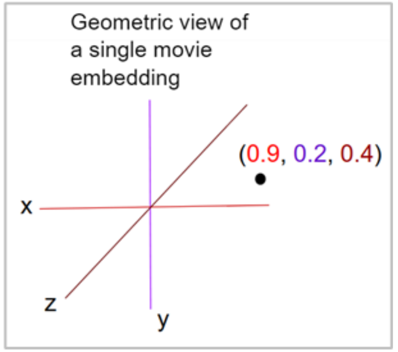
Word2Vec

- “objective is to have words with similar context occupy close spatial positions.
- Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.”

[“Introduction to Word Embedding and Word2Vec”, Dhruvil Karani, 2018]



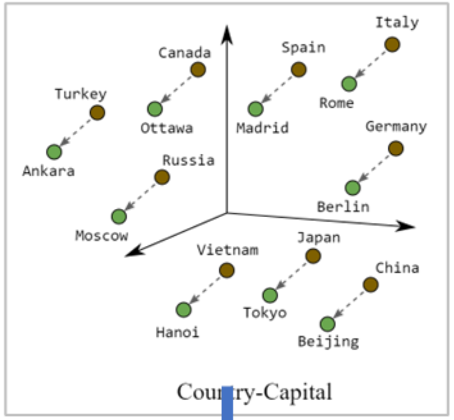


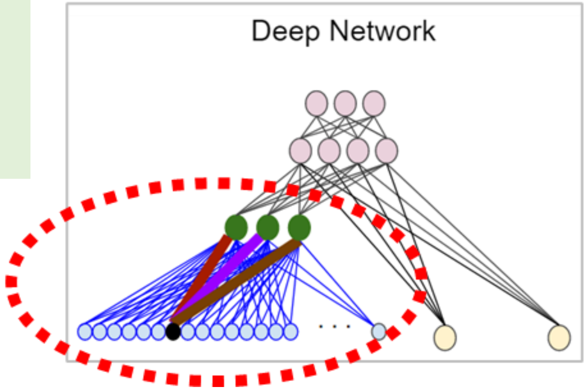


Trainable Embeddings in Recommenders

- User/movie matrix
- All moves (dictionary) to be embed into high-dim space
- (Some) implicite distance measures
- Make the embedding matrix (mapping) completely trainable to optimize a task-specific loss function (labeled data, supervised)

[developers.google.com/machine-learning/crash-course/embeddings, retrieved 2023]



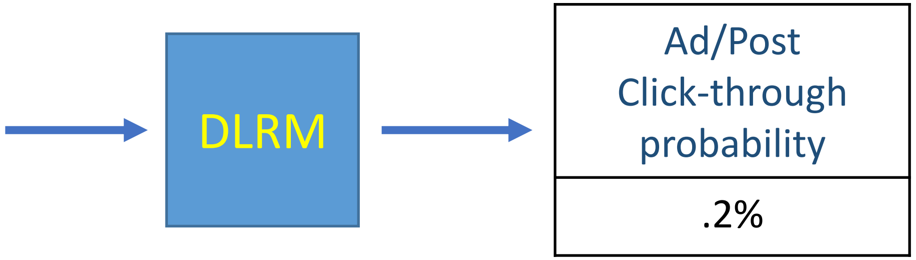


An Synthetic Example for DLRM (Meta, User-Item Interaction Projection)

- Question to address:
  If recommend a “Health” related advertisement or post to user “Alice”, what’s the probability that Alice will click the ad/post?

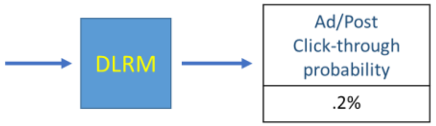
- The inference process:

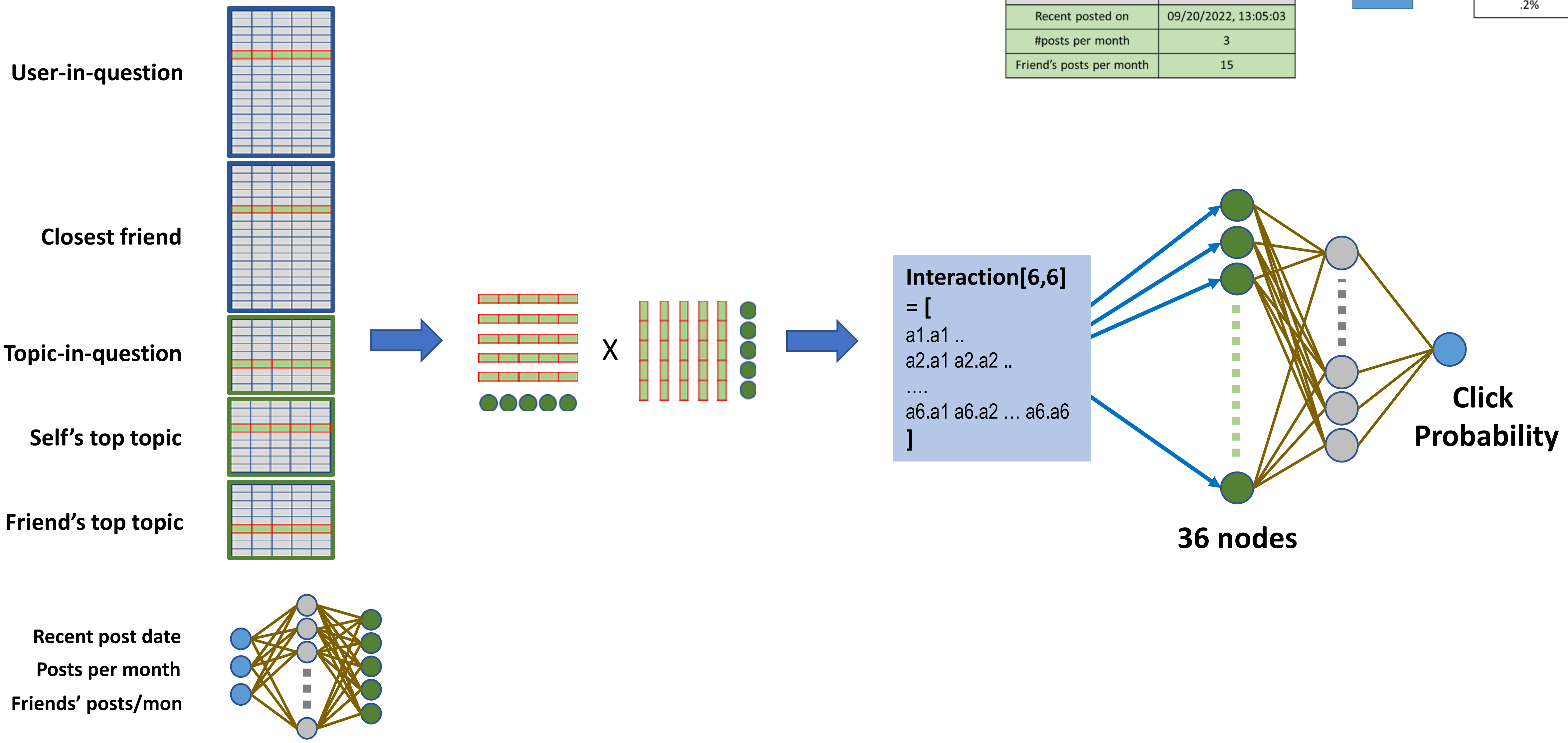
	Fields	**Values**
User-Item Pair	**User name**	**Alice**
	Ad/Post topic	**Health**
Categorical data	Cloest friend	Bob
	Self’s favorate topic	Soccer
	Friend’s favorate topic	Technology
Numerical data	Recent posted on	09/20/2022, 13:05:03
	#posts per month	3
	Friend’s posts per month	15



An Synthetic DLRM Inference

Fields	Values
User name	Alice
Ad/Post topic	Health
Cloest friend	Bob
Self's favorate topic	Soccer
Friend's favorate topic	Technology
Recent posted on	09/20/2022, 13:05:03
#posts per month	3
Friend's posts per month	15





Confused? Layers, State-transition graph, Matrix Multiplication

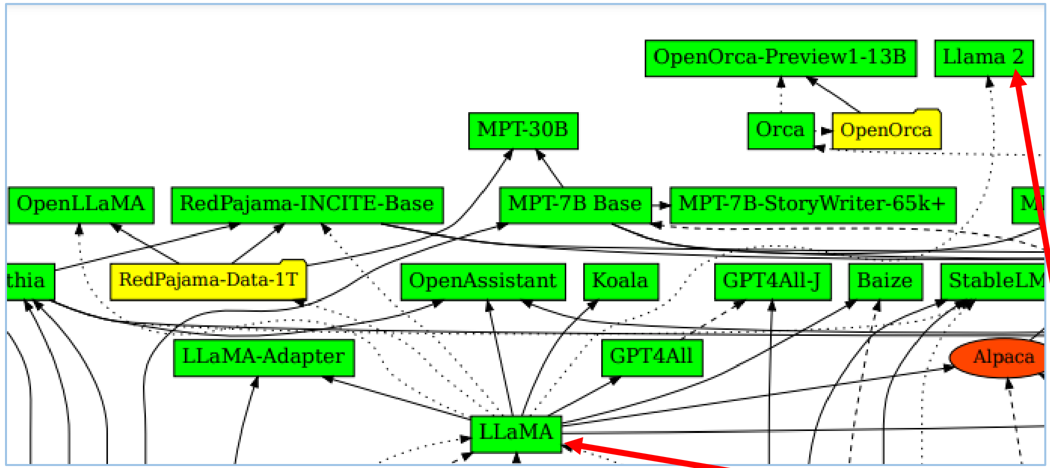
- Can you have a mental visualization of 3D/4D tensor computation?
- How about adding "loop tiling" to the complexity?
- Adding Tensor Parallelism ?

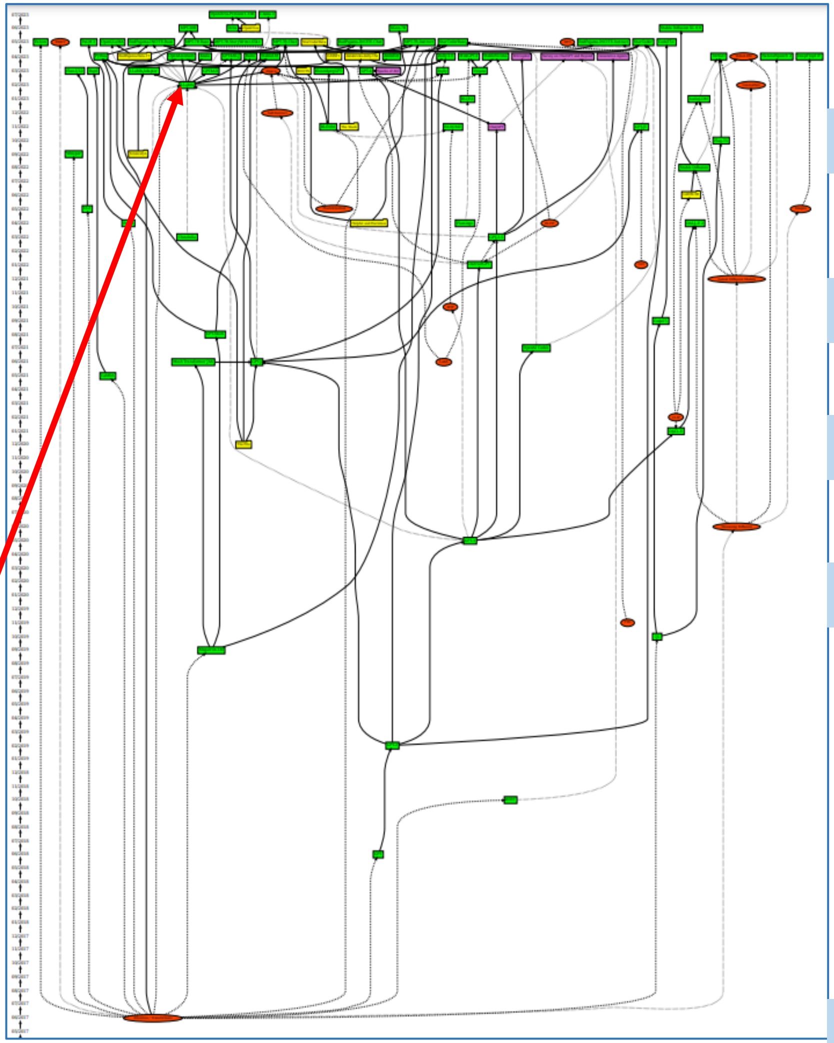
- "Inside the Matrix: Visualizing Matrix Multiplication, Attention and Beyond", team pytorch, 2023
  - https://pytorch.org/blog/inside-the-matrix/
  - Play with it
  - A great software effort (AI burst to technology and science)

NLP/LLM/FM explosive development

- An explosive 7-year development of LLM models
  - https://github.com/rain-1

exponential growth





2023

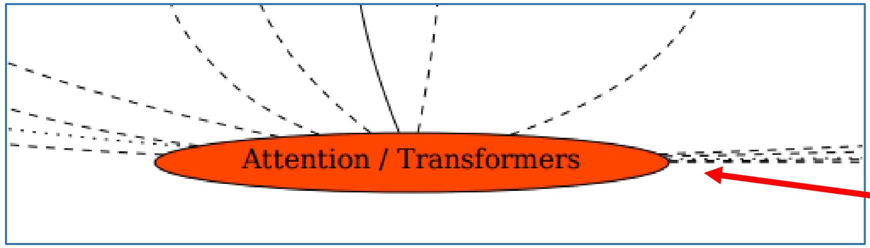
2022

2021

2020

2017

Current focus



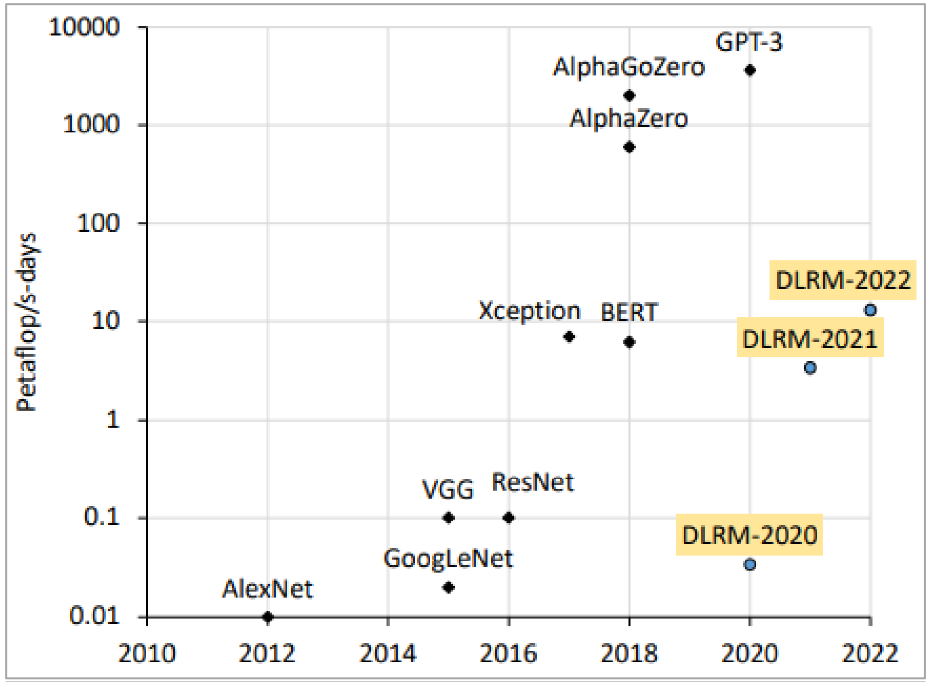
Starting point

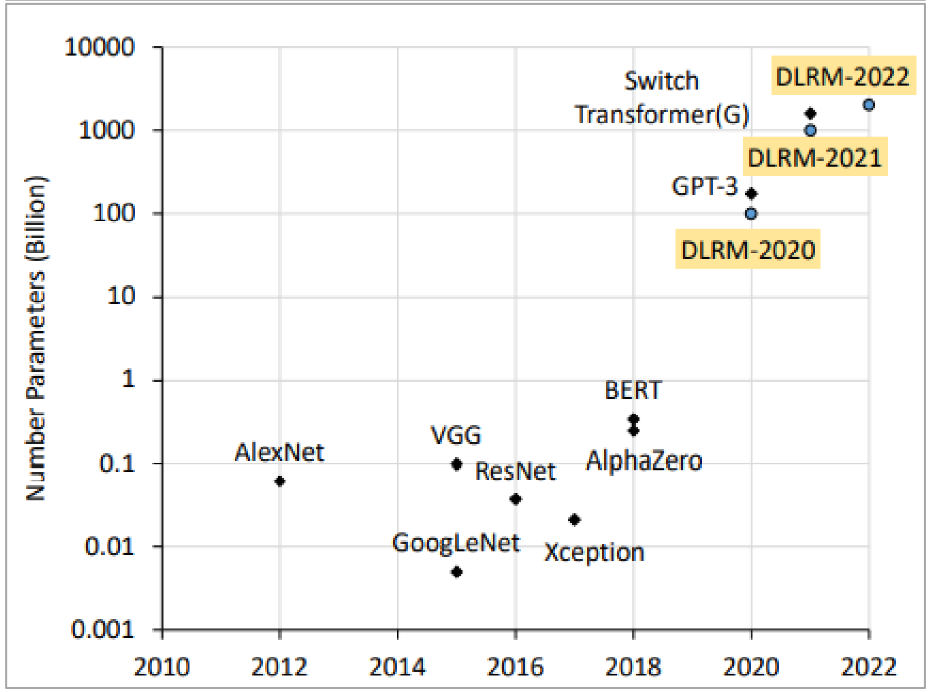
Model Size Scaling

The scaring trend in 2022

Model Name	# tunable params	Primary precision	Type	Organization	Open source available	Accelerator types	Announcing publishing dates
BERT	340 M		Dense	Google	y		Nov-18
GPT3	175 B		Dense	Open AI	N		Jun-20
Jurassic	178 B		Dense	AI21			Aug-21
Gopher	280 B		Dense	DeepBrain			Jan-22
MT-NLG	530 B		Dense	Nvidia Microsoft	y		Feb-22
LaMDA	137 B		Dense	Google	n	TPU v4	Feb-22
Chinchilla	1.4 T, 70 B		Sparse	DeepBrain	n		Mar-22
OPT	175 B		Dense	Meta	y (gh, hf)		Jun-22
BLOOM	176 B		Dense	BigScience	y (hf)		Aug-22
GLM	130 B		Dense	Tsinghua	y	GPU	Aug-22
GLaM	1.2 T		Sparse	Google	n	TPU v3	Aug-22
PaLM	540 B		Dense	Google	n	TPU v4	Oct-22
MoE (meta)	1.1 T		Sparse	Meta	y (gh)		Nov-22

Nov. 2022





[June 2022 DLRM co-design, ISCA]

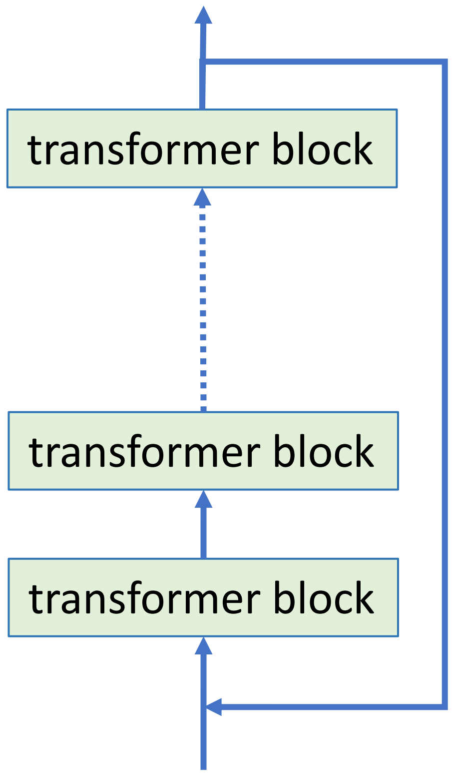
Foundation Model: the naming

- **Foundation model (FM)** Definition (wiki):
  - “a large machine learning (ML) model trained on a vast quantity of data at scale (often by self-supervised learning or semi-supervised learning)[2] such that it can be adapted to a wide range of downstream tasks[3][4] .”
  - By: CRFM, in On the Opportunities and Risks of Foundation Models, 2021
- **Transformer model** centered
  - Attention block
  - Repeated transformer blocks
- Developed in **NLP** field with 20 years of slow brewing.
- Ever growing model sizes: **LLM**
  - For a given family models (e.g. gpt2, bert), the predictive capacity grows with it model size.

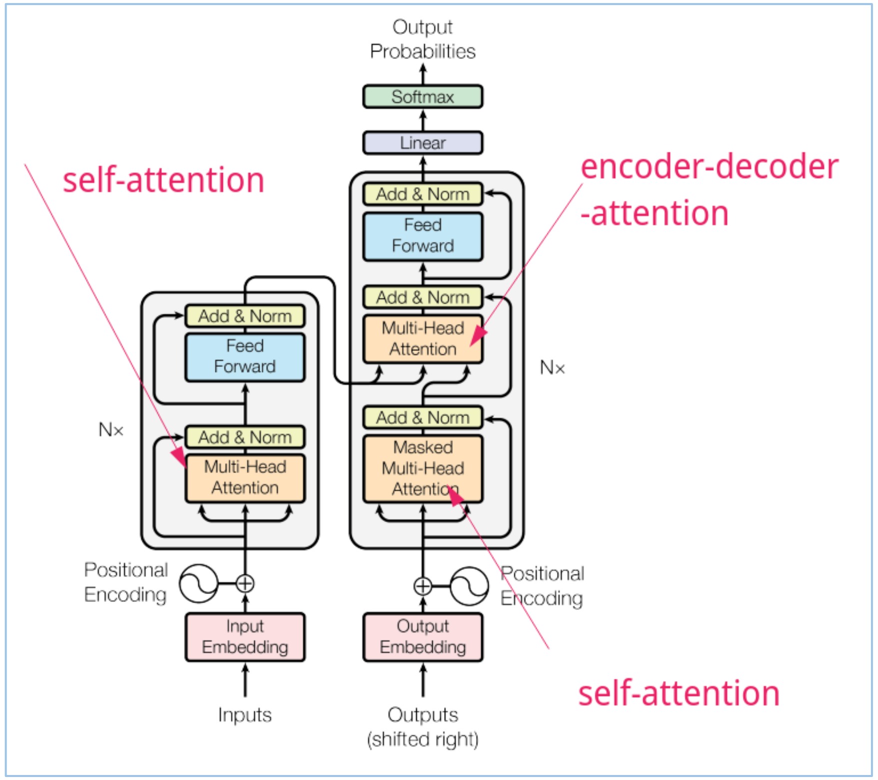
Transformer model architecture

- Repeated transformer blocks, with key configuration parameters:
  - Number of transformer block layers
  - Transformer block: Encoder, decoder, encoder-decoder
  - Embedding (hidden, reduction) dimension size

Typical transformer model



transformer block architecture

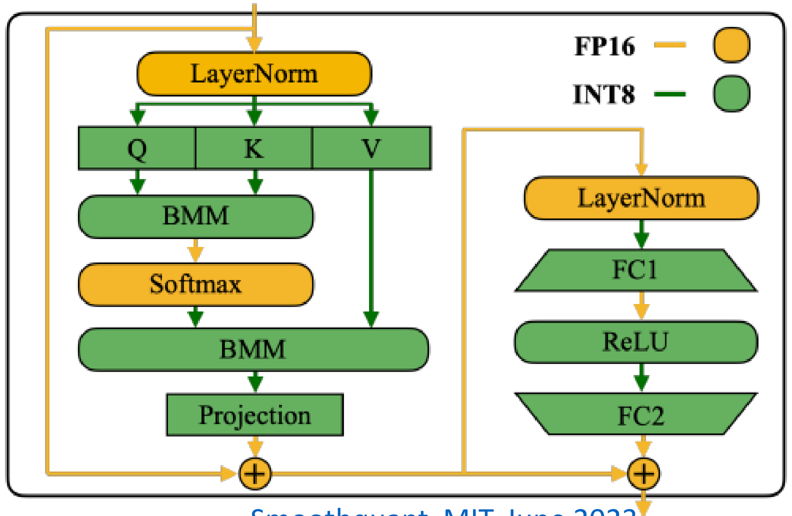


Attention is all you need (oversimplified), Aayush Neupane

Dot-product attention

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

Attention is.. , Google, June 2017



Smoothquant, MIT, June 2023

FM Applications

- Extensible usage pattern (typical):
  - Unsupervised **pretraining** over huge amount of data
    +
    Supervised **finetuning** over domain/task specific labeled data
    - Fine tuning, changes model weights
  - Pretrain + multi-shot : prompt engineering
    - Prompt eng. “getting the model to do what you want at inference time by providing enough context, instruction and examples without changing the underlying weights. fine-tuning”
    - Fine tuning vs prompt engineering LLM, Niels Bantilan, may 2023
- Application/Tasks
  - NLP tasks for training/tuning: language modeling, QA, reading, sentiment, paraphrasing.
  - In more than NLP: translation, summarization, writing, image segmentation, bio-sequence, coding, etc.

Transformer models in practice

- HF/transformers:
  - transformer-based NN architectures + pretrained models.
- See the architecture: FX Graph, NSYS, torch-profiler, model-print, abstract code,
  - Transformer explained
  - Terms: encoder, decoder, encoder-decoder, position embedding (where the words are in the input sequence),

Transformer models in practice

- HF/transformers:
  - transformer-based NN architectures + pretrained models.

Selected list of pretrained models hosted out of HuggingFace

bert-base-cased	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on cased English text.
bert-large-cased	24-layer, 1024-hidden, 16-heads, 340M parameters. Trained on cased English text.
gpt2	12-layer, 768-hidden, 12-heads, 117M parameters. OpenAI GPT-2 English model
gpt2-medium	24-layer, 1024-hidden, 16-heads, 345M parameters. OpenAI's Medium-sized GPT-2 English model
gpt2-large	36-layer, 1280-hidden, 20-heads, 774M parameters. OpenAI's Large-sized GPT-2 English model
gpt2-xl	48-layer, 1600-hidden, 25-heads, 1558M parameters. OpenAI's XL-sized GPT-2 English model
t5-large	~770M parameters with 24-layers, 1024-hidden-state, 4096 feed-forward hidden-state, 16-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-3B	~2.8B parameters with 24-layers, 1024-hidden-state, 16384 feed-forward hidden-state, 32-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-11B	~11B parameters with 24-layers, 1024-hidden-state, 65536 feed-forward hidden-state, 128-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)

https://huggingface.co/transformers/v3.3.1/pretrained_models.html

Huggingface Transformer Lib

Super easy starting point

Google: huggingface gpt2 → https://huggingface.co/gpt2

```
from transformers import GPT2Tokenizer, GPT2Model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2Model.from_pretrained('gpt2')
text = "Replace me by any text you'd like."
encoded_input = tokenizer(text, return_tensors='pt')
output = model(**encoded_input)
```

```
from transformers import pipeline, set_seed
generator = pipeline('text-generation', model='gpt2')
set_seed(42)
generator("Hello, I'm a language model,", max_length=30, num_return_sequences=5)
```

Getting serious?

https://github.com/huggingface/transformers/blob/main/examples/pytorch

- Manage realistic input datasets
- Inference performance boost
  - Float-16 vs flaot-32 (bits)
  - Model computation graph optimization (torch.jit.trace)
- Reproducible and reusable effort (code)

Visual of Transformer Models

```
DistilBertForMaskedLM(
  (activation): GELUActivation()
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (vocab_transform): Linear(in_features=768, out_features=768, bias=True)
  (vocab_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (vocab_projector): Linear(in_features=768, out_features=30522, bias=True)
  (mlm_loss_fct): CrossEntropyLoss()
)
```

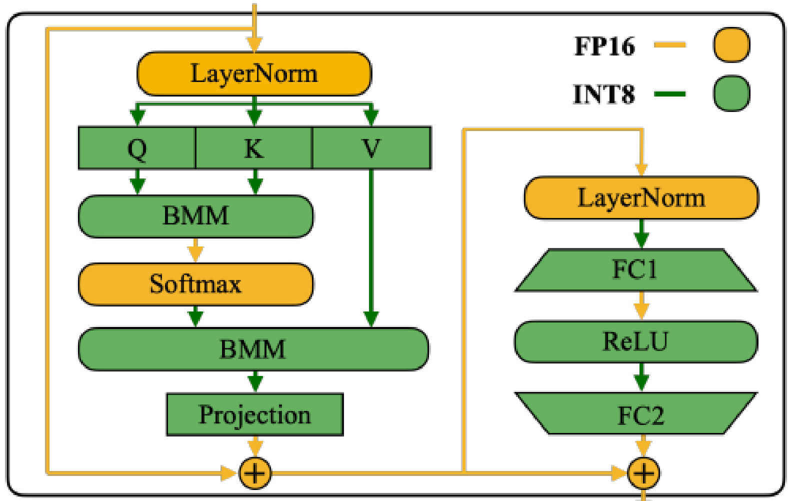
https://huggingface.co/docs/transformers/v4.34.0/en/model_doc/distilbert#transformers.DistilBertForMaskedLM

- Model file only keeps the trained weights (parameters)
- There are no trainbles for softmax BMM, GELU layer. Some time model print won't show.

Get the visual memory

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

Attention is.. , Google, June 2017

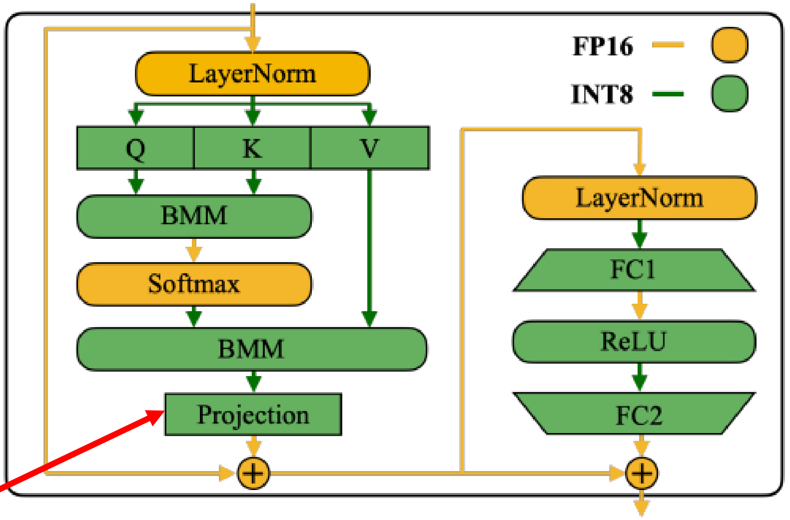


Smoothquant, MIT, June 2023

Visual of Transformer Models

```
(0-5): 6 x TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
    (activation): GELUActivation()
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
```

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



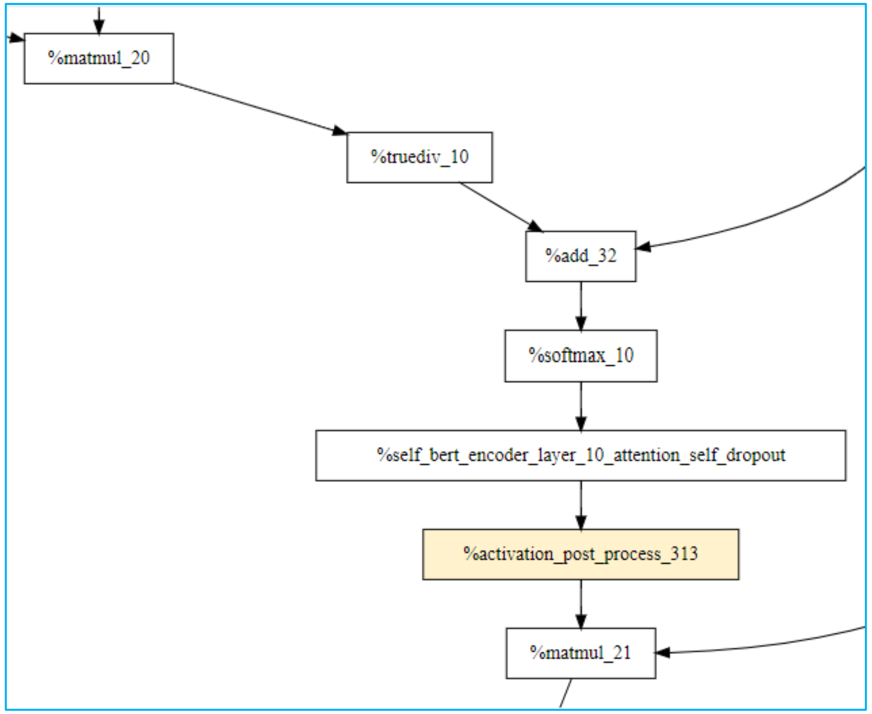

```
name: distilbert.transformer.layer.3.attention.q_lin.weight ----------  weights: torch.Size([768, 768])
name: distilbert.transformer.layer.3.attention.q_lin.bias ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.attention.k_lin.weight ----------  weights: torch.Size([768, 768])
name: distilbert.transformer.layer.3.attention.k_lin.bias ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.attention.v_lin.weight ----------  weights: torch.Size([768, 768])
name: distilbert.transformer.layer.3.attention.v_lin.bias ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.attention.out_lin.weight ----------  weights: torch.Size([768, 768])
name: distilbert.transformer.layer.3.attention.out_lin.bias ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.sa_layer_norm.weight ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.sa_layer_norm.bias ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.ffn.lin1.weight ----------  weights: torch.Size([3072, 768])
name: distilbert.transformer.layer.3.ffn.lin1.bias ----------  weights: torch.Size([3072])
name: distilbert.transformer.layer.3.ffn.lin2.weight ----------  weights: torch.Size([768, 3072])
name: distilbert.transformer.layer.3.ffn.lin2.bias ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.output_layer_norm.weight ----------  weights: torch.Size([768])
name: distilbert.transformer.layer.3.output_layer_norm.bias ----------  weights: torch.Size([768])
```

Viewpoint of Coders, Researchers, and Engineers

Make sense to torch.**compile**

[bert_model_prep_cleanup.svg]

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



K/Q/V

OUT_proj

FeedForward