香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

# CSC5051/MDS5110/CSC6052 : Natural Language Processing

## Lecture 4: Deep Learning for NLP

## Architecture Engineering: Transformer and Beyond

Spring 2026

Benyou Wang
School of Data Science

# To recap…

# Example of LM: A bigram model

**Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'**

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{tired}) = 1$

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{use}) = 1$

$P(w^{(i)} = \textbf{sister} \mid w^{(i-1)} = \textbf{her}) = 1$

$P(w^{(i)} = \textbf{beginning} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{reading} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{bank} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{book} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{use} \mid w^{(i-1)} = \textbf{the}) = 1/3$

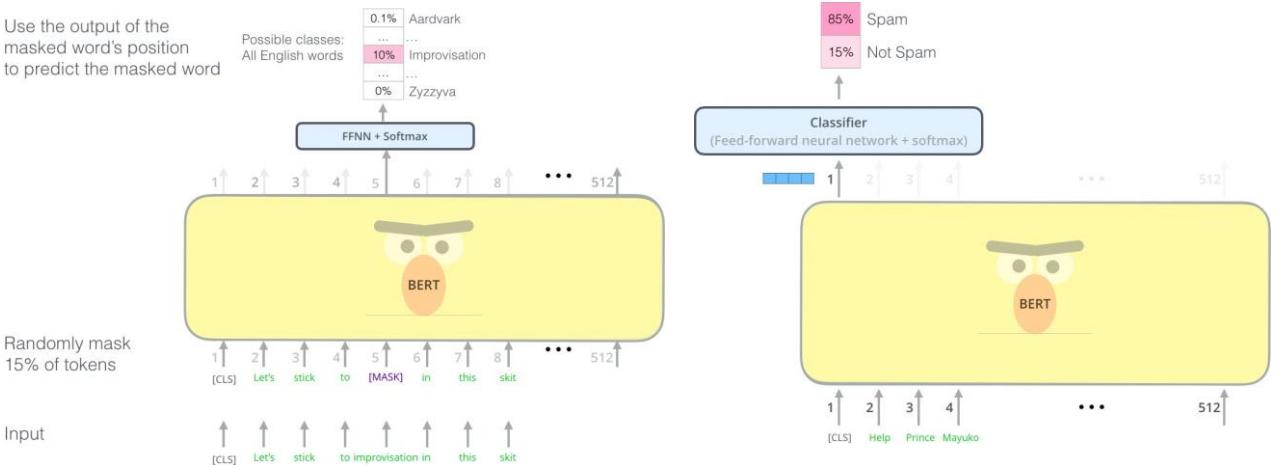
# Generating Shakespeare

| | |
|---|---|
| Unigram | • To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have<br>• Every enter now severally so, let<br>• Hill he late speaks; or! a more to leg less first you enter<br>• Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like |
| Bigram | • What means, sir. I confess she? then all sorts, he is trim, captain.<br>•Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.<br>•What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?<br>•Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt |
| Trigram | • Sweet prince, Falstaff shall die. Harry of Monmouth's grave.<br>• This shall forbid it should be branded, if renown made it empty.<br>• Indeed the duke; and had a very good friend.<br>• Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. |
| Quadrigram | • King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;<br>• Will you not tell me who I am?<br>• It cannot be but so.<br>• Indeed the short and the long. Marry, 'tis a noble Lepidus. |

# Masked language models (MLMs)

- Solution: Mask out 15% of the input words, and then predict the masked words

# Pre-training and fine-tuning



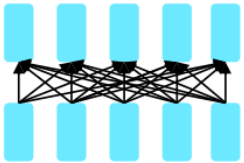

Pre-training

Fine-tuning

Key idea: all the weights could be fine-tuned on downstream tasks

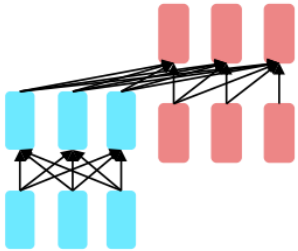
# Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



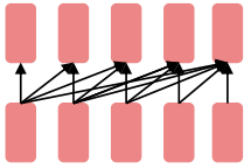
**Encoders**

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



**Encoder-Decoders**

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



**Decoders**

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

# Contents

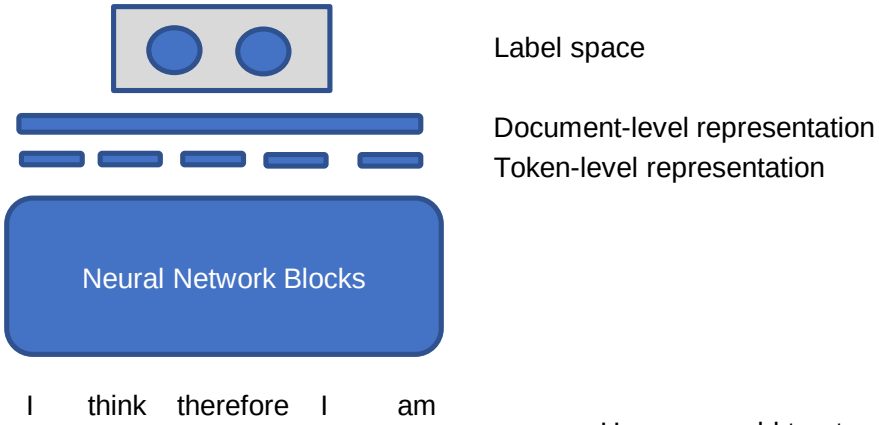
- **Neural Network in NLP and applications**
- Inductive bias of Neural Network for NLP
  Semantic Abstraction and Semantic composition
- MLP, CNN & RNN
- Transformer
- Scaling law and emergent ability

# Typical NLP Applications

NLU (**N**atural Language **U**nderstanding)

- Sentence classification
- Question answering (span prediction)
- NER (sequential labeling)
- Retrieval (document pair classification)

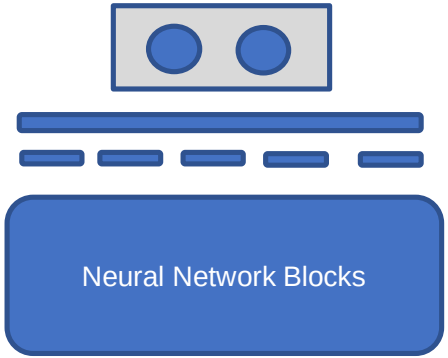
These tasks need either a **sentence-level** or (contextualized) **token-level** representation



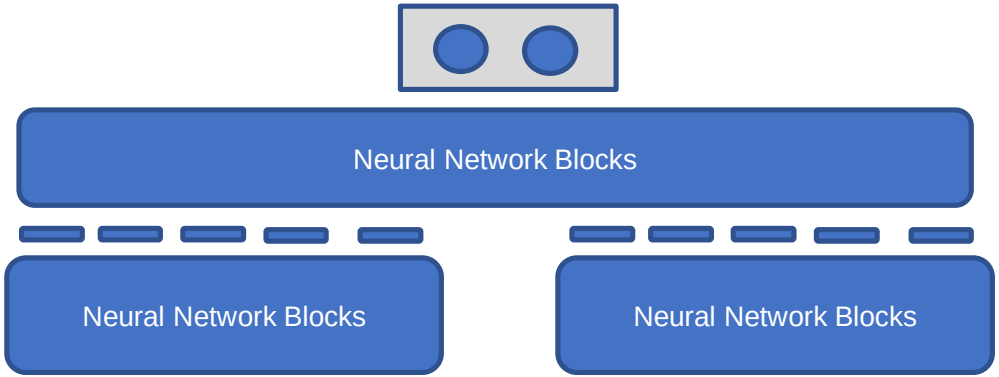

Here we could treat a document as a longer “sentence”

# Sentence-level tasks

Fusion at input tokens (BERT)
Fusion at the middle (Condenser)
Fusion at output (SimCSE)

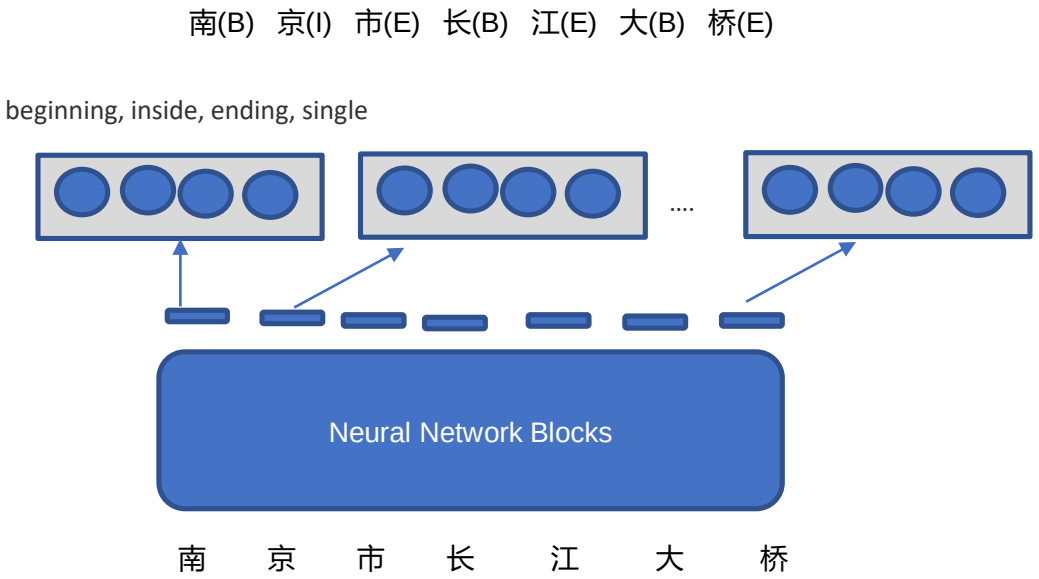



**Single Sentence**

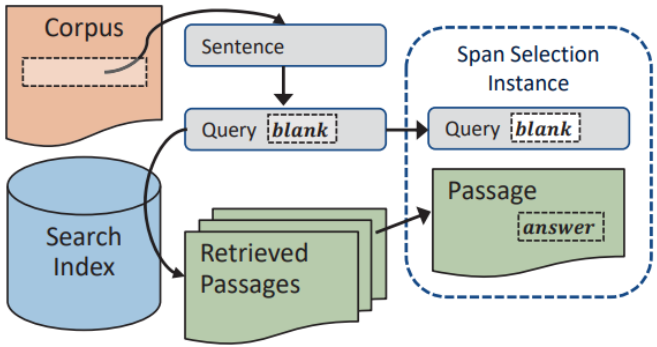



**Sentence pair classification**

# Token-level tasks, e.g. word segmentation



# Token-level tasks, e.g. span prediction



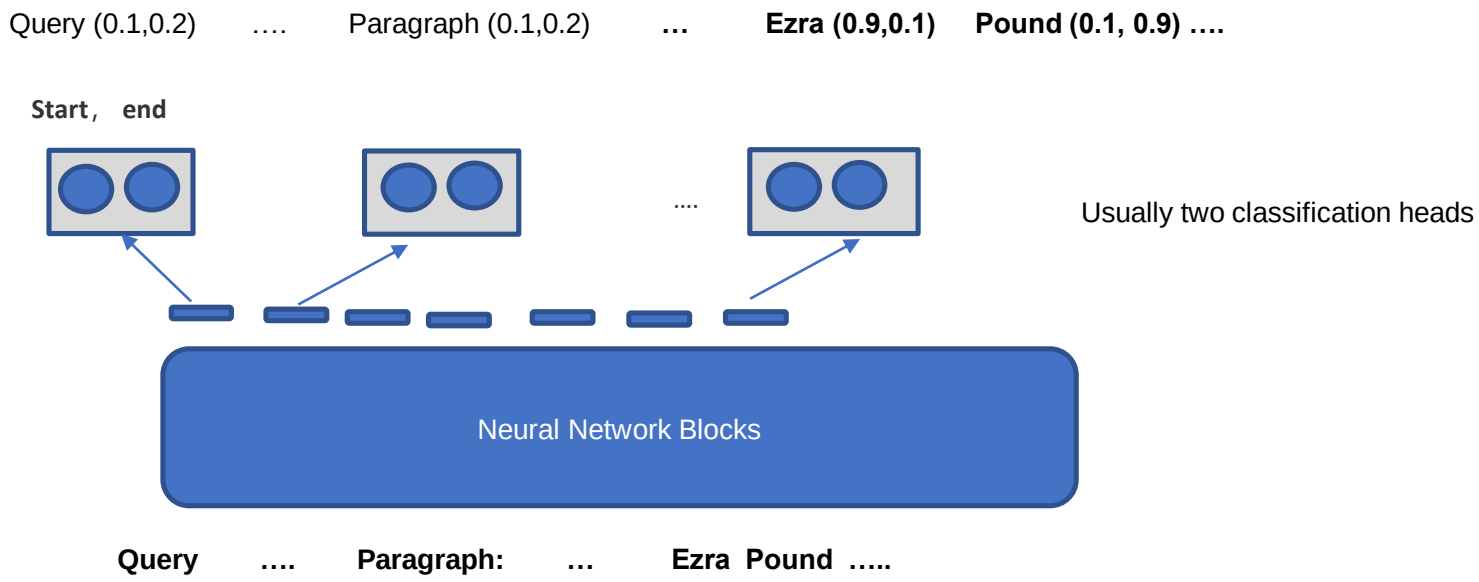

**Example:**

| | |
|---|---|
| **Query** | "In a station of the metro" is an Imagist poem by [BLANK] published in 1913 in the literary magazine Poetry |
| **Passage** | . . . Ezra Pound 's famous Imagist poem, "In a station of the metro", was inspired by this station . . . |
| **Answer** | Ezra Pound |

**Background**

https://aclanthology.org/2020.acl-main.247.pdf

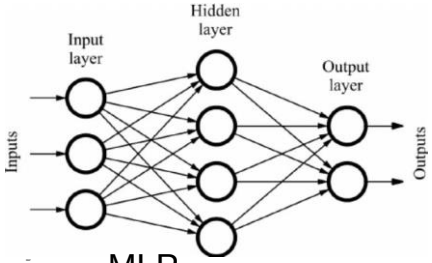
# Token-level tasks, e.g. span prediction



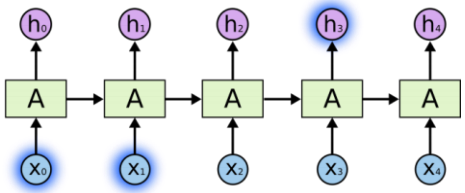

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15792151.pdf

# Which neural networks?

- ~~✔ Multilayer Perceptron (MLP)~~
- ~~✔ Convolutional neural network~~
- ~~✔ Recurrent neural network~~
- ✔ **Transformer**

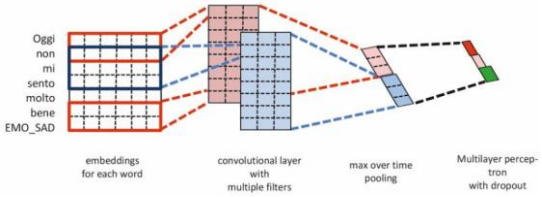



MLP

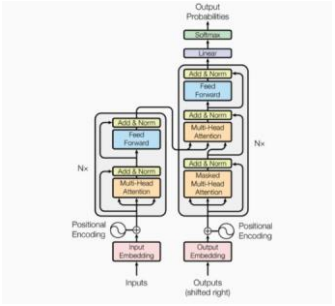



Recurrent NNs

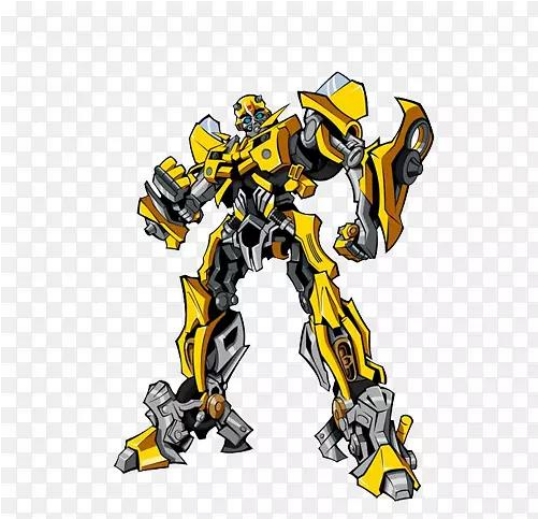
Convolutional NNs

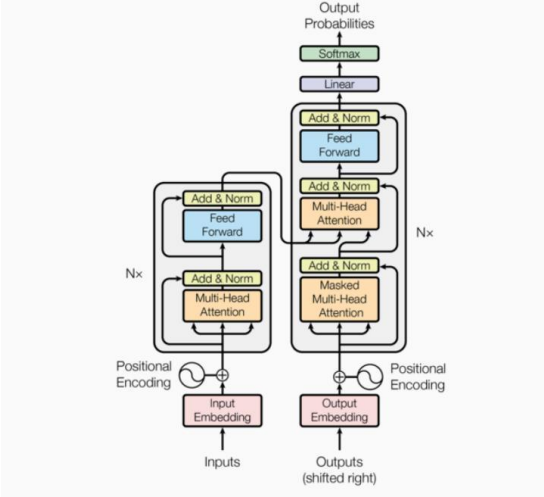



Transformer



# Why Transformer?





# Contents

- Neural Network in NLP and applications
- **Inductive bias of Neural Network for NLP**
  Semantic Abstraction and Semantic composition
- MLP, CNN & RNN
- Transformer
- Scaling law and emergent ability

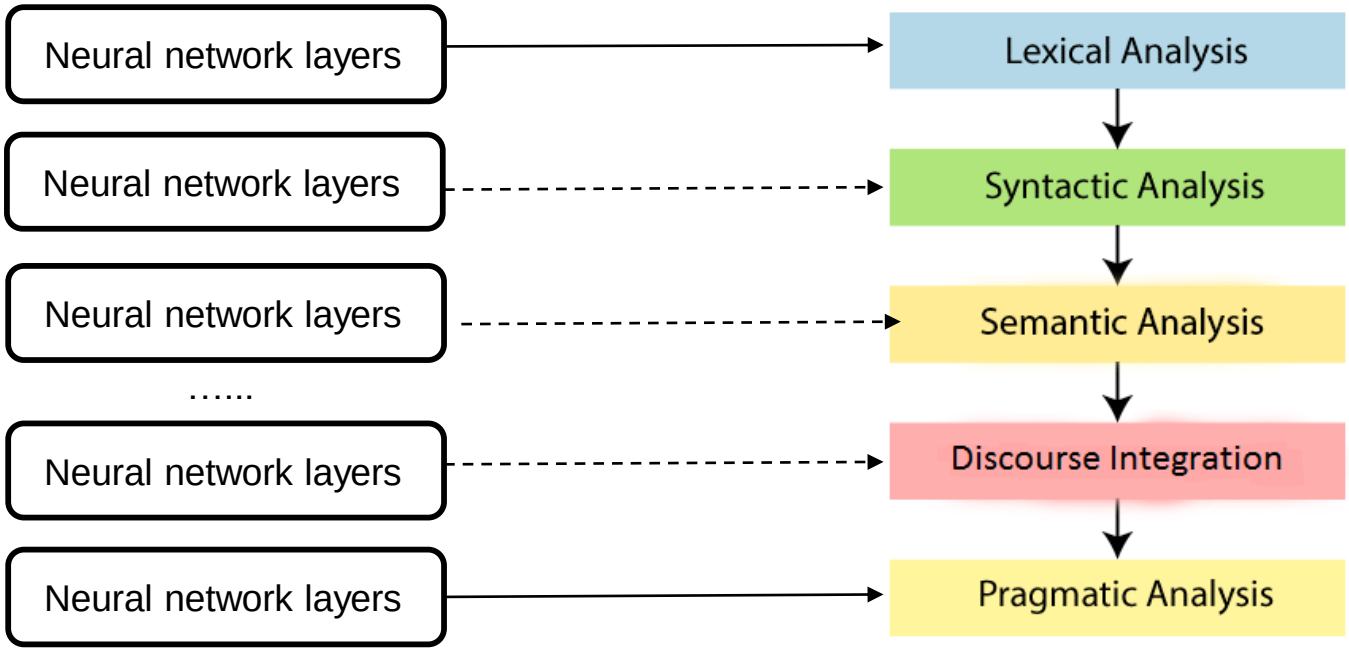
# (I) What is Semantic abstraction?


Pixel -> texture -> region -> object -> relation -> semantics-> …

Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. https://arxiv.org/pdf/1707.03816. ICCV 2015

# Higher-level layers deal with higher-degree abstraction

**Input**: I think therefore I

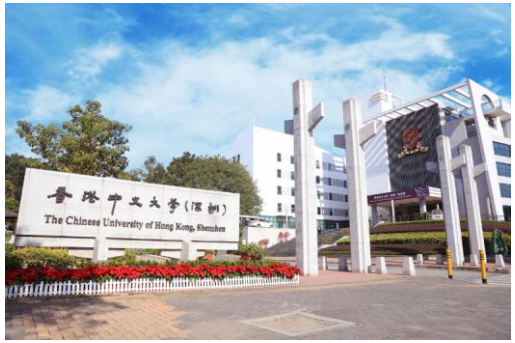



**Output**: am

# (II) What is Semantic composition?

Ivory (象牙)

tower (塔)

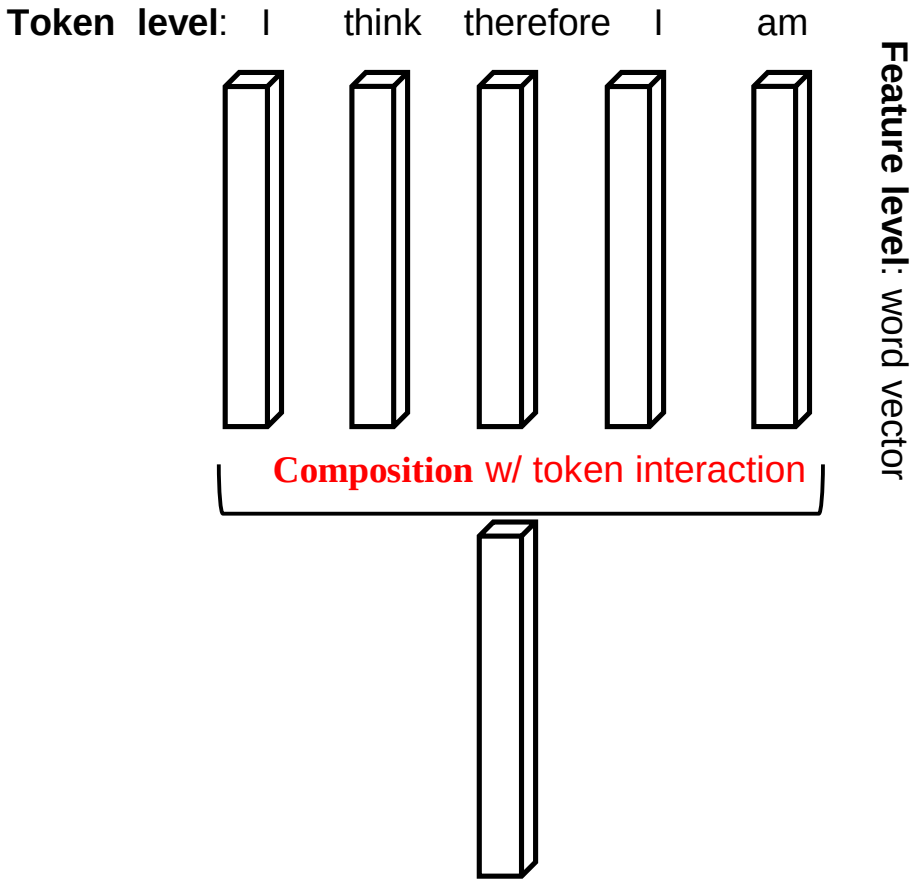


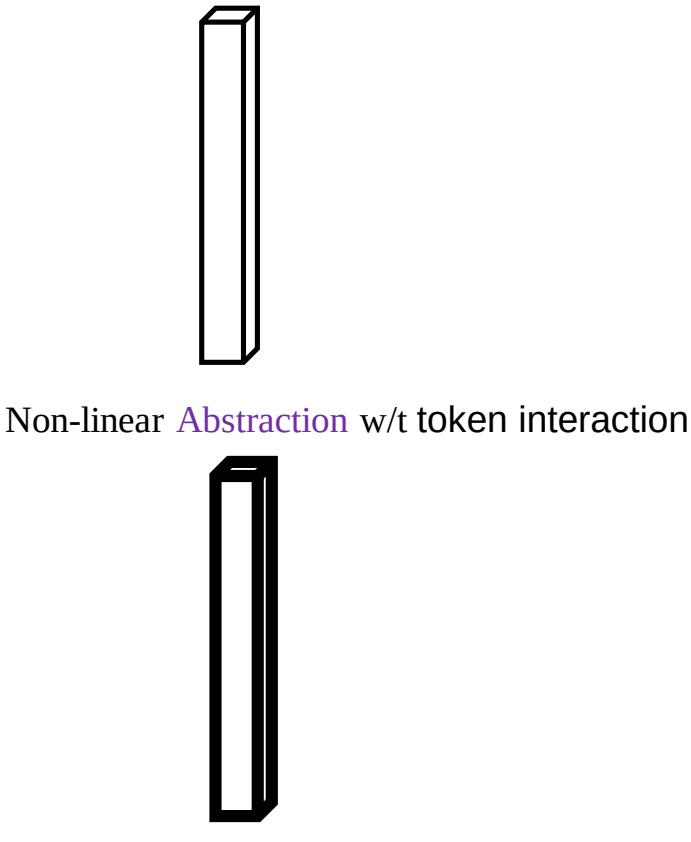

Ivory tower(象牙塔)

Semantic composition is the task of understanding the meaning of text by composing the meanings of the individual words in the text.

It involves **token interaction**

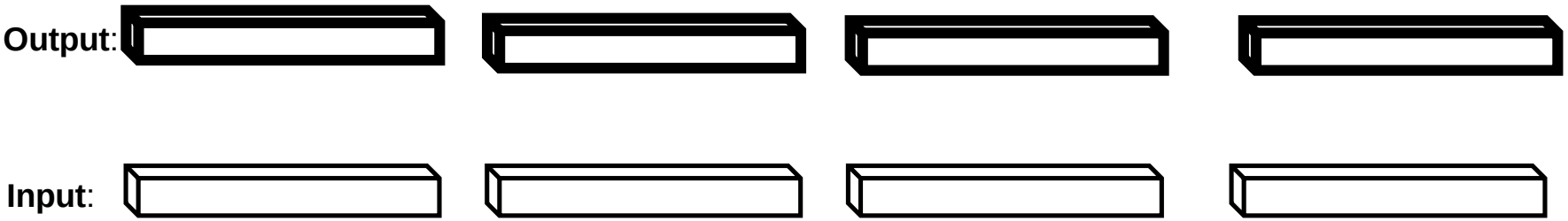
# Semantic composition vs. Semantic Abstraction



# How to combine composition and Abstraction

**A flatten solution: MLP (e.g. NNLM)**

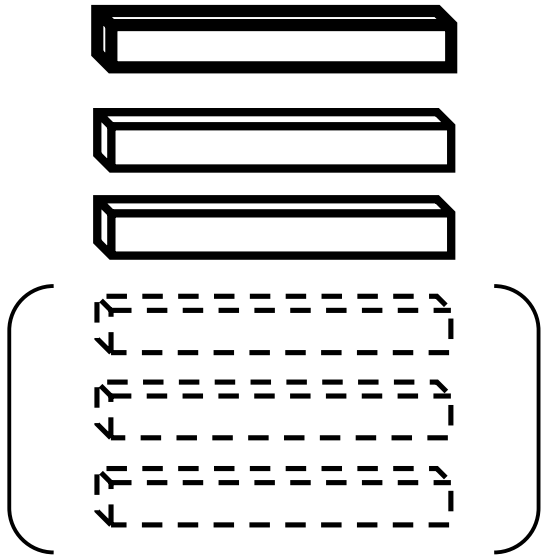



**Complexity: O($D^2L^2$)**

Yoshua Bengio et.al A Neural Probabilistic Language Model. NIPS 2003

# How to combine composition and Abstraction

**A variant of MLP (**e.g. CBoW**)**



**Remove** token interaction in deeper layers

Mean pooling (token interaction) in the first layer

**Complexity: O($D^2$)**

T Mikolov et.al Efficient Estimation of Word Representations in Vector Space. https://arxiv.org/abs/1301.3781

# Inductive bias of composition

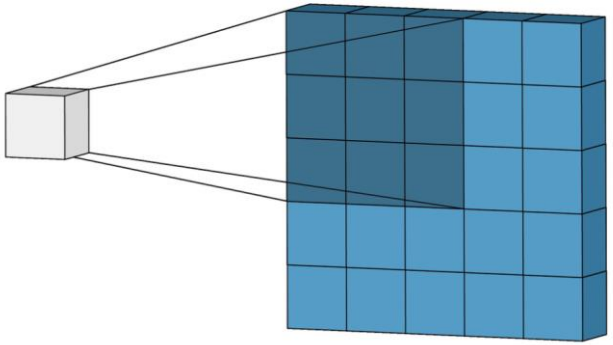
How we believe **tokens should be interacted** as the inductive bias, also considering semantic abstraction simultaneously?

Definition: The inductive bias (a.k.a learning bias) of a learning algorithm is the set of assumptions that a machine learning algorithm makes about the relationship between input variables (features) and output variables (labels) based on the training data.

# Inductive bias of composition

CNN: **local** composition within a window

RNN: **recurrently** compose tokens from left to right or right to left.



# Issues of CNN and RNN

CNN: **local** composition：

unfriendly to long-term/global token interaction

RNN: **recurrent** composition

what if we forget tokens checked 10 timestamp ago?

# Long-term token interaction

**上下文**：随后他立即想到自己为什么如此气愤——他之所以气愤，是因为他害怕了。在他个人处于巨大危险的情况下，贝思抛弃了他。在海底深处只剩下他们三个人，他们互相需要——他们得互相依靠。
**目标句**：然而贝思不可信赖，这使他感到害怕，而且
**目标词**：气愤

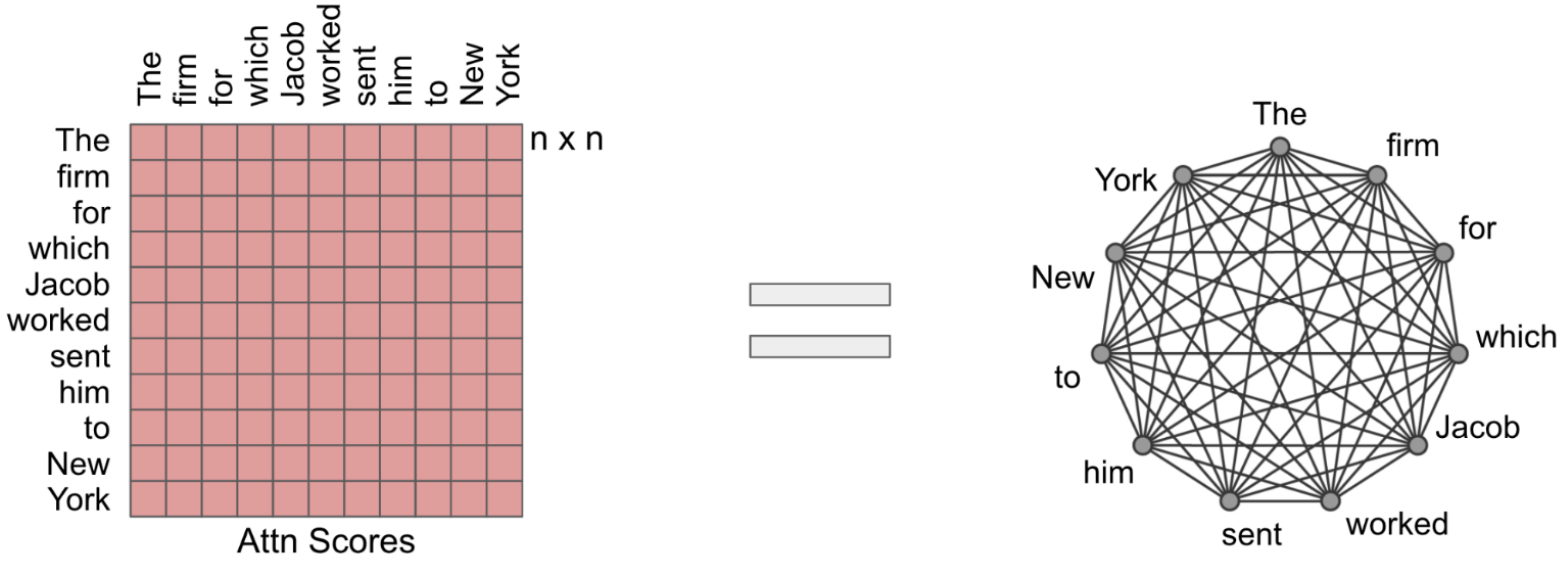
**上下文**：公主喜欢出宫打猎，雷章采便和公主从中原武功谈到中土风物，不但显示风流文采，还极尽体贴周到，公主就对他心生好感了。与此同时，他又物色了一个美貌妓女去勾引准驸马，让公主亲眼见了准驸马的下流丑态。
**目标句**：公主一怒之下，回宫禀告，就此废了这个
**目标词**：驸马

Huibin Ge, Chenxi Sun, Deyi Xiong, Qun Liu. Chinese WPLC: A Chinese Dataset for Evaluating Pretrained Language Models on Word Prediction **Given Long-Range Context**. https://aclanthology.org/2021.emnlp-main.306/

# How can we freely compose tokens without constraints (weaker inductive bias) ?

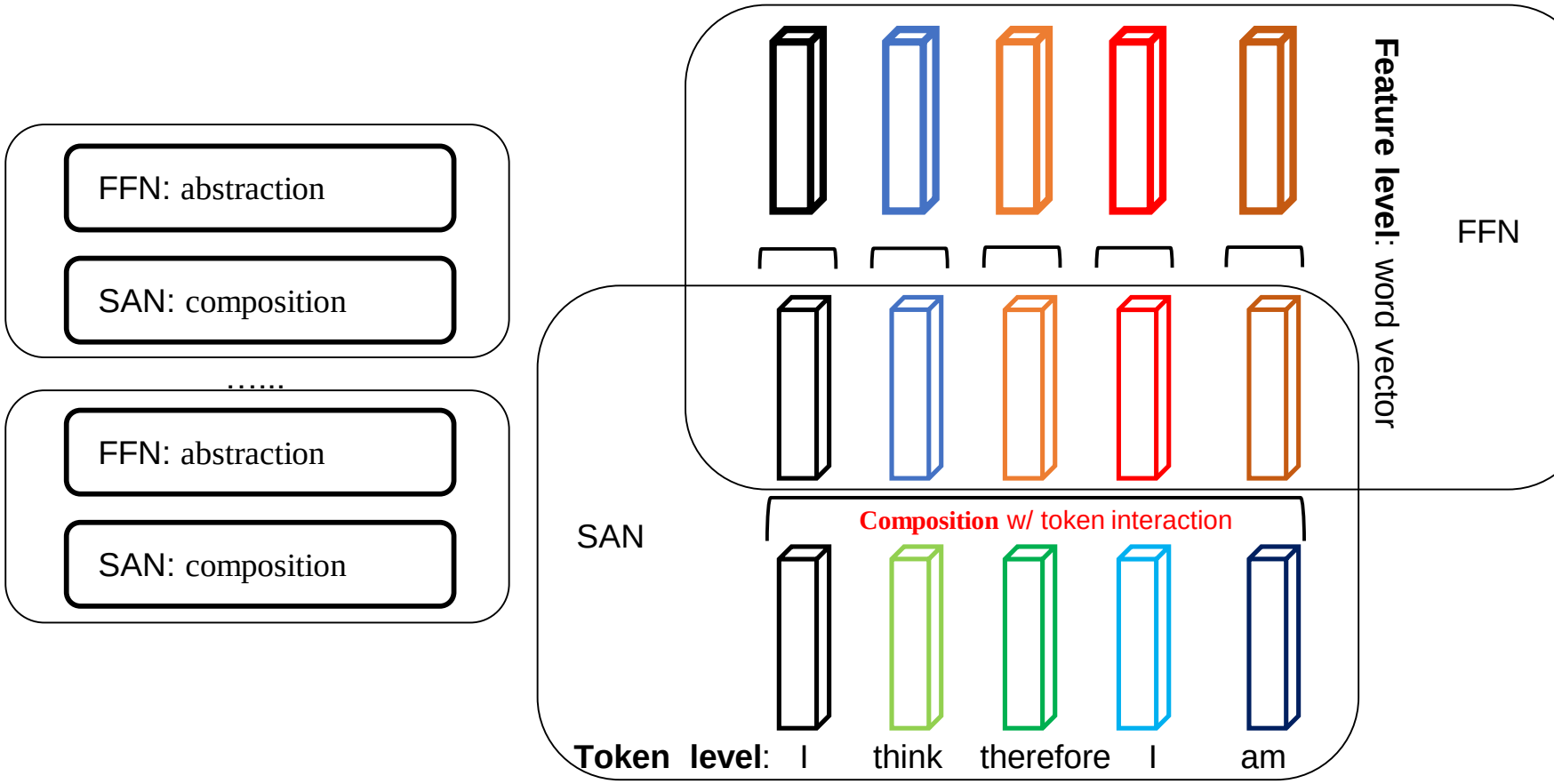
The modern deep learning is just using weaker inductive biases and make more data-driven instead of prior-driven.

# Make each token to see every other token




Difference between BERT and GPT?

# Efficiency: Decompose abstraction and composition



# Today's lecture

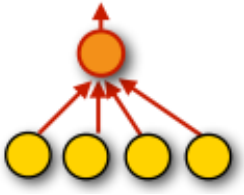
- **MLP**
  +: Strongest inductive bias: if all words are concatenated
  +: Weakest inductive bias: if all words are averaged
  - : The interaction at the token-level is too weak
- CNN & RNN
  +: The interaction at the token-level is slightly better.
    CNN: Bringing the global token-level interaction to the window-level
      - : Make simplifications, its global dependencies are limited
    RNN: An ideal method for processing token sequences
      - : Its recursive nature has the problem of disaster forgetting.
- Transformer
  +: Achieve **global dependence** at the **token-level** by **decoupling** token-level interaction and feature-level abstraction into two components, in **SAN** and **FNN**.

- Scaling law and emergent ability

# Multilayer Perceptron (MLP)

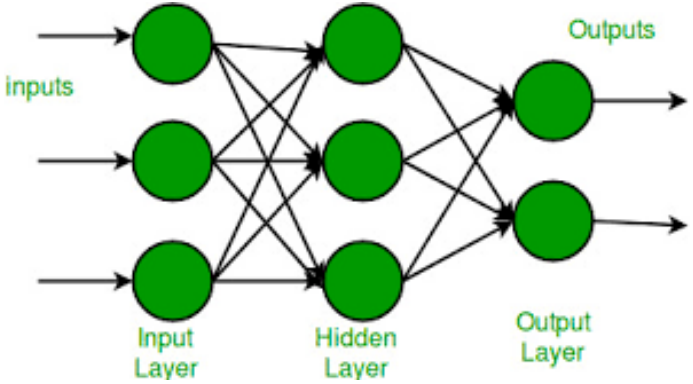
**Definition:** The Multilayer Perceptron (MLP) is a type of artificial neural network (ANN) that consists of multiple layers of interconnected artificial neurons or perceptrons.

A **perceptron** can be seen as a single neuron
(one output unit with a vector or **layer** of input units):



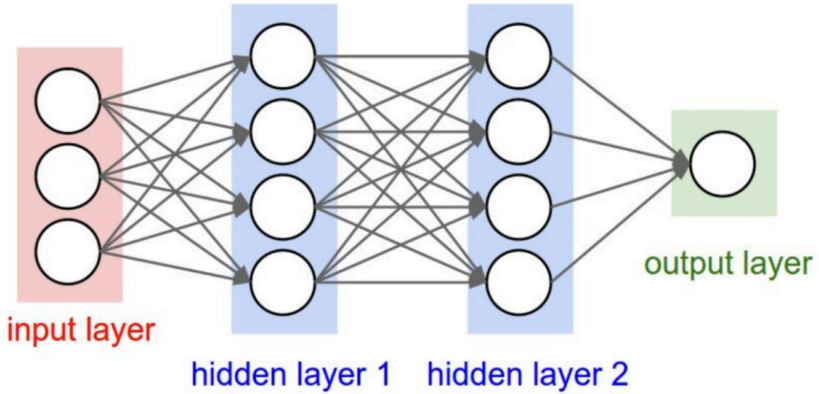
**Output unit:** scalar $y = f(\mathbf{x})$

**Input layer:** vector $\mathbf{x}$



# Feed-forward NNs

- The units are connected with no cycles
- The outputs from units in each layer are passed to units in the next higher layer. No outputs are passed back to lower layers




**Fully-connected (FC) layers:**

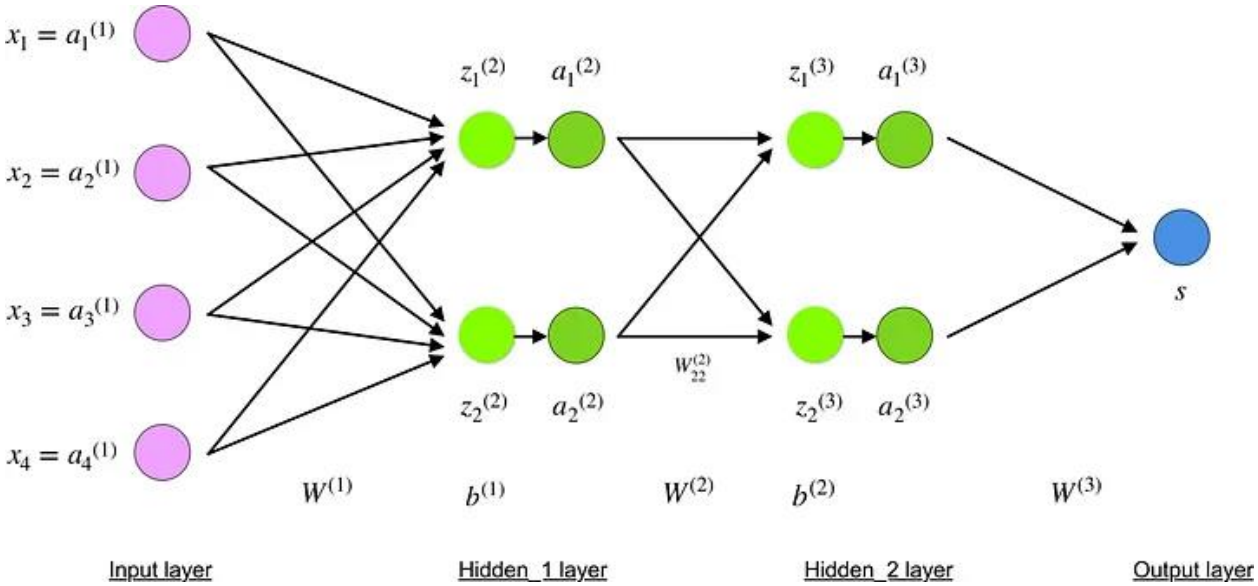
All the units from one layer are fully connected to every unit of the next layer.

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

# Backpropagation

**Definition:**

Backpropagation, short for "backward propagation of errors," is a supervised learning algorithm used for training artificial neural networks, including deep learning models like Multilayer Perceptrons (MLPs).




https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

# Common Challenges in Backward Propagation

- Vanishing Gradients
- Exploding Gradient
- Overfitting
- Local Minima
- Gradient Descent Variants
- Training Time
- Poor Initialization

**Summary:**
- Backward propagration is a critical but challenging step in training neural networks
- Addressing these issues requires a combination of architectural choices, optimization techniques, and regularization methods.

# Feedforward neural language models

**A Neural Probabilistic Language Model**

(Bengio et al., 2003)


**Yoshua Bengio** BENGIOY@IRO.UMONTREAL.CA
**Réjean Ducharme** DUCHARME@IRO.UMONTREAL.CA
**Pascal Vincent** VINCENTP@IRO.UMONTREAL.CA
**Christian Jauvin** JAUVINC@IRO.UMONTREAL.CA


**Yoshua Bengio**

Probabilistic models of sequences: In the 1990s, Bengio combined neural networks with probabilistic models of sequences, such as hidden Markov models. These ideas were incorporated into a system used by AT&T/NCR for reading handwritten checks, were considered a pinnacle of neural network research in the 1990s, and modern deep learning speech recognition systems are extending these concepts.

High-dimensional word embeddings and attention: In 2000, Bengio authored the landmark paper, "A Neural Probabilistic Language Model," that introduced high-dimension word embeddings as a representation of word meaning. Bengio's insights had a huge and lasting impact on natural language processing tasks including language translation, question answering, and visual question answering. His group also introduced a form of attention mechanism which led to breakthroughs in machine translation and form a key component of sequential processing with deep learning.

Generative adversarial networks: Since 2010, Bengio's papers on generative deep learning, in particular the Generative Adversarial Networks (GANs) developed with Ian Goodfellow, have spawned a revolution in computer vision and computer graphics. In one fascinating application of this work, computers can actually create original images, reminiscent of the creativity that is considered a hallmark of human intelligence.

https://awards.acm.org/about/2018-turing

# Feedforward neural language models

**A Neural Probabilistic Language Model** (Bengio et al., 2003)


**Yoshua Bengio** BENGIOY@IRO.UMONTREAL.CA
**Réjean Ducharme** DUCHARME@IRO.UMONTREAL.CA
**Pascal Vincent** VINCENTP@IRO.UMONTREAL.CA
**Christian Jauvin** JAUVINC@IRO.UMONTREAL.CA


Key idea: Instead of estimating raw probabilities, let's use a

**neural network** to fit the **probabilistic distribution of language**!

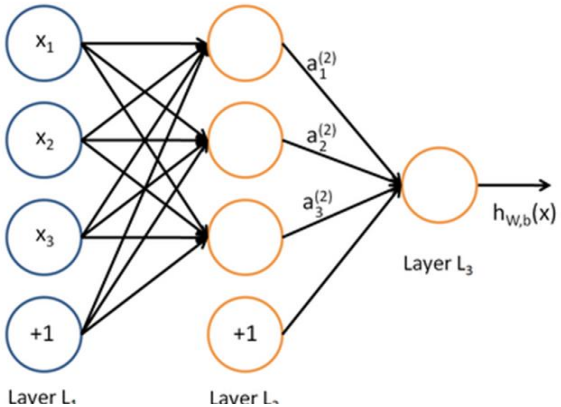
$P(w \mid \text{I am a good})$ $P(w \mid \text{I am a great})$

**Key ingredient**: word embeddings **e(good) ≈ e(great)**

Hope: this would give us similar distributions for similar contexts!

# Review: MLP

1. Brief introduction of MLP;
2. Forward propagation and backward propagation;



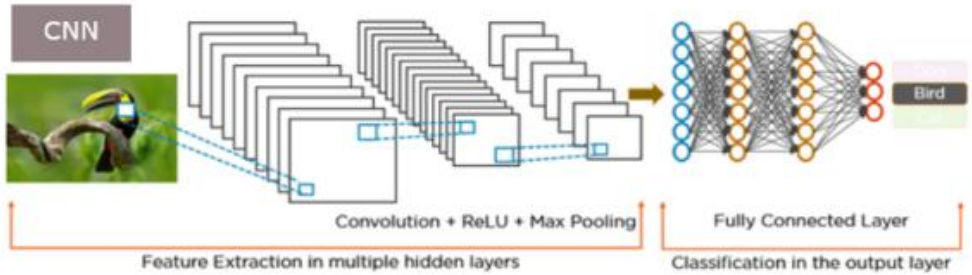

**Limitations of MLP:**

1. Limited Spatial Invariance (vs. CNNs)
2. Sequential Information Handling (vs. RNNs)
3. Positional Encoding (vs. Transformers)
4. Attention Mechanism (vs. Transformers)
5. Hierarchical Feature Extraction (vs. CNNs and Transformers)
6. Parameter Efficiency (vs. Transformers)
7. Pre-training Efficiency (vs. Transformers)
8. Structured Input Bias (vs. CNNs and Transformers)

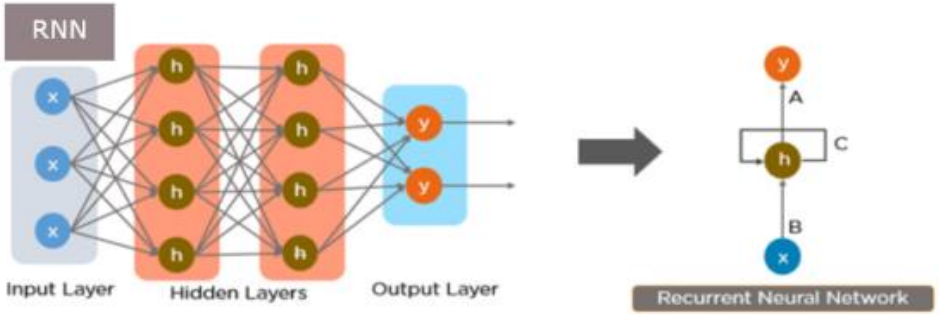
# CNN&RNN

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)

**Convolutional Neural Network**




**Recurrent Neural Network**

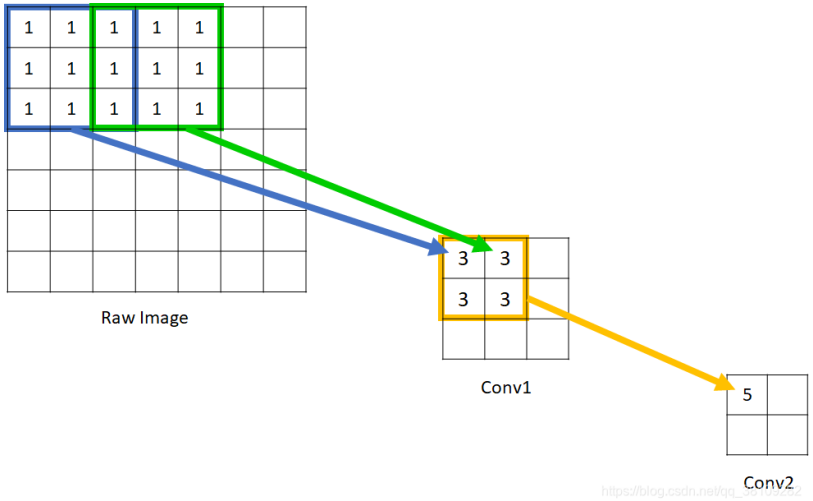


# Today's lecture

- MLP
  +: Strongest inductive bias: if all words are concated
  +: Weakest inductive bias: if all words are averaged
  - : The interaction at the token-level is too weak
- **CNN & RNN**
  +: The interaction at the token-level is slightly better.
    CNN: Bringing the global token-level interaction to the window-level
      - : Make simplifications, its global dependencies are limited
    RNN: An ideal method for processing token sequences
      - : Its recursive nature has the problem of disaster forgetting.
- Transformer
  +: Achieve **global dependence** at the **token-level** by **decoupling** token-level interaction and feature-level abstraction into two components, in **SAN** and **FNN**.

- Scaling law and emergent ability

# **Local** receptive field

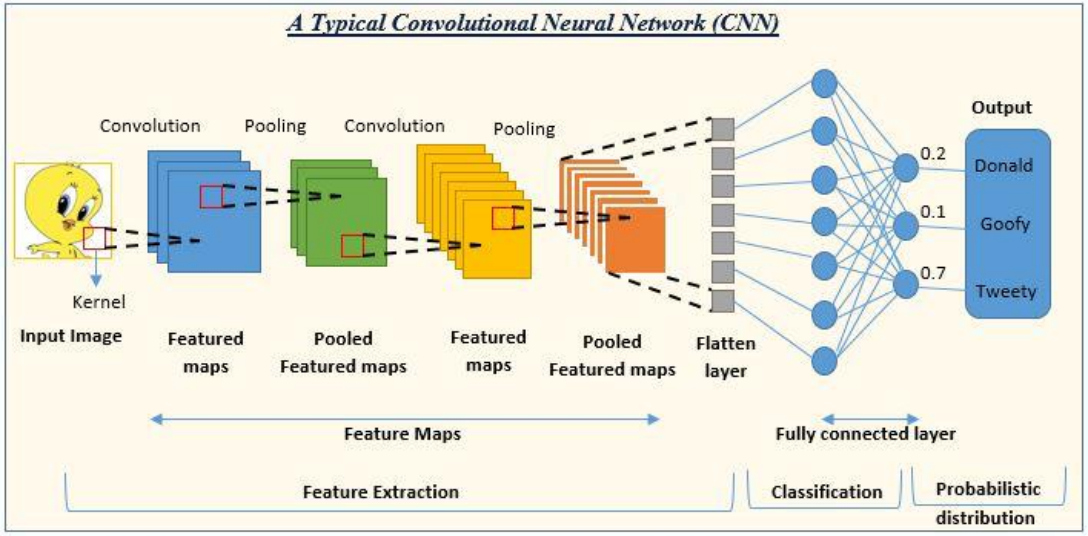



At each time， it only perceives part of the input.

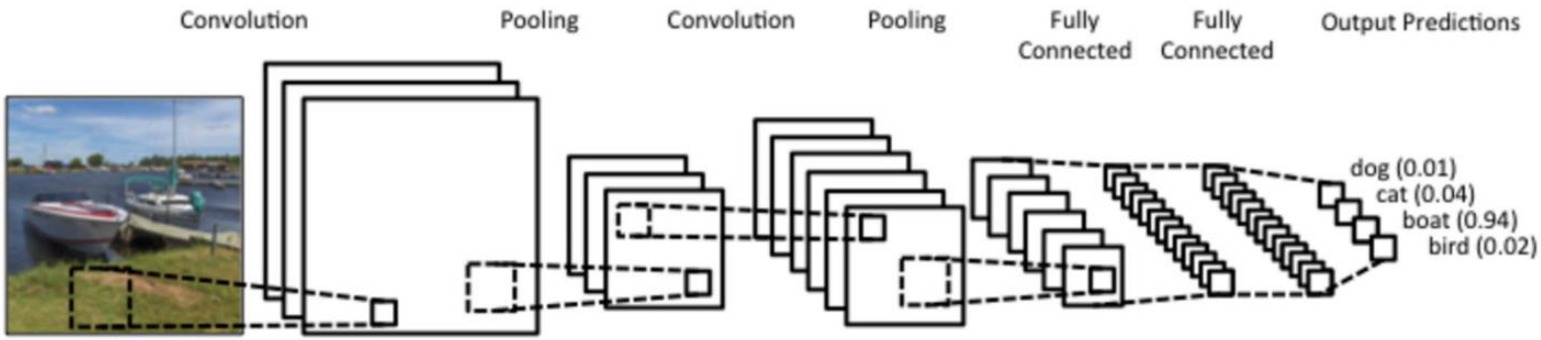
# CNN

## Convolutional Neural Network

- What is CNN?
- Motivation: Image Processing
- Key Components
  - Convolutional Layers
  - Pooling Layers
  - Fully Connected Layers
- Hierarchical Feature Extraction

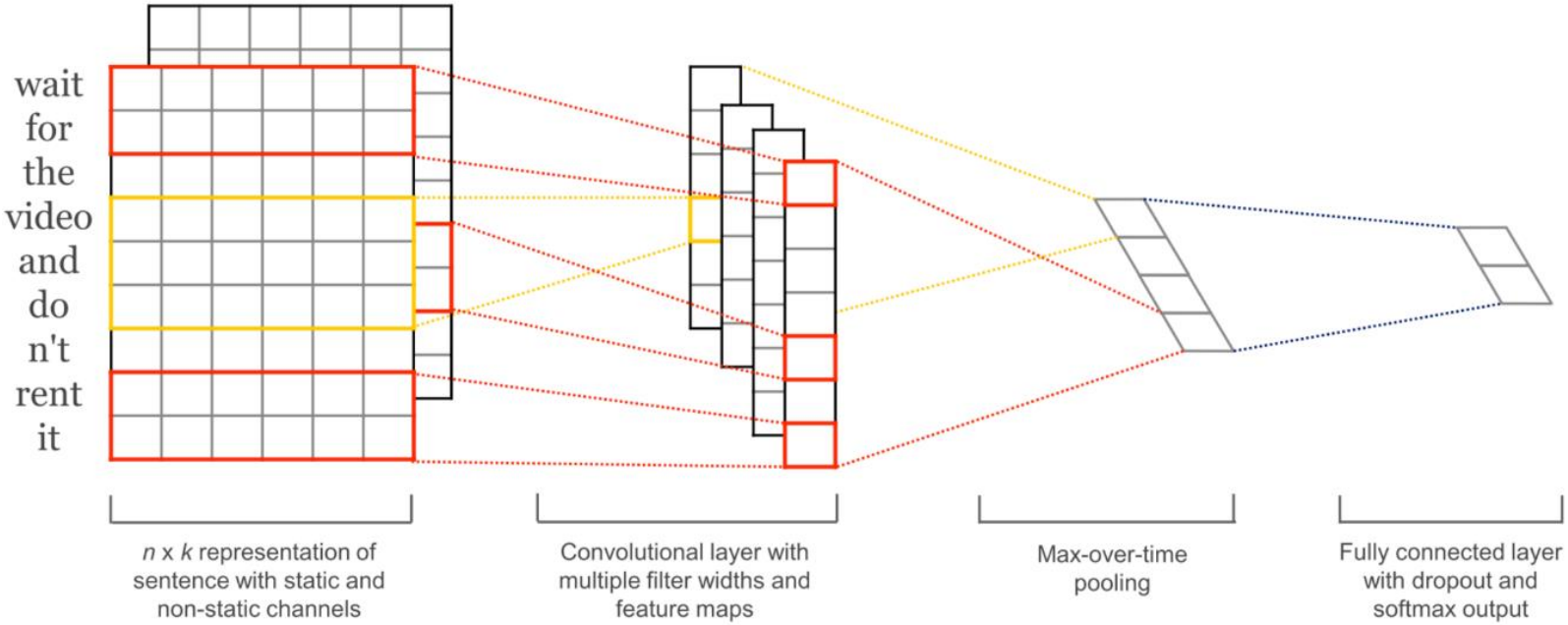


# Convolutional NNs in image classification



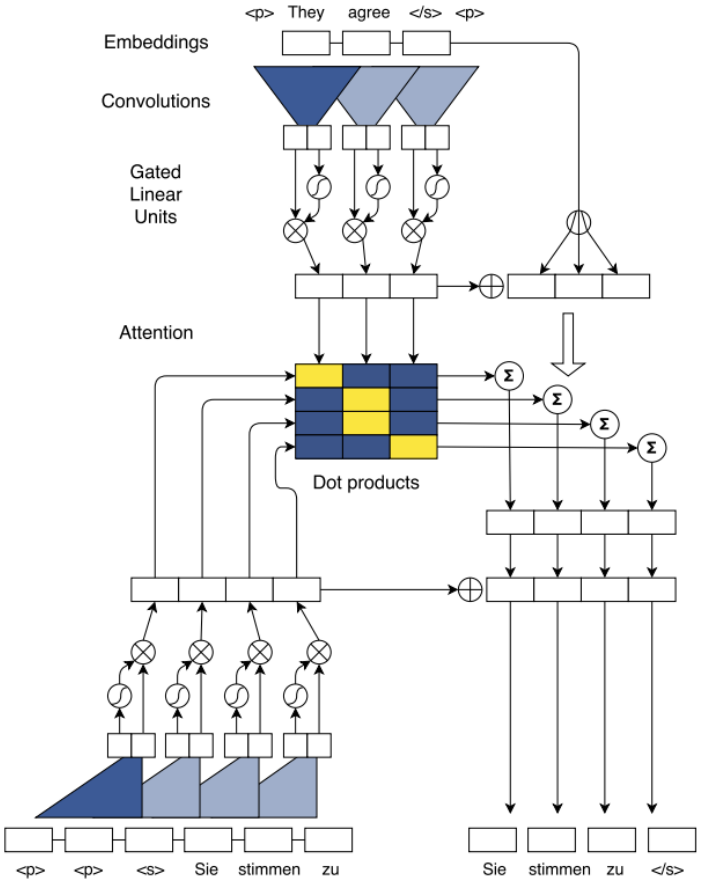

Key components: 1) convolution; 2) pooling; 3) multiple channels (feature maps)

# Convolutional NNs for text classification




(Kim 2014): Convolutional Neural Networks for Sentence Classification

# Convolutional Sequence to Sequence Learning



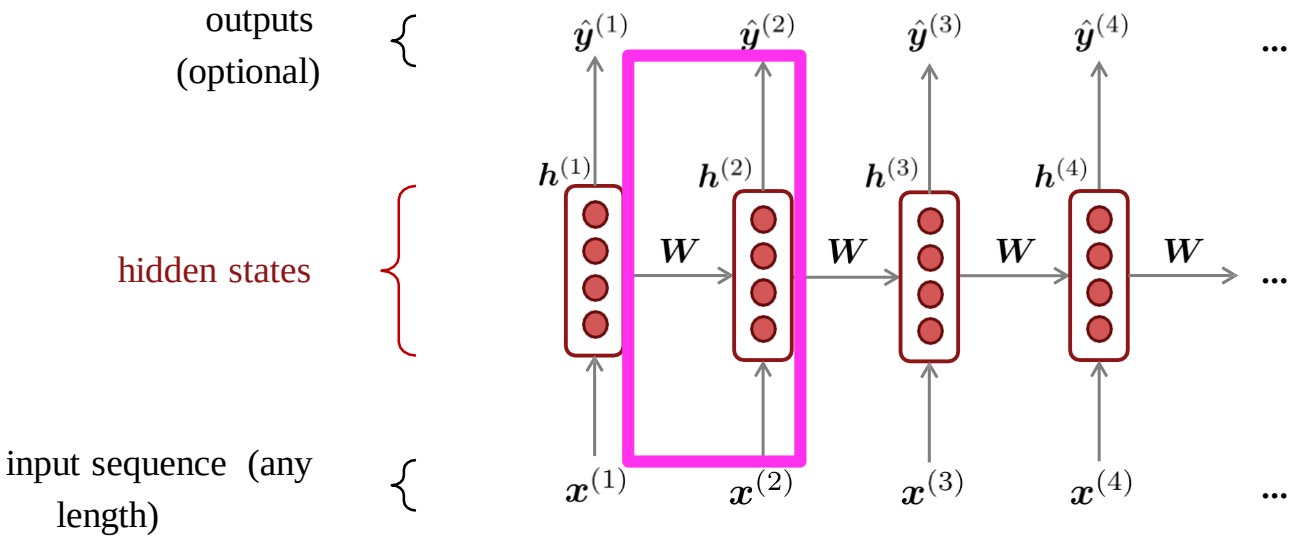

- Encoder and decoder are simple blocks of convolution operation followed by nonlinearity on fixed size of input.
- Introduce a concept of order preservation as a positional vectors p = (p_1,p_2 ...,p_m). In combination of both input elements are represented as E = (e_1=w_1+p_1, e_2=w_2+p_2, ....,e_m=w_m+p_m).
- Adds a linear mapping to project between the embedding size f and the convolution outputs that are size 2d.
- Computes a distribution over the T possible next target elements y_i+1 by transforming the top decoder output h_i_l via a linear layer with weights and bias.

https://browse.arxiv.org/pdf/1705.03122.pdf

# RNN

## Recurrent Neural Network

**Core idea:** Apply the same weights *W repeatedly*



# A Simple RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

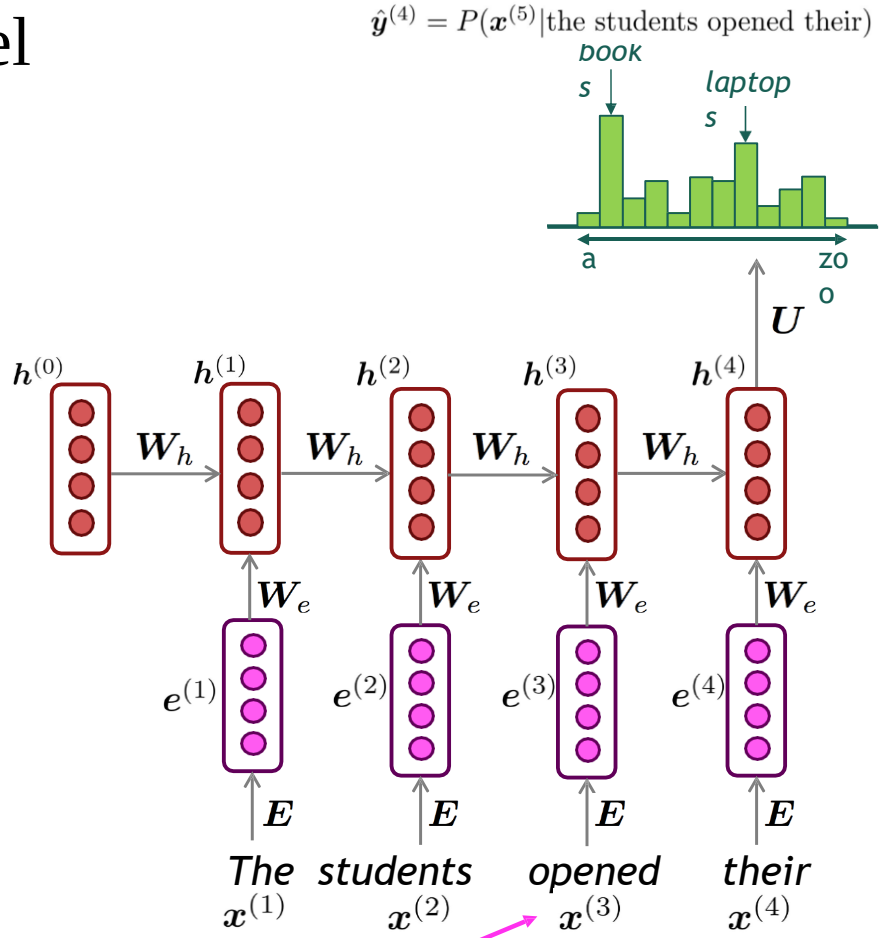
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$




***Note**: this input sequence could be much longer now!*

# RNN Language Models

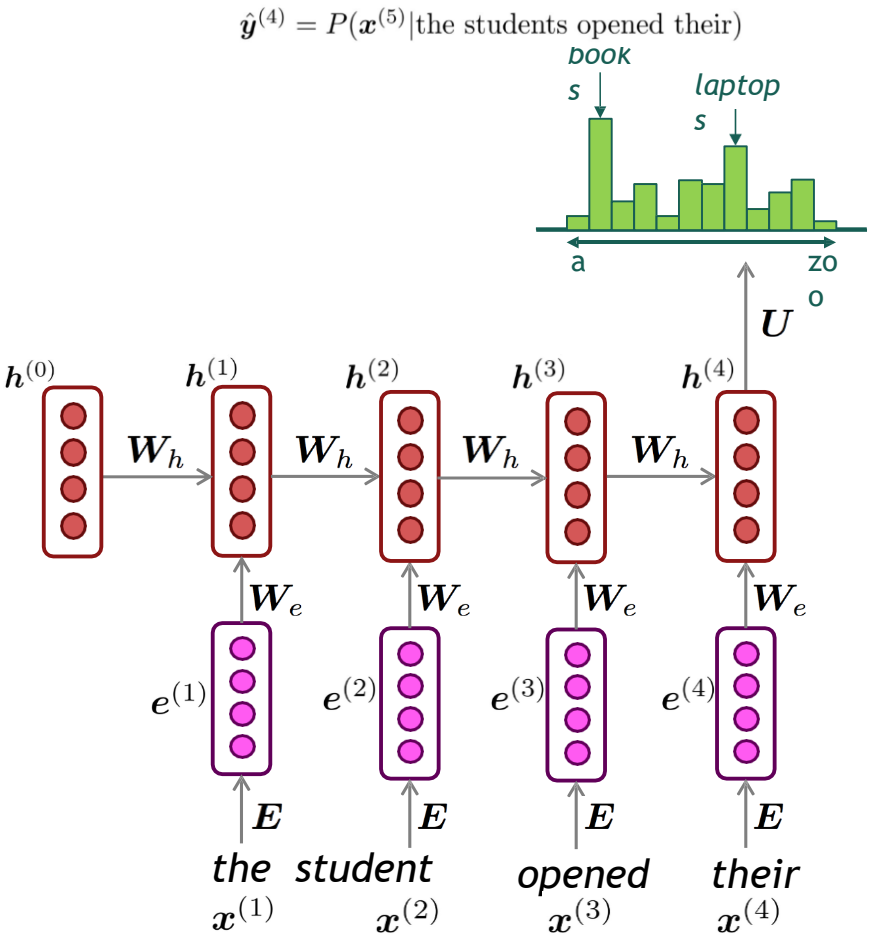
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

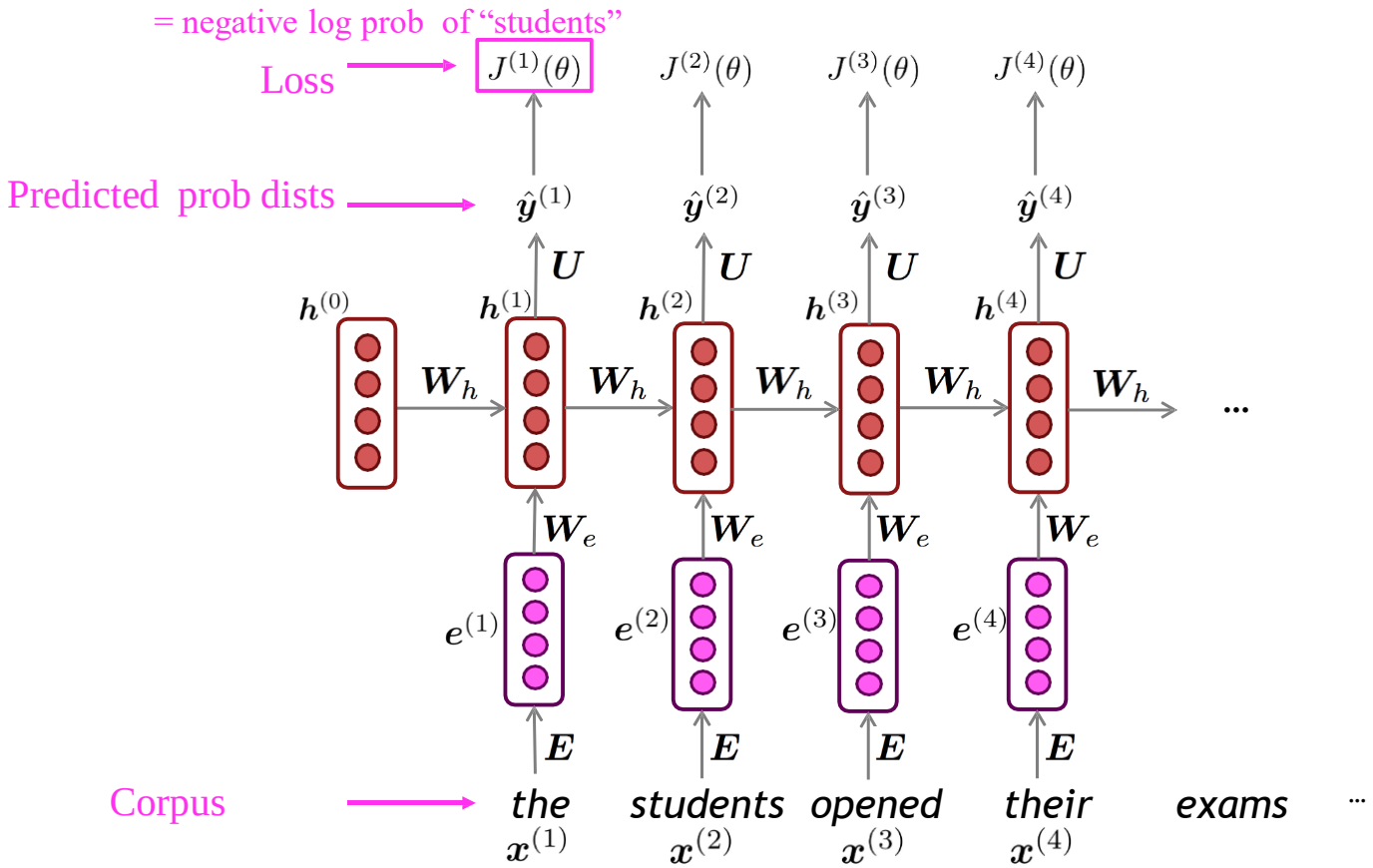
RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later

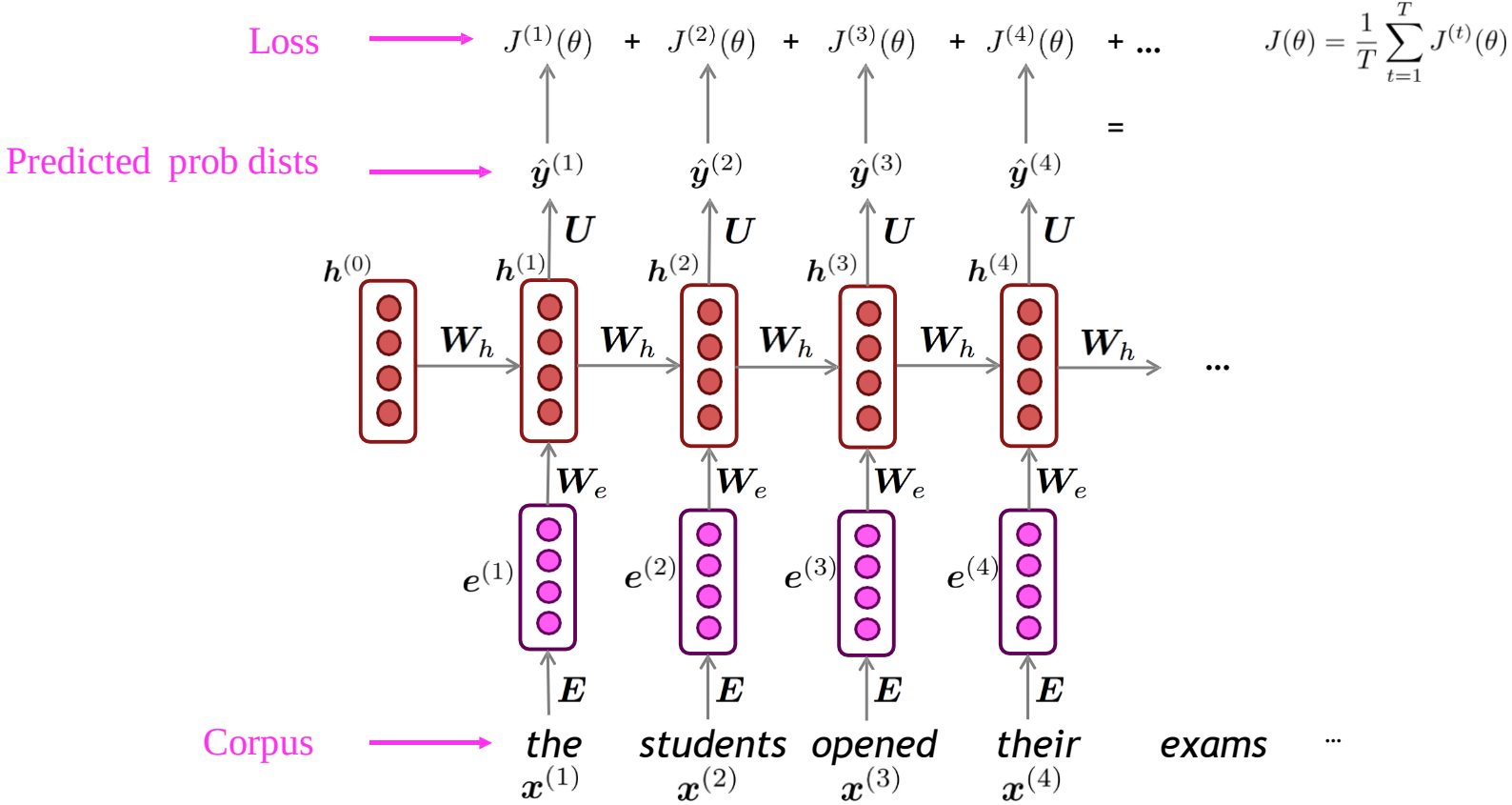


# Training an RNN Language Model

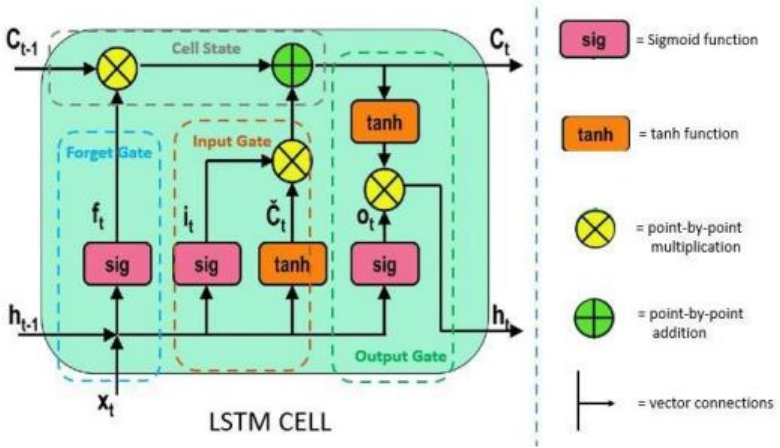


# Training an RNN Language Model

"Teacher forcing"



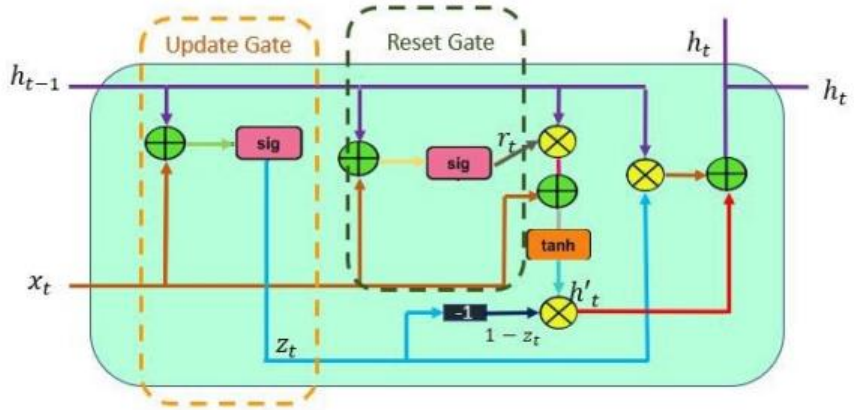
# Special case of RNN - LSTM




**Forget Gate**: – This gate decides what information should be carried out forward or what information should be ignored. Information from previous hidden states and the current state information passes through the sigmoid function. Values that come out from sigmoid are always between 0 and 1. – if the value is closer to 1 means information should proceed forward and if value closer to 0 means information should be ignored.
**Input Gate:** – After deciding the relevant information, the information goes to the input gate, Input gate passes the relevant information, and this leads to updating the cell states. simply saving updating the weight. Input gate adds the new relevant information to the existing information by updating cell states.
**Output Gate**: – After the information is passed through the input gate, now the output gate comes into play. – Output gate generates the next hidden states. and cell states are carried over the next time step.

https://mitu.co.in/wp-content/uploads/2022/12/8.-GRU-and-LSTM.pdf

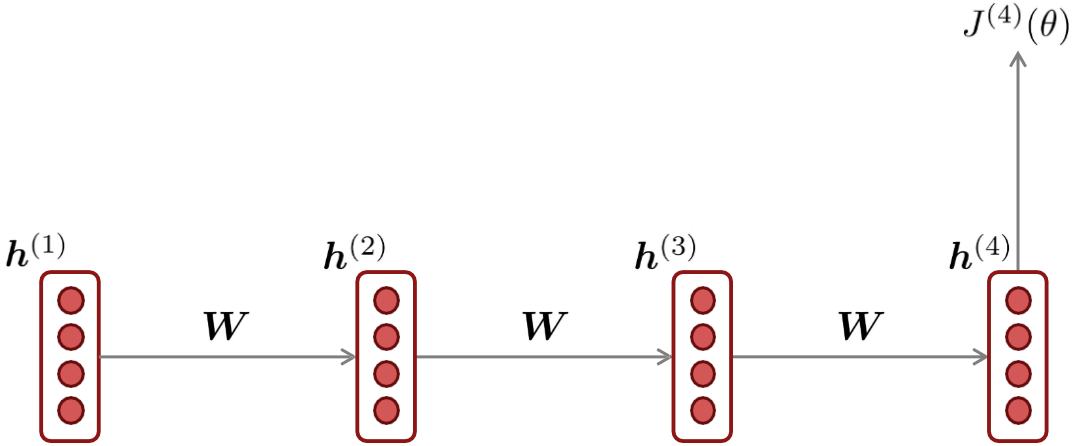
# Special case of RNN - GRU



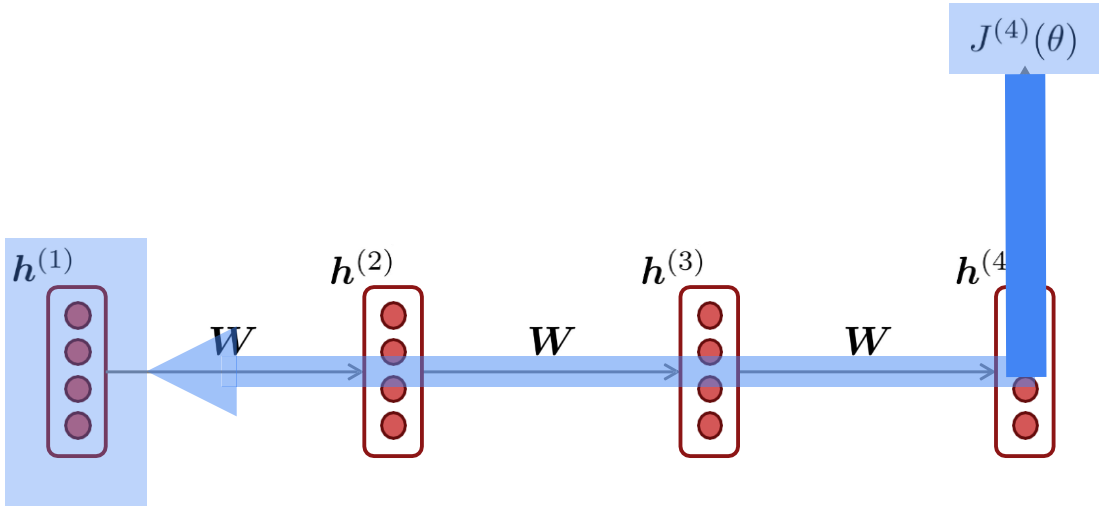

GRU ( Gated Recurrent Units ) are similar to the LSTM networks. GRU is a kind of newer version of RNN. However, there are some differences between GRU and LSTM.

– GRU doesn't contain a cell state
– GRU uses its hidden states to transport information
– It Contains only 2 gates(Reset and Update Gate)
– GRU is faster than LSTM – GRU has lesser tensor's operation that makes it faster

# Problems with RNNs: Vanishing and Exploding Gradients

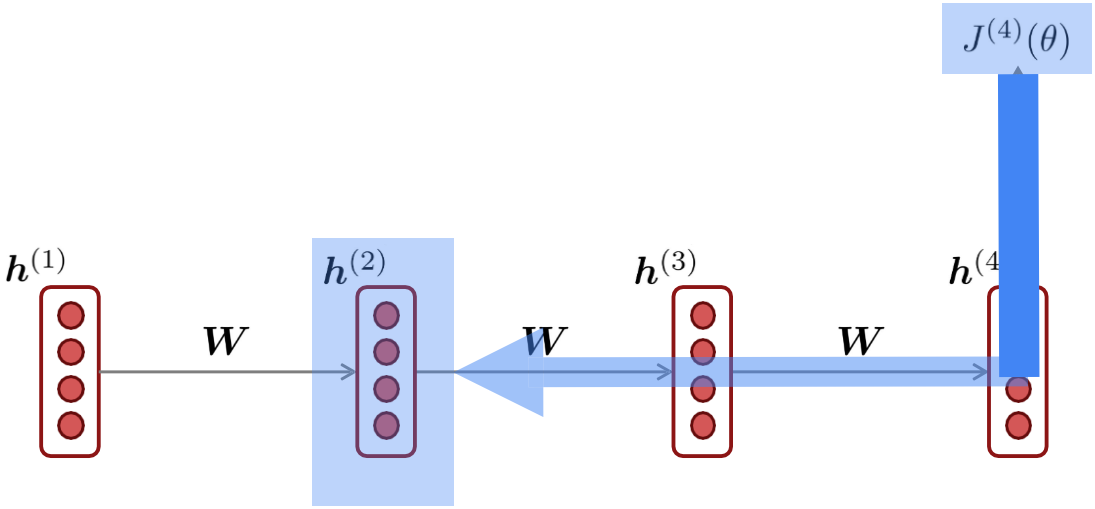


# Vanishing gradient intuition




$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = ?$$

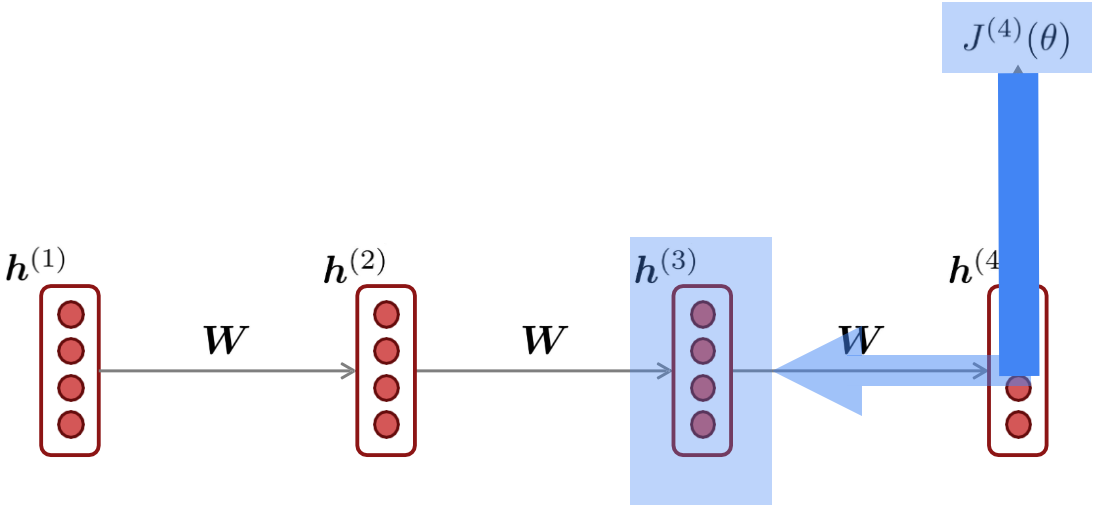
# Vanishing gradient intuition




$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(2)}}$$

chain rule!

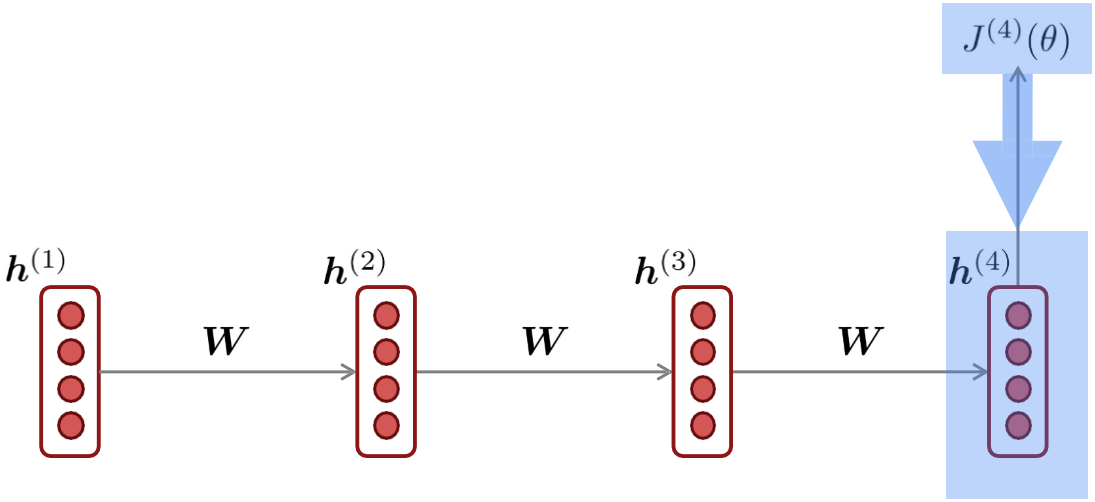
# Vanishing gradient intuition




$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

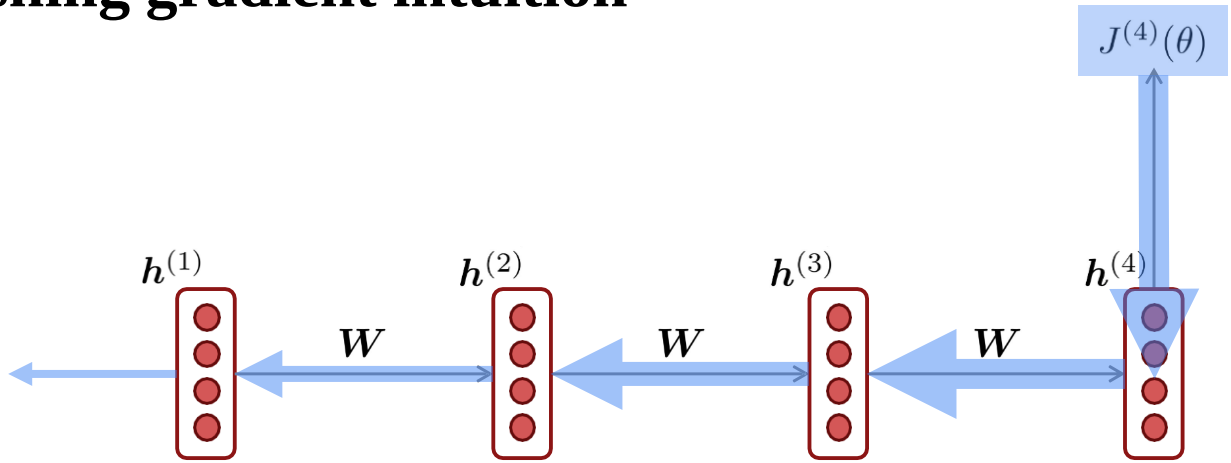
# Vanishing gradient intuition




$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

chain rule!

# Vanishing gradient intuition




$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

**Vanishing gradient problem:**
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

# Is vanishing/exploding gradient just an RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (i.e., are hard to train)
- Another solution: lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

For example:

- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

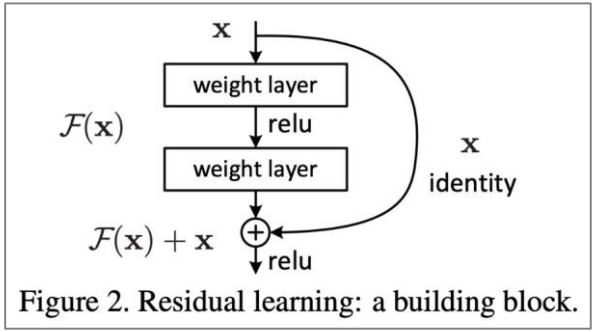


Figure 2. Residual learning: a building block.

"Deep Residual Learning for Image Recognition", He et al, 2015.
https://arxiv.org/pdf/1512.03385.pdf

# CNN vs. RNN

| | | | performance | lr | hidden | batch | sentLen | filter_size | margin |
|---|---|---|---|---|---|---|---|---|---|
| TextC | SentiC (acc) | CNN | 82.38 | 0.2 | 20 | 5 | 60 | 3 | – |
| | | GRU | **86.32** | 0.1 | 30 | 50 | 60 | – | – |
| | | LSTM | 84.51 | 0.2 | 20 | 40 | 60 | – | – |
| | RC (F1) | CNN | 68.02 | 0.12 | 70 | 10 | 20 | 3 | – |
| | | GRU | **68.56** | 0.12 | 80 | 100 | 20 | – | – |
| | | LSTM | 66.45 | 0.1 | 80 | 20 | 20 | – | – |
| SemMatch | TE (acc) | CNN | 77.13 | 0.1 | 70 | 50 | 50 | 3 | – |
| | | GRU | **78.78** | 0.1 | 50 | 80 | 65 | – | – |
| | | LSTM | 77.85 | 0.1 | 80 | 50 | 50 | – | – |
| | AS (MAP & MRR) | CNN | **(63.69,65.01)** | 0.01 | 30 | 60 | 40 | 3 | 0.3 |
| | | GRU | (62.58,63.59) | 0.1 | 80 | 150 | 40 | – | 0.3 |
| | | LSTM | (62.00,63.26) | 0.1 | 60 | 150 | 45 | – | 0.1 |
| | QRM (acc) | CNN | **71.50** | 0.125 | 400 | 50 | 17 | 5 | 0.01 |
| | | GRU | 69.80 | 1.0 | 400 | 50 | 17 | - | 0.01 |
| | | LSTM | 71.44 | 1.0 | 200 | 50 | 17 | - | 0.01 |
| SeqOrder | PQA (hit@10) | CNN | 54.42 | 0.01 | 250 | 50 | 5 | 3 | 0.4 |
| | | GRU | **55.67** | 0.1 | 250 | 50 | 5 | – | 0.3 |
| | | LSTM | 55.39 | 0.1 | 300 | 50 | 5 | – | 0.3 |
| ContextDep | POS tagging (acc) | CNN | 94.18 | 0.1 | 100 | 10 | 60 | 5 | – |
| | | GRU | 93.15 | 0.1 | 50 | 50 | 60 | – | – |
| | | LSTM | 93.18 | 0.1 | 200 | 70 | 60 | – | – |
| | | Bi-GRU | 94.26 | 0.1 | 50 | 50 | 60 | – | – |
| | | Bi-LSTM | **94.35** | 0.1 | 150 | 5 | 60 | – | – |

Table 1: Best results or CNN, GRU and LSTM in NLP tasks

https://arxiv.org/pdf/1702.01923

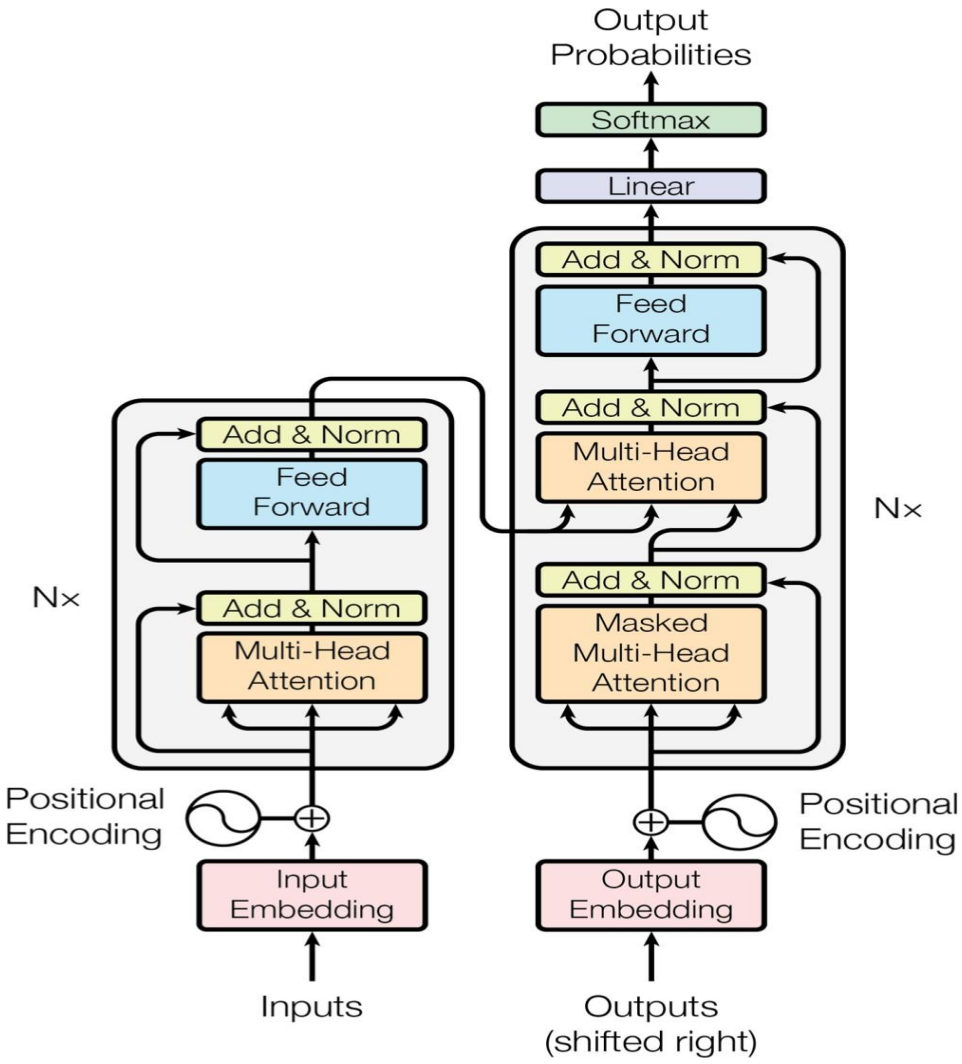
# Today's lecture

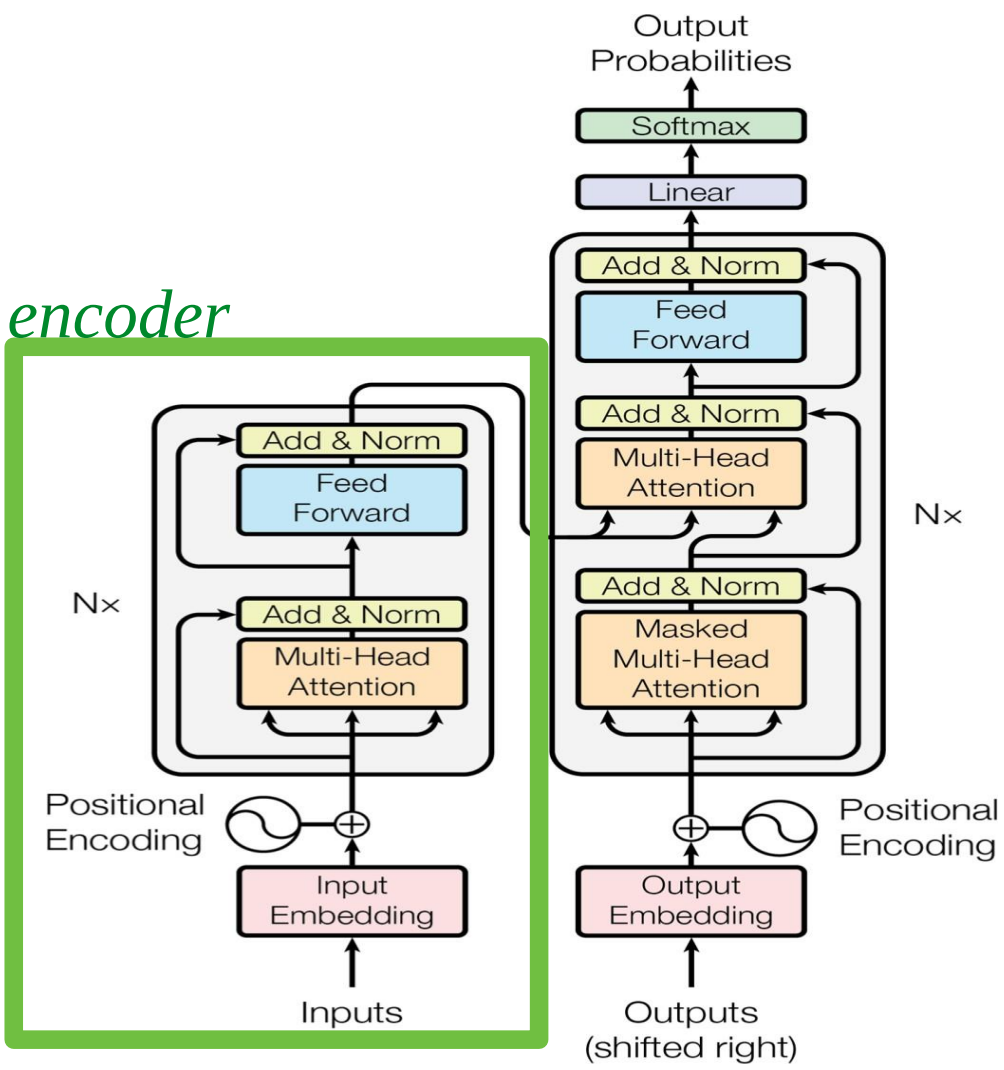
- MLP
  +: Strongest inductive bias: if all words are concated
  +: Weakest inductive bias: if all words are averaged
  - : The interaction at the token-level is too weak
- CNN & RNN
  +: The interaction at the token-level is slightly better.
    CNN: Bringing the global token-level interaction to the window-level
      - : Make simplifications, its global dependencies are limited
    RNN: An ideal method for processing token sequences
      - : Its recursive nature has the problem of disaster forgetting.
- **Transformer**
  +: Achieve **global dependence** at the **token-level** by **decoupling** token-level interaction and feature-level abstraction into two components, in **SAN** and **FNN**.

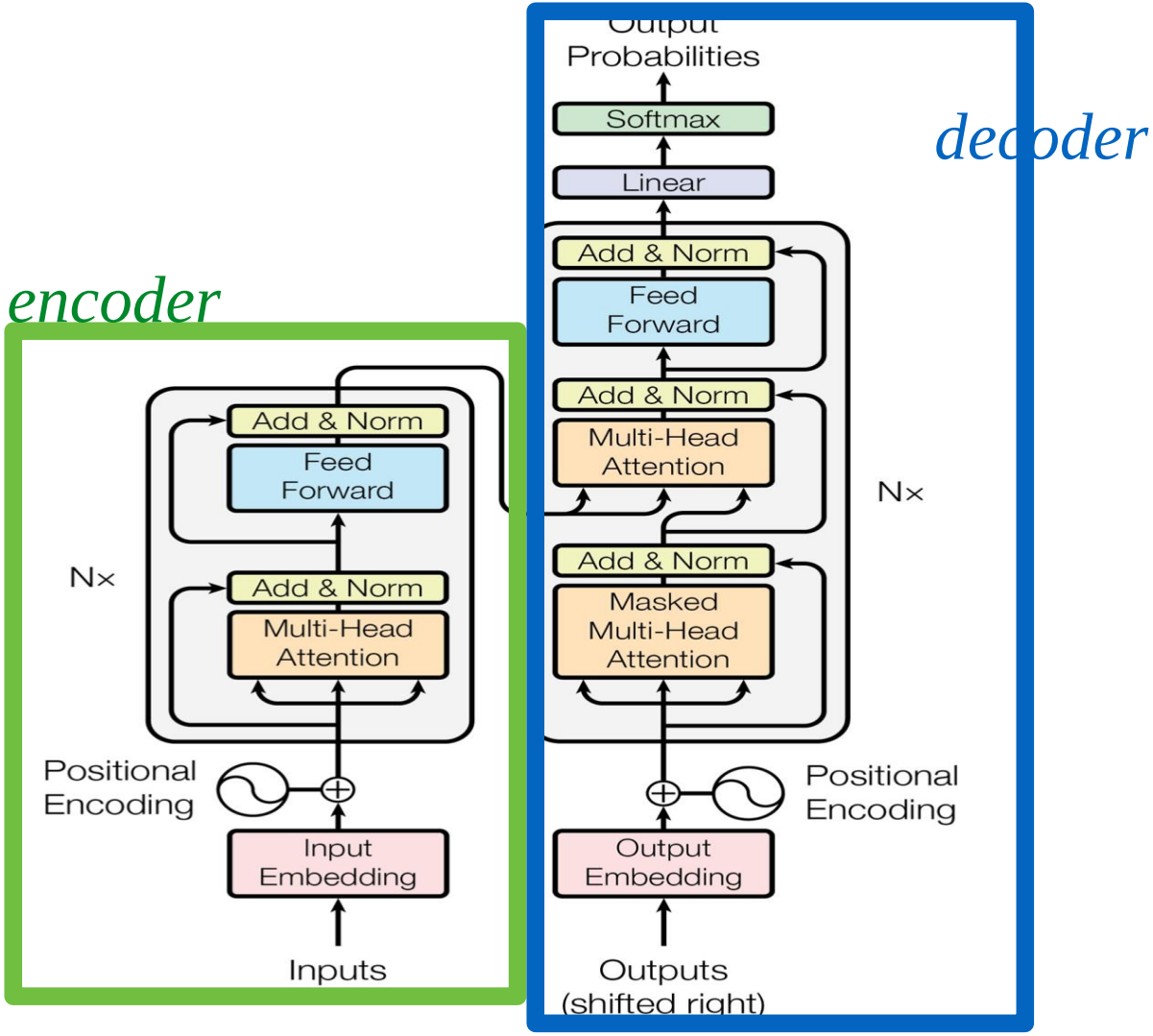
- Scaling law and emergent ability

# Transformer

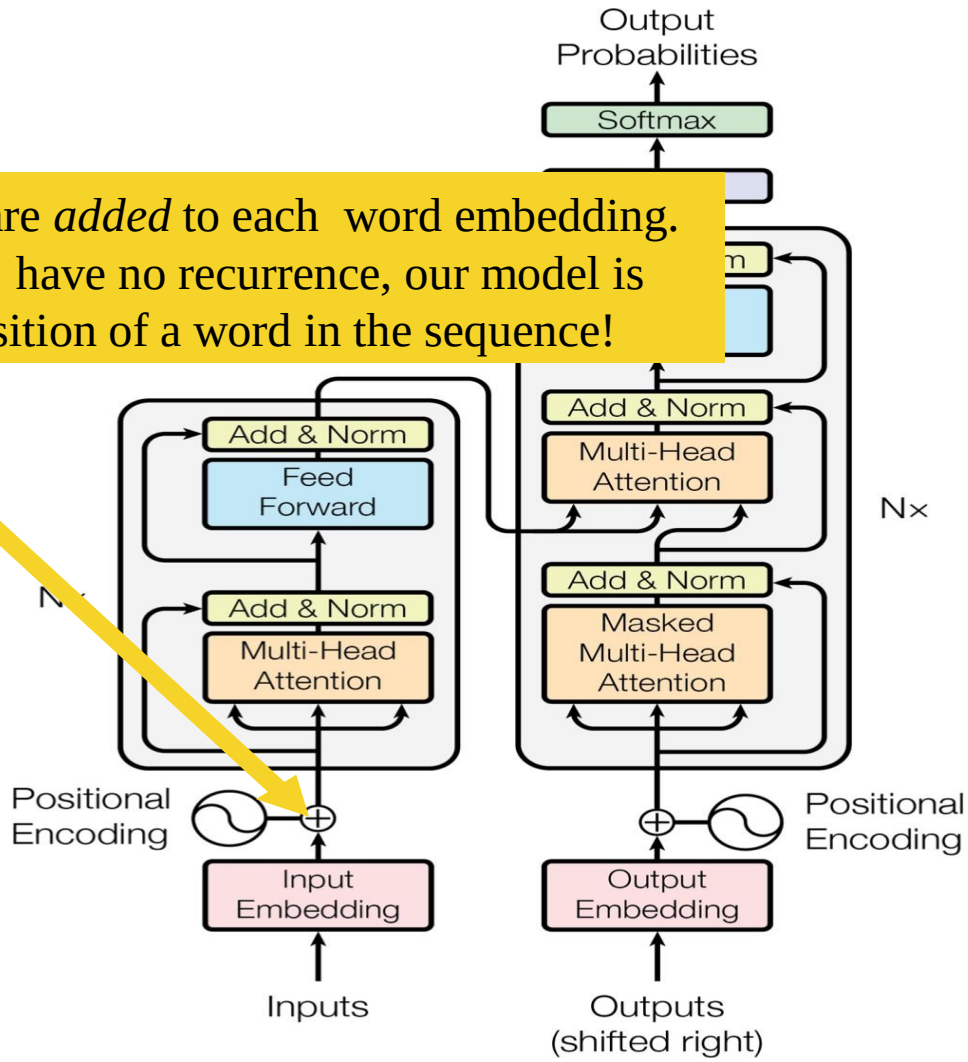
- **Encoder**
- **Decoder**
- **Self-attention**
- **Multi-head self-attention**
- **Positional Encoding**


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

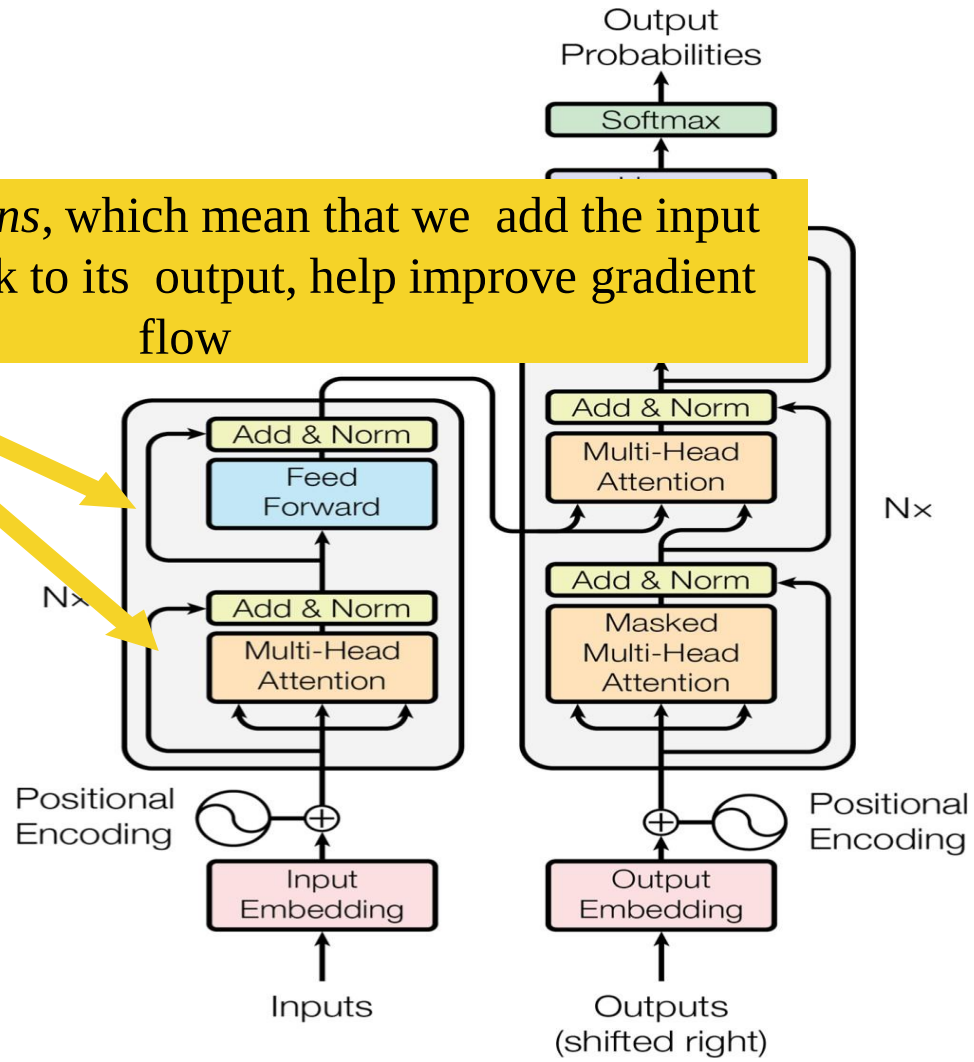

encoder
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs


encoder
decoder
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs

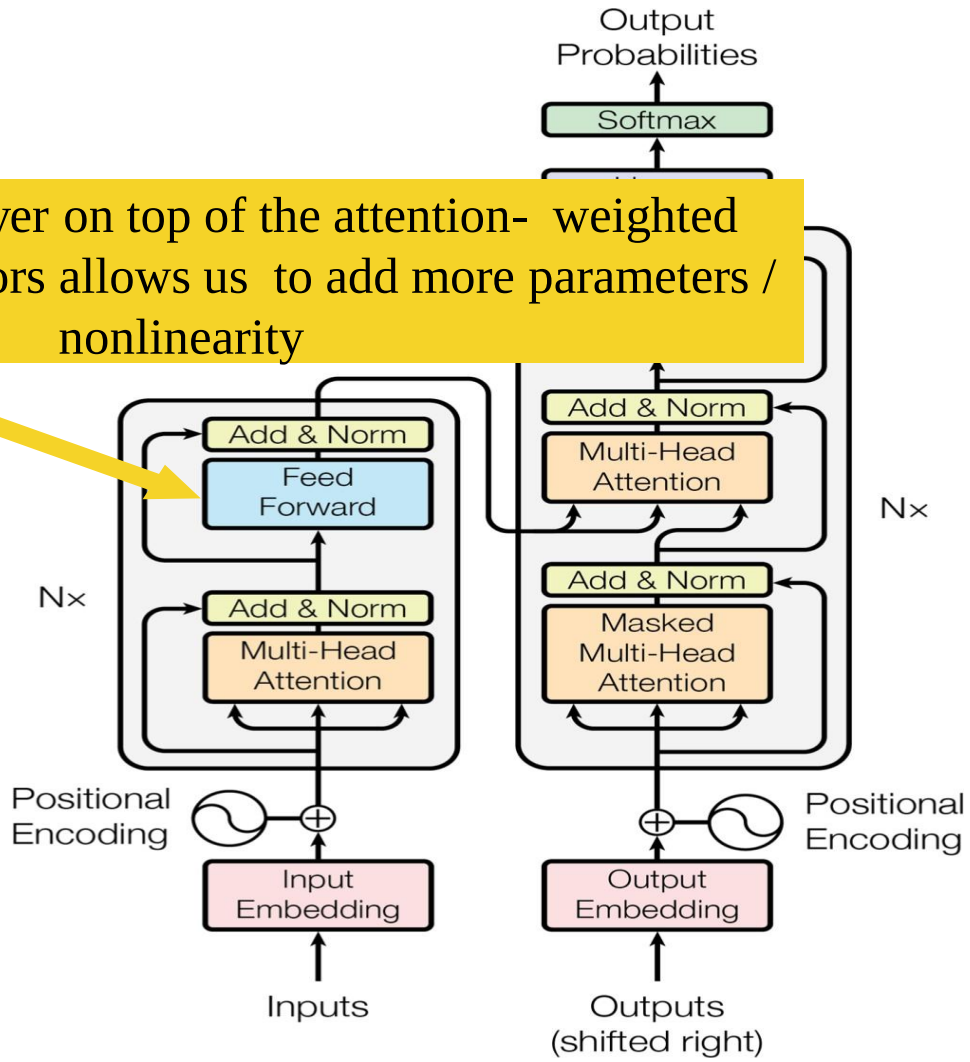
Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!



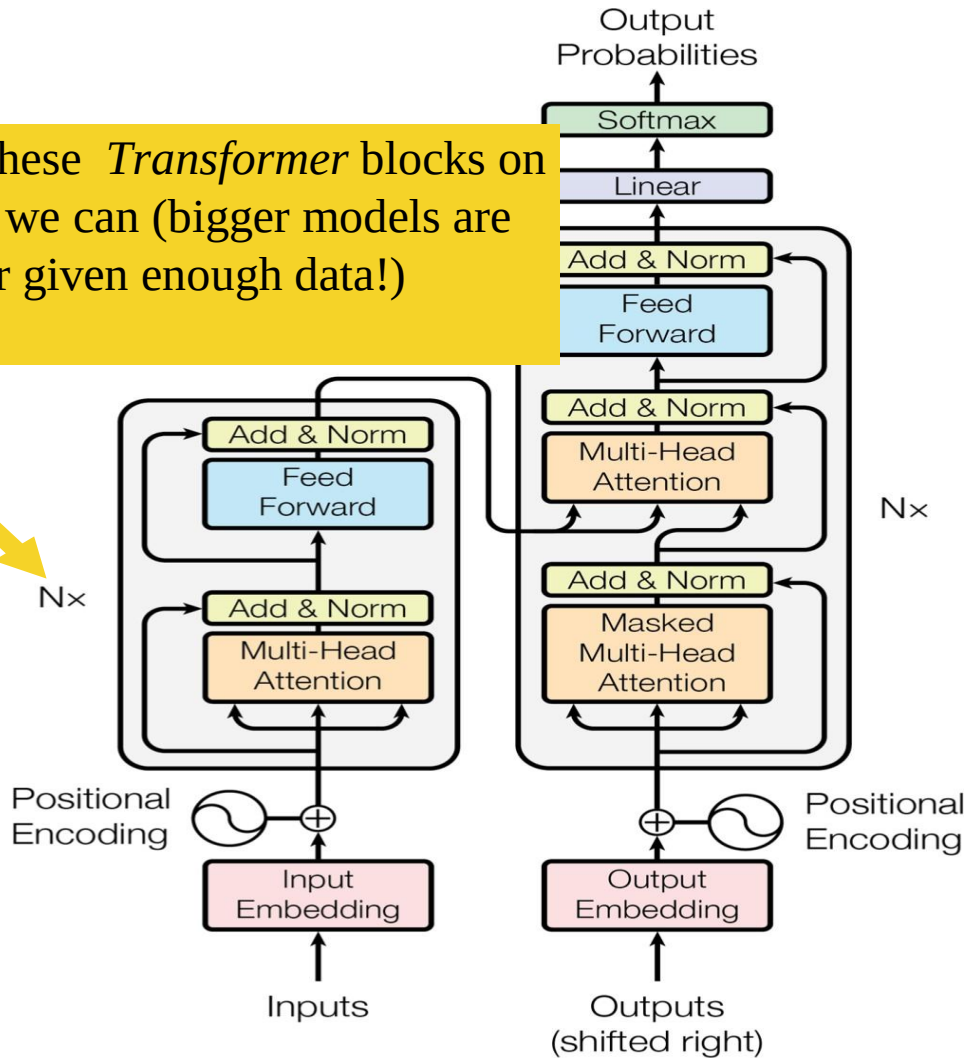
*Residual connections*, which mean that we add the input to a particular block to its output, help improve gradient flow



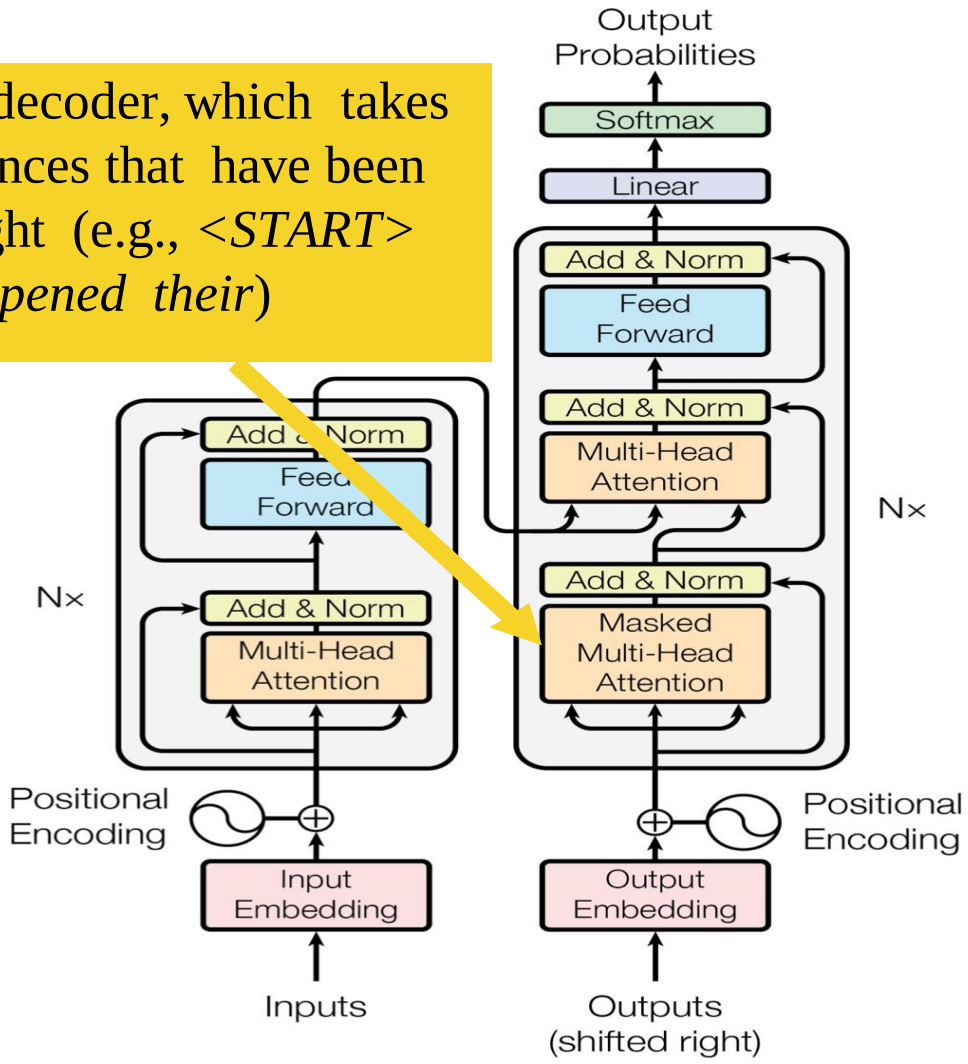
A feed-forward layer on top of the attention- weighted averaged value vectors allows us to add more parameters / nonlinearity



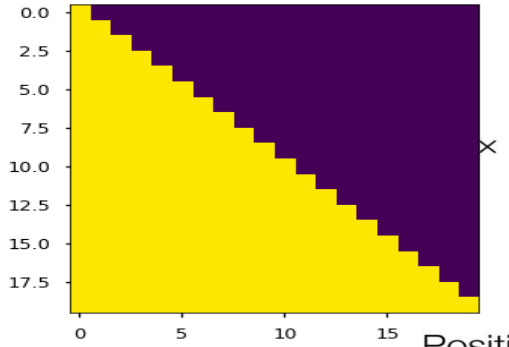
We stack as many of these *Transformer* blocks on top of each other as we can (bigger models are generally better given enough data!)

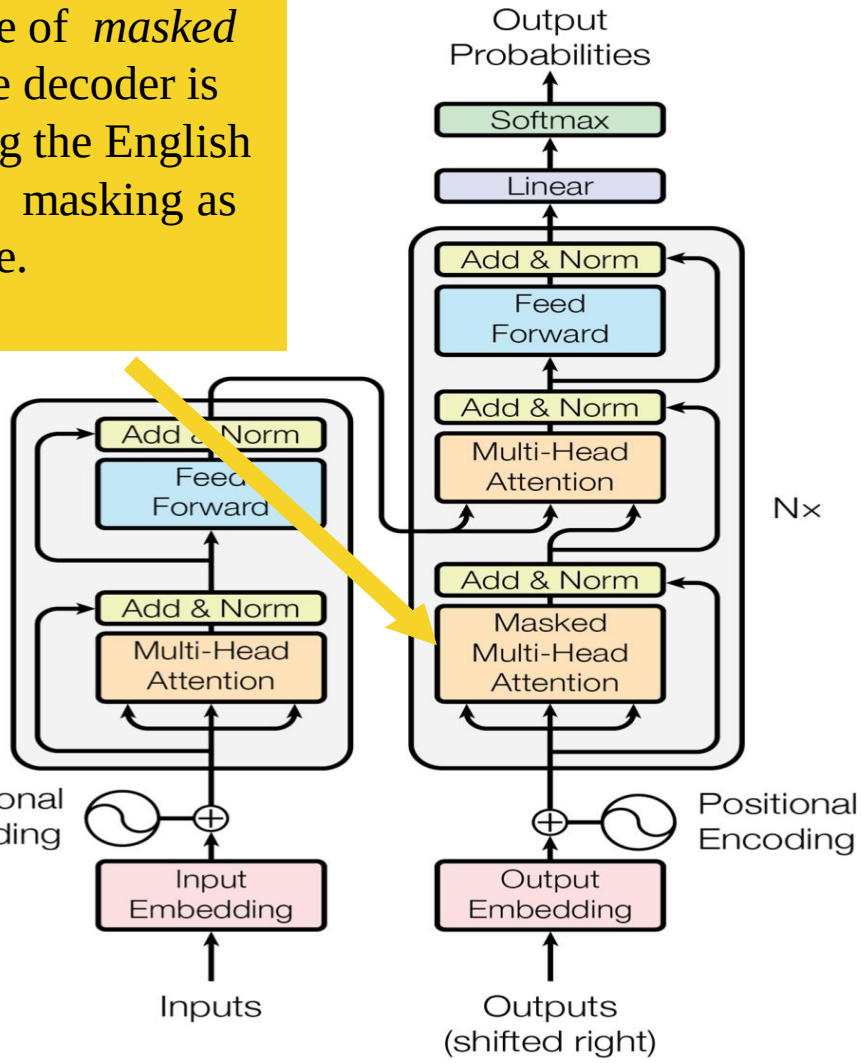


Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., <*START*> *schools opened their*)



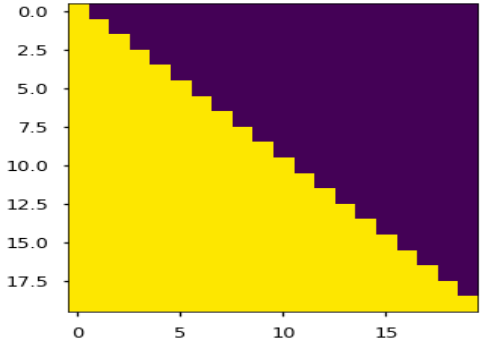
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

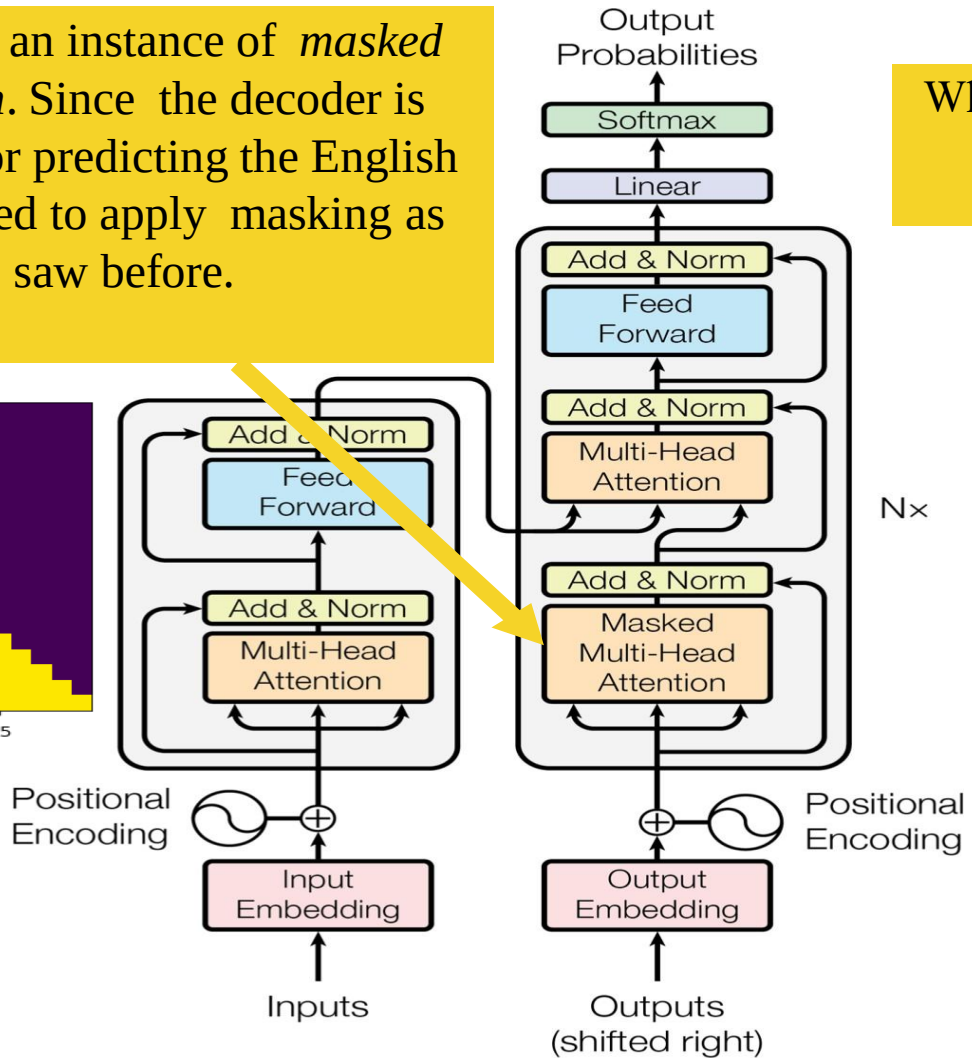


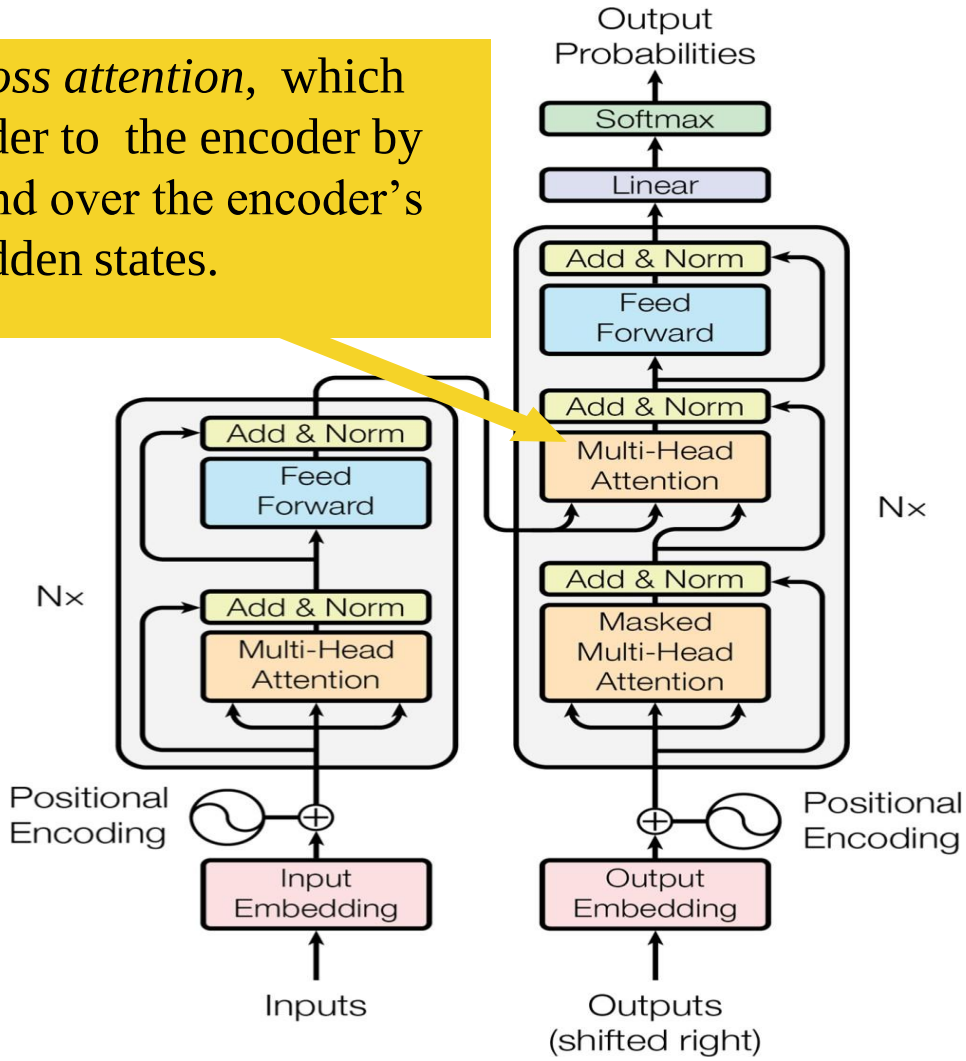
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

Why don't we do masked self-attention in the encoder?

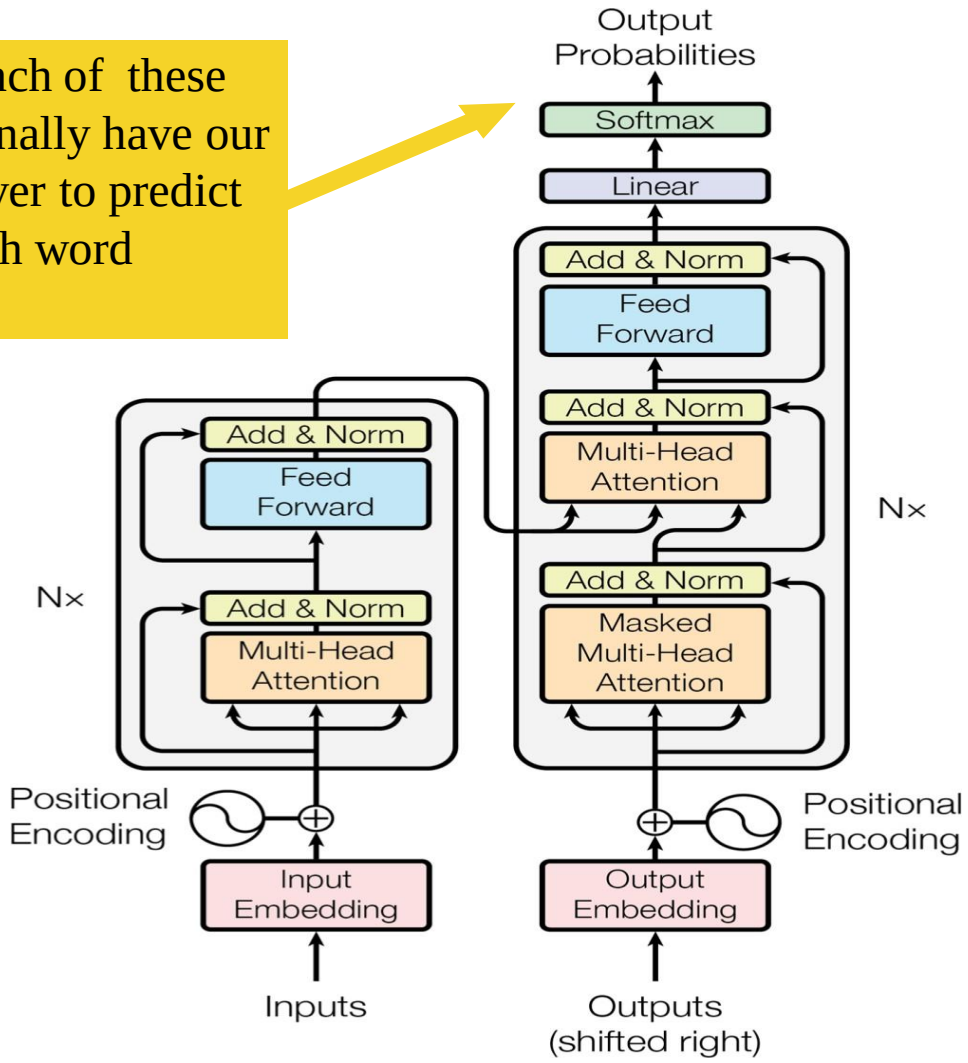


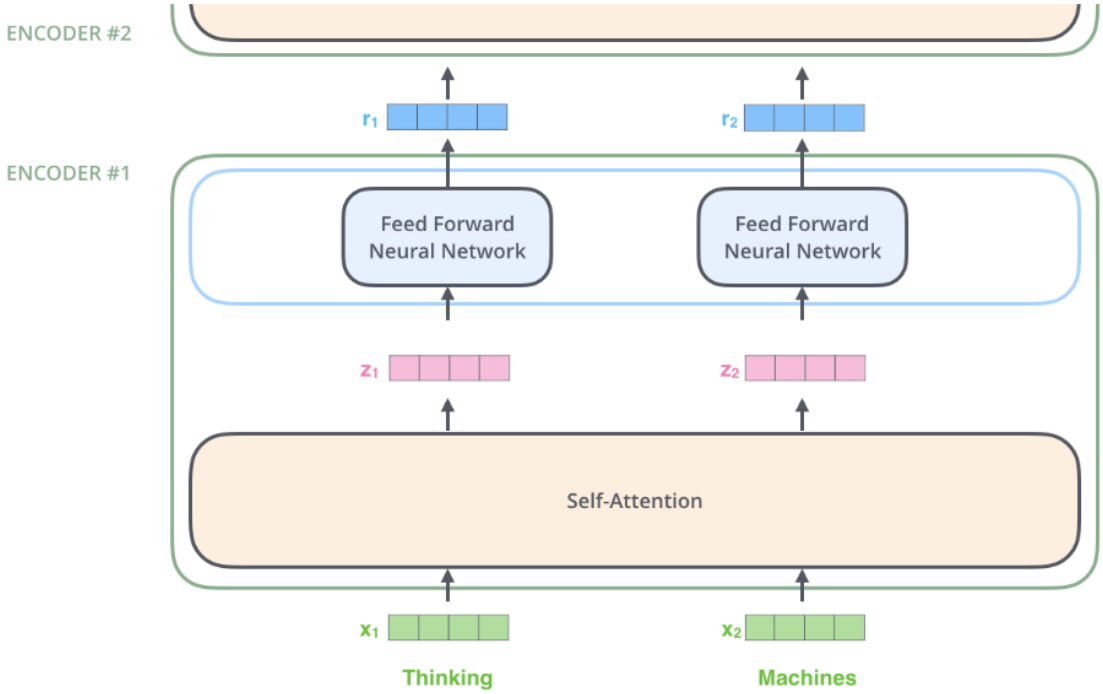



Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



After stacking a bunch of these decoder blocks, we finally have our familiar Softmax layer to predict the next English word



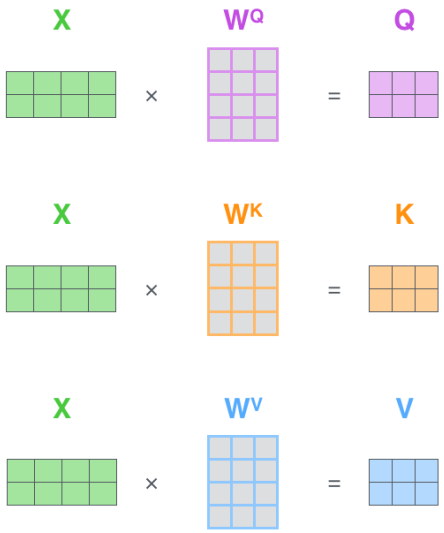


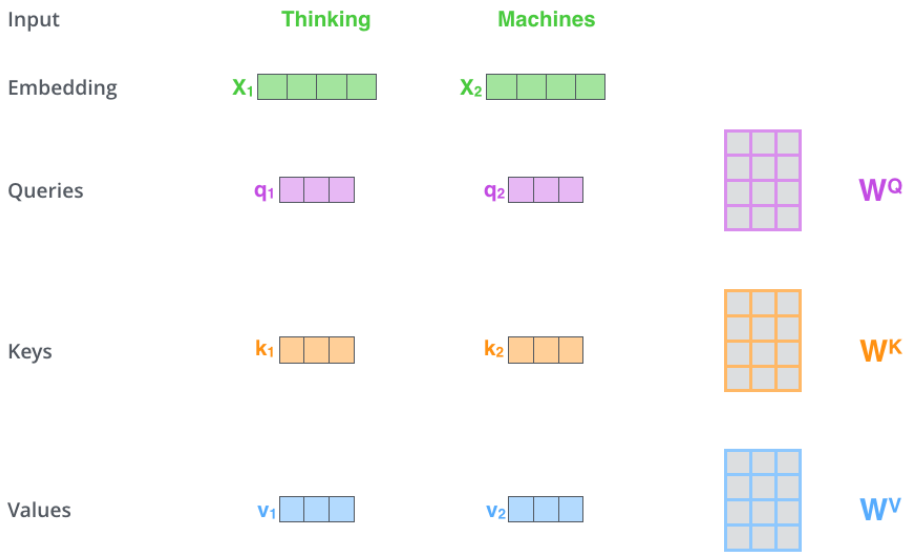

The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

Token abstraction

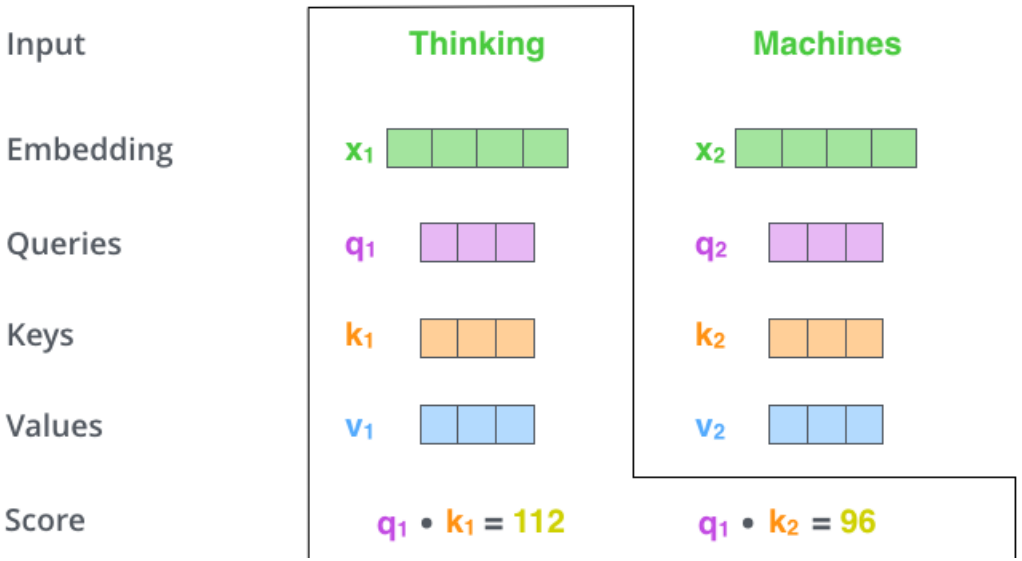
Between-token interaction

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

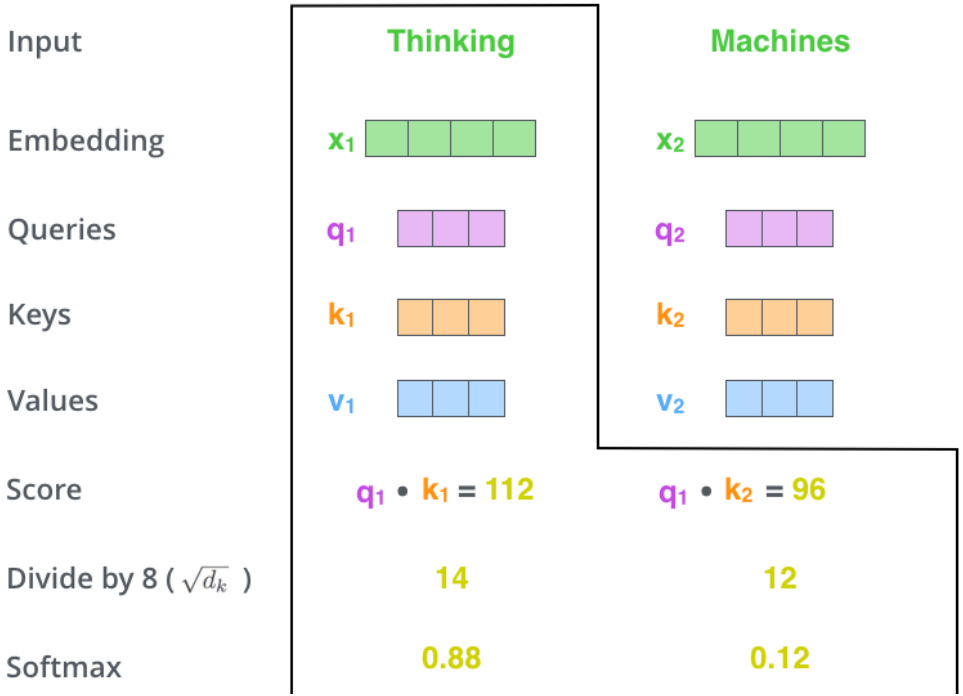



# Multi-faced token representation (QKV)

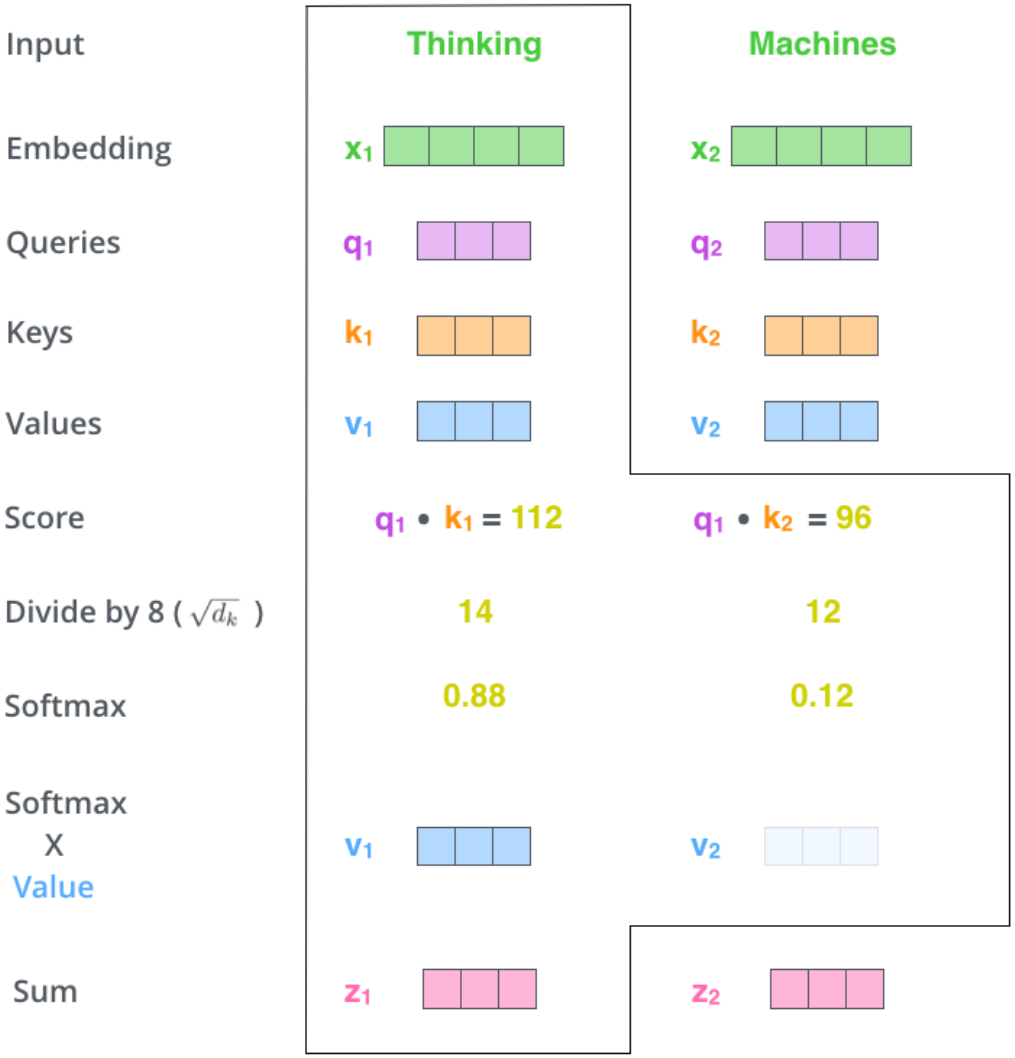



**Key-value** interaction between tokens



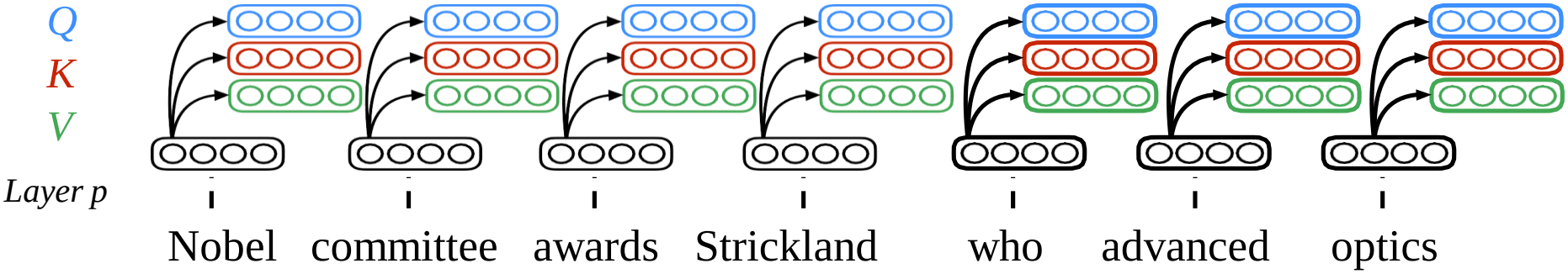

**Normalize** the **Key-value** interaction as attention (a probability distribution)

p.s. why softmax makes a probability distribution?


Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ( √dk )
14
12
Softmax
0.88
0.12
Softmax X Value
v1
v2
Sum
z1
z2

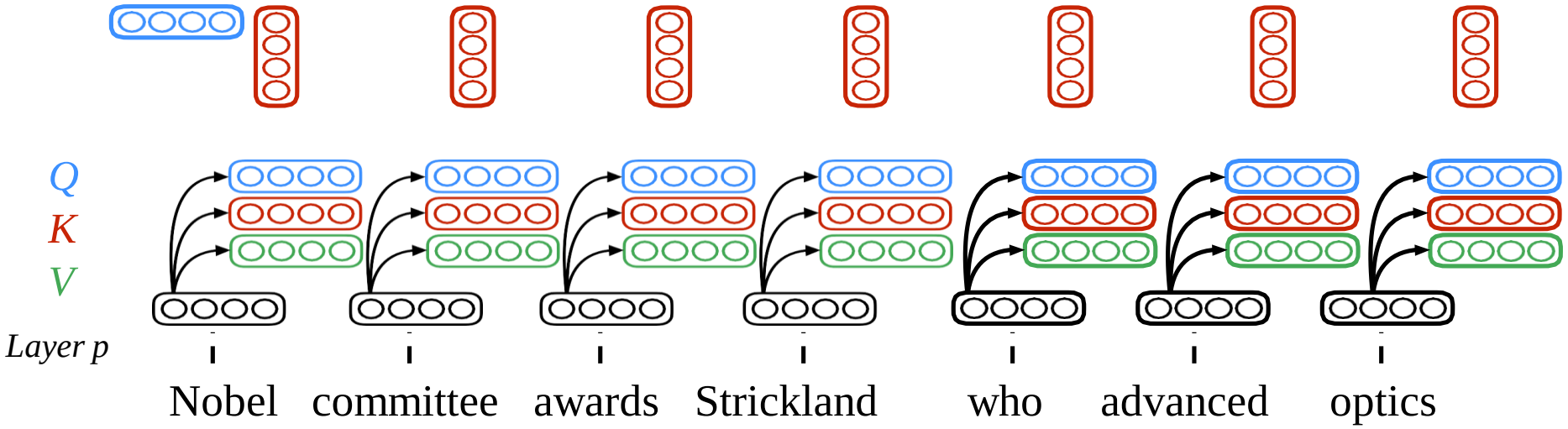
[Vaswani et al. 2017]

# Self-attention (in encoder)



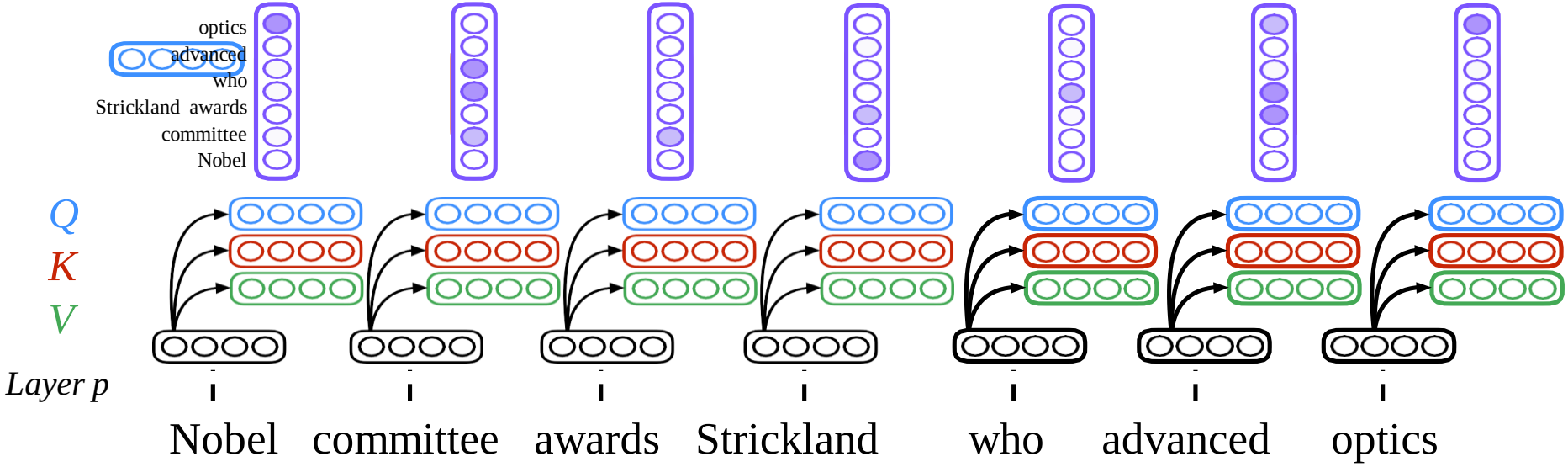
[Vaswani et al. 2017]

# Self-attention (in encoder)



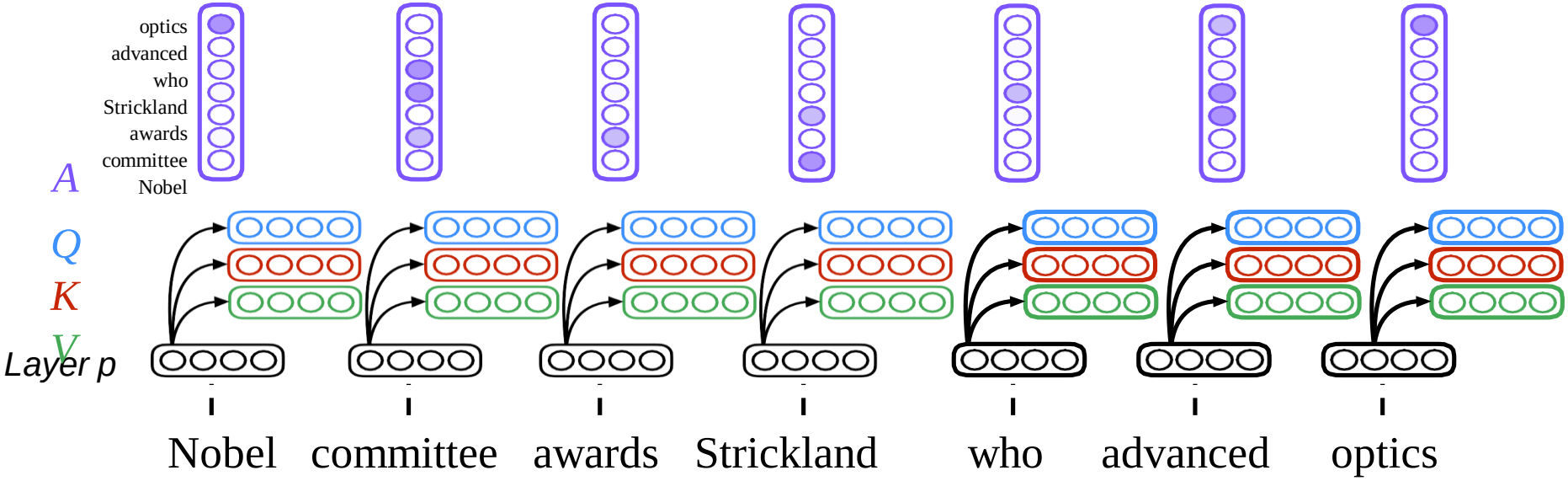
[Vaswani et al. 2017]

# Self-attention (in encoder)



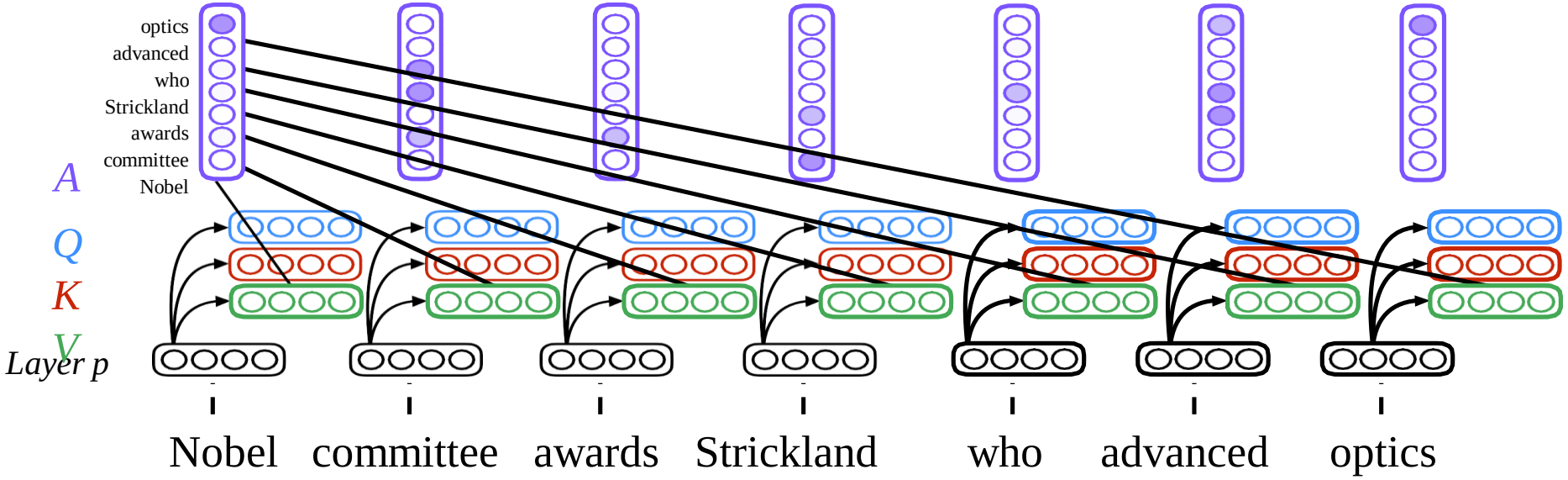
[Vaswani et al. 2017]

# Self-attention (in encoder)



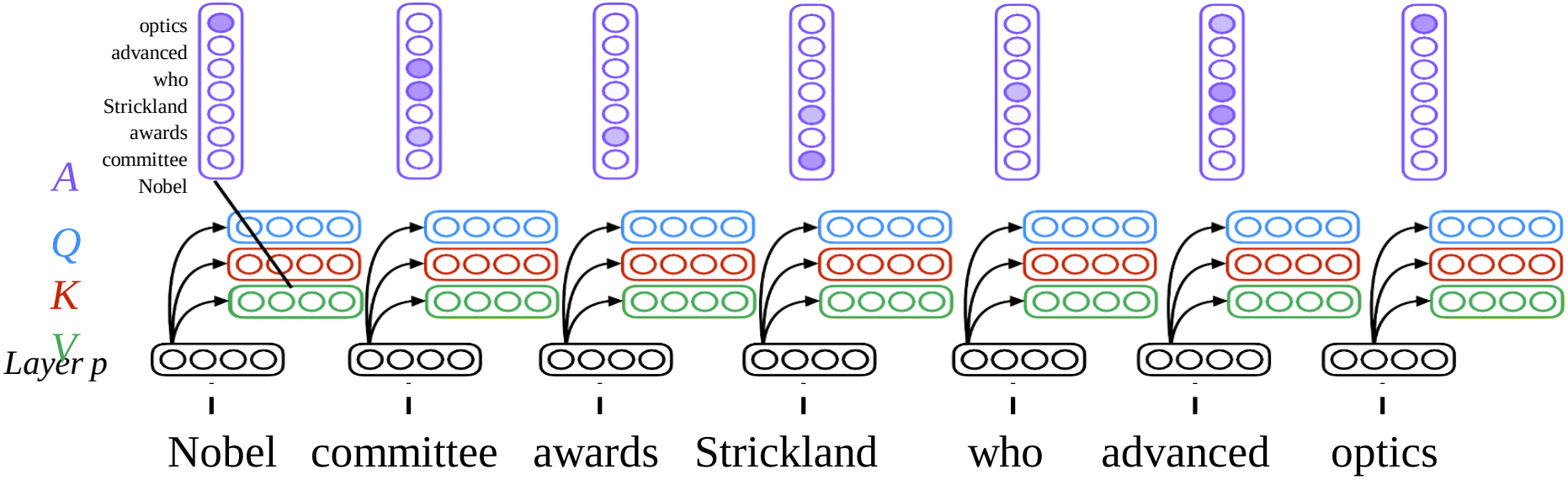
[Vaswani et al. 2017]

# Self-attention (in encoder)



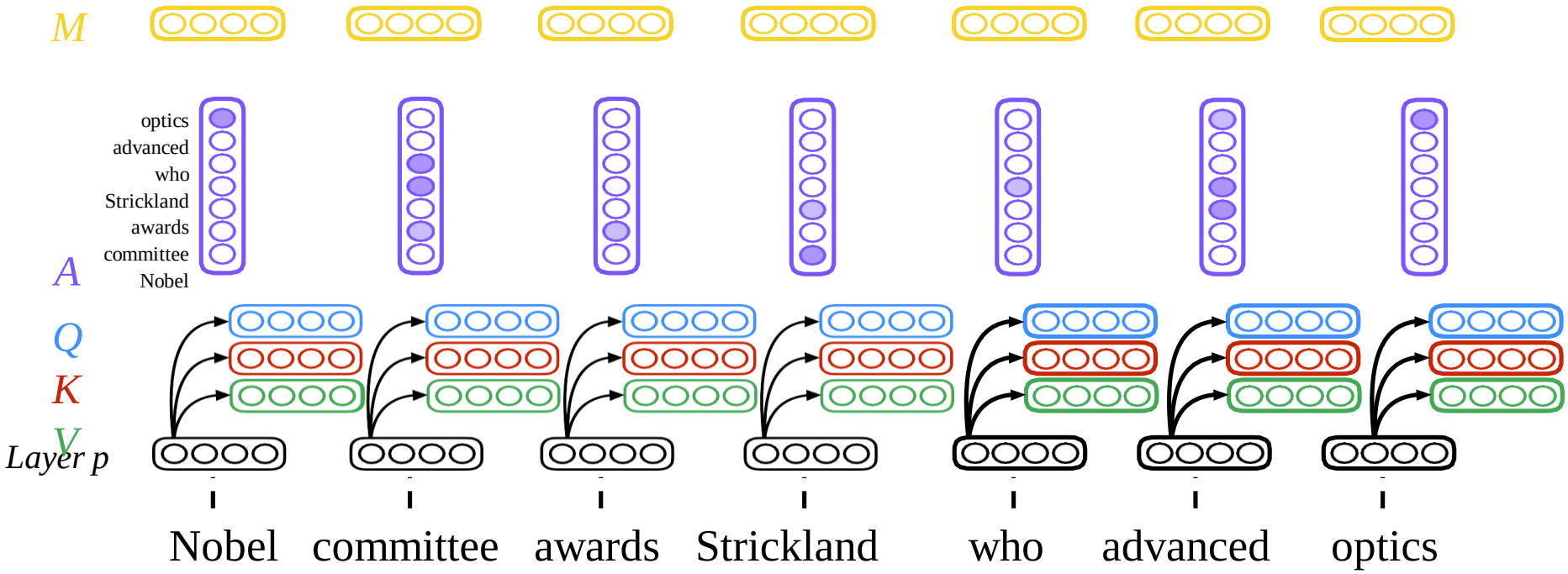
[Vaswani et al. 2017]

# Self-attention (in encoder)



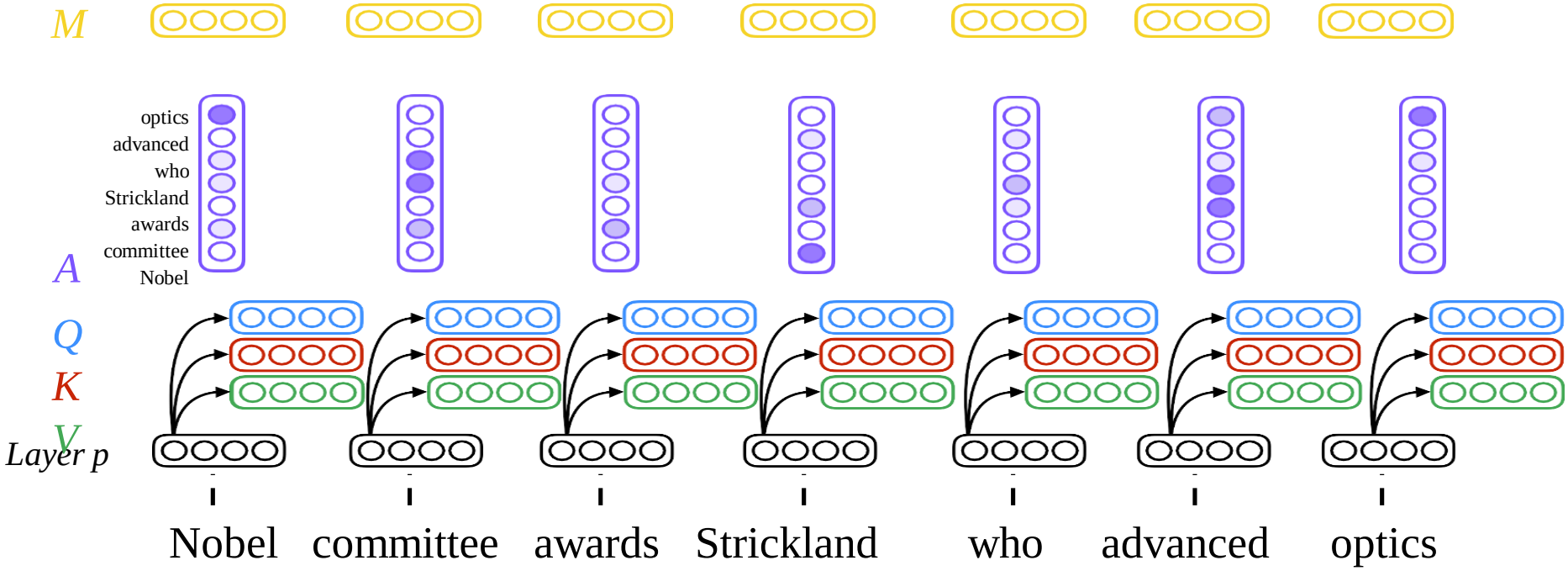
[Vaswani et al. 2017]

# Self-attention (in encoder)



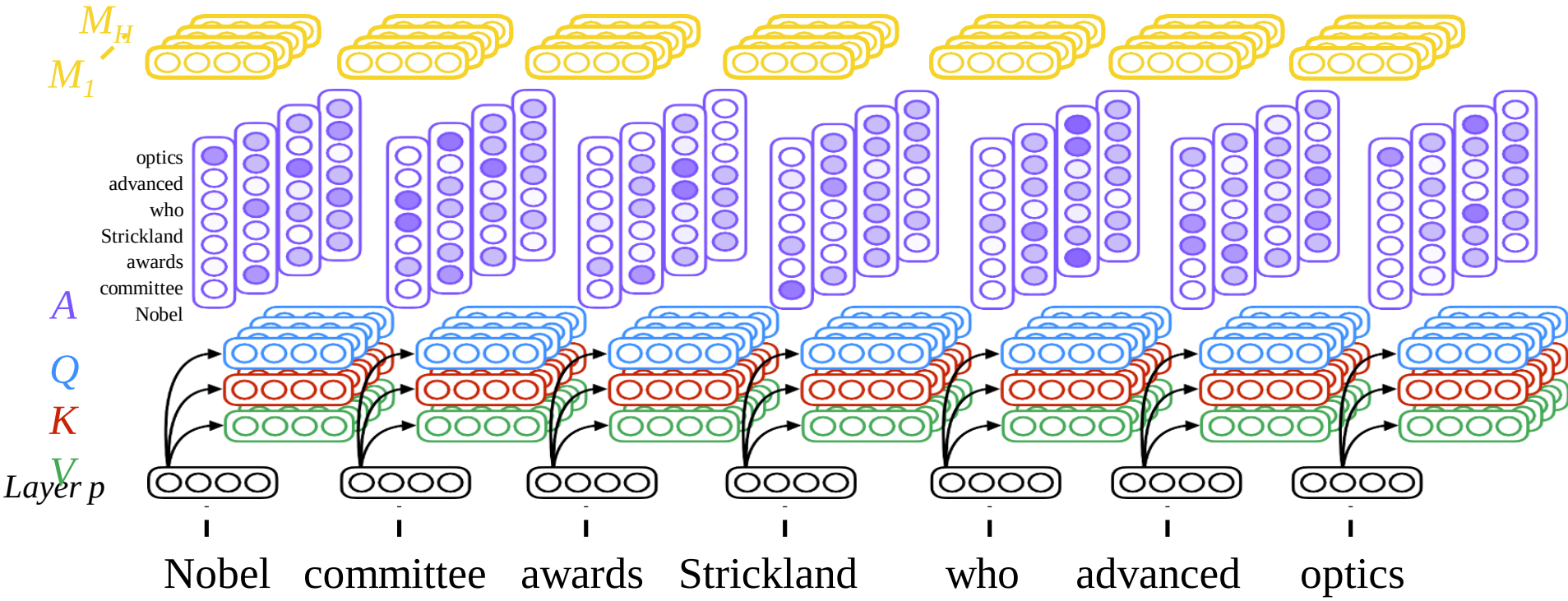
[Vaswani et al. 2017]

# Self-attention (in encoder)



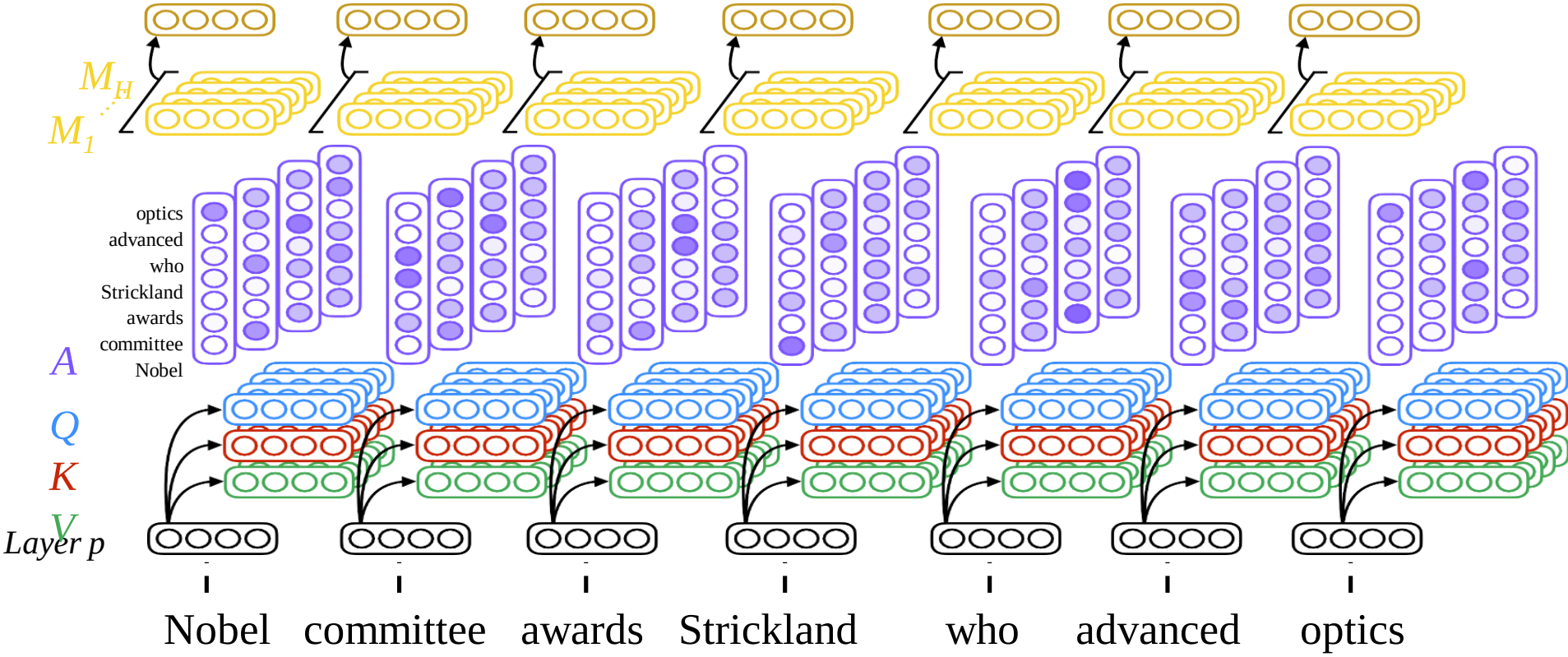
[Vaswani et al. 2017]

# Multi-head self-attention



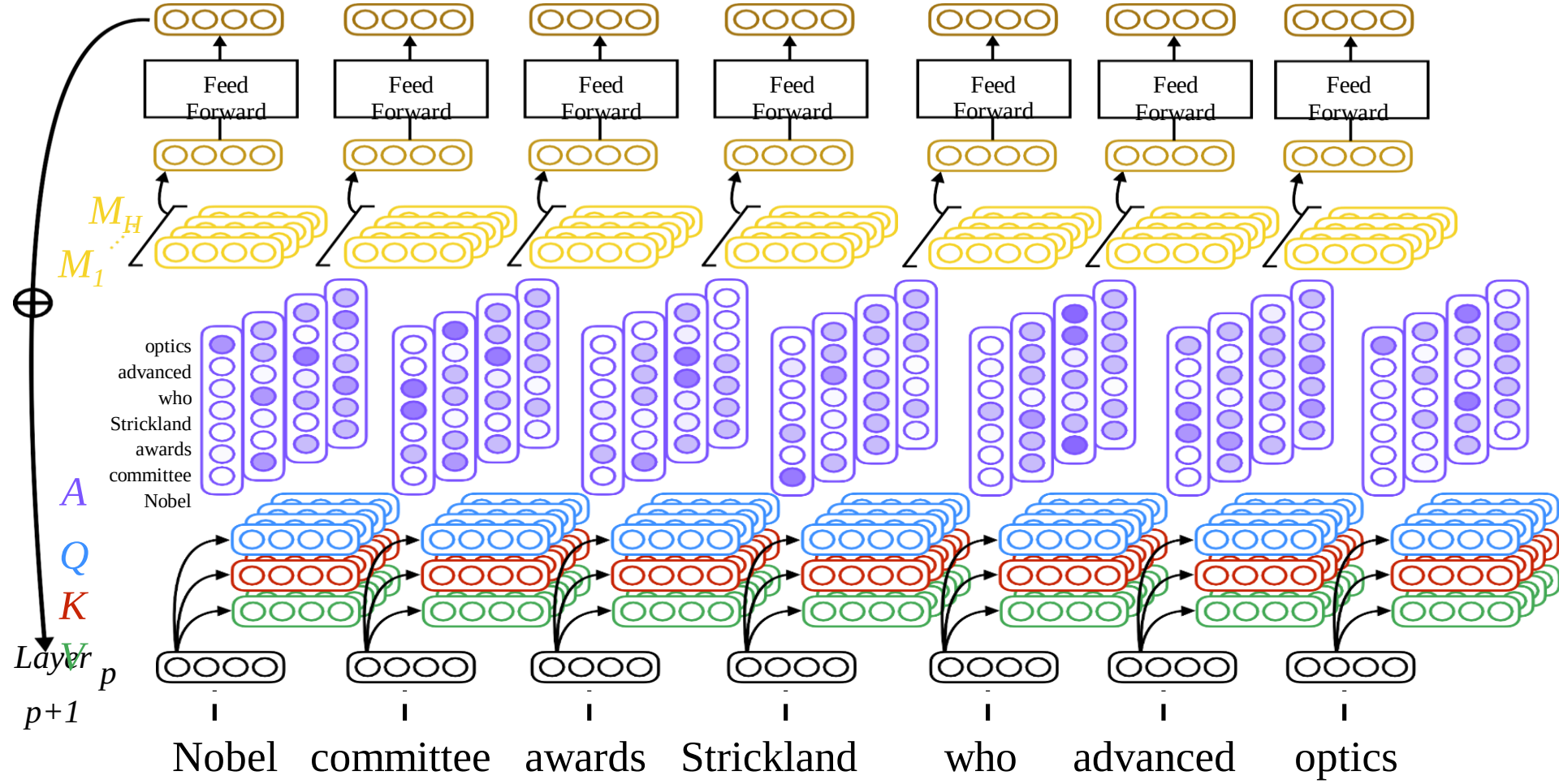
[Vaswani et al. 2017]

# Multi-head self-attention

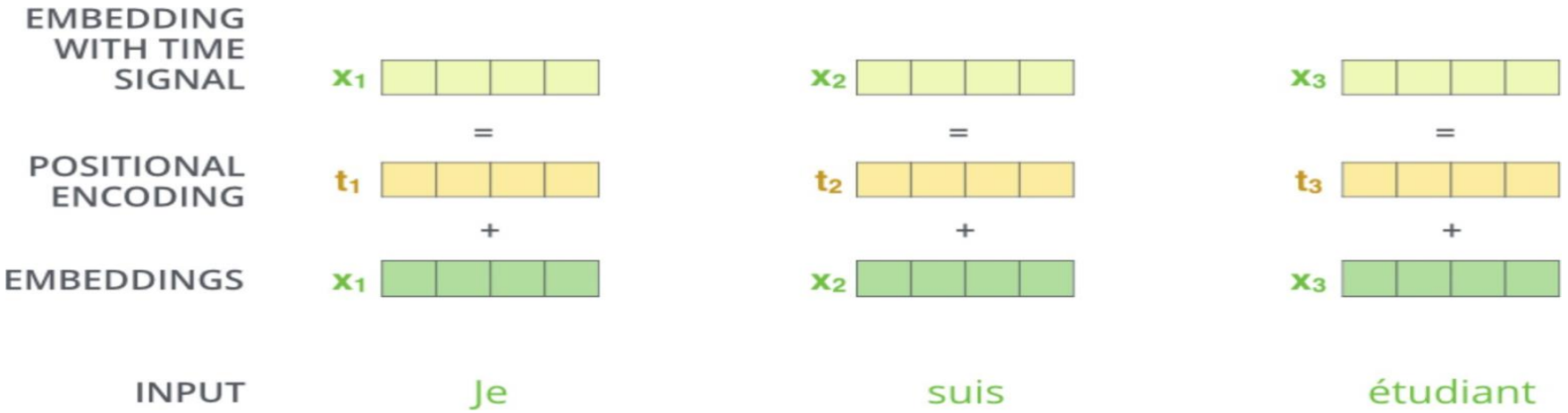


[Vaswani et al. 2017]

# Multi-head self-attention



# Positional encoding



# Intuitive example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 : | 0 | 0 | 0 | 0 | 8 : | 1 | 0 | 0 | 0 |
| 1 : | 0 | 0 | 0 | 1 | 9 : | 1 | 0 | 0 | 1 |
| 2 : | 0 | 0 | 1 | 0 | 10 : | 1 | 0 | 1 | 0 |
| 3 : | 0 | 0 | 1 | 1 | 11 : | 1 | 0 | 1 | 1 |
| 4 : | 0 | 1 | 0 | 0 | 12 : | 1 | 1 | 0 | 0 |
| 5 : | 0 | 1 | 0 | 1 | 13 : | 1 | 1 | 0 | 1 |
| 6 : | 0 | 1 | 1 | 0 | 14 : | 1 | 1 | 1 | 0 |
| 7 : | 0 | 1 | 1 | 1 | 15 : | 1 | 1 | 1 | 1 |

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

# Transformer positional encoding

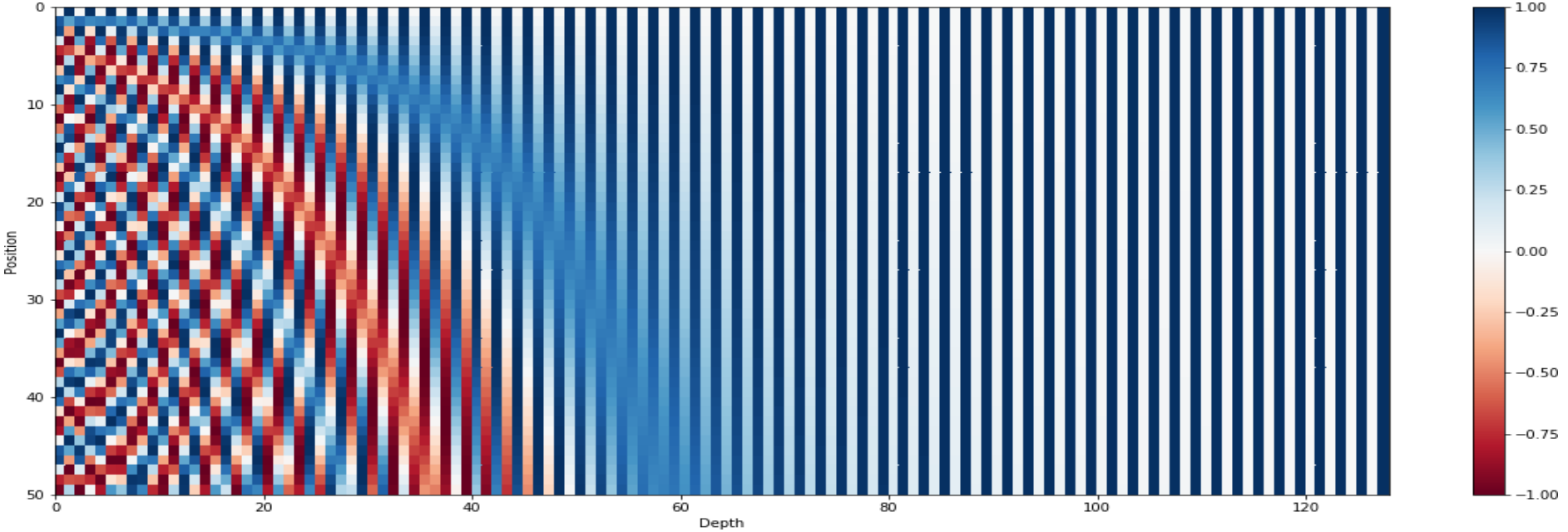
$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d_{model}}})$$

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

Positional encoding is a 512d vector
*i* = a particular dimension of this vector
*pos* = dimension of the word
*d_model* = 512

# What does this look like?

*(each row is the pos. emb. of a 50-word sentence)*




https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

# Modern Position Embedding： Rotary embedding

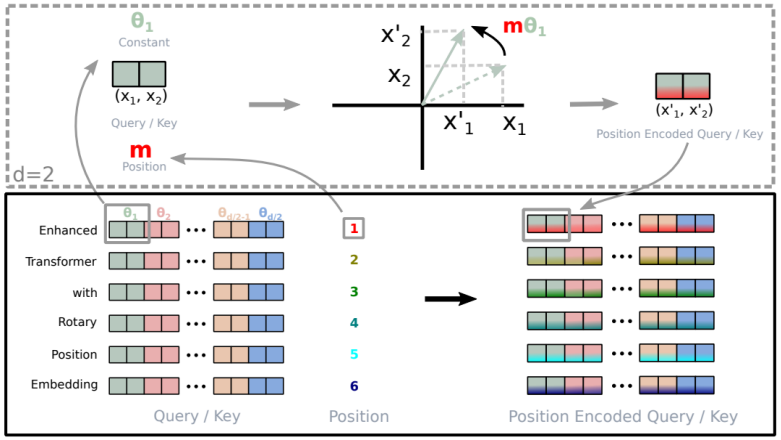



Figure 1: Implementation of Rotary Position Embedding(RoPE).


Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. https://arxiv.org/abs/2104.09864

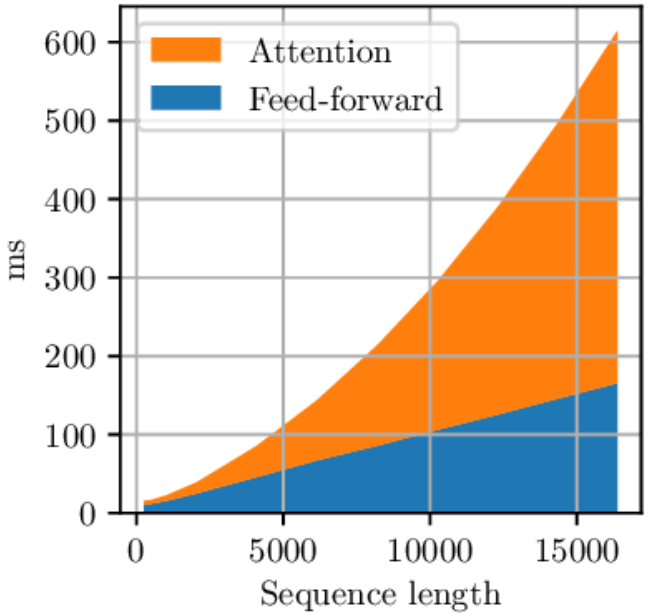
Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, Jakob Grue Simonsen. Encoding word order in complex embeddings. https://arxiv.org/abs/1912.12333

# More on new-Transformer

## What would we like to fix about the Transformer?

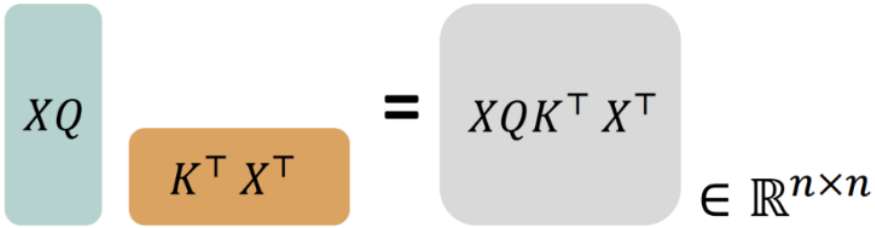
**Quadratic compute in self-attention (today):**

- Computing all pairs of interactions means our computation grows quadratically with the sequence length!
- For recurrent models, it only grew linearly!



# Quadratic computation as a function of sequence length

- One of the benefits of self-attention over recurrence was that it's highly parallelizable.
- However, its total number of operations grows as $O(n^2 d)$, where $n$ is the sequence length, and $d$ is the dimensionality.



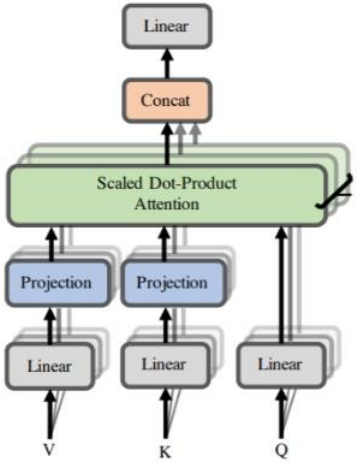

Need to compute all pairs of interactions!
$O(n^2 d)$

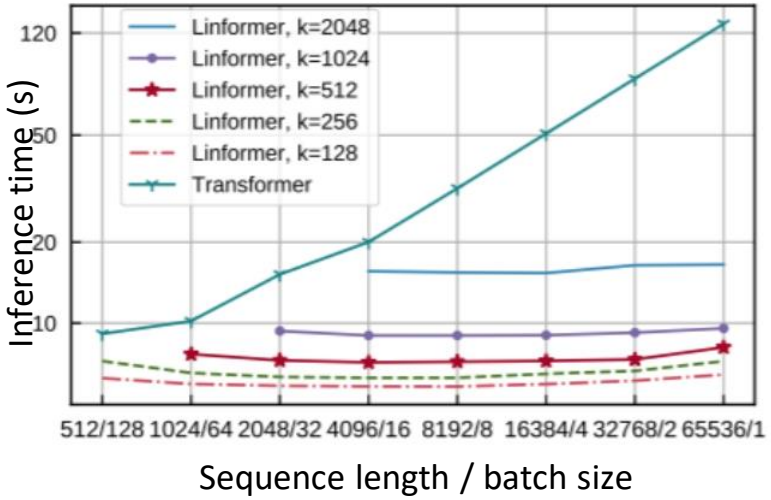
- Think of $d$ as around $\mathbf{1,000}$ (though for large language models it's much larger!).
  - So, for a single (shortish) sentence, $n \leq 30; n^2 \leq \mathbf{900}$.
  - In practice, we set a bound like $n = 512$.
  - **But what if we'd like $n \geq 50,000$?** For example, to work on long documents?

# Work on improving on quadratic self-attention cost

Considerable recent work has gone into the question, *Can we build models like Transformers without paying the all-pairs self-attention cost?*
For example, **Linformer** [Wang et al., 2020]

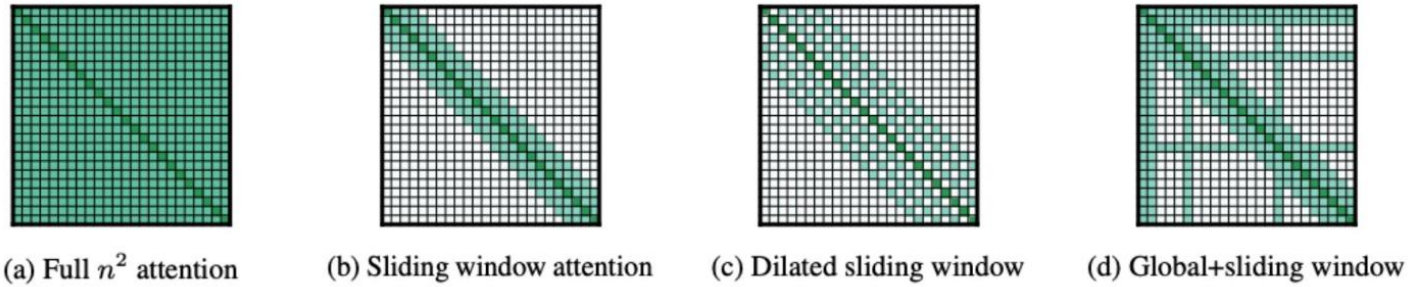
**Key Idea:**
- Linformer introduces a novel concept called "compressed" or "linearized" self-attention.
- Instead of computing attention scores for all pairs of input elements, it employs linear projections to reduce the complexity.



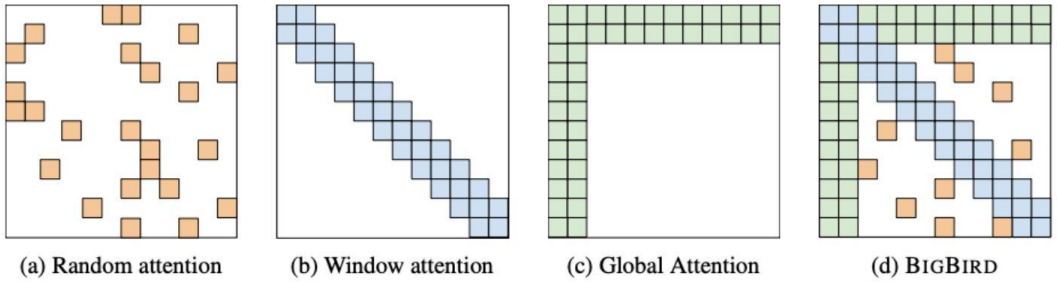
# Example: Longformer / Big Bird

Key idea: use sparse attention patterns!



(a) Full $n^2$ attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

(Beltagy et al., 2020): Longformer: The Long-Document Transformer



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

(Zaheer et al., 2021): Big Bird: Transformers for Longer Sequences

# Do we even need to remove the quadratic cost of attention?

- **As Transformers Scale Up:** When Transformers are scaled to larger sizes, an increasingly significant portion of computational resources is allocated to tasks outside of the self-attention mechanism, despite its quadratic computational cost.
- **Current Practice:** In practice, nearly all large Transformer-based language models continue to rely on the traditional quadratic-cost attention mechanism that has been presented.
- **Challenges with Cost-Efficiency:** Alternative, more computationally efficient methods often do not perform as effectively when applied at a large scale.
- **Exploring Cheaper Alternatives:** Is there value in exploring cost-efficient alternatives to self-attention, or could we unlock the potential for significantly improved models with much longer contextual information (e.g., >100k tokens) if we find the right approach?

# Do Transformer Modifications Transfer?

- "Surprisingly, we find that most modifications do not meaningfully improve performance."

| Model | Params | Ops | Step/s | Early loss | Final loss | SGLUE | XSum | WebQ | WMT EnDe |
|---|---|---|---|---|---|---|---|---|---|
| Vanilla Transformer | 223M | 11.1T | 3.50 | 2.182 ± 0.005 | 1.838 | 71.66 | 17.78 | 23.02 | 26.62 |
| GeLU | 223M | 11.1T | 3.58 | 2.179 ± 0.003 | 1.838 | **75.79** | **17.86** | **25.13** | 26.47 |
| Swish | 223M | 11.1T | 3.62 | 2.186 ± 0.003 | 1.847 | **73.77** | 17.74 | **24.34** | **26.75** |
| ELU | 223M | 11.1T | 3.56 | 2.270 ± 0.007 | 1.932 | 67.83 | 16.73 | 23.02 | 26.08 |
| GLU | 223M | 11.1T | 3.59 | 2.174 ± 0.003 | **1.814** | **74.20** | **17.42** | 24.34 | **27.12** |
| GeGLU | 223M | 11.1T | 3.55 | 2.130 ± 0.006 | **1.792** | **75.96** | **18.27** | **24.87** | **26.87** |
| ReGLU | 223M | 11.1T | 3.57 | 2.145 ± 0.004 | **1.803** | **76.17** | **18.36** | **24.87** | **27.02** |
| SeLU | 223M | 11.1T | 3.55 | 2.315 ± 0.004 | 1.948 | 68.76 | 16.76 | 22.75 | 25.99 |
| SwiGLU | 223M | 11.1T | 3.53 | 2.127 ± 0.003 | **1.789** | **76.00** | **18.20** | **24.34** | **27.02** |
| LiGLU | 223M | 11.1T | 3.59 | 2.149 ± 0.005 | **1.798** | **75.34** | **17.97** | **24.34** | 26.53 |
| Sigmoid | 223M | 11.1T | 3.63 | 2.291 ± 0.019 | 1.867 | **74.31** | 17.51 | 23.02 | 26.30 |
| Softplus | 223M | 11.1T | 3.47 | 2.207 ± 0.011 | 1.850 | **72.45** | 17.65 | **24.34** | **26.89** |
| RMS Norm | 223M | 11.1T | 3.68 | 2.167 ± 0.008 | **1.821** | **75.45** | **17.94** | **24.07** | **27.14** |
| Rezero | 223M | 11.1T | 3.51 | 2.262 ± 0.003 | 1.939 | 61.69 | 15.64 | 20.90 | 26.37 |
| Rezero + LayerNorm | 223M | 11.1T | 3.26 | 2.223 ± 0.006 | 1.858 | 70.42 | 17.58 | 23.02 | 26.29 |
| Rezero + RMS Norm | 223M | 11.1T | 3.34 | 2.221 ± 0.009 | 1.875 | 70.33 | 17.32 | 23.02 | 26.19 |
| Fixup | 223M | 11.1T | 2.95 | 2.382 ± 0.012 | 2.067 | 58.56 | 14.42 | 23.02 | 26.31 |
| 24 layers, $d_{ff} = 1536$, $H = 6$ | 224M | 11.1T | 3.33 | 2.200 ± 0.007 | 1.843 | **74.89** | 17.75 | **25.13** | **26.89** |
| 18 layers, $d_{ff} = 2048$, $H = 8$ | 223M | 11.1T | 3.38 | 2.185 ± 0.005 | **1.831** | **76.45** | 16.83 | **24.34** | **27.10** |
| 8 layers, $d_{ff} = 4608$, $H = 18$ | 223M | 11.1T | 3.69 | 2.190 ± 0.005 | 1.847 | **74.58** | 17.69 | **23.28** | **26.85** |
| 6 layers, $d_{ff} = 6144$, $H = 24$ | 223M | 11.1T | 3.70 | 2.201 ± 0.010 | 1.857 | **73.55** | 17.59 | **24.60** | **26.66** |
| Block sharing | 65M | 11.1T | 3.91 | 2.497 ± 0.037 | 2.164 | 64.50 | 14.53 | 21.96 | 25.48 |
| + Factorized embeddings | 45M | 9.4T | 4.21 | 2.631 ± 0.305 | 2.183 | 60.84 | 14.00 | 19.84 | 25.27 |
| + Factorized & shared embeddings | 20M | 9.1T | 4.37 | 2.907 ± 0.313 | 2.385 | 53.95 | 11.37 | 19.84 | 25.19 |
| Encoder only block sharing | 170M | 11.1T | 3.68 | 2.298 ± 0.023 | 1.929 | 69.60 | 16.23 | 23.02 | 26.23 |
| Decoder only block sharing | 144M | 11.1T | 3.70 | 2.352 ± 0.029 | 2.082 | 67.93 | 16.13 | **23.81** | 26.08 |
| Factorized Embedding | 227M | 9.4T | 3.80 | 2.208 ± 0.006 | 1.855 | 70.41 | 15.92 | 22.75 | 26.50 |
| Factorized & shared embeddings | 202M | 9.1T | 3.92 | 2.320 ± 0.010 | 1.952 | 68.69 | 16.33 | 22.22 | 26.44 |
| Tied encoder/decoder input embeddings | 248M | 11.1T | 3.55 | 2.192 ± 0.002 | 1.840 | **71.70** | 17.72 | **24.34** | 26.49 |
| Tied decoder input and output embeddings | 248M | 11.1T | 3.57 | 2.187 ± 0.007 | **1.827** | **74.86** | 17.74 | **24.87** | **26.67** |
| Untied embeddings | 273M | 11.1T | 3.53 | 2.195 ± 0.005 | **1.834** | **72.99** | 17.58 | **23.28** | 26.48 |
| Adaptive input embeddings | 394M | 9.2T | 3.55 | 2.250 ± 0.002 | 1.899 | 66.57 | 16.21 | **24.07** | **26.66** |
| Adaptive softmax | 394M | 9.2T | 3.60 | 2.364 ± 0.005 | 1.982 | **72.91** | 16.67 | 21.16 | 25.56 |
| Adaptive softmax without projection | 223M | 10.8T | 3.43 | 2.229 ± 0.009 | 1.914 | **71.82** | 17.10 | 23.02 | 25.72 |
| Mixture of softmaxes | 232M | 16.3T | 2.24 | 2.227 ± 0.017 | **1.821** | **76.77** | 17.62 | 22.75 | **26.82** |
| Transparent attention | 223M | 11.1T | 3.33 | 2.181 ± 0.014 | 1.874 | 54.31 | 10.40 | 21.16 | **26.80** |
| Dynamic convolution | 257M | 11.8T | 2.65 | 2.403 ± 0.009 | 2.047 | 58.30 | 12.67 | 21.16 | 17.03 |
| Lightweight convolution | 234M | 10.4T | 4.07 | 2.370 ± 0.010 | 1.989 | 63.07 | 14.86 | 23.02 | 24.73 |
| Evolved Transformer | 217M | 9.9T | 3.09 | 2.220 ± 0.003 | 1.863 | **73.67** | 10.76 | **24.07** | 26.58 |
| Synthesizer (dense) | 224M | 11.4T | 3.47 | 2.334 ± 0.021 | 1.962 | 61.03 | 14.27 | 16.14 | **26.63** |
| Synthesizer (dense plus) | 243M | 12.6T | 3.22 | 2.191 ± 0.010 | 1.840 | **73.98** | 16.96 | **23.81** | **26.71** |
| Synthesizer (dense plus alpha) | 243M | 12.6T | 3.01 | 2.180 ± 0.007 | **1.828** | **74.25** | 17.02 | **23.28** | 26.61 |
| Synthesizer (factorized) | 207M | 10.1T | 3.94 | 2.341 ± 0.017 | 1.968 | 62.78 | 15.39 | **23.55** | 26.42 |
| Synthesizer (random) | 254M | 10.1T | 4.08 | 2.326 ± 0.012 | 2.009 | 54.27 | 10.35 | 19.56 | 26.44 |
| Synthesizer (random plus) | 292M | 12.0T | 3.63 | 2.189 ± 0.004 | 1.842 | **73.32** | 17.04 | **24.87** | 26.43 |
| Synthesizer (random plus alpha) | 292M | 12.0T | 3.42 | 2.186 ± 0.007 | **1.828** | **75.24** | 17.08 | **24.08** | 26.39 |
| Universal Transformer | 84M | 40.0T | 0.88 | 2.406 ± 0.036 | 2.053 | 70.13 | 14.09 | 19.05 | 23.91 |
| Mixture of experts | 648M | 11.7T | 3.20 | 2.148 ± 0.006 | **1.785** | **74.55** | **18.13** | **24.08** | **26.94** |
| Switch Transformer | 1100M | 11.7T | 3.18 | 2.135 ± 0.007 | **1.758** | **75.38** | **18.02** | **26.19** | **26.81** |
| Funnel Transformer | 223M | 1.9T | 4.30 | 2.288 ± 0.008 | 1.918 | 67.34 | 16.26 | 22.75 | 23.20 |
| Weighted Transformer | 280M | 71.0T | 0.59 | 2.378 ± 0.021 | 1.989 | 69.04 | 16.98 | 23.02 | 26.30 |
| Product key memory | 421M | 386.6T | 0.25 | 2.155 ± 0.003 | **1.798** | **75.16** | 17.04 | **23.55** | **26.73** |

**Do Transformer Modifications Transfer Across Implementations and Applications?**

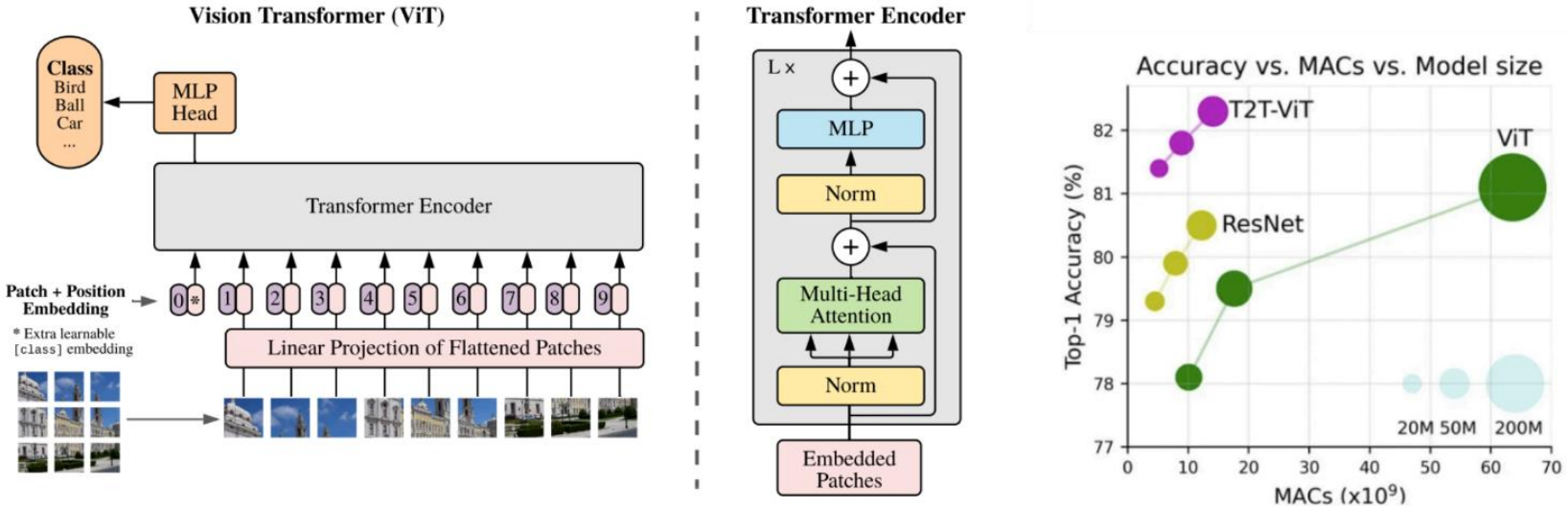

Sharan Narang* Hyung Won Chung Yi Tay William Fedus

Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel

Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li

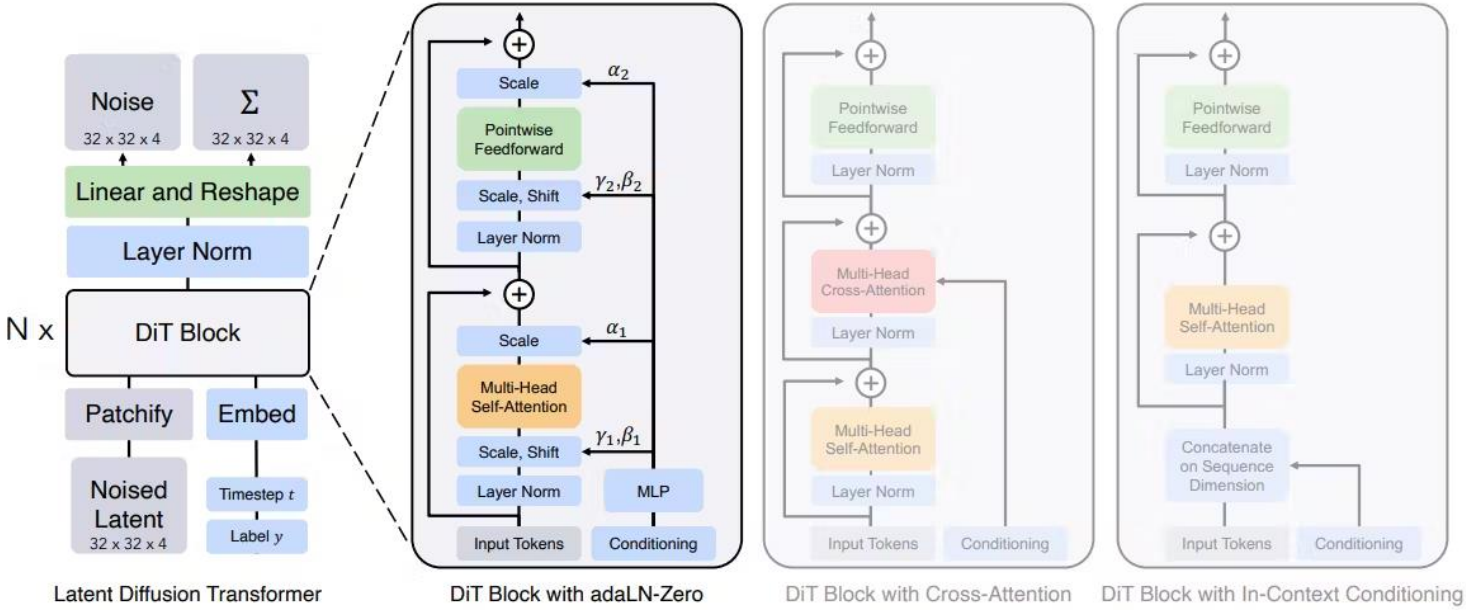
Nan Ding Jake Marcus Adam Roberts Colin Raffel†

# Vision Transformer (ViT)




(Dosovitskiy et al., 2021): An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

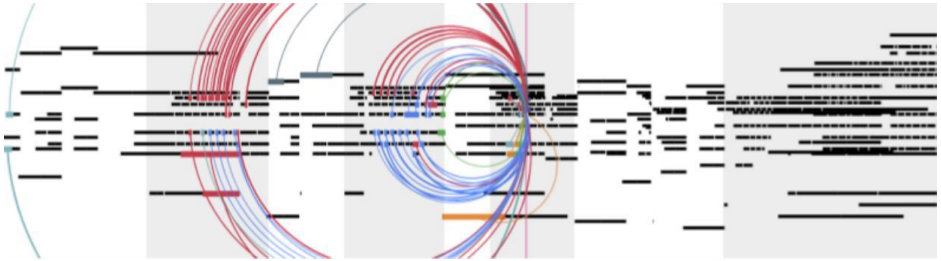
# Diffusion Transformer (DiT)




(William Peebles et al., 2022): Scalable Diffusion Models with Transformers.

DiT aims to improve the performance of diffusion models by replacing the commonly used U-Net backbone with a transformer.

# Music Transformer



https://magenta.tensorflow.org/music-transformer

(Huang et al., 2018): Music Transformer: Generating Music with Long-Term Structure

# Why transformer

# Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

Credits from 杨植麟, Recurrent AI

# Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

- 2. Because transformers are better on machine translation?

RNNs and CNNs are equally good in machine translations

# Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

- 2. Because transformers are better on machine translation?

RNNs and CNNs are equally good in machine translations

- 3. Because transformers use nothing but attention?

So what?

# Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

- 2. Because transformers are better on machine translation?

RNNs and CNNs are equally good in machine translations

- 3. Because transformers use nothing but attention?

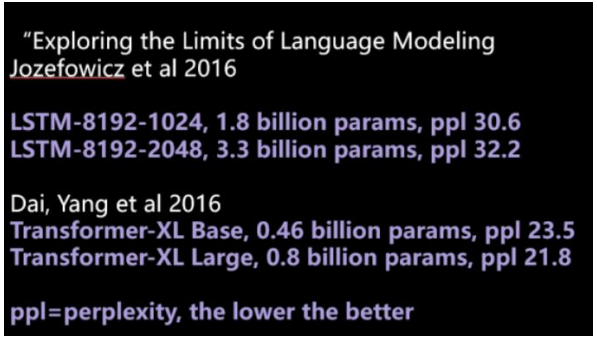
So what?

- 4. Because transformers learns contextualised word embeddings?

RNN also can learn contextualised word embeddings (See ELMO)

# Why Pretraining + Transformers

- **Capacity**: The model has sufficient expressive capabilities
- **Optimization**: Can optimize and obtain better solutions in a huge expression space
- **Generalization**: Better solutions can generalize on test data



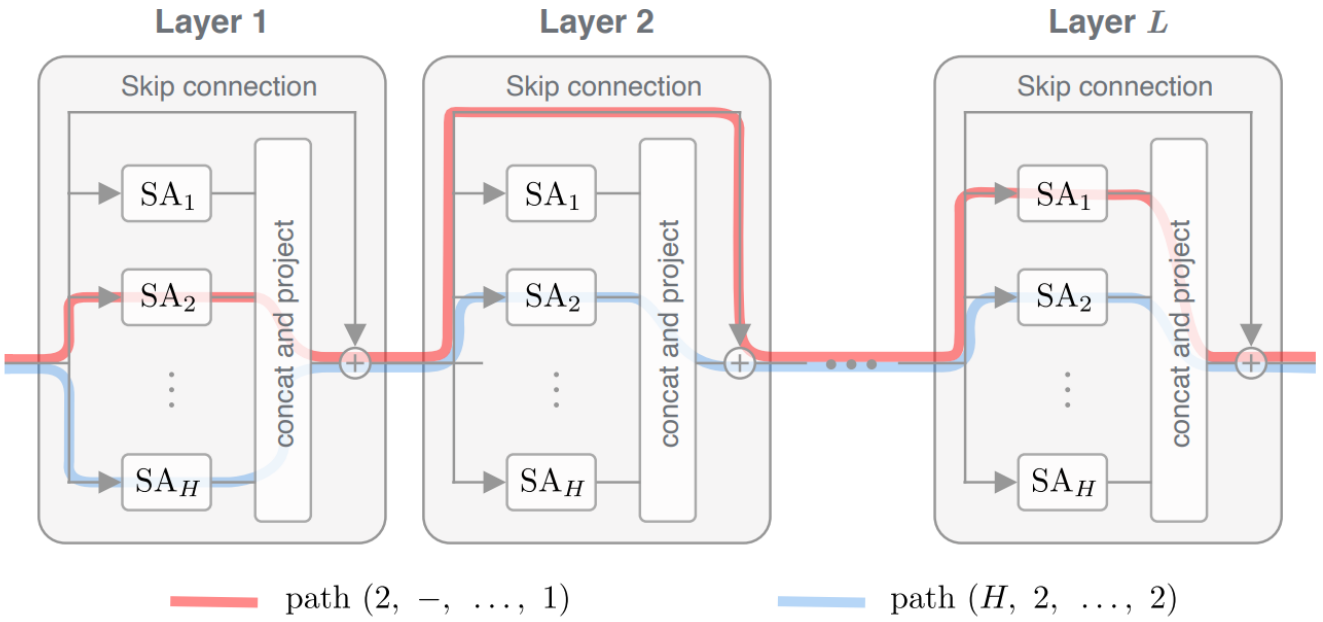

Scalability: Transformers scale much better with more parameters

# Deep understanding of transformer

What if

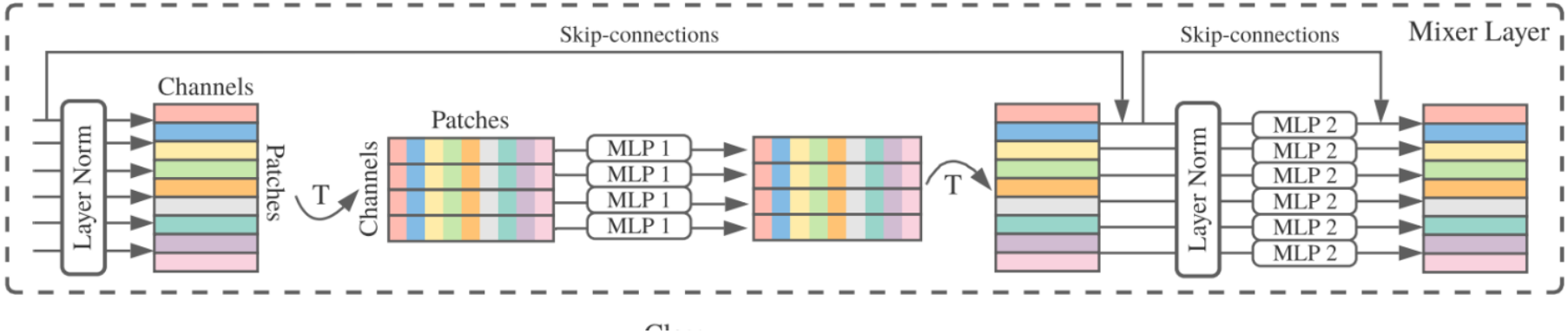
- ✔ removing SAN
- ✔ removing FFN
- ✔ removing PE
- ✔ and many others?

# Without FFN, pure SAN




Y Dong, JB Cordonnier, A Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. https://browse.arxiv.org/pdf/2103.03404.pdf

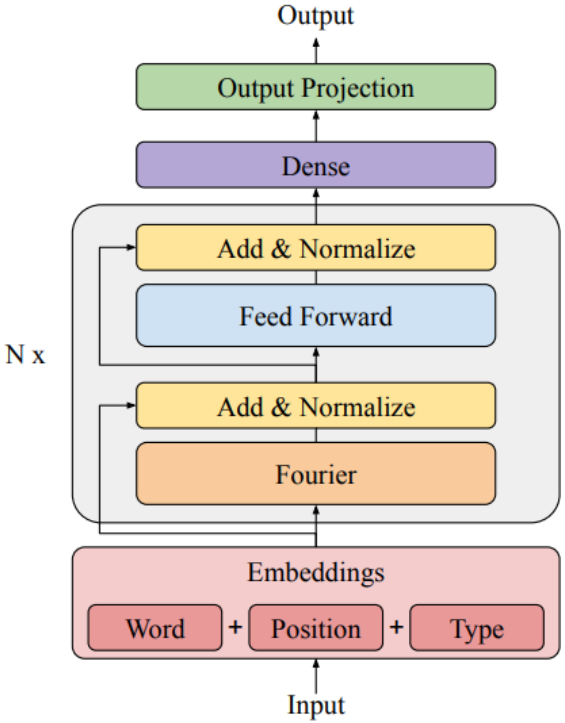
# Without SAN, pure FNN




At least it works for computer vision.

Ilya Tolstikhin et.al MLP-Mixer: An all-MLP Architecture for Vision https://browse.arxiv.org/pdf/2105.01601.pdf

# Replace SAN with fourier




- Highlight the potential of linear units as a drop-in replacement for the attention mechanism **in text classification** tasks.
- Unclear if it could generalize to more tasks

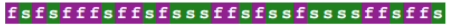
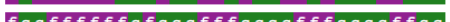
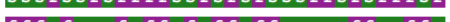
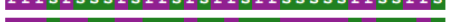
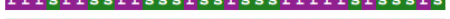
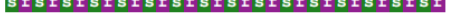
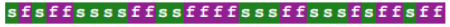
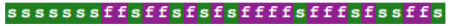
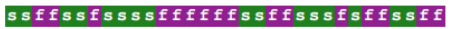
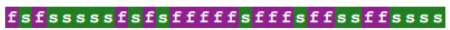
James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, Santiago Ontanon .
FNet: Mixing Tokens with Fourier Transforms. NAACL 2022

# How to place FFN and SAN

(a) Interleaved Transformer

(b) Sandwich Transformer

Figure 1: A transformer model (a) is composed of interleaved self-attention (green) and feedforward (purple) sublayers. Our sandwich transformer (b), a reordering of the transformer sublayers, performs better on language modeling. Input flows from left to right.

| Model | PPL |
|---|---|
|  | 20.74 |
|  | 20.64 |
|  | 20.33 |
|  | 20.27 |
|  | 19.98 |
|  | 19.92 |
|  | 19.69 |
|  | 19.54 |
|  | **19.13** |
|  | 19.08 |
|  | 18.90 |
|  | **18.83** |
|  | 18.83 |
|  | 18.77 |
|  | 18.68 |
|  | 18.64 |
|  | 18.61 |
|  | 18.60 |
|  | 18.55 |
|  | **18.54** |
|  | **18.49** |
|  | 18.38 |
|  | 18.28 |
|  | **18.25** |
|  | 18.19 |

Ofir Press, Noah A. Smith, Omer Levy. Improving Transformer Models by Reordering their Sublayers.
https://browse.arxiv.org/pdf/1911.03864.pdf

# What if position embedding is removed or changed?

Table 3: Experiments on GLUE. The evaluation metrics are following the official GLUE benchmark (Wang et al., 2018). The best performance of each task is bold.

| PEs | single sentence | | sentence pair | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CoLA acc | SST-2 acc | MNLI acc | MRPC F1 | QNLI acc | QQP F1 | RTE acc | STS-B spear. cor. | WNLI acc | mean ± std |
| BERT without PE | 39.0 | 86.5 | 80.1 | 86.2 | 83.7 | 86.5 | 63.0 | 87.4 | 33.8 | 76.6 ± 0.41 |
| fully learnable (BERT-style) APE | 60.2 | **93.0** | 84.8 | 89.4 | 88.7 | 87.8 | **65.1** | **88.6** | 37.5 | 82.2 ± 0.30 |
| fixed sin. APE | 57.1 | 92.6 | 84.3 | 89.0 | 88.1 | 87.5 | 58.4 | 86.9 | 45.1 | 80.5 ± 0.71 |
| learnable sin. APE | 56.0 | 92.8 | 84.8 | 88.7 | 88.5 | 87.7 | 59.1 | 87.0 | 40.8 | 80.6 ± 0.29 |
| fully-learnable RPE | 58.9 | 92.6 | 84.9 | **90.5** | 88.9 | **88.1** | 60.8 | 88.6 | 50.4 | 81.7 ± 0.31 |
| fixed sin. RPE | 60.4 | 92.2 | 84.8 | 89.5 | 88.8 | 88.0 | 62.9 | 88.1 | 45.1 | 81.8 ± 0.53 |
| learnable sin. RPE | 60.3 | 92.6 | **85.2** | 90.3 | **89.1** | **88.1** | 63.5 | 88.3 | 49.9 | 82.2 ± 0.40 |
| fully learnable APE + fully-learnable RPE | 59.8 | 92.8 | 85.1 | 89.6 | 88.6 | 87.8 | 62.5 | 88.3 | **51.5** | 81.8 ± 0.17 |
| fully learnable APE + fixed sin. RPE | 59.2 | 92.4 | 84.8 | 89.9 | 88.8 | 87.9 | 61.0 | 88.3 | 48.2 | 81.5 ± 0.20 |
| fully learnable APE+ learnable sin. RPE | **61.1** | 92.8 | **85.2** | **90.5** | 89.5 | 87.9 | **65.1** | 88.2 | 49.6 | **82.5** ± 0.44 |
| learnable sin. APE + fully-learnable RPE | 57.2 | 92.7 | 84.8 | 88.9 | 88.5 | 87.8 | 58.6 | 88.0 | 51.3 | 80.8 ± 0.44 |
| learnable sin. APE + fixed sin. RPE | 57.6 | 92.6 | 84.5 | 88.8 | 88.6 | 87.6 | 63.1 | 87.4 | 48.7 | 81.3 ± 0.43 |
| learnable sin. APE + learnable sin. RPE | 57.7 | 92.7 | 85.0 | 89.6 | 88.7 | 87.8 | 62.3 | 87.5 | 50.1 | 81.4 ± 0.33 |

Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, Jakob Grue Simonsen. On Position Embeddings in BERT. https://openreview.net/pdf?id=onxoVA9FxMw ICLR 2021.

# Improvements for Norm

## DeepNet - 1000 layer Transformers

A new normalization function (DEEPNORM) is introduced
[replacing it is not Layer Norm! Instead, modify it similarly to:

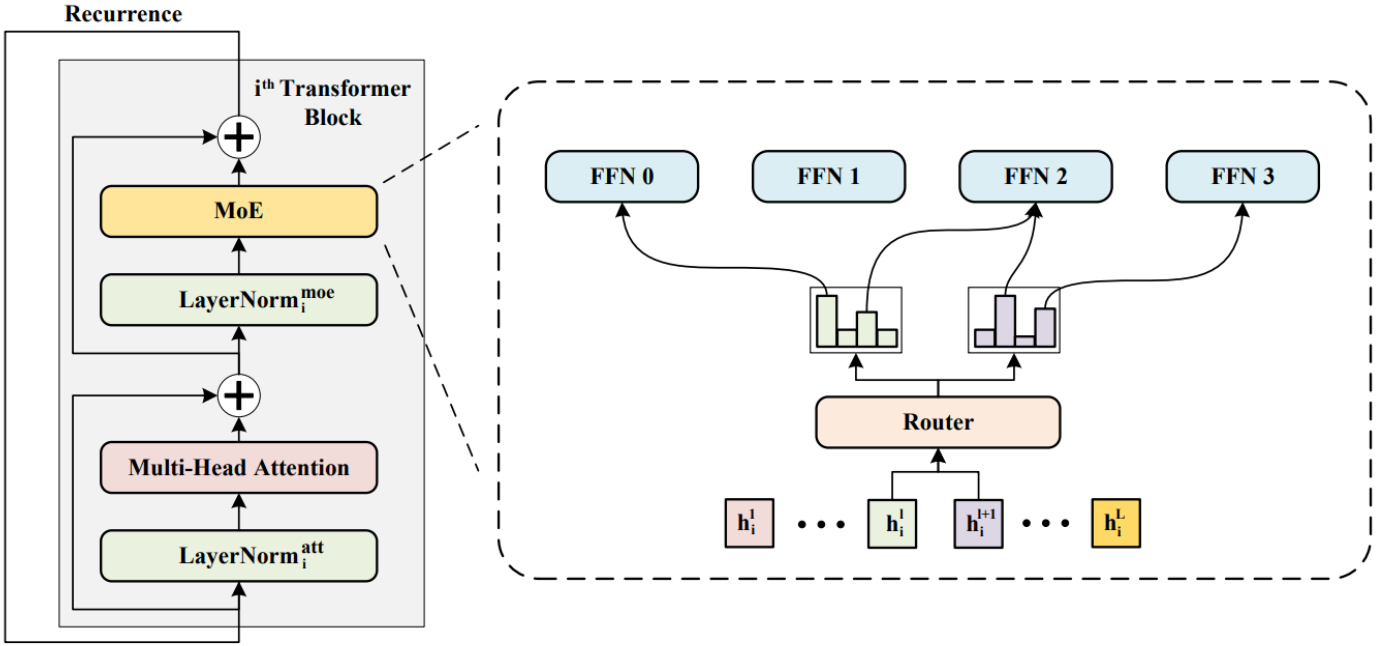
**layernorm (x + f(x)) ---> layernorm(x*alpha + f(x)).**

The proposed method combines the advantages of both schools, namely the good performance of **Post-LN** and the stable training of **Pre-LN**, making DEEPNORM the preferred alternative.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, Furu Wei. DeepNet: Scaling Transformers to 1,000 Layers. https://browse.arxiv.org/pdf/2203.00555.pdf

# Is the model deeper or wider?

Go Wider Instead of Deeper




- WideNet first compresses trainable parameters along with depth by parameter-sharing across transformer blocks.
- Each expert requires enough tokens to train.

Fuzhao Xue, Ziji Shi, Futao Wei, Yuxuan Lou, Yong Liu, Yang You. **Go Wider Instead of Deeper. https://arxiv.org/abs/2107.11817**

# Contents

- Neural Network in NLP and applications
- Inductive bias of Neural Network for NLP
  Semantic Abstraction and Semantic composition
- MLP, CNN & RNN
- Transformer
- **Scaling law and emergent ability？**

# Scaling law？

# Scaling Law for Neural Language Models

Performance depends strongly on scale! We keep getting better performance as we scale the model, data, and compute up!

**Scaling Laws for Neural Language Models**


**Jared Kaplan** *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

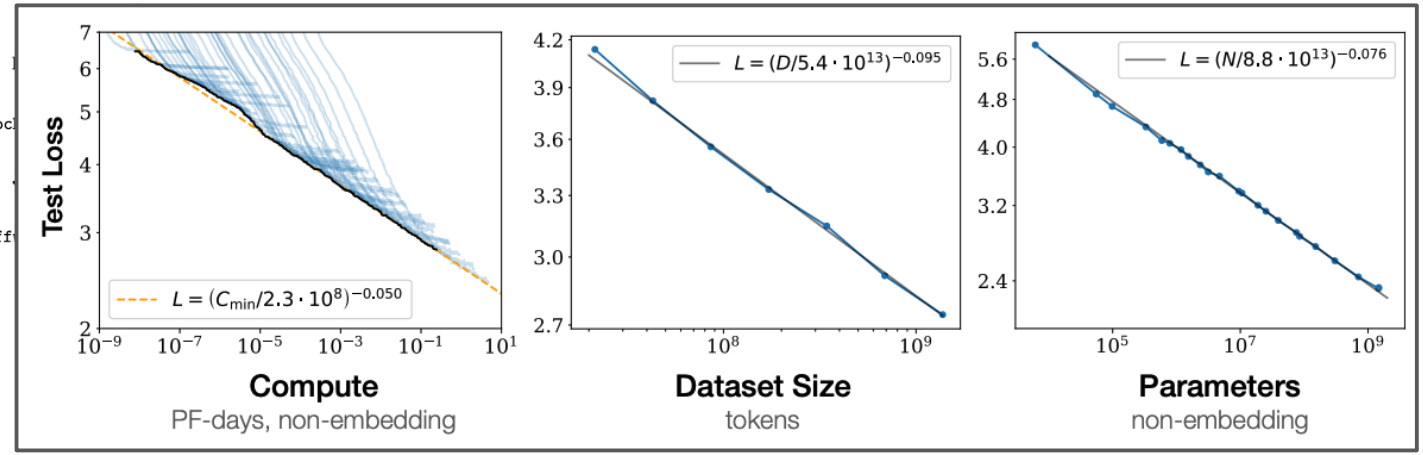
**Sam McCandlish***
OpenAI

**Tom Henighan**
OpenAI
henighan@openai.com

**Tom B. Brown**
OpenAI
tom@openai.com

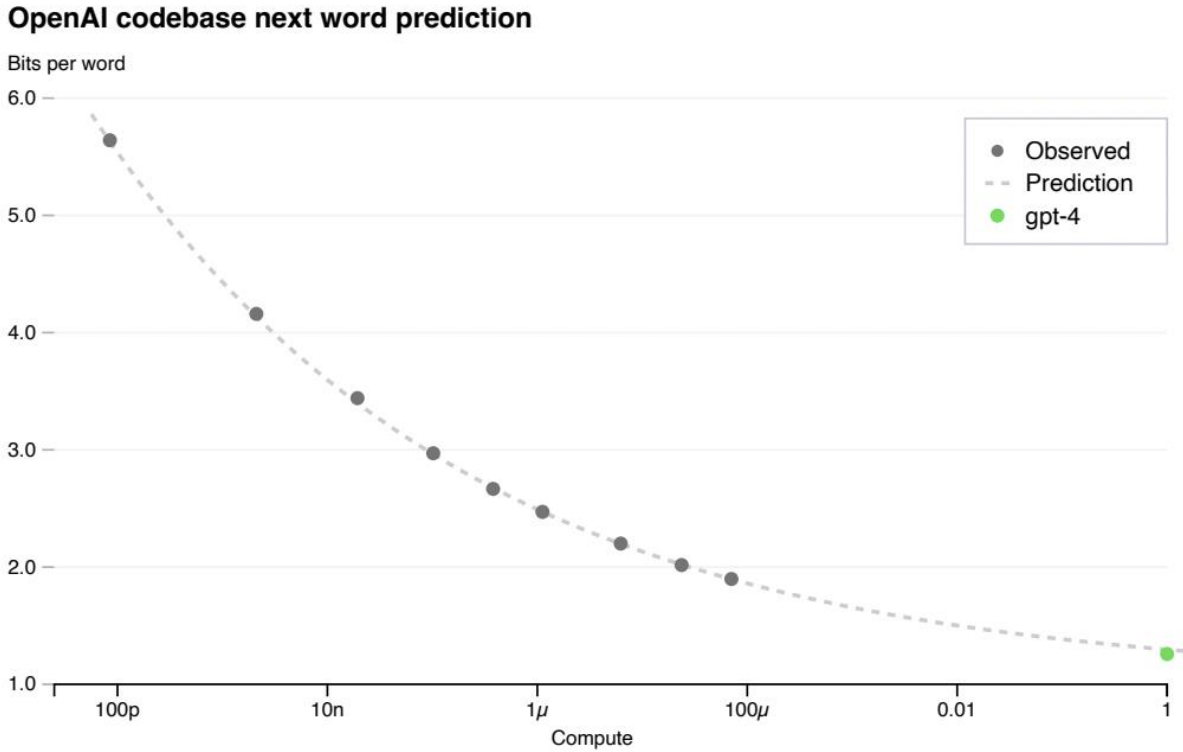
**Scott Gray**
OpenAI
scott@openai.com

**Alec Radford**
OpenAI
alec@openai.com

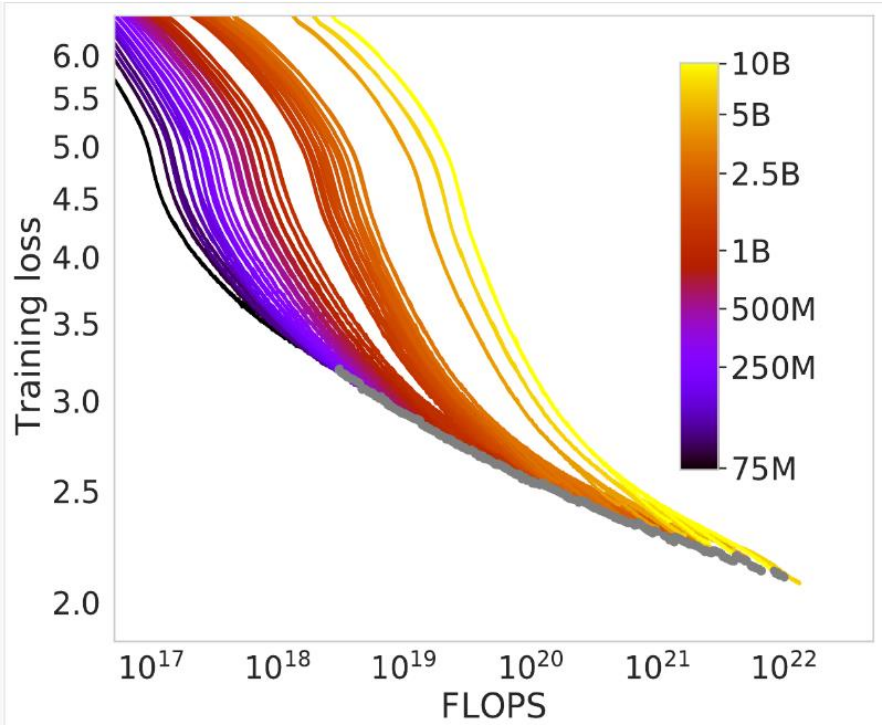

Emergent abilities of large language models (TMLR ‘22).
J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, & W. Fedus.

# Scaling laws




*GPT-4 Technical Report*, *OpenAI (2023)*

# Challenge to scaling law: Chinchilla's Death

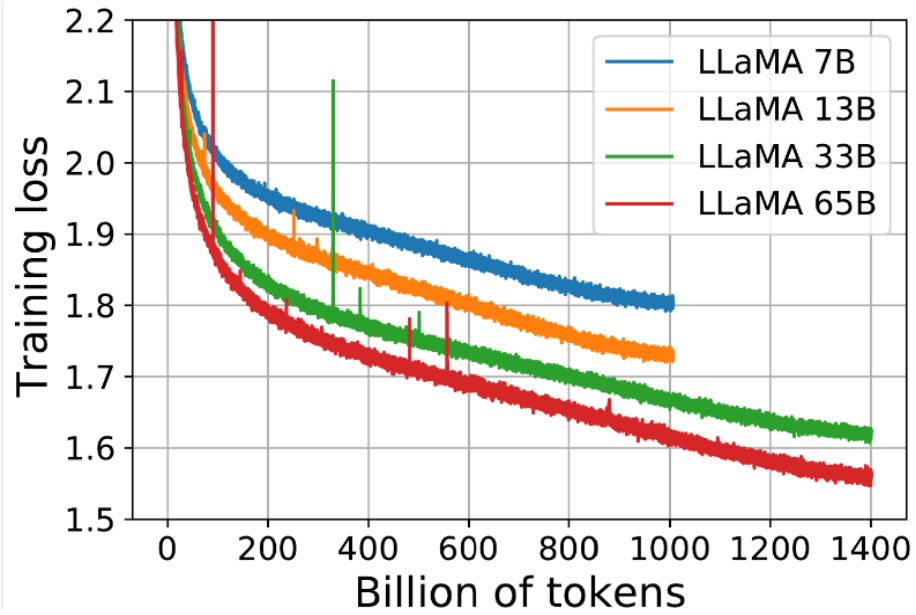



Smaller models eventually reach the limit of their capacity for knowledge, and their learning slows,
while that of a **larger model, with a larger capacity, will overtake them** and reach better performance past a given amount of training time.

https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute
Training Compute-Optimal Large Language Models. https://arxiv.org/abs/2203.15556
https://espadrine.github.io/blog/posts/chinchilla-s-death.html

# Challenge to scaling law: Chinchilla's Death

## Can Chinchillas picture a Llama's sights?

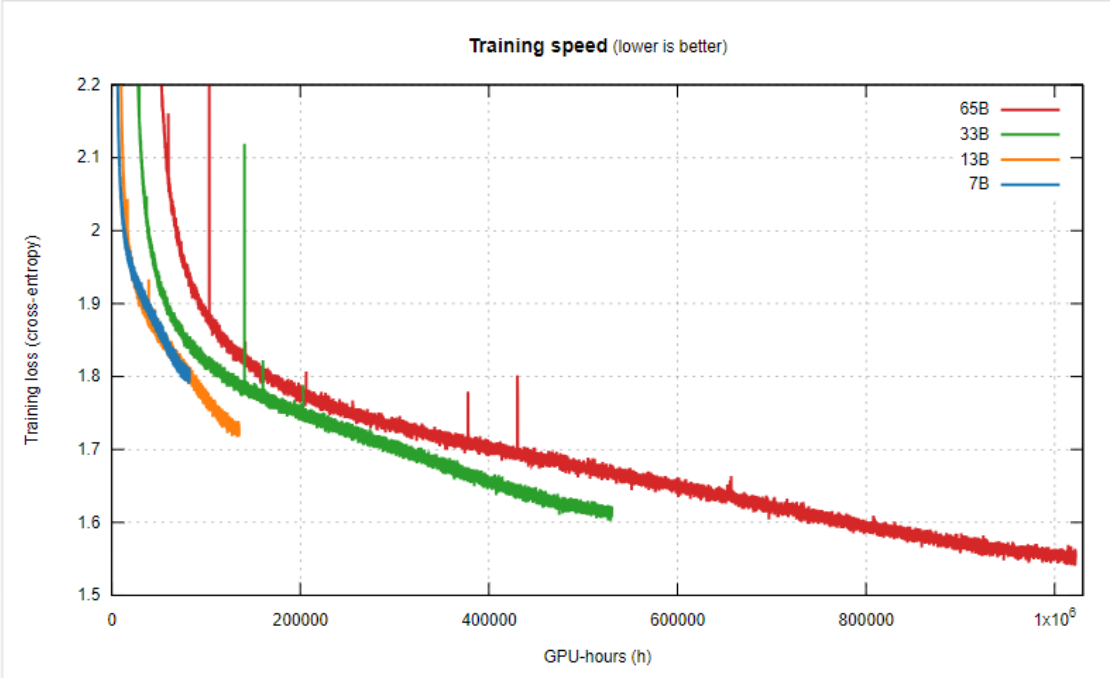



- Each curve first plummets in a **power law**,
- and then seemingly enters a **nearly-linear** decrease in loss (corresponding to a fairly constant rate of knowledge acquisition).
- At the very tip of the curve, they all break this line by **flattening** slightly.

- This should consider the cosine LR schedule.

https://espadrine.github.io/blog/posts/chinchilla-s-death.html

# Challenge to scaling law: Chinchilla's Death




Let's picture instead a race: All those models start at the same time, and we want to know which one crosses the finish line first.

In other words, when throwing a fixed amount of compute at the training, who learns the most in that time?

**the 7B enters a near-linear regime, with a steep downward trend, and seems on its way to maybe overpass the 13B again?**

https://espadrine.github.io/blog/posts/chinchilla-s-death.html

# We should find the **new scaling dimensions**

**~~Parallel~~** ~~scaling~~ [1]:

Sampling multiple times and find the right answers (with verifiers)

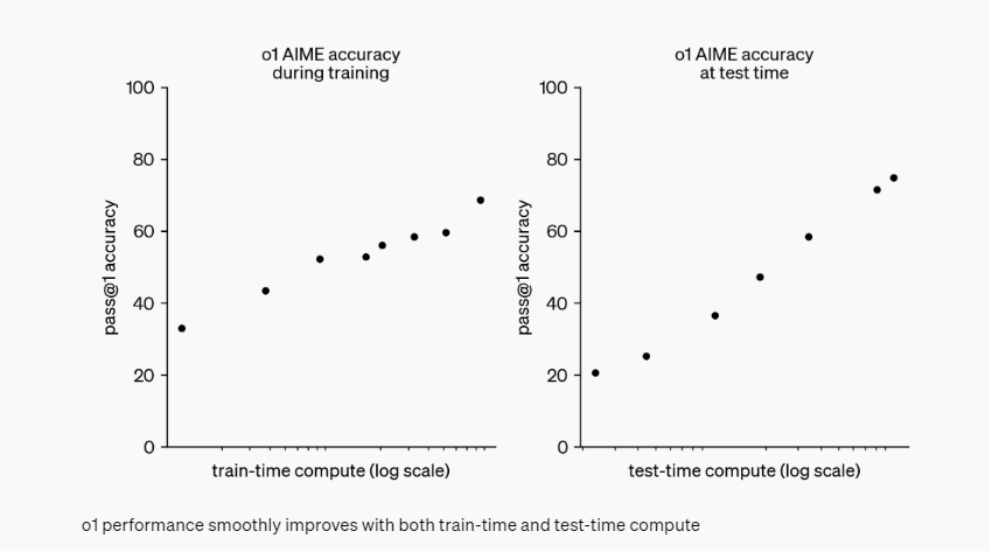
**Sequential** scaling: generating more tokens (thinking longer?)

e.g., Open O1 and DeepSeek R1

[1] Fei Yu, Yingru Li, Benyou Wang. Scaling Flaws of Verifier-Guided Search. https://arxiv.org/abs/2502.00271

# New Scaling law from OpenAI o1

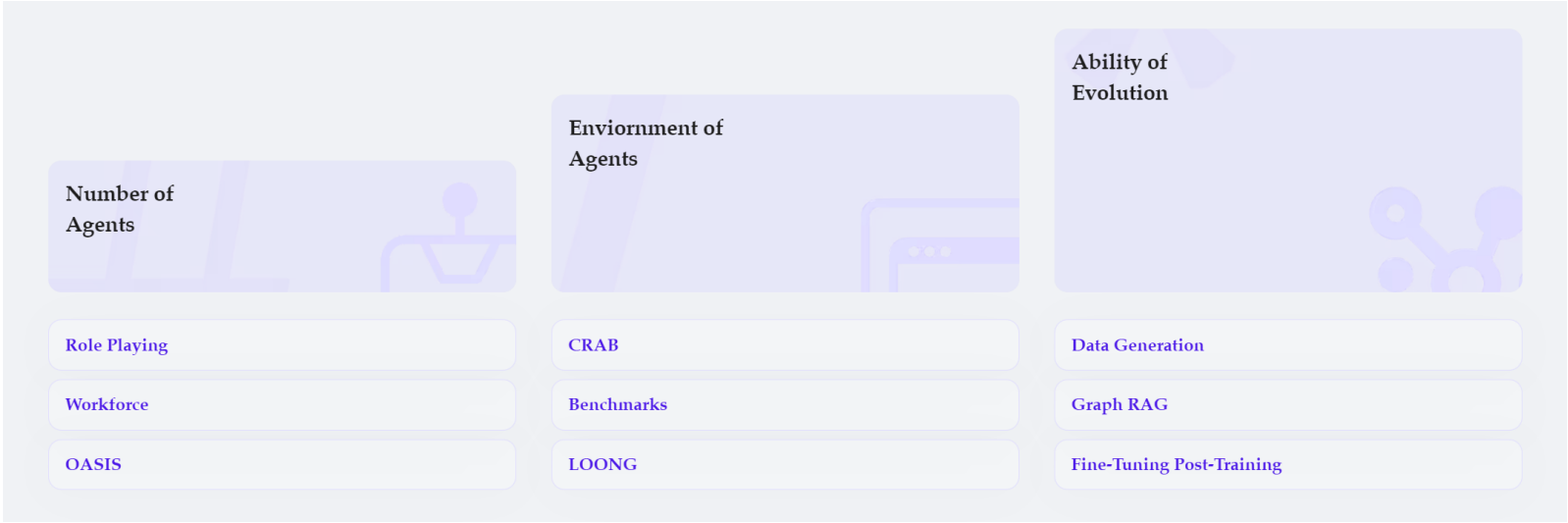
Our **large-scale reinforcement learning** algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with **more reinforcement learning** (train-time compute) and with **more time spent thinking** (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining.




o1 performance smoothly improves with both train-time and test-time compute

https://openai.com/index/learning-to-reason-with-llms/

# Scaling Law of Agents?




Not work that well

**Promising!**
see Embodied AI and robotics

Evolution needs feedback from real-word **environment**.

https://www.camel-ai.org/

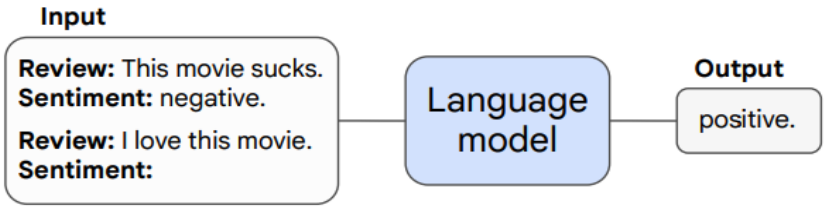
**New Scaling laws** might drive the next-generation AI revolution !

# Emergent ability?

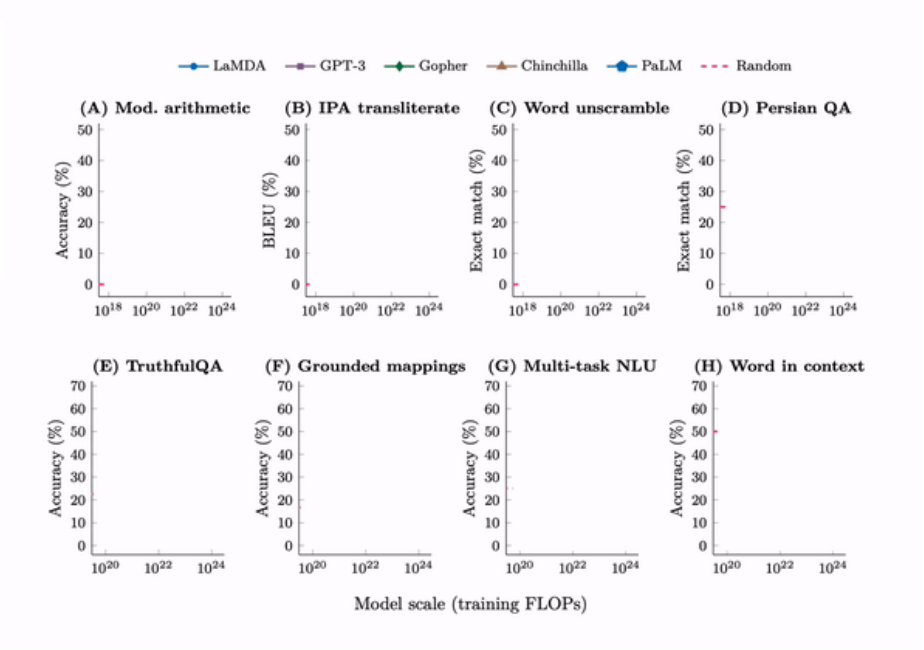
**Emergent properties in LLMs**:

Some ability of LM is not present in smaller models but is present in larger models

Example **Emergent Capability**: Few-shot prompting

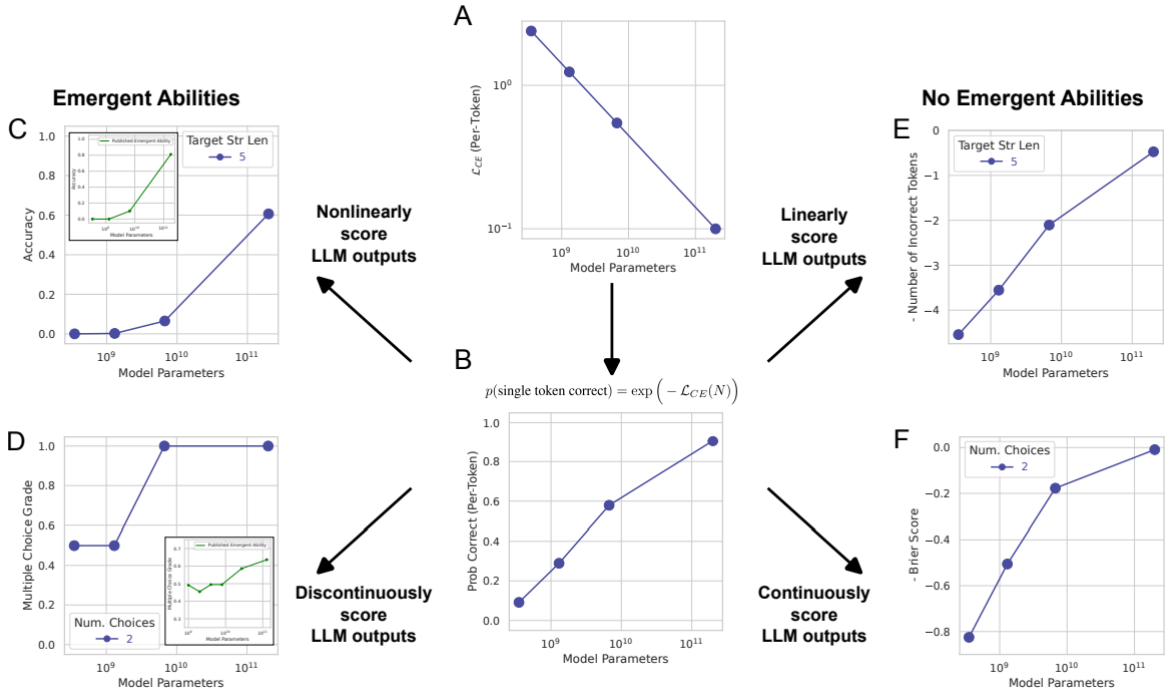



*> A few-shot prompted task is emergent if it achieves random accuracy for small models and above-random accuracy for large models.*




https://docs.google.com/presentation/d/1yzbmYB5E7G8lY2-KzhmArmPYwwl7o7CUST1xRZDUu1Y/edit?resourcekey=0-6_TnUMoK_WCk_FN2BiPxmbw#slide=id.g1fc34b3ac18_0_27

# Emergent capabilities may be a consequence of metric choice




It seems that emergent ability of a model only occurs if the measure of per-token error rate of any model is scaled **non-linearly or discontinuously**.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are Emergent Abilities of Large Language Models a Mirage? https://browse.arxiv.org/pdf/2304.15004.pdf

# Acknowledgement


- Princeton COS 484: Natural Language Processing. Contextualized Word Embeddings. Fall 2019
- CS447: Natural Language Processing. Language Models. *http://courses.engr.illinois.edu/cs447*
- *http://cs231n.stanford.edu/*
- *https://medium.com/@gautam.karmakar/summary-seq2seq-model-using-convolutional-neural-network-b1eb100fb4c4*
- Transformers and sequence- to-sequence learning. CS 685, Fall 2021. Mohit Iyyer. College of Information and Computer Sciences. University of Massachusetts Amherst. https://people.cs.umass.edu/~miyyer/cs685_f21/slides/05-transformers.pdf

# Some Discussions

- About your final reports
  - research track
    - Invite some researchers for suggestions
  - application track
    - Invite VC, startup, and engineers for suggestions
- Assignment
  - Any issues?

# Examples of back-propagation

# Backpropagation: a simple example

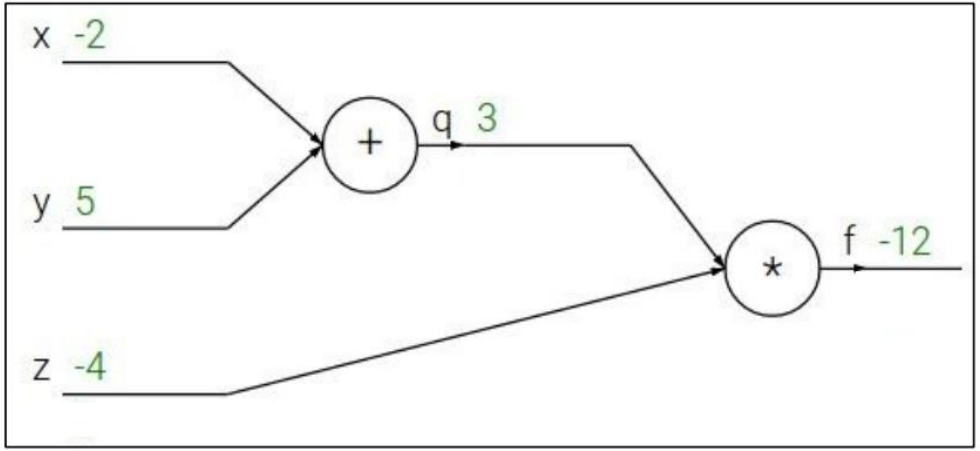
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Backpropagation: a simple example

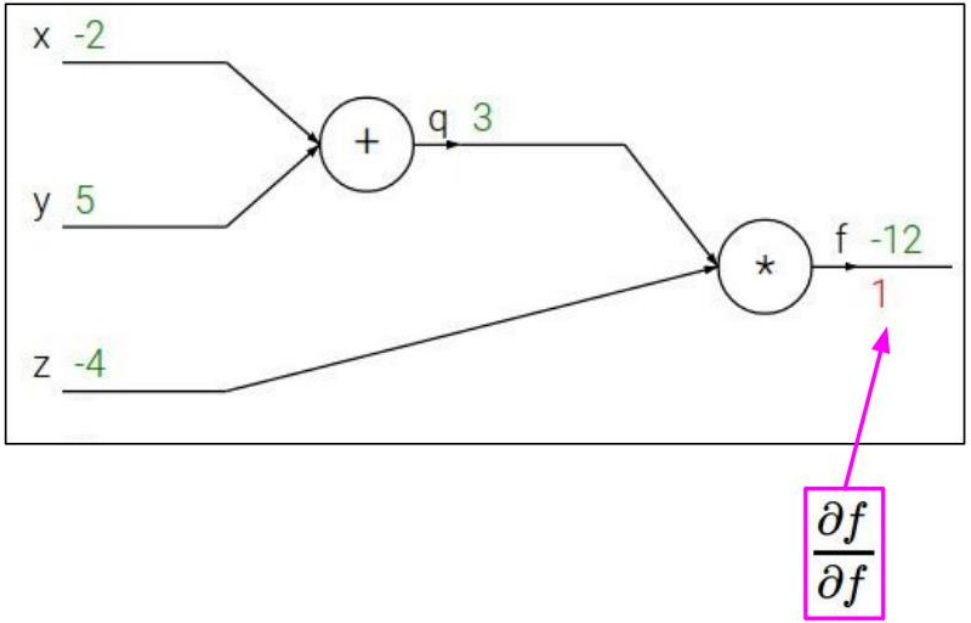
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Backpropagation: a simple example

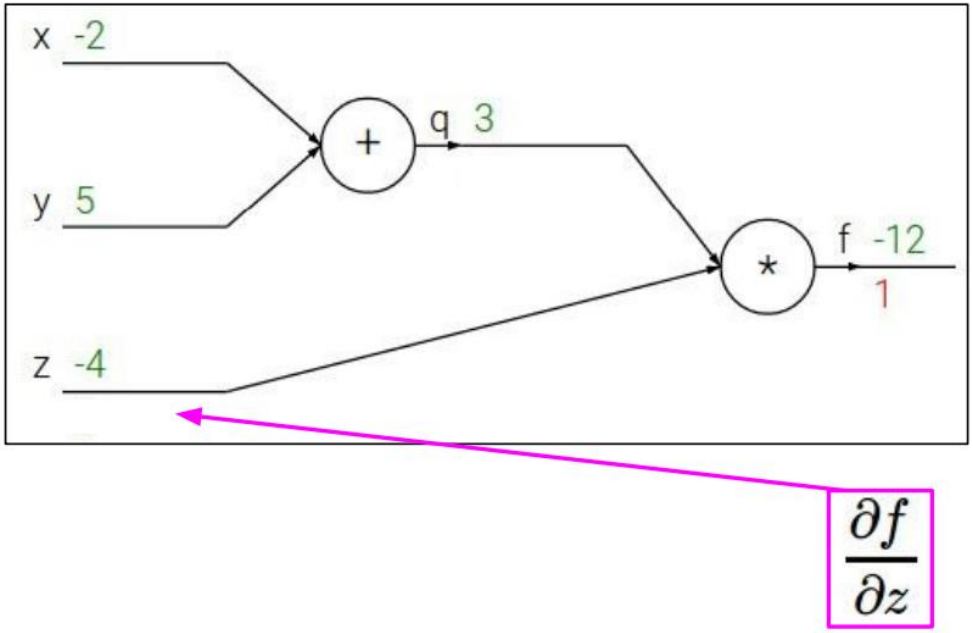
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Backpropagation: a simple example

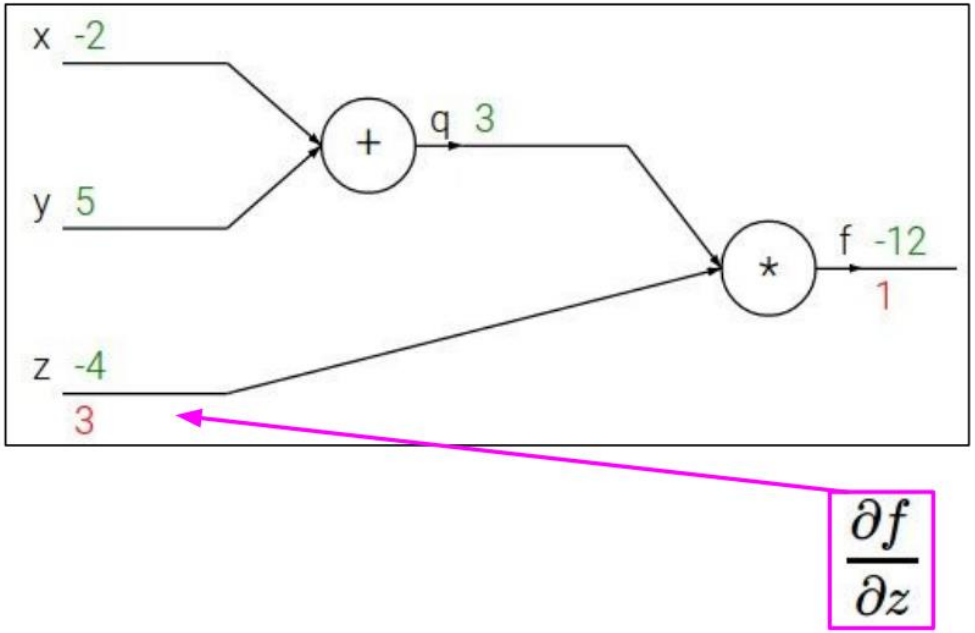
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Backpropagation: a simple example

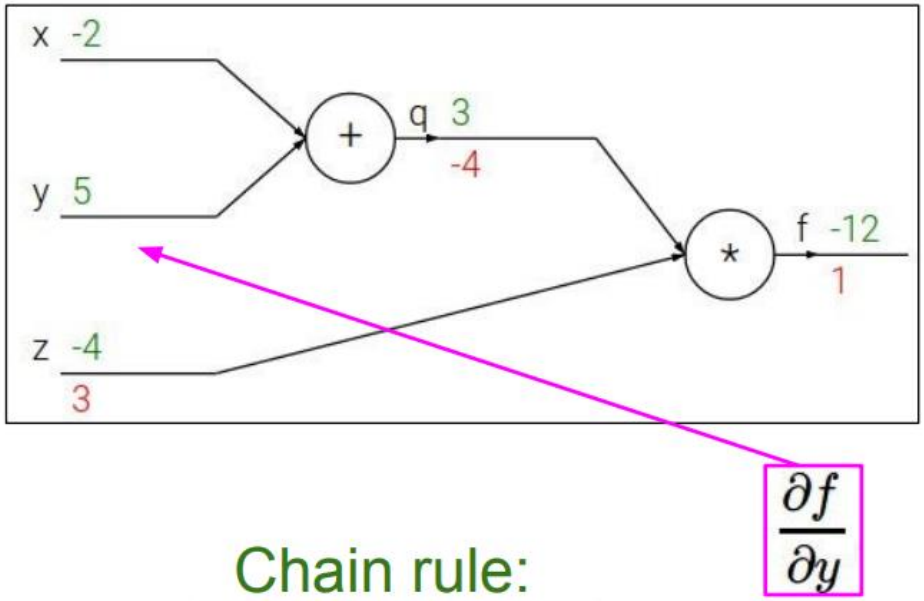
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$




Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

# Backpropagation: a simple example

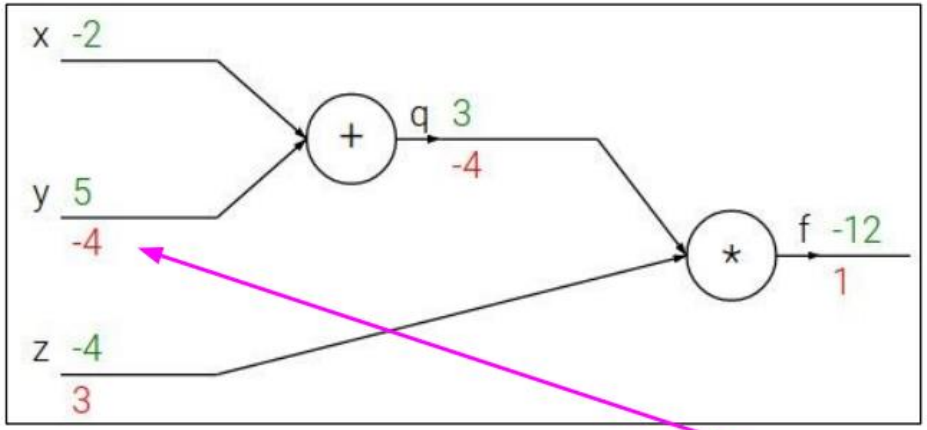
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$




Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

# Backpropagation: a simple example

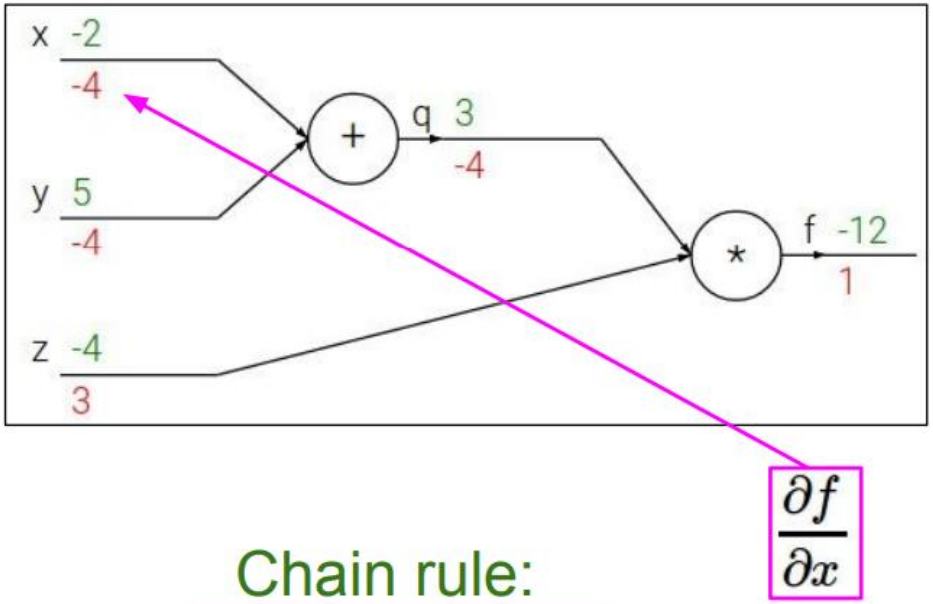
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

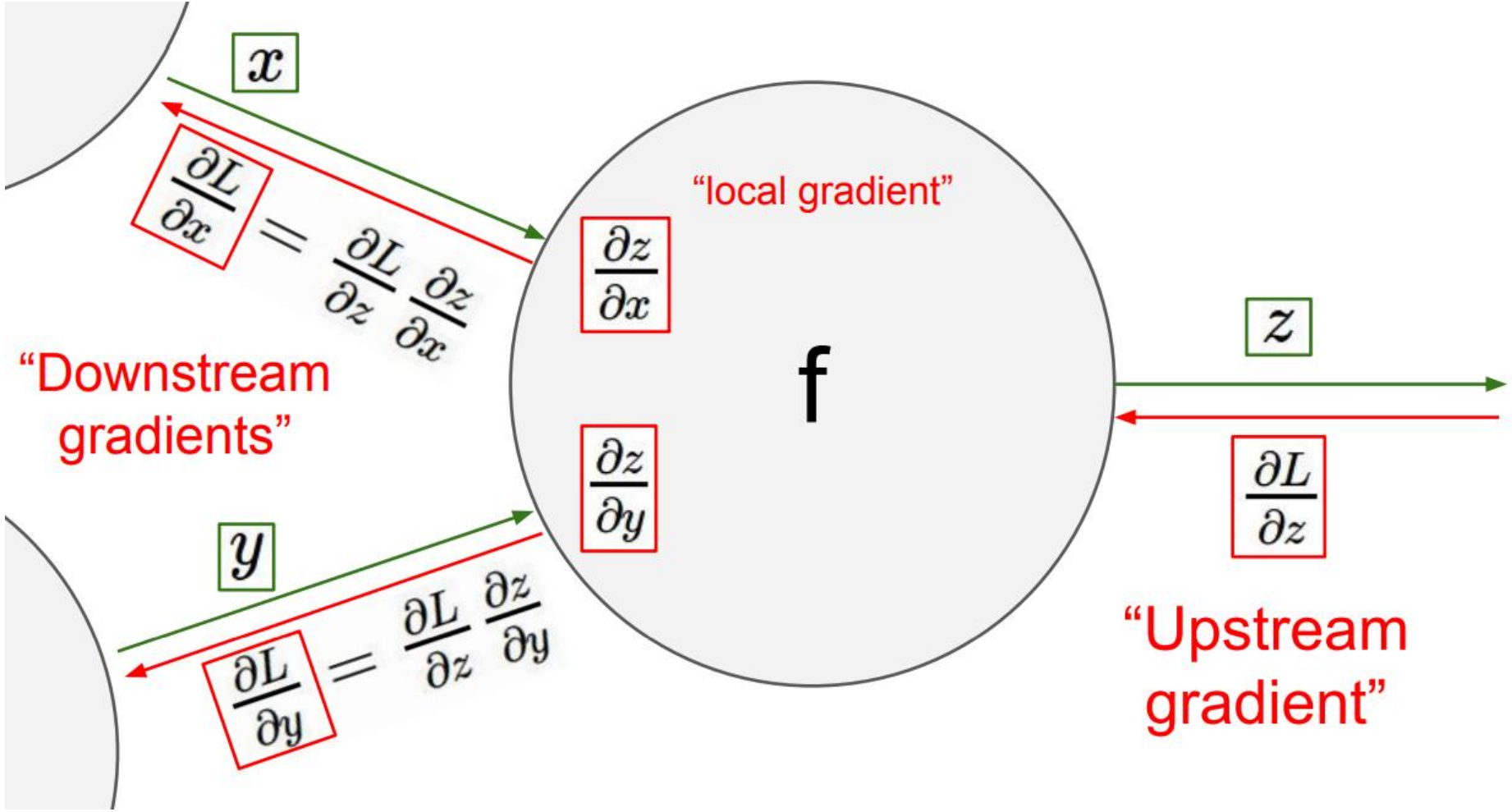



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

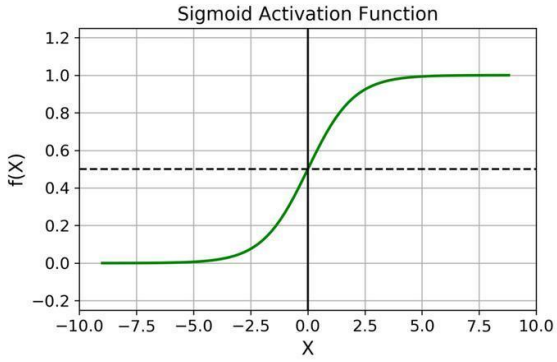
Upstream gradient

Local gradient


$x$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"Downstream gradients"
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
$z$
$\frac{\partial L}{\partial z}$
"Upstream gradient"
$y$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

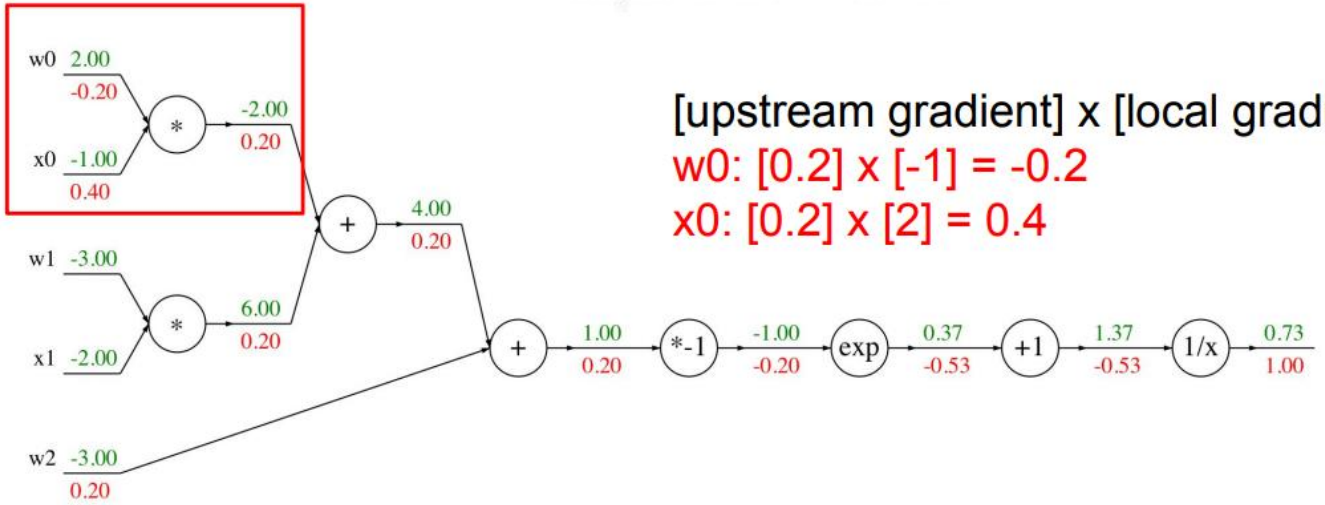
# Another example

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



# Another example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




[upstream gradient] x [local gradient]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

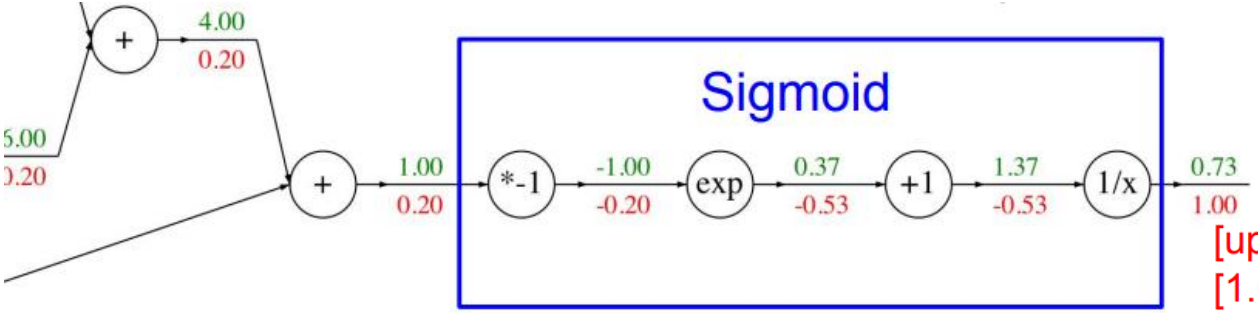
# Another example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

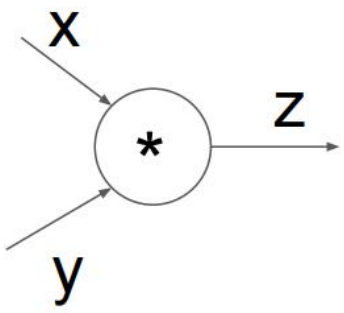



[upstream gradient] x [local gradient]
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

# Modularized implementation: forward / backward API




(x,y,z are scalars)

Gate / Node / Function object: Actual PyTorch code

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

Need to cache some values for use in backward

Upstream gradient

Multiply upstream and local gradients

# PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Forward

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

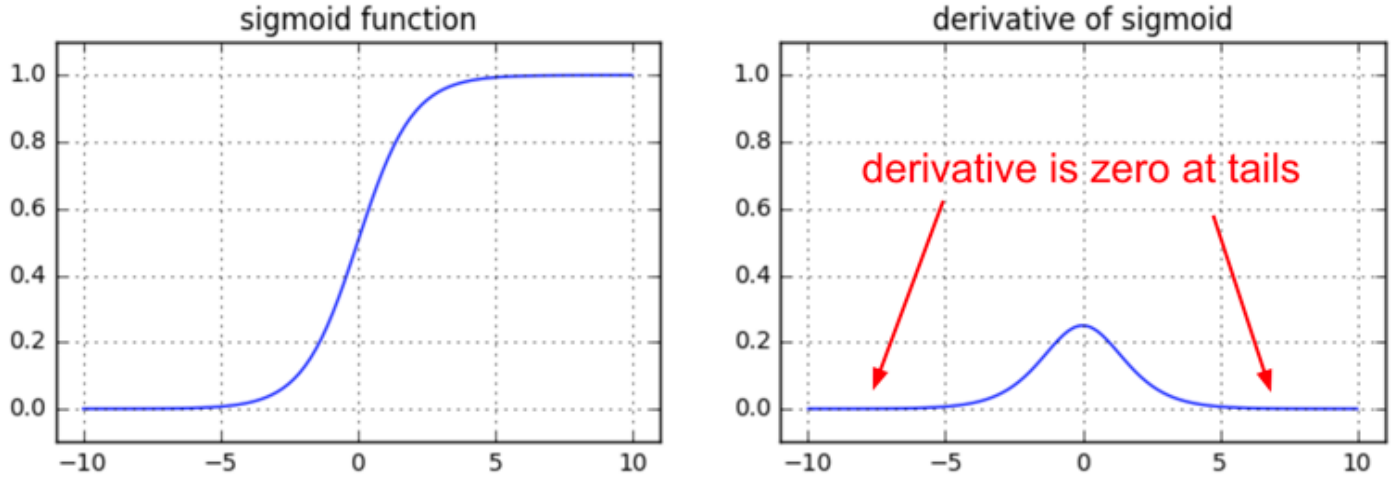
Forward actually defined [elsewhere](https://github.com/pytorch/pytorch/blob/517c7c98610402e2746586c78987c64c28e024aa/aten/src/THNN/generic/Sigmoid.c)...

```
static void sigmoid_kernel(TensorIterator& iter) {
  AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
    unary_kernel_vec(
        iter,
        [=](scalar_t a) -> scalar_t { return (1 / (1 + std::exp((-a)))); },
        [=](Vec256<scalar_t> a) {
          a = Vec256<scalar_t>((scalar_t)(0)) - a;
          a = a.exp();
          a = Vec256<scalar_t>((scalar_t)(1)) + a;
          a = a.reciprocal();
```

Backward

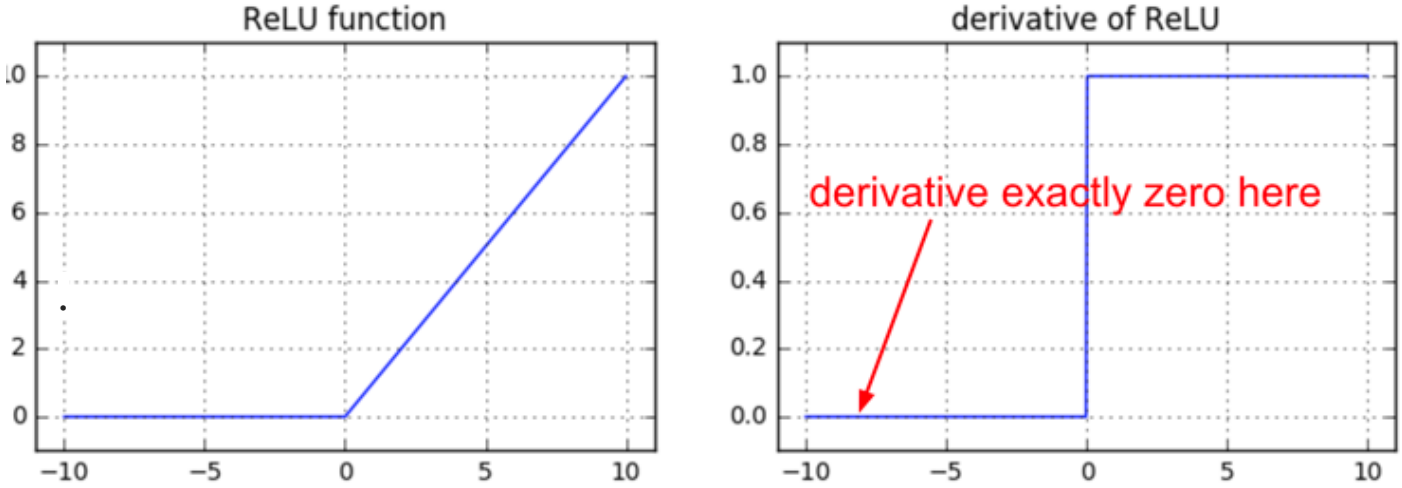
$$(1-\sigma(x))\,\sigma(x)$$

## Trap1: Vanishing gradients on sigmoids




if you're using **sigmoids** or **tanh** non-linearities in your network and you understand backpropagation you should always be nervous about making sure that the initialization doesn't cause them to be fully saturated.

## Trap2: Dying ReLUs




If you understand backpropagation and your network has ReLUs, you're always nervous about dead ReLUs. These are neurons that never turn on for any example in your entire training set and will remain permanently dead. Neurons can also die during training, usually as a symptom of aggressive learning rates.

# Trap3: Exploding gradients in RNNs

```python
H = 5    # dimensionality of hidden state
T = 50   # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

if the largest eigenvalue is > 1, gradient will explode
if the largest eigenvalue is < 1, gradient will vanish

If you understand backpropagation and you're using RNNs you are nervous about having to do gradient clipping, or you prefer to use an LSTM.