

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

CSC5051 / MDS5110 / CSC6052
Natural Language Processing

Lecture 4: Transformer and Beyond

Fall 2026

Benyou Wang

School of Data Science

Language Models: Recap

Example of a Bigram Language 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?'

$P(of \mid tired) = 1$
$P(of \mid use) = 1$
$P(sister \mid her) = 1$
$P(beginning \mid was) = 1/2$
$P(reading \mid was) = 1/2$

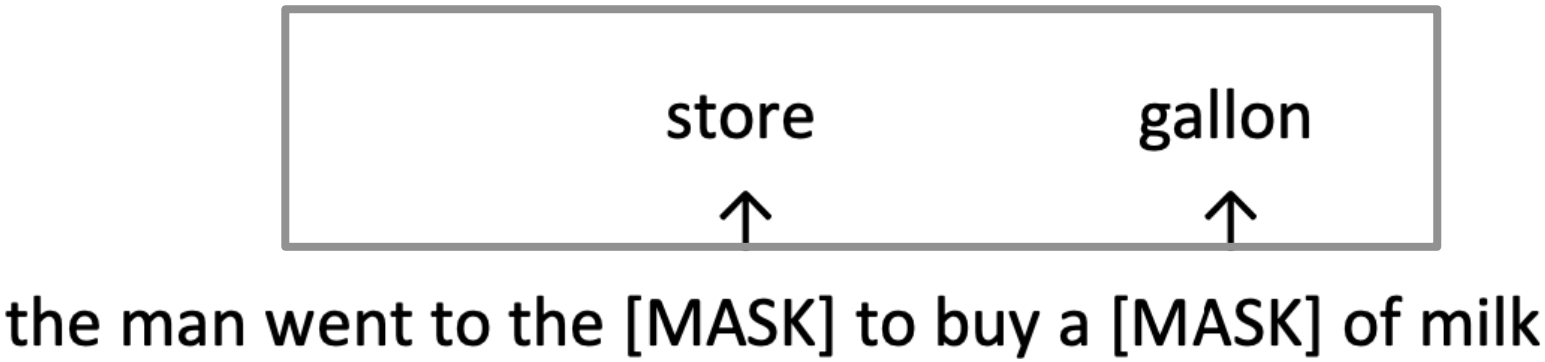
$P(bank \mid the) = 1/3$
$P(book \mid the) = 1/3$
$P(use \mid the) = 1/3$

Generating Shakespeare

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • 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. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •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. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • 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; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

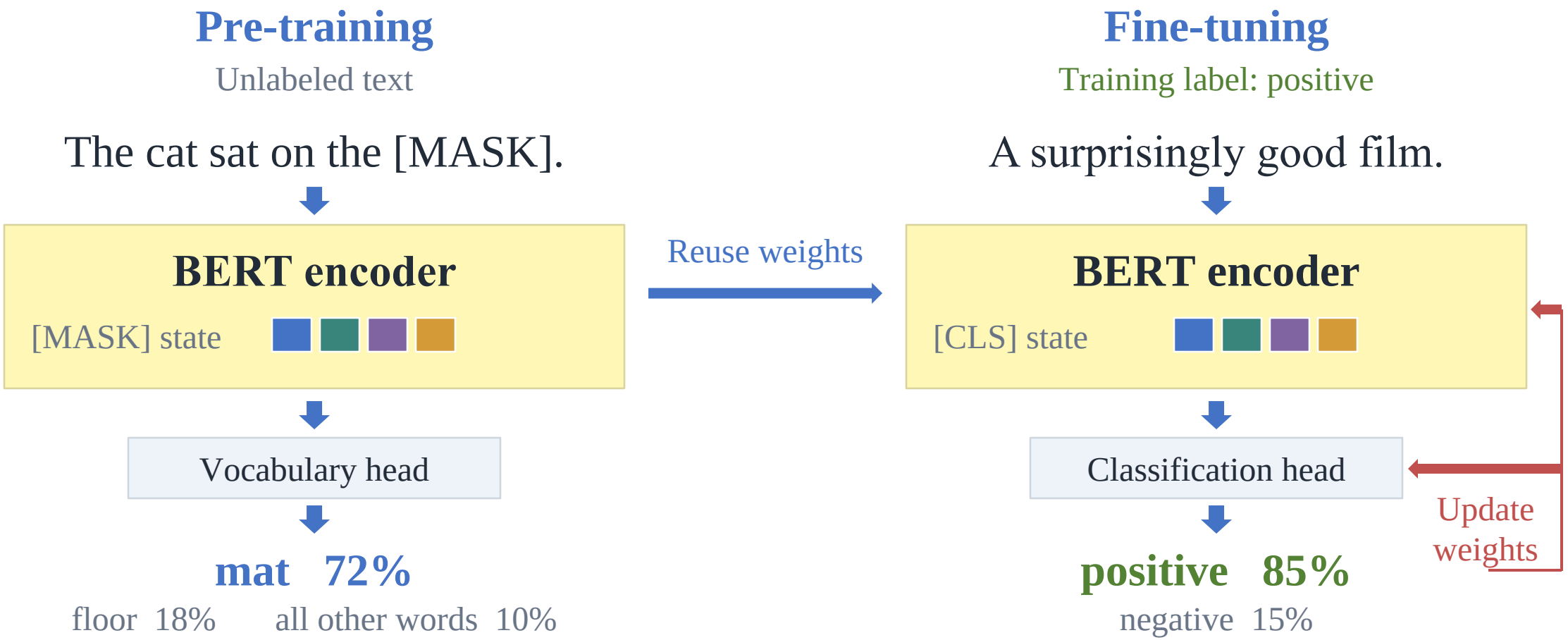
Masked Language Models

Mask 15% of the input words and predict the masked words.



Pre-training and Fine-tuning

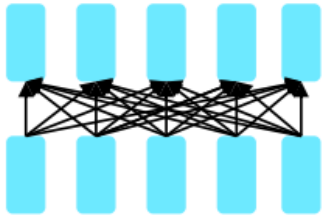
One encoder learns language patterns, then adapts to a labeled task.



Illustrative probabilities. Fine-tuning updates the encoder and its task head together.

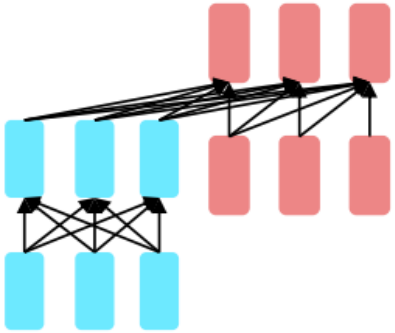
Three Pre-training Architectures

Architecture determines the available context and the pre-training objective.



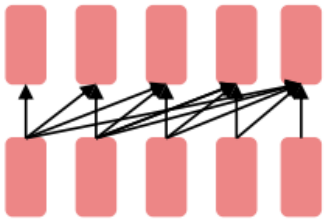
Encoders

Bidirectional context
Representation learning



Encoder-Decoders

Source encoding and target generation
Conditional pre-training



Decoders

Causal context
Autoregressive language modeling

Lecture Roadmap

1 NLP tasks and representations

2 Inductive bias: semantic abstraction and composition

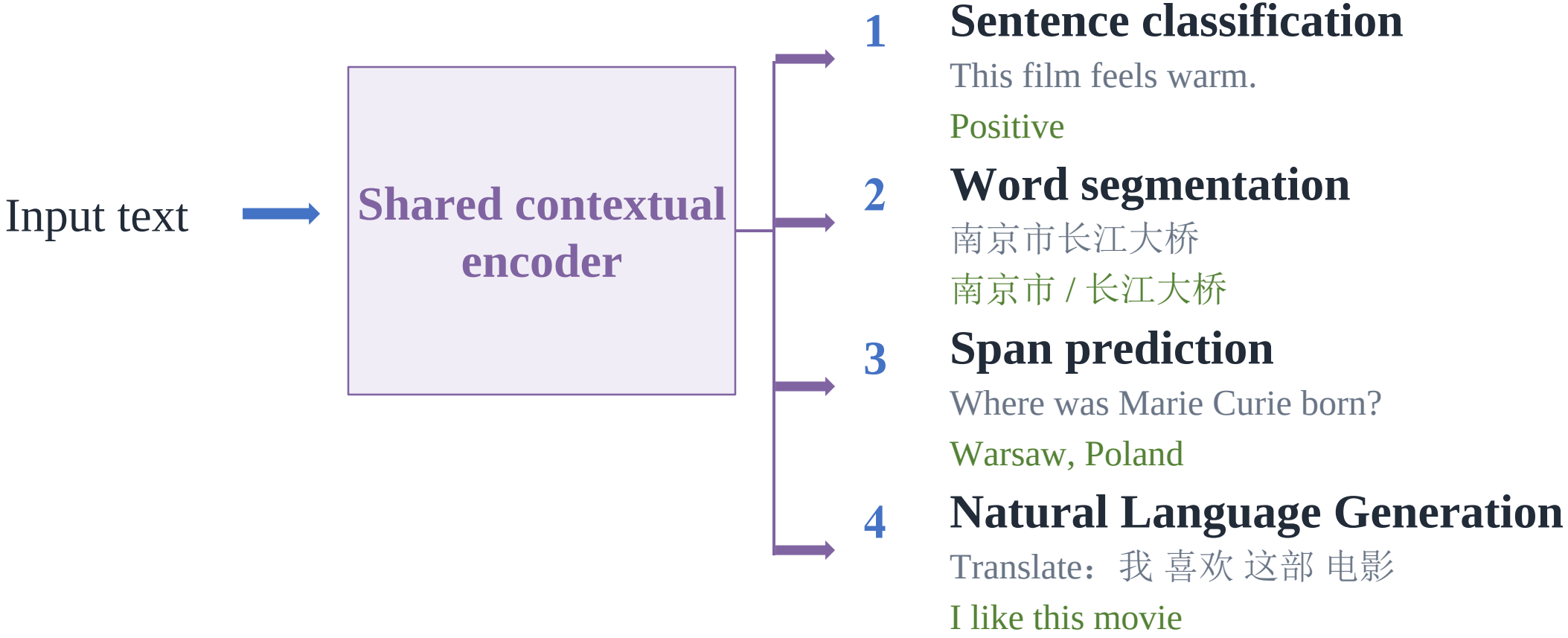
3 Architectures for Neural representations

- 3.1 Multilayer perceptrons
- 3.2 Convolutional networks
- 3.3 Recurrent networks
- 3.4 Transformer

4 Scaling laws and emergence

Typical NLP Applications

Four output structures, four different prediction problems.



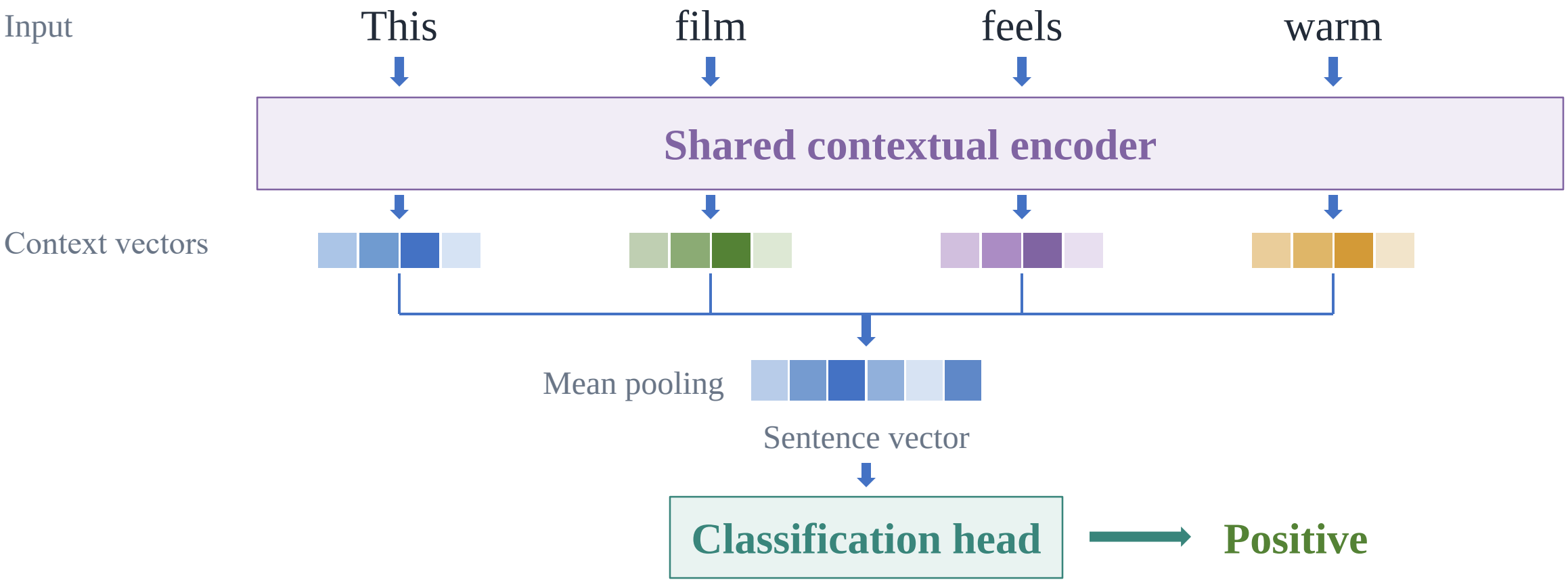
A task head or decoder turns contextual representations into the required output.

Reading: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Sentence-level Classification

TASK 1

One sentence produces one class label.



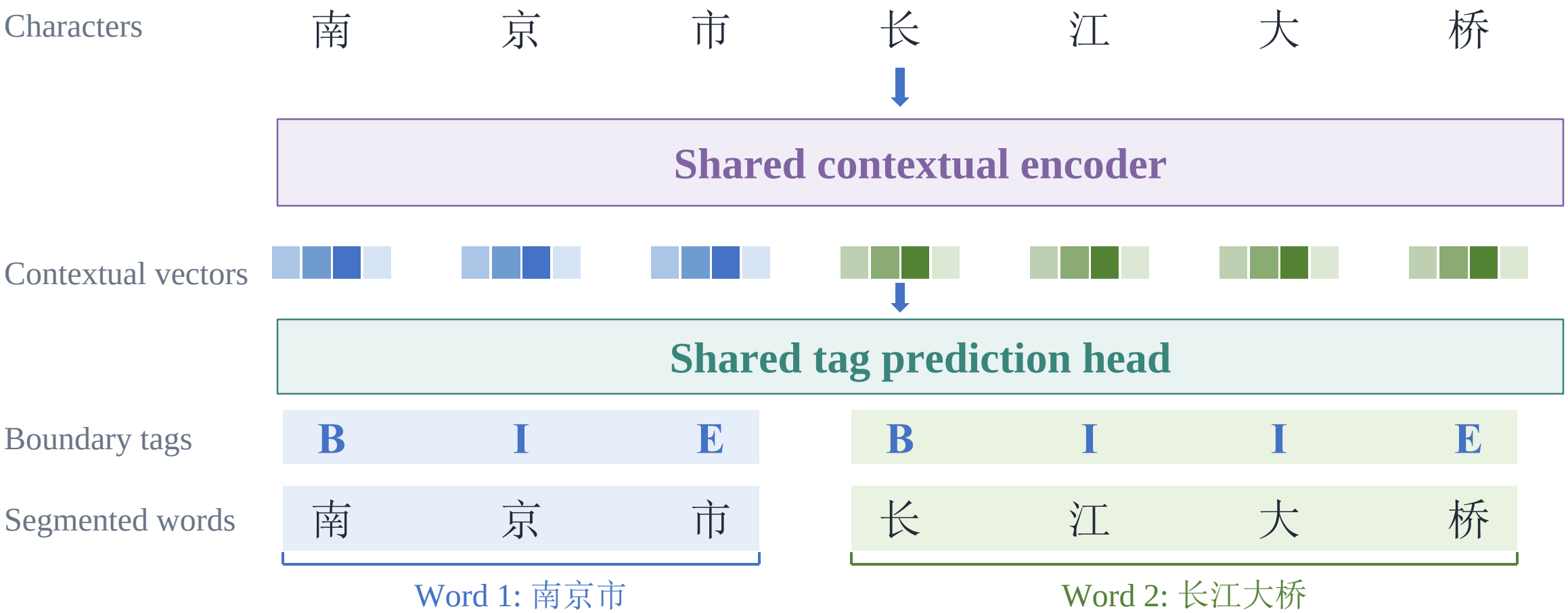
Pooling combines the token axis. The head predicts scores for the sentiment classes.

Reading: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Word Segmentation

TASK 2

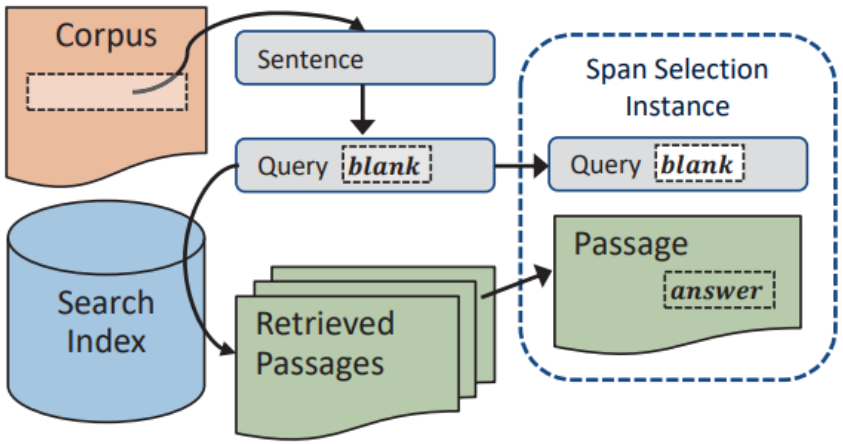
Predict a boundary tag for each character.



B = beginning I = inside E = ending S = a single-character word

Reading: Chinese Word Segmentation as Character Tagging

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

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

Span Prediction

TASK 3

Where was Marie Curie born?

Wikipedia evidence

1	2	3	4	5	6	7
Marie	Curie	was	born	in	Warsaw,	Poland

Shared contextual encoder

Start head

6 Warsaw: 0.60

7 Poland: 0.25

End head

6 Warsaw: 0.25

7 Poland: 0.65

Candidate spans	End 6: Warsaw	End 7: Poland
Start 6: Warsaw	Warsaw 0.60 × 0.25 = 0.1500	Warsaw, Poland 0.60 × 0.65 = 0.3900 (best valid)
Start 7: Poland	Invalid: start > end 0.25 × 0.25 = 0.0625 → discard	Poland 0.25 × 0.65 = 0.1625

Top two boundaries per head shown. Rank valid pairs by P(start) × P(end).

Reading: SQuAD: 100,000+ Questions for Machine Comprehension of Text

Translation: Generate One Token at a Time

TASK 4

Predict from the current prefix, then append the new token.

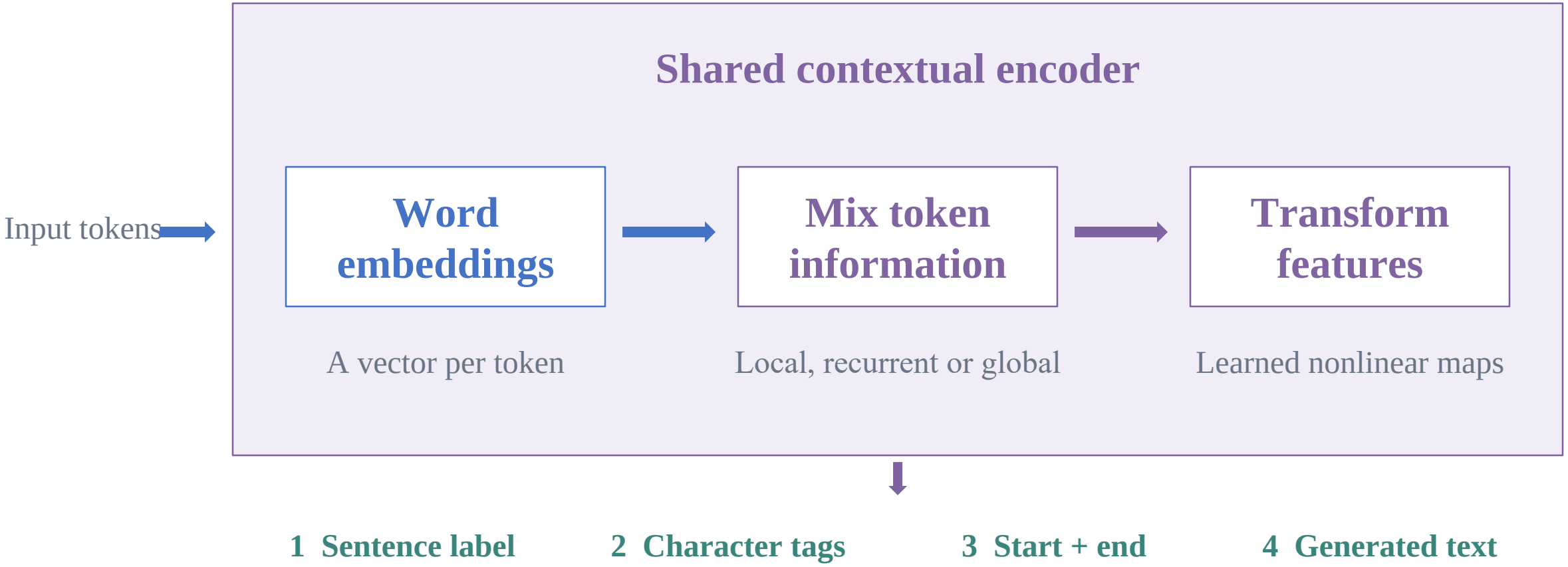


Every prediction uses the source context and only the target tokens already generated.

Reading: Convolutional Sequence to Sequence Learning

Building a Contextual Encoder

The architecture determines how tokens exchange information.

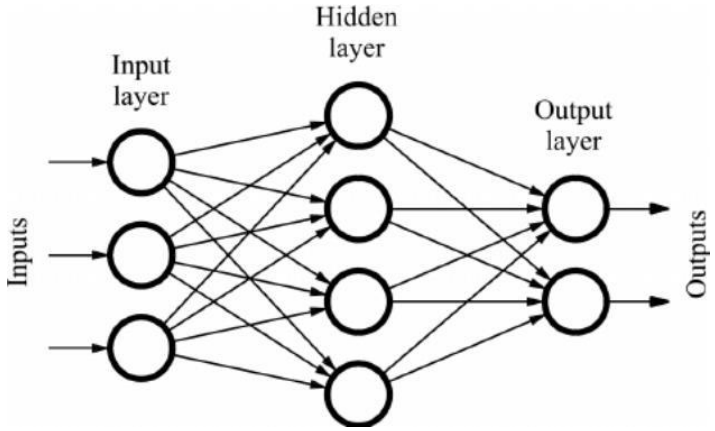


MLPs, CNNs, RNNs and Transformers use different rules for building these representations.

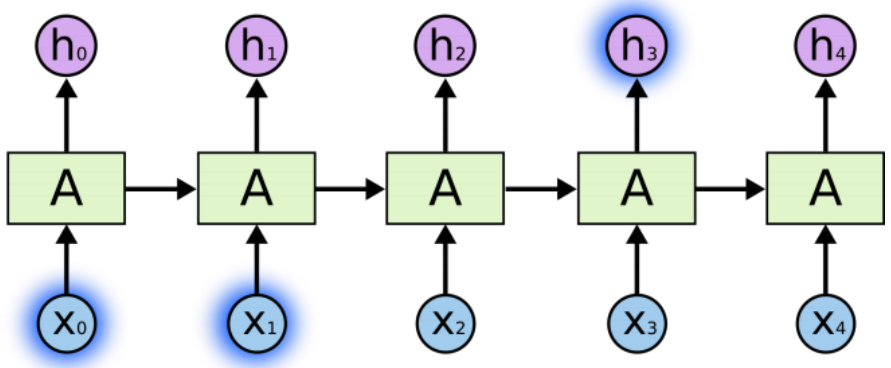
Reading: A Neural Probabilistic Language Model

Neural Network Architectures

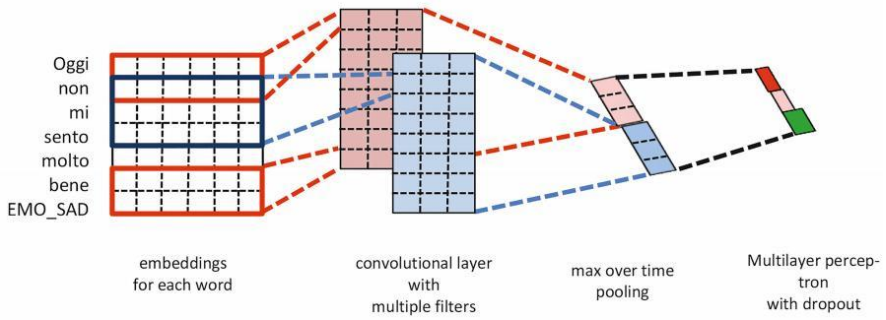
Multilayer perceptron



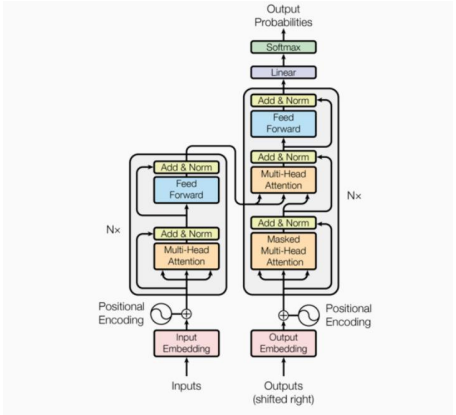
Recurrent network



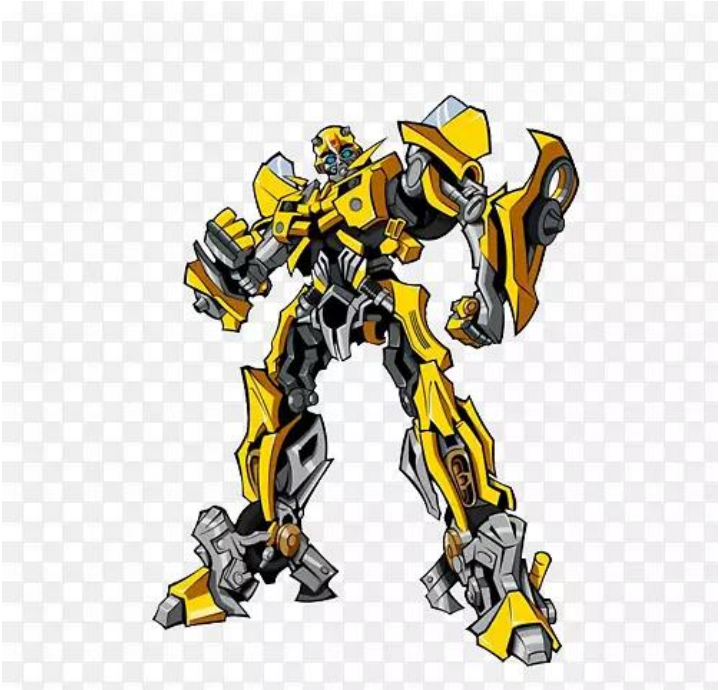
Convolutional network

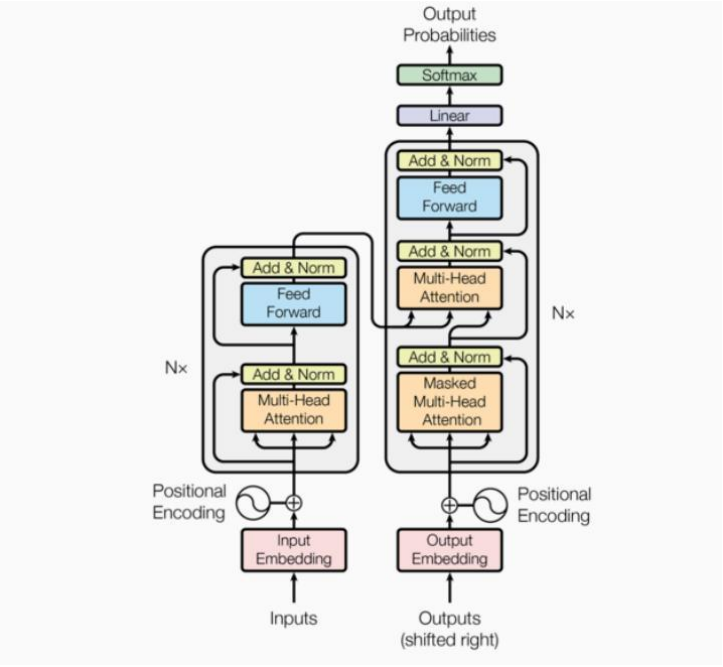


Transformer



Why Transformer?





Lecture Roadmap

1 NLP tasks and representations

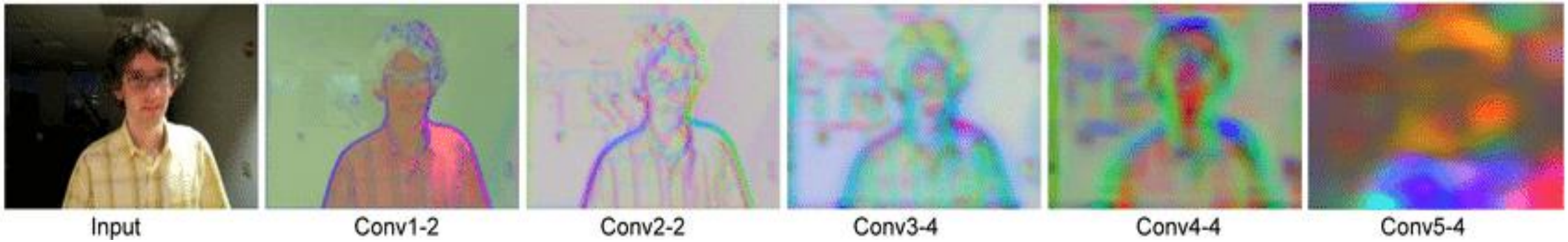
2 Inductive bias: semantic abstraction and composition

3 Architectures for Neural representations

- 3.1 Multilayer perceptrons
- 3.2 Convolutional networks
- 3.3 Recurrent networks
- 3.4 Transformer

4 Scaling laws and emergence

Semantic Abstraction



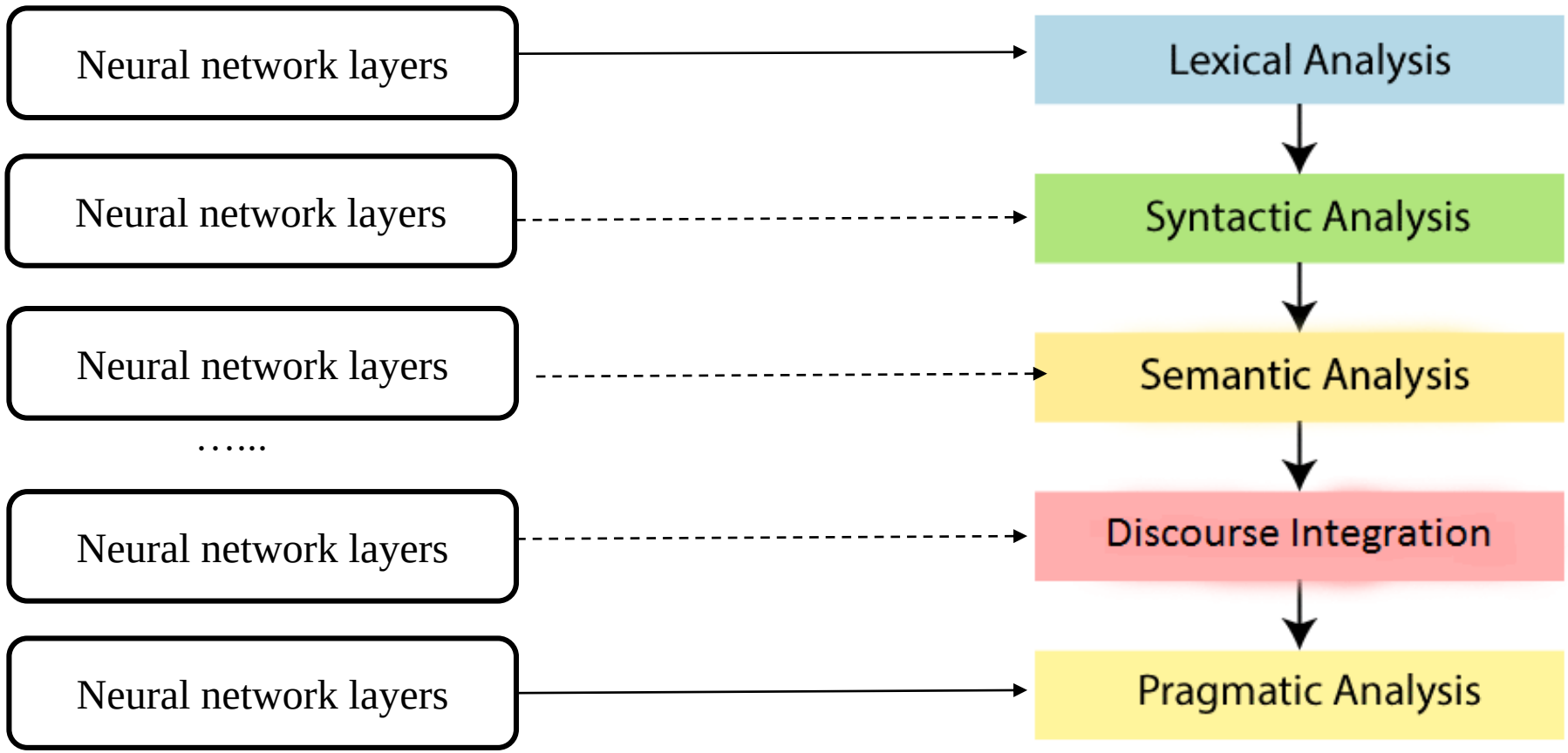
Pixels, textures, regions, objects, relations and semantics

像素、纹理、区域、物体、关系与语义

Ma et al. (2017), Robust Visual Tracking via Hierarchical Convolutional Features

Abstraction Across Network Layers

Input: I think therefore I



Output: am

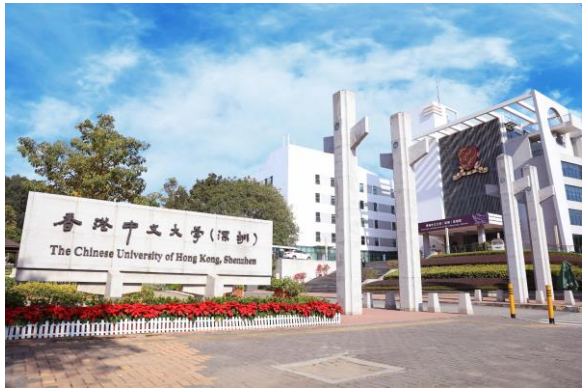
Semantic Composition



Ivory (象牙)



Tower (塔)



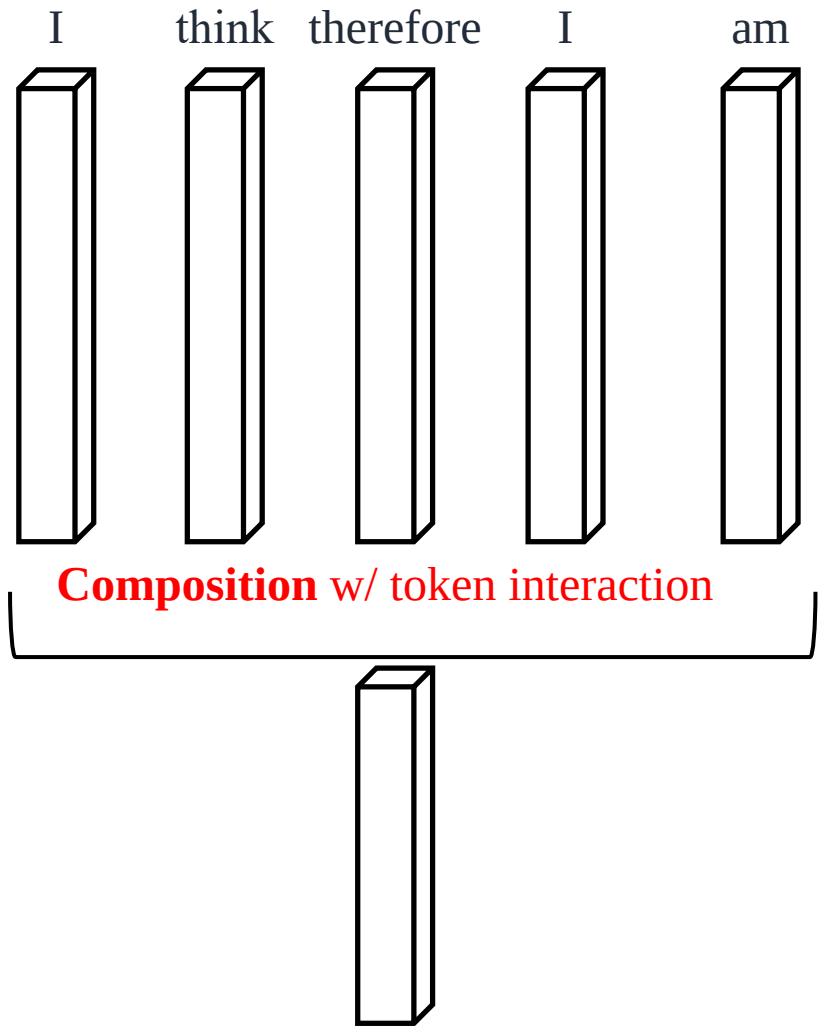
Ivory tower (象牙塔)

Semantic composition combines word meanings to interpret a phrase or sentence.

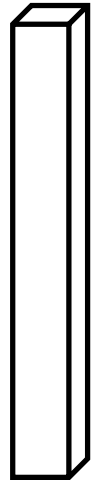
Composition requires token interaction.

Composition and Abstraction

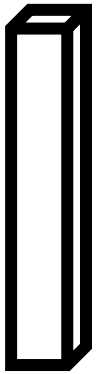
Token level



Feature level:
word vector



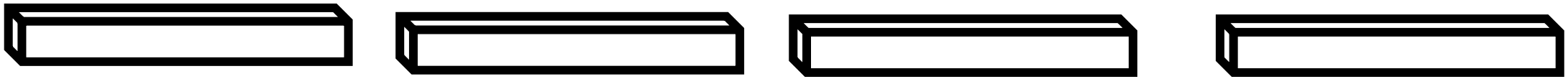
Nonlinear abstraction within each token



Combining Composition and Abstraction

Concatenate the context, then apply an MLP

Output:



Input:

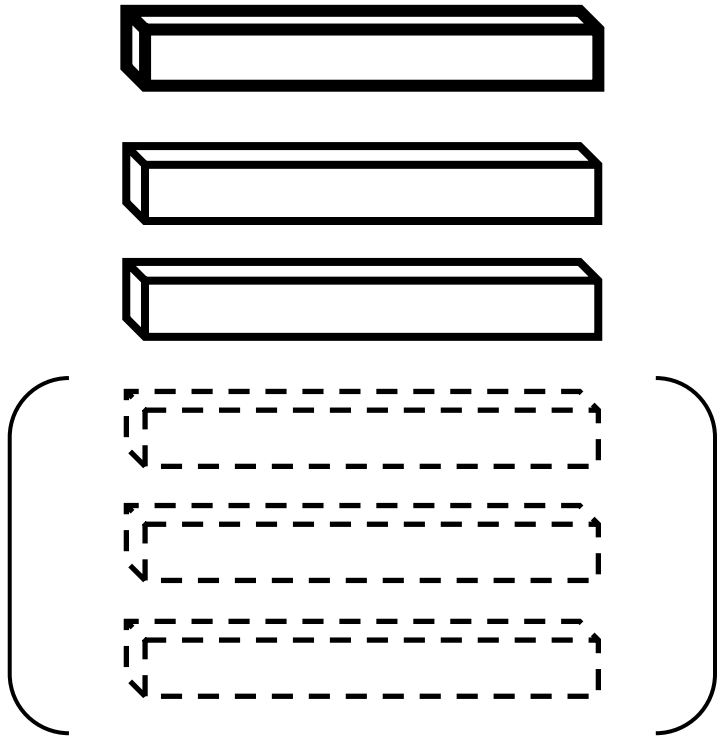


Complexity: $O(D^2L^2)$

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

Mean Pooling in CBOW

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



Remove token interaction in deeper layers

Mean pooling (token interaction) in the first layer

Complexity: $O(D^2)$

Mikolov et al. (2013), Efficient Estimation of Word Representations in Vector Space

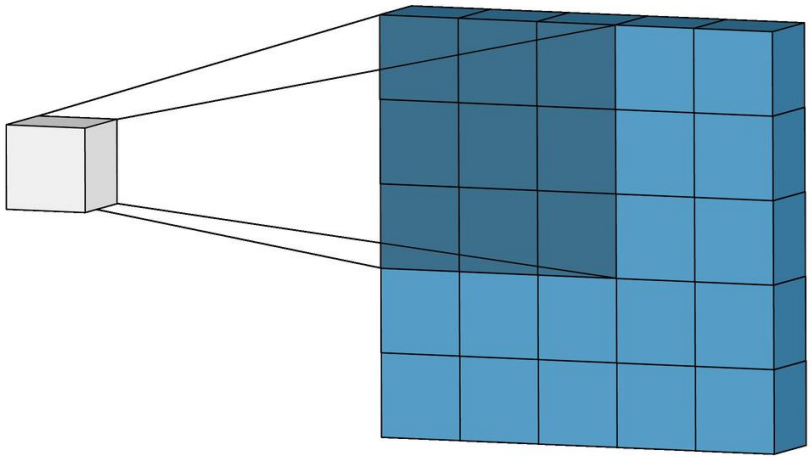
Inductive Bias in Composition

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.

Local and Recurrent Composition

CNN: local composition within a window
RNN: recurrent composition along the sequence





Limitations of CNNs and RNNs

CNN: local composition

Long-range interaction requires multiple layers or larger windows.

RNN: recurrent composition

Distant information must survive repeated state updates.

Long-range 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

Ge et al. (2021), Chinese WPLC: A Chinese Dataset for Evaluating Pretrained Language Models on Word Prediction Given Long-Range Context

Weaker Inductive Bias

How can we compose tokens freely, with fewer structural constraints?

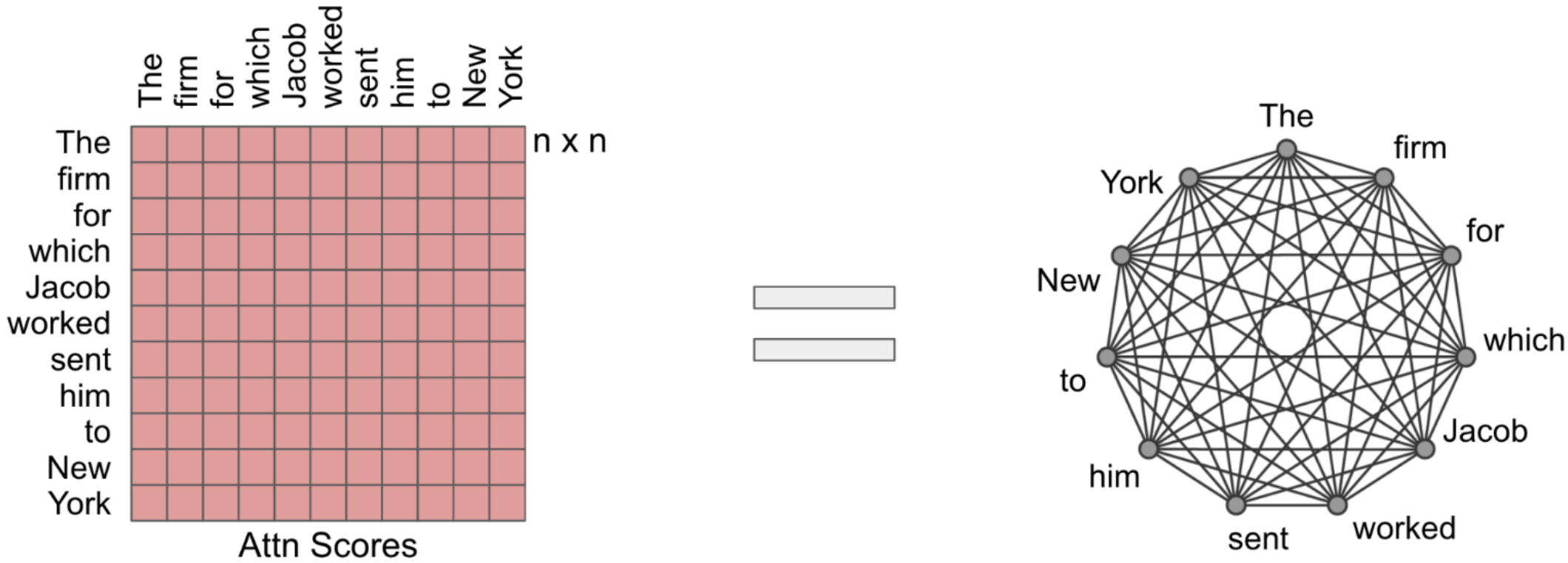
Let data determine more of the token interactions.

“general methods that leverage computation are ultimately the most effective, and by a large margin.”

Rich Sutton · The Bitter Lesson · March 13, 2019

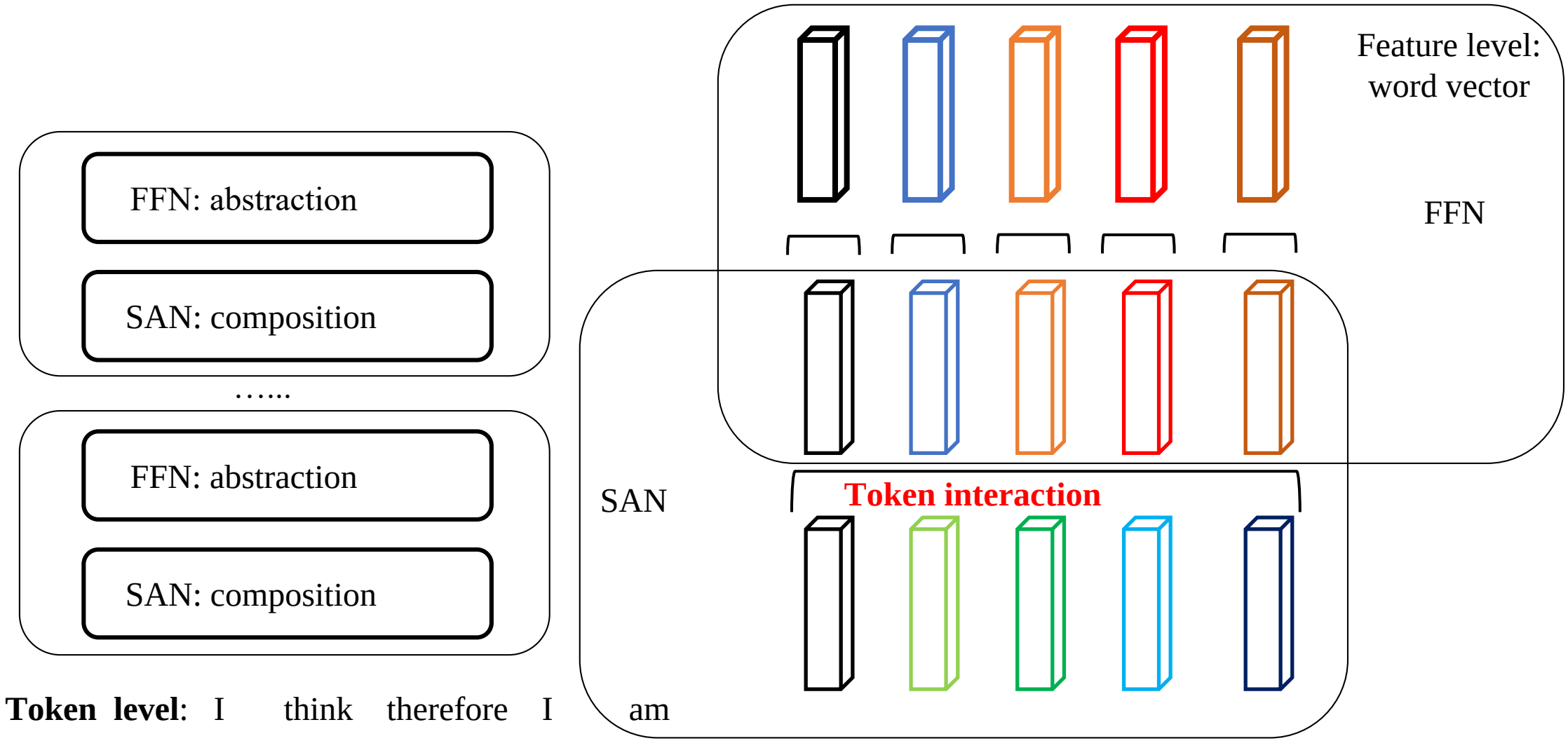
Weaker constraints still leave design choices about representation and learning.

Global Token Interaction



How does the attention mask differ between BERT and GPT?

Separating Composition and Abstraction



Lecture Roadmap

1 NLP tasks and representations

2 Inductive bias: semantic abstraction and composition

3 Architectures for Neural representations

3.1 Multilayer perceptrons

3.2 Convolutional networks

3.3 Recurrent networks

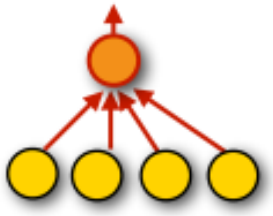
3.4 Transformer

4 Scaling laws and emergence

Multilayer Perceptron

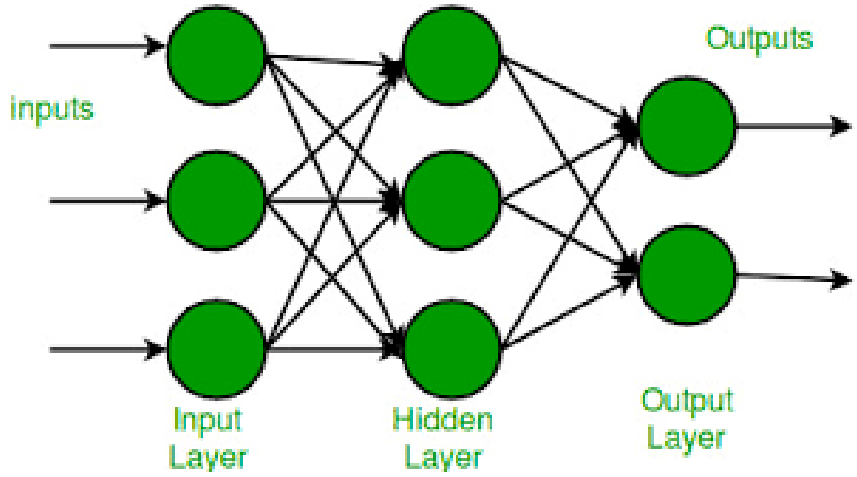
An MLP stacks fully connected layers and nonlinear activations.

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



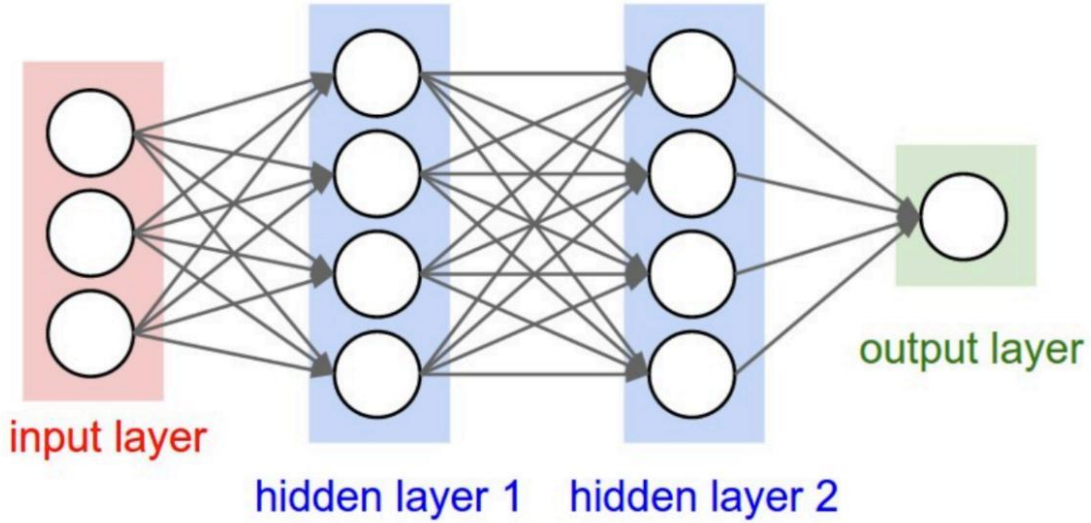
Output unit: scalar $y = f(\mathbf{x})$

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



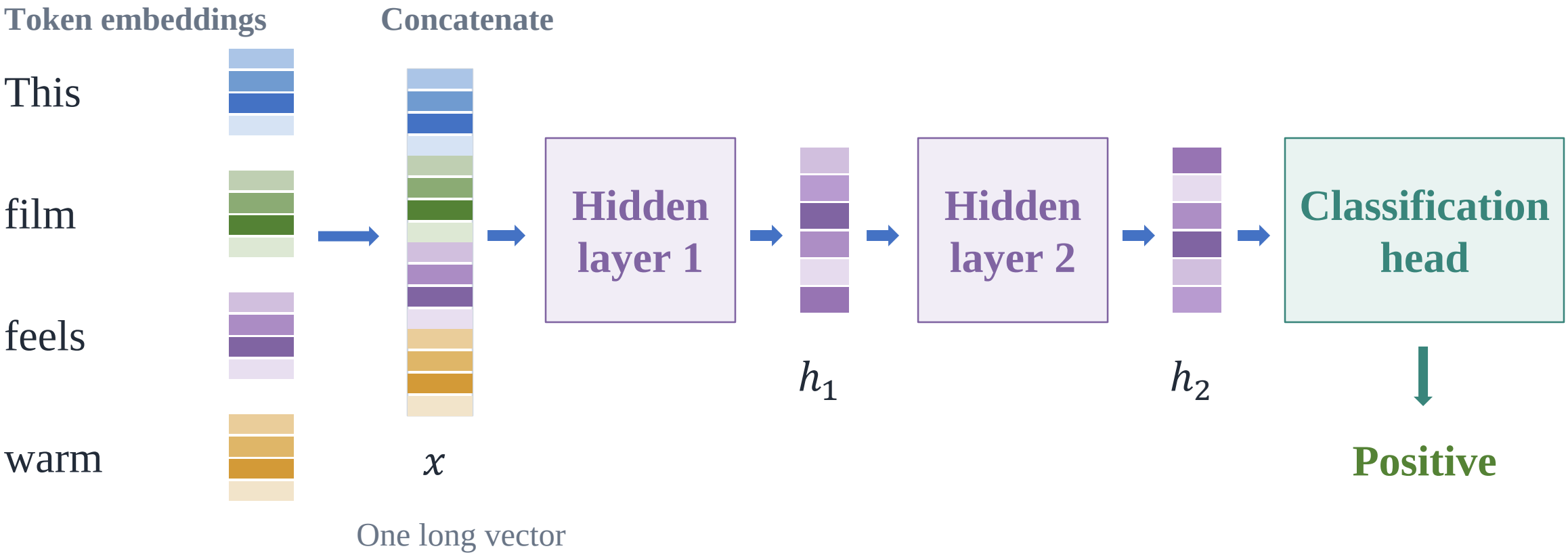
Fully-connected (FC) layers:

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)
```

From Word Embeddings to a Classifier

Concatenate embeddings, transform the vector, then classify.



Concatenation preserves every component in token order; each hidden layer outputs a vector.

Reading: A Neural Probabilistic Language Model

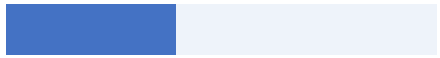
Learning from Prediction Error

The loss measures error. Gradients guide the parameter update.

Same training example; target = Positive

1 Loss

P(Positive) = 0.40



Loss = 0.93

Cross-entropy

2 Gradients



$g = \nabla_\theta L$

Chain rule through every trainable layer

3 Update

$\theta \leftarrow \theta - \eta\, g$

Gradient descent

Take a small step against the gradient

4 Predict again

P(Positive) = 0.81



Loss = 0.21

Closer to the target

The input stays the same. The learned parameters change.

Reading: Forward Propagation, Backward Propagation, and Computational Graphs

Challenges in Neural Network Training

Different failures call for different interventions.

Challenge	What goes wrong	Typical response
Vanishing or exploding gradients	Earlier layers receive tiny or unstable learning signals.	Residual paths, suitable activations, normalization and gradient clipping.
Poor initialization	Activations or gradients start at an unhelpful scale.	Use an initialization suited to the layer and activation.
Slow / unstable optimization	Updates oscillate, overshoot or make little progress.	Tune the learning rate, optimizer and schedule.
Overfitting	Training loss improves while held-out performance worsens.	Use validation, regularization, more data and early stopping.
Training cost	Memory and compute limit the experiments we can run.	Efficient architectures, batching, precision and implementation.

Architecture, optimization and generalization require separate diagnoses.

Reading: Dive into Deep Learning, Chapter 5: Multilayer Perceptrons

Why Sequence Models?

Language varies in length, position and contextual dependence.

Variable length

"good"
"a surprisingly good film"

→ **Reuse a computation as the sequence grows.**

Patterns at many positions

"not good"
"the ending was not good"

→ **Share parameters across token positions.**

Context beyond one window

"The film was not, despite its beautiful images, good."

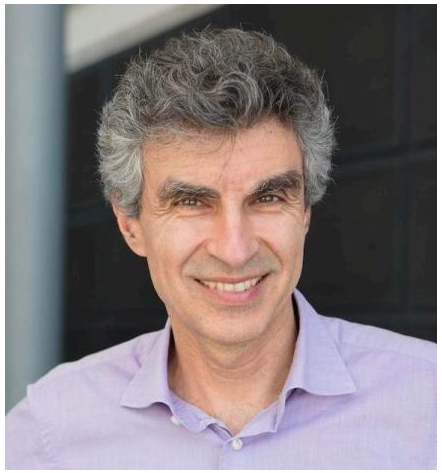
→ **Choose how information travels between words.**

CNN: local RNN: recurrent
Attention: content-dependent

These choices organize representations. Training still needs sound optimization and regularization.

Reading: A Neural Probabilistic Language Model; Convolutional Sequence to Sequence Learning

A Neural Probabilistic Language Model



Yoshua Bengio

Bengio et al. (2003)

Word embeddings let similar words share statistical strength.

An MLP maps a fixed context to a next-word distribution.

Bengio et al. (2003), A Neural Probabilistic Language Model. JMLR 3:1137–1155.

Generalization Through Word Embeddings

A Neural Probabilistic Language Model

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



A neural network models a distribution over the next word.

P(w | I am a good …)
P(w | I am a great …)

Word embeddings share information:
e(good) ≈ e(great)
Similar contexts can yield similar distributions.

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

From MLPs to Sequence Architectures

A connection rule determines how information moves between tokens.

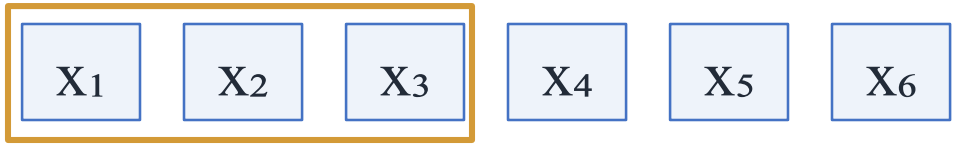
MLP



Fixed input layout

Token order comes from how the input is arranged.

CNN

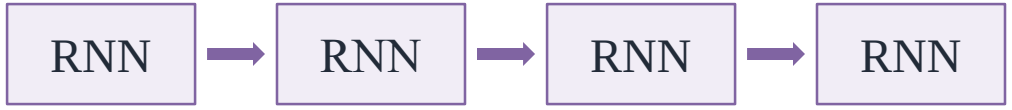


Shared local window

Local patterns

The same filter is reused at different positions.

RNN



State passes along the sequence

Sequential memory

One shared update combines the current token with history.

Bengio et al. (2003), A Neural Probabilistic Language Model. Kim (2014), Convolutional Neural Networks for Sentence Classification. Elman (1990), Finding Structure in Time.

Lecture Roadmap

1 NLP tasks and representations

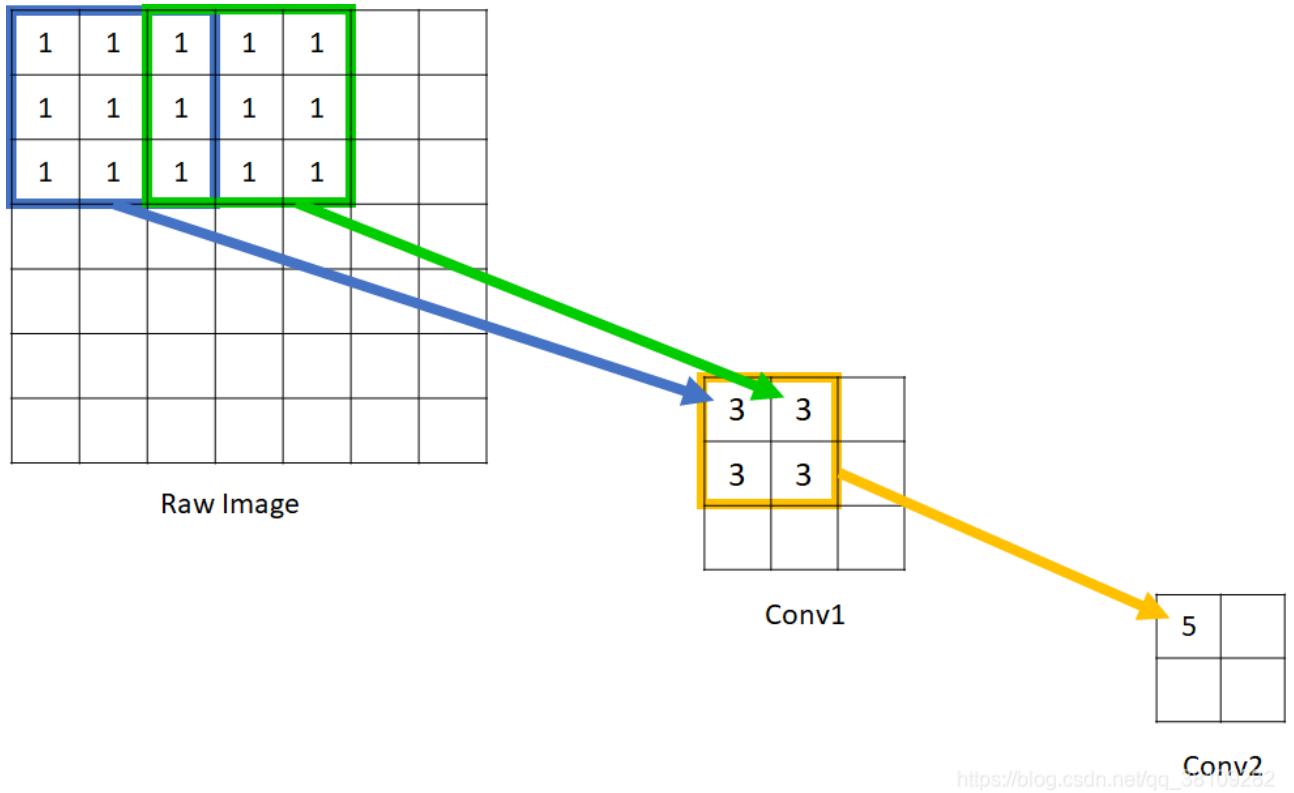
2 Inductive bias: semantic abstraction and composition

3 Architectures for Neural representations

- 3.1 Multilayer perceptrons
- **3.2 Convolutional networks**
- 3.3 Recurrent networks
- 3.4 Transformer

4 Scaling laws and emergence

Convolution: Multiply, Then Sum

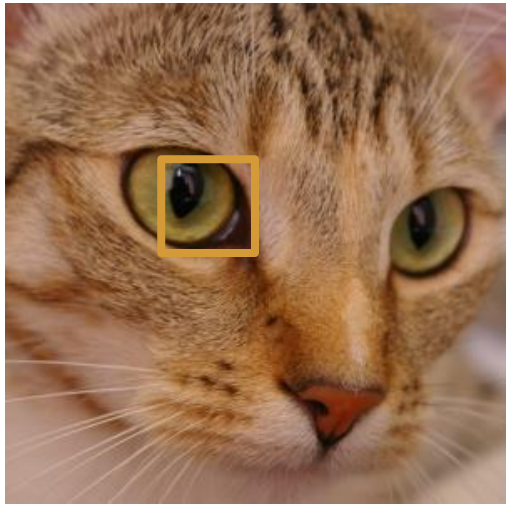


At each time， it only perceives part of the input.

Convolution: Multiply, Then Sum

One 3 × 3 patch and one shared filter produce one response.

Image



16 × 16 grayscale grid

Patch values

2	2	3
1	3	2
3	2	1

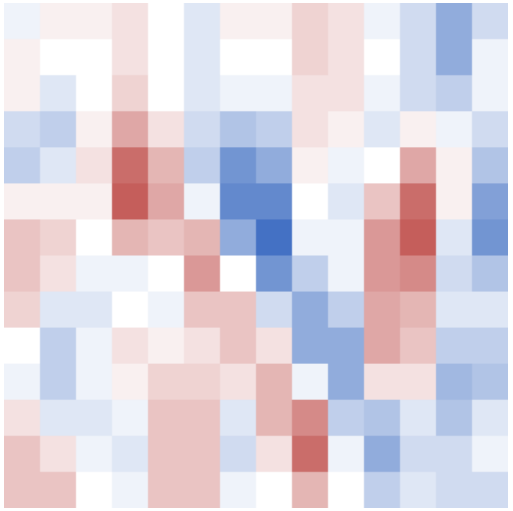
×

Filter weights

-1	0	1
-2	0	2
-1	0	1

→

Complete response map



14 × 14 positions

The same filter is reused
at every valid image position.

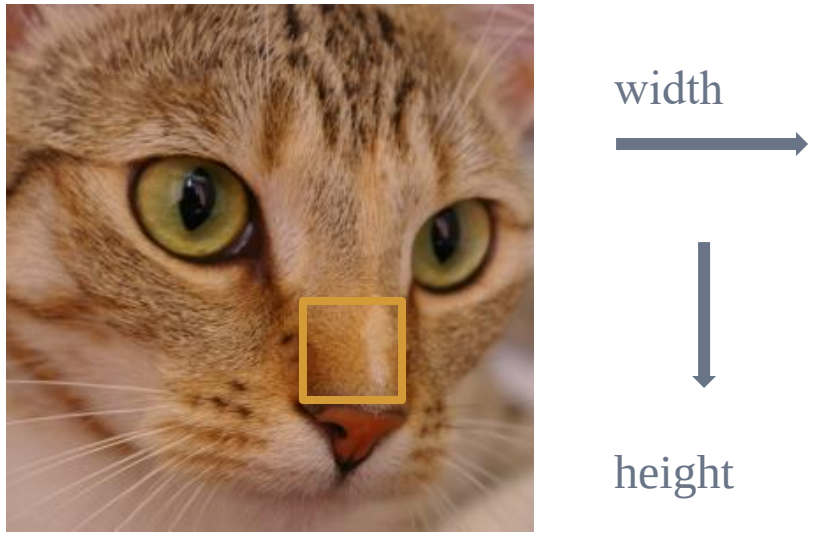
Repeat at each position: stride 1 and no padding give 14 × 14 responses.

Stanford CS231n, Convolutional Neural Networks. Photograph: Stefan van der Walt, Chelsea (CC0).

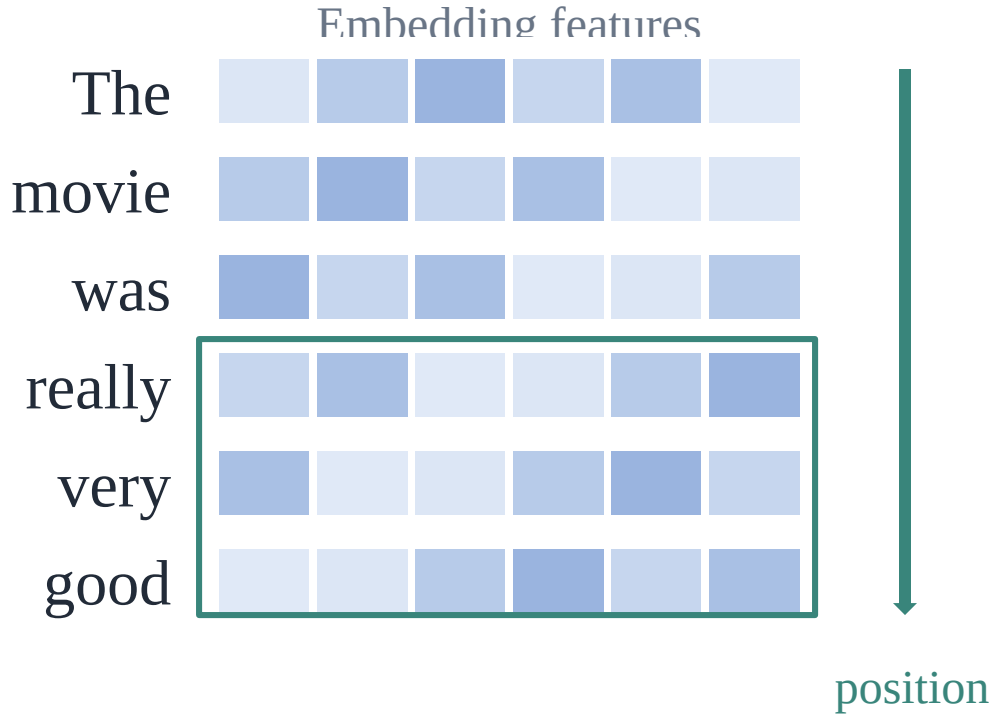
2D and 1D Convolution: Move the Window

Both reuse one filter. Only the axes that represent position are different.

Image: width and height



Text: token positions



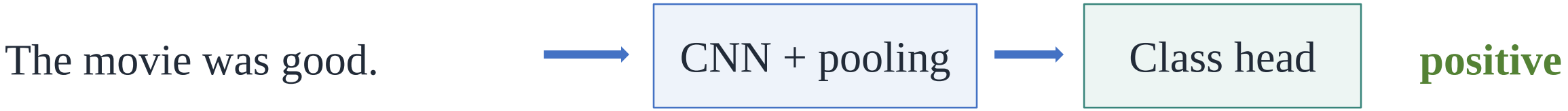
Click 3: another image position; text now covers really, very, good.

Kim (2014), Convolutional Neural Networks for Sentence Classification. EMNLP.

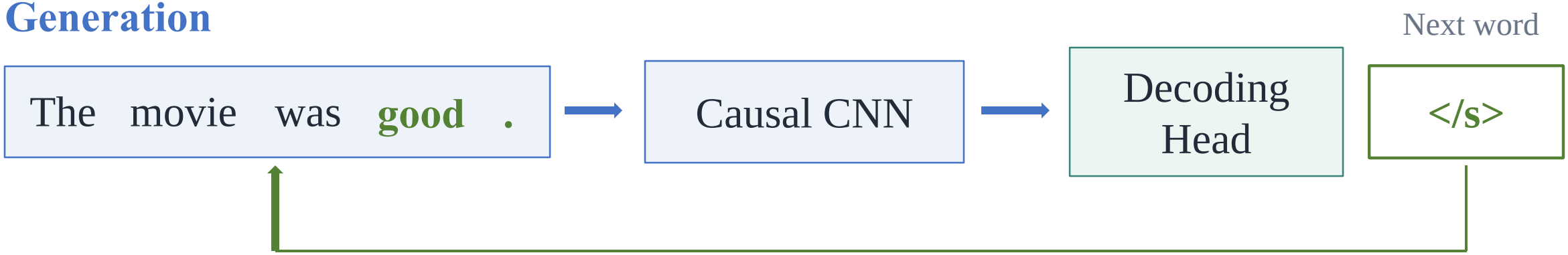
From Classification to Autoregressive Generation

Predict one word, append that word, then compute the next prediction.

Classification



Generation

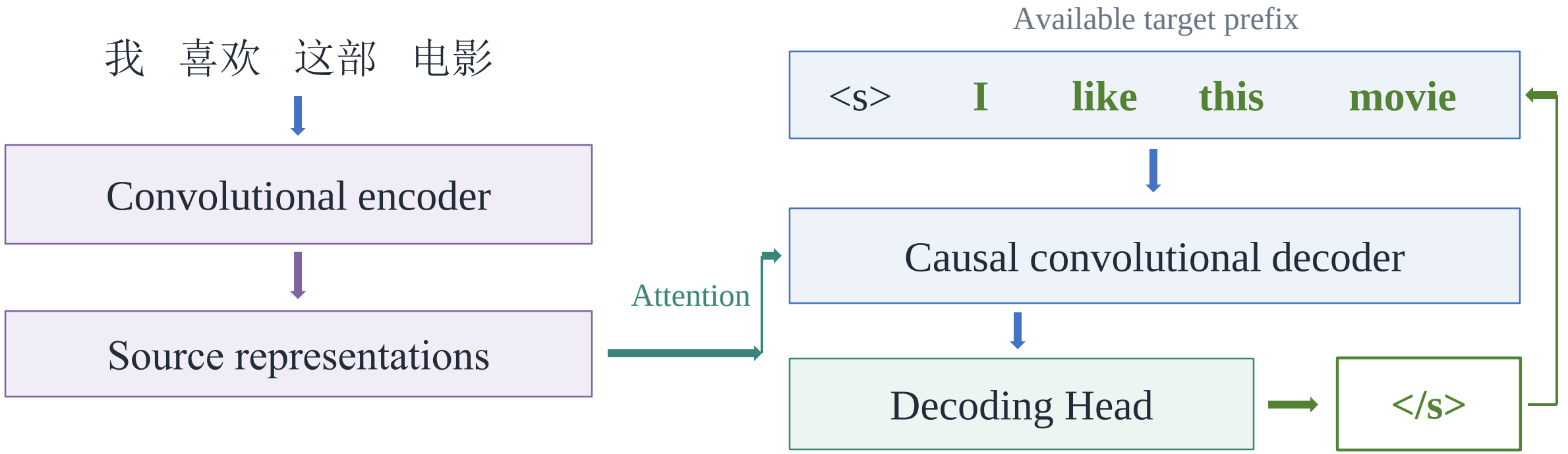


5 Predict the end token and stop. Future words never enter the earlier prefix.

Gehring et al. (2017), Convolutional Sequence to Sequence Learning. ICML.

ConvS2S: Predict, Append, Repeat

Task 4: the source stays fixed while the target prefix grows.

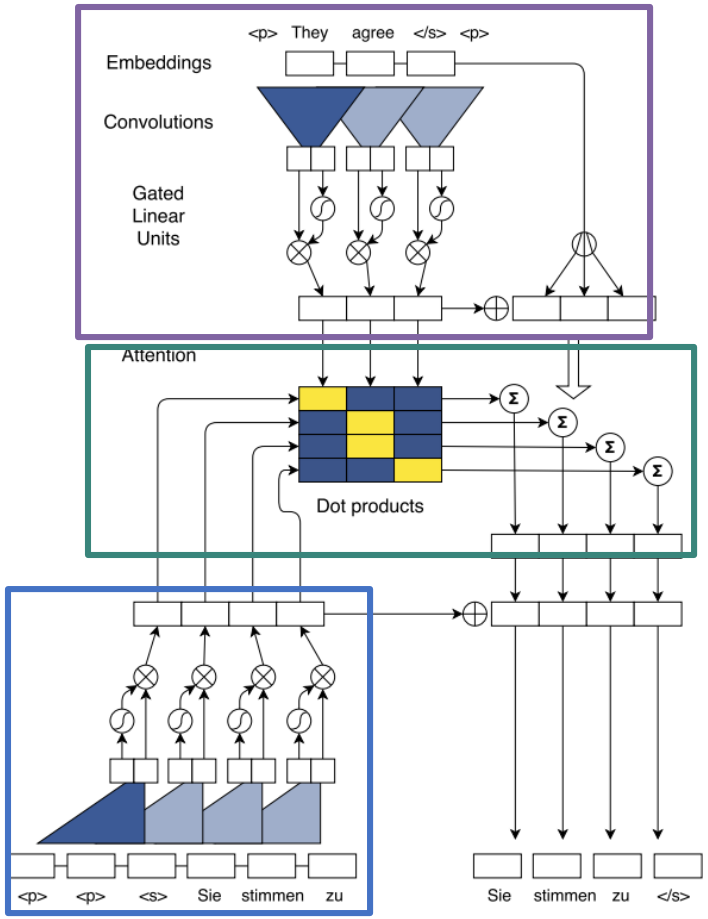


The completed prefix predicts </s>. Translation stops.

Gehring et al. (2017), Convolutional Sequence to Sequence Learning. ICML.

ConvS2S in the Original Paper

Figure 1 shows parallel teacher-forced training, not simultaneous generation.



1 Source encoder

Convolutions build contextual source features.

2 Causal target decoder

Shifted gold targets supply the prefix.
Each position still excludes future tokens.

3 Attention and prediction

Every decoder position reads the source and predicts its next target word.

Gehring et al. (2017), Convolutional Sequence to Sequence Learning. ICML. Figure 1, p. 3.

Lecture Roadmap

1 NLP tasks and representations

2 Inductive bias: semantic abstraction and composition

3 Architectures for Neural representations

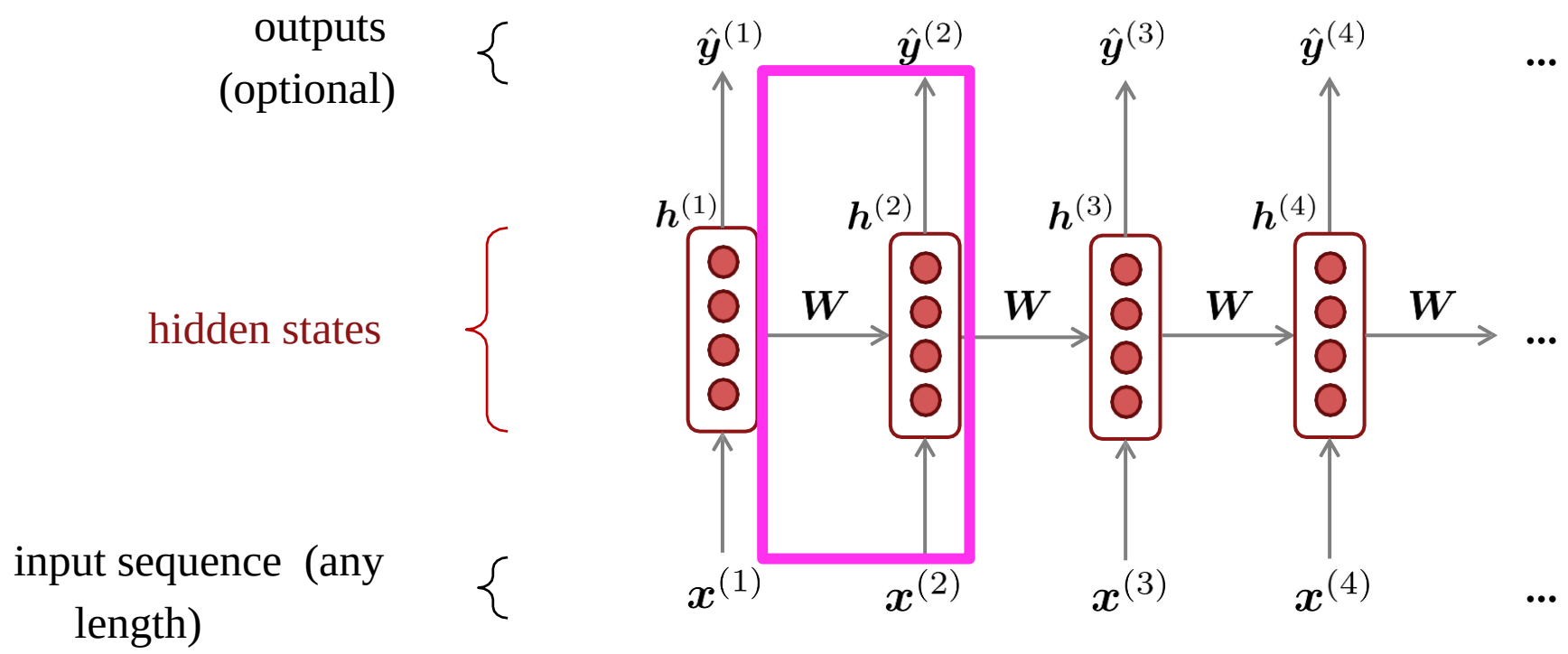
- 3.1 Multilayer perceptrons
- 3.2 Convolutional networks
- **3.3 Recurrent networks**
- 3.4 Transformer

4 Scaling laws and emergence

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

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

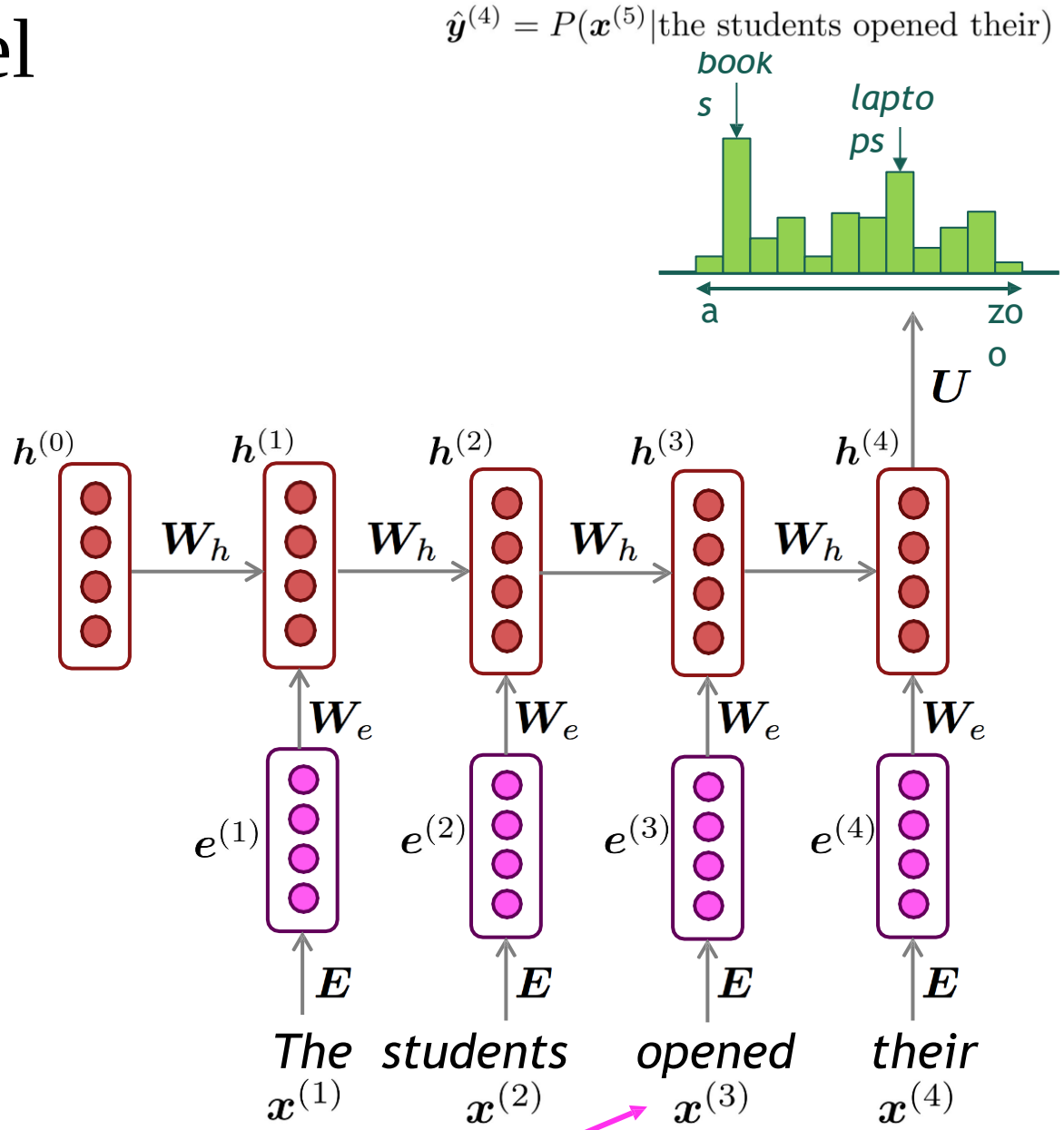
$\boldsymbol{h}^{(0)}$ is the initial hidden state

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

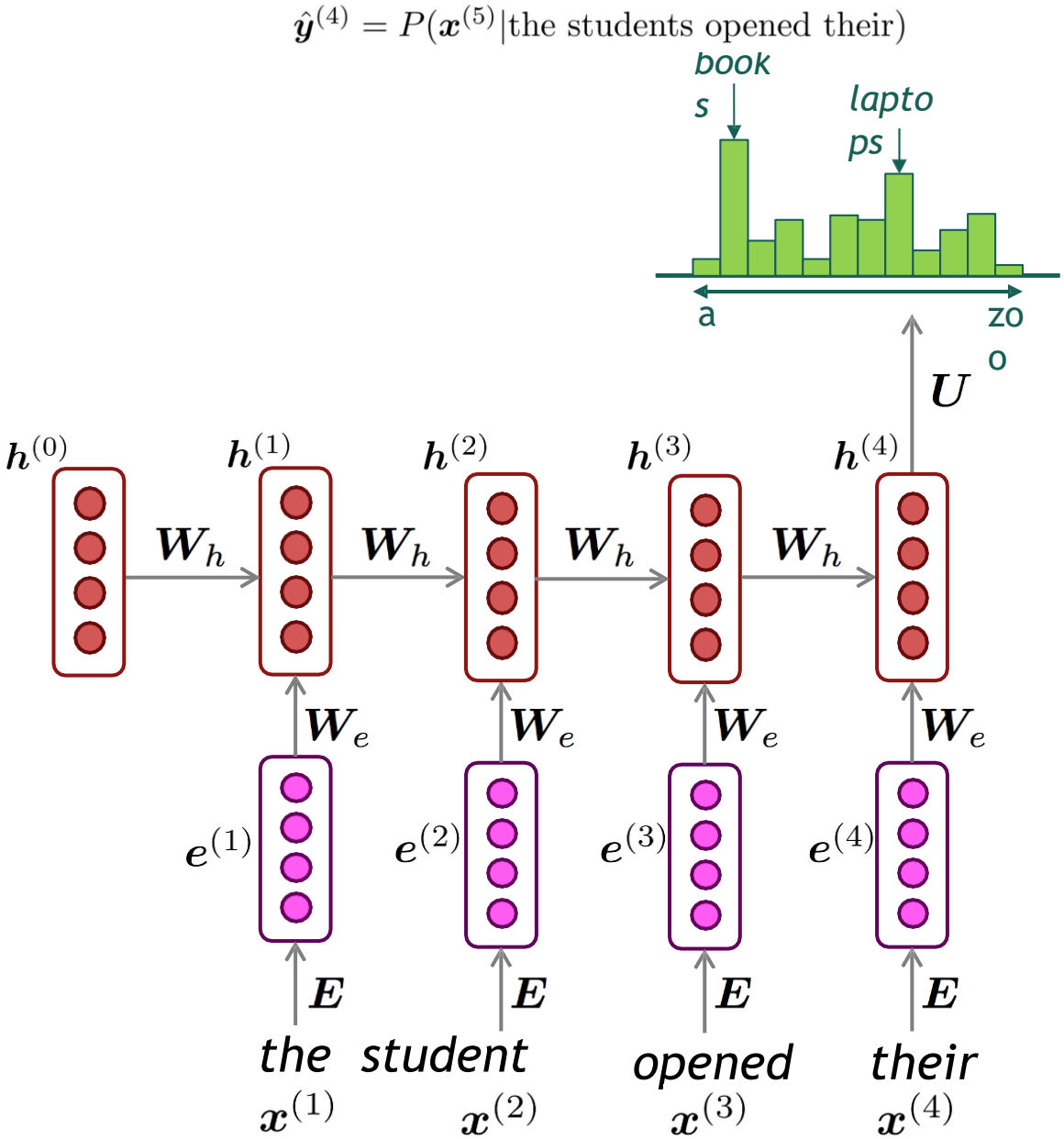
RNN **Advantages**:

- 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.

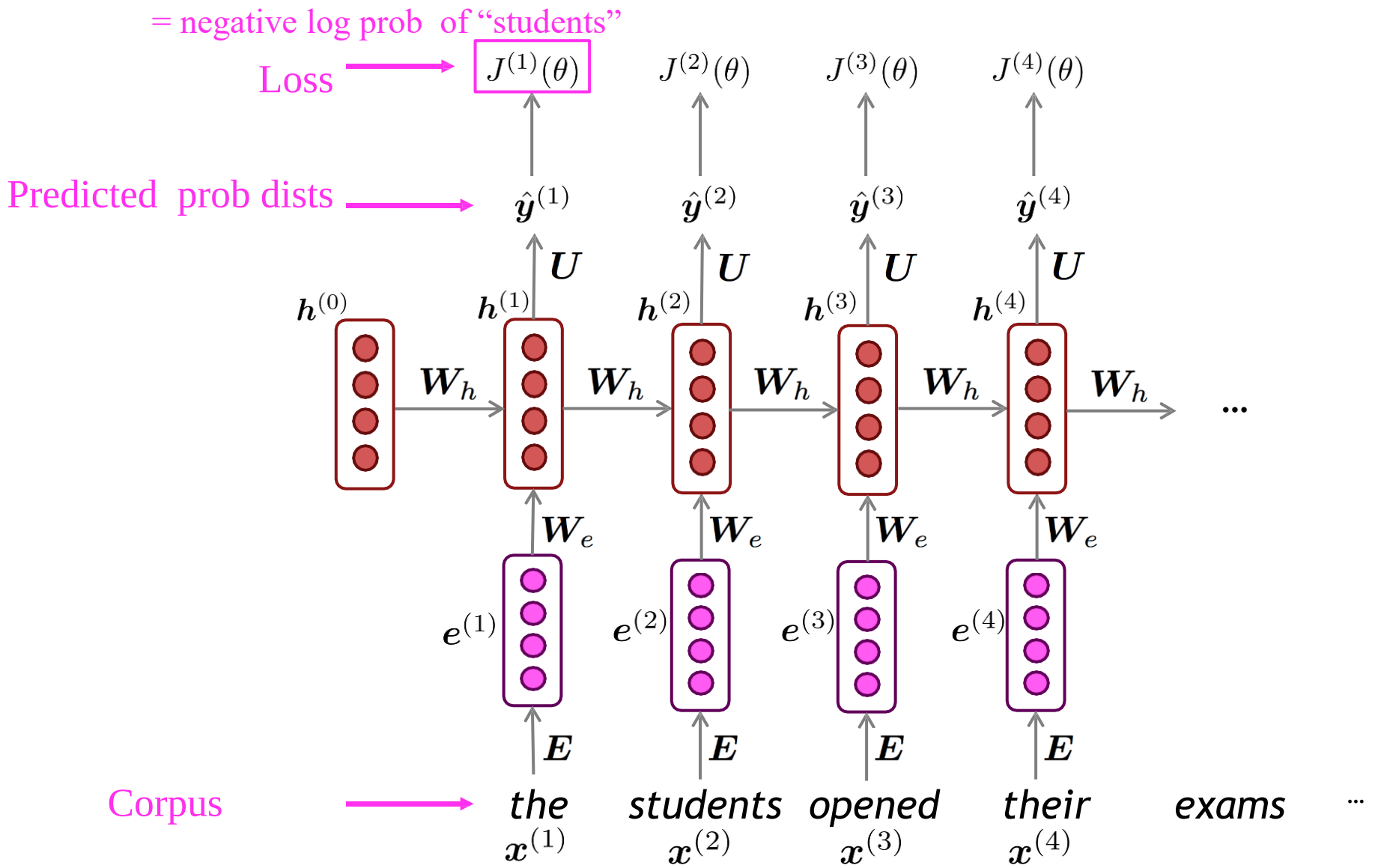
RNN **Disadvantages**:

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

More on these later

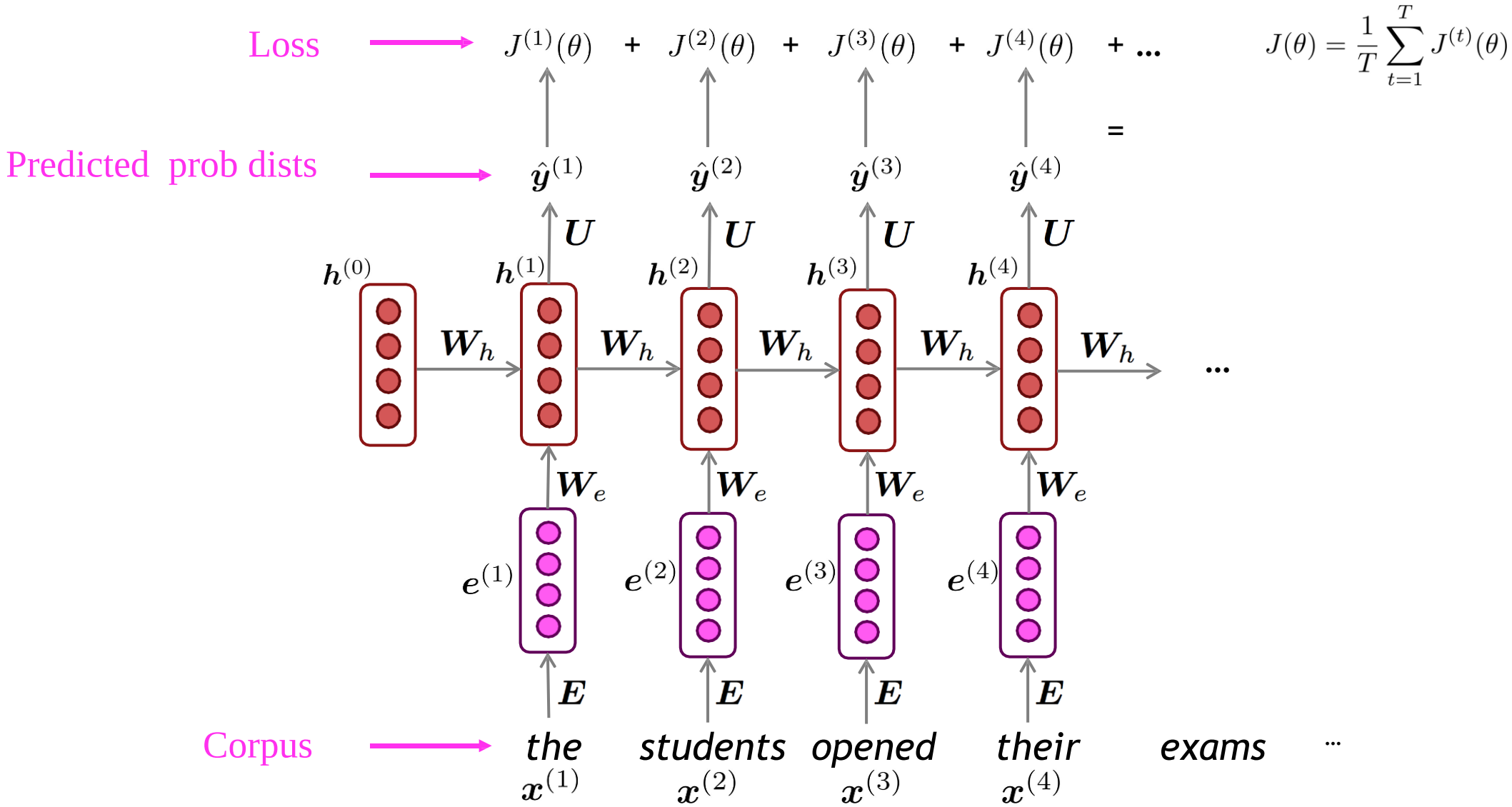


Training an RNN Language Model



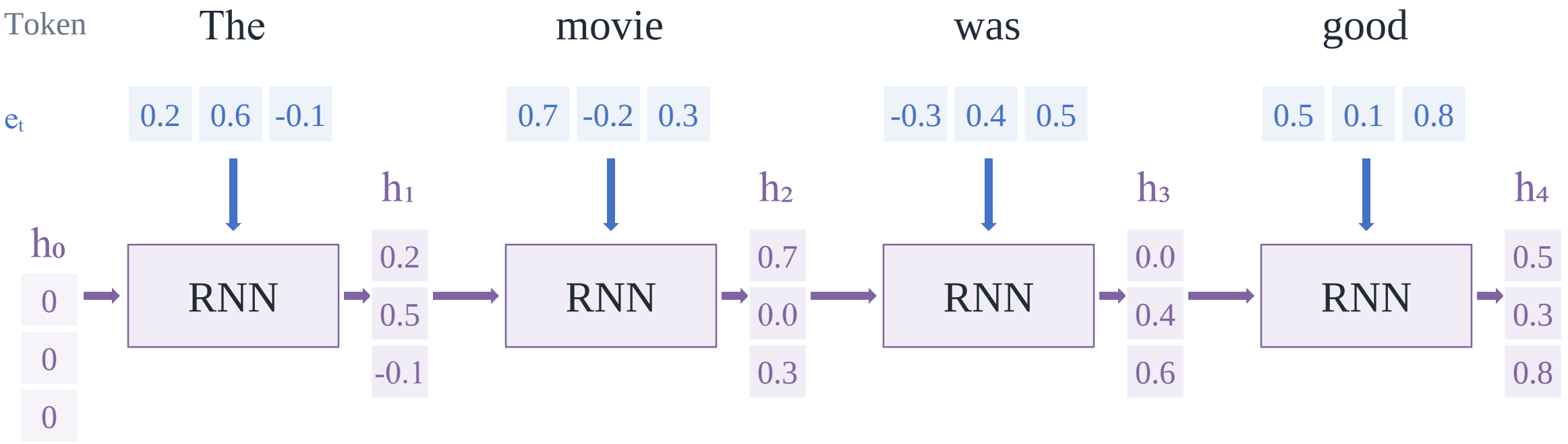
Training an RNN Language Model

"Teacher forcing"



RNN: A New Context Vector at Every Step

Each step combines a token embedding with the previous hidden vector.



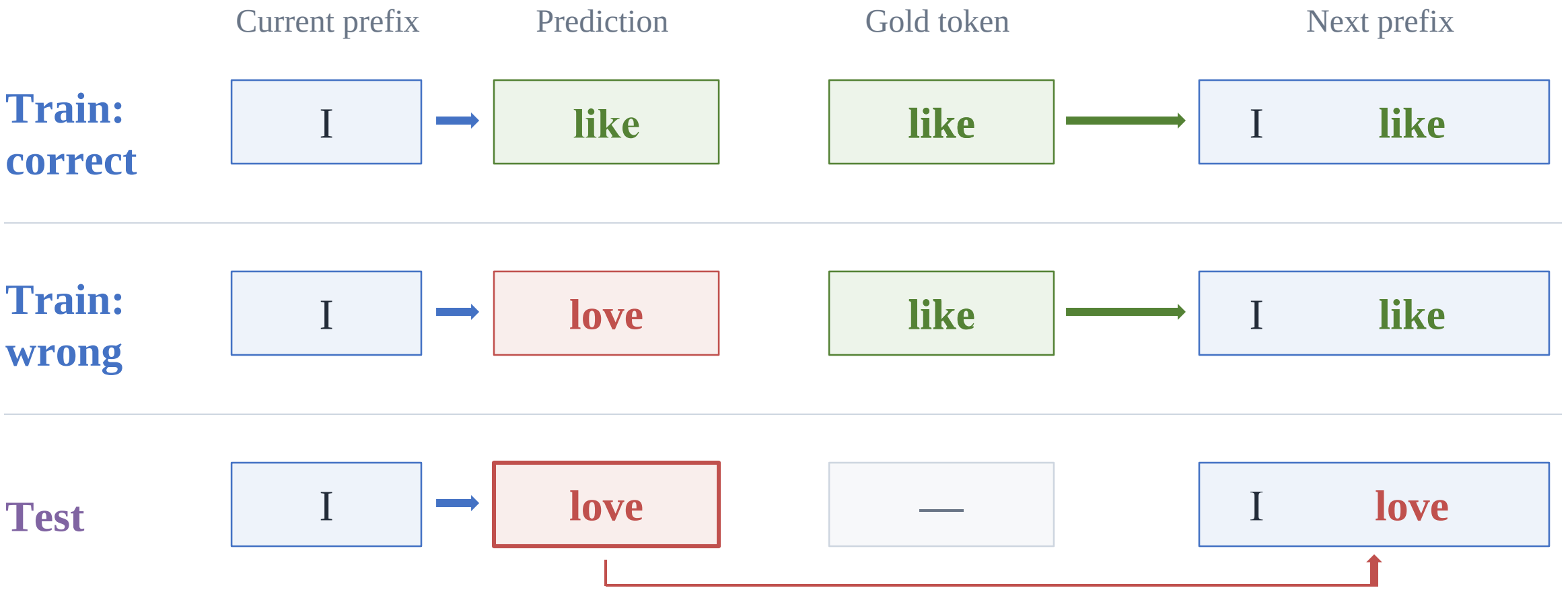
$$h_t = \tanh(W_h\, h_{t-1} + W_x\, e_t + b)$$

Same W_h, W_x and b at every step

Elman (1990), Finding Structure in Time. Cognitive Science. Demonstration values, rounded to one decimal.

Teacher Forcing and Free-Running Generation

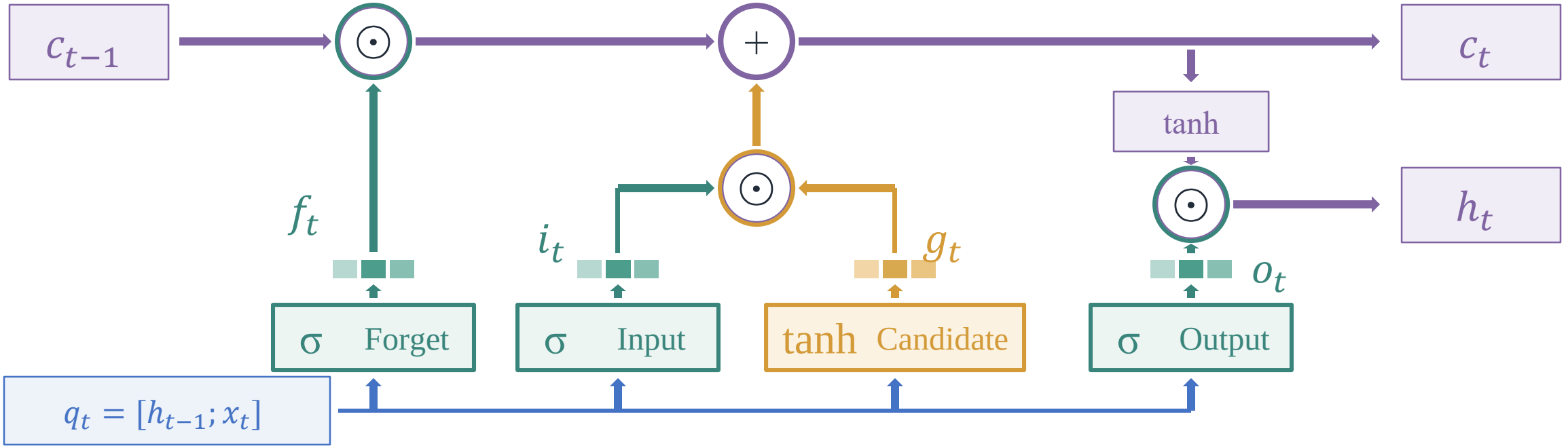
Training appends the gold token whether the prediction is correct or wrong.



Bengio et al. (2015), Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. NeurIPS.

LSTM: Keep, Write and Expose Memory

Vector gates (0–1) control what memory is kept, written and exposed.



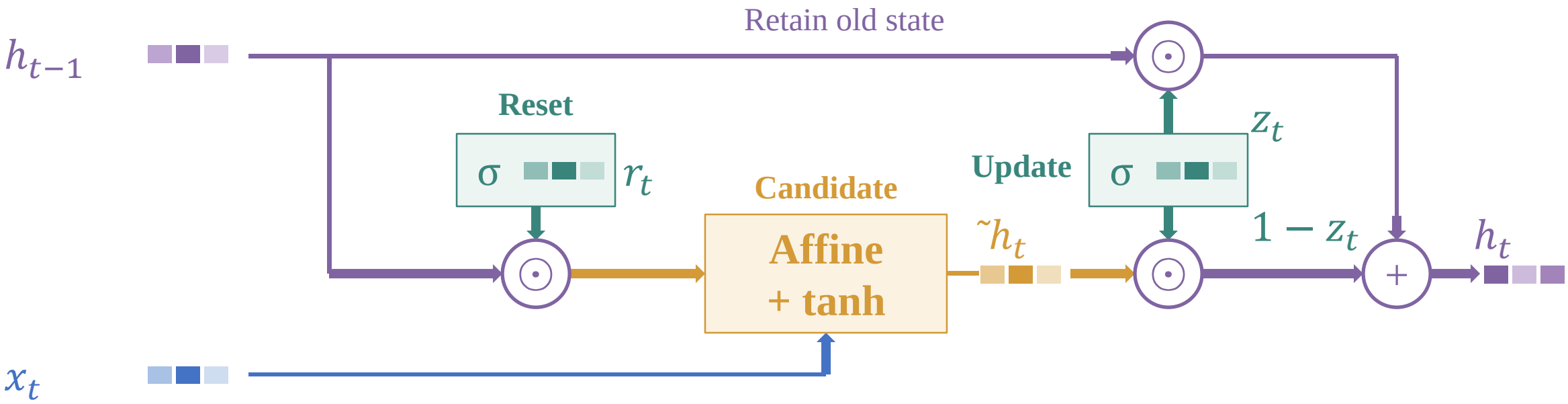
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

Kute, GRU and LSTM. Hochreiter & Schmidhuber (1997), Long Short-Term Memory.
Gers et al. (2000), Learning to Forget: Continual Prediction with LSTM.

GRU: Reset, Then Blend

Reset controls the candidate. Update blends old and new.



Same context: $q_t = [h_{t-1}; x_t]$ $r_t = \sigma(W_r q_t + b_r)$ $z_t = \sigma(W_z q_t + b_z)$

$$\tilde{h}_t = \tanh(W_x x_t + W_h(r_t \odot h_{t-1}) + b_h)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

$\odot$ means element-wise multiplication

Cho et al. (2014), Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

RNN, LSTM and GRU: What the Gates Control

Each gate has one role in retaining, writing or exposing information.

Mechanism	RNN	LSTM	GRU
Stored state	h	c and h	h
Retain history	Implicit update	Forget gate f	Update gate z
Write content	Whole candidate	Input gate i	1 − z blends candidate
Shape candidate	Input + old h	Input + old h	Reset gate r
Expose output	State h	Output gate o	State h

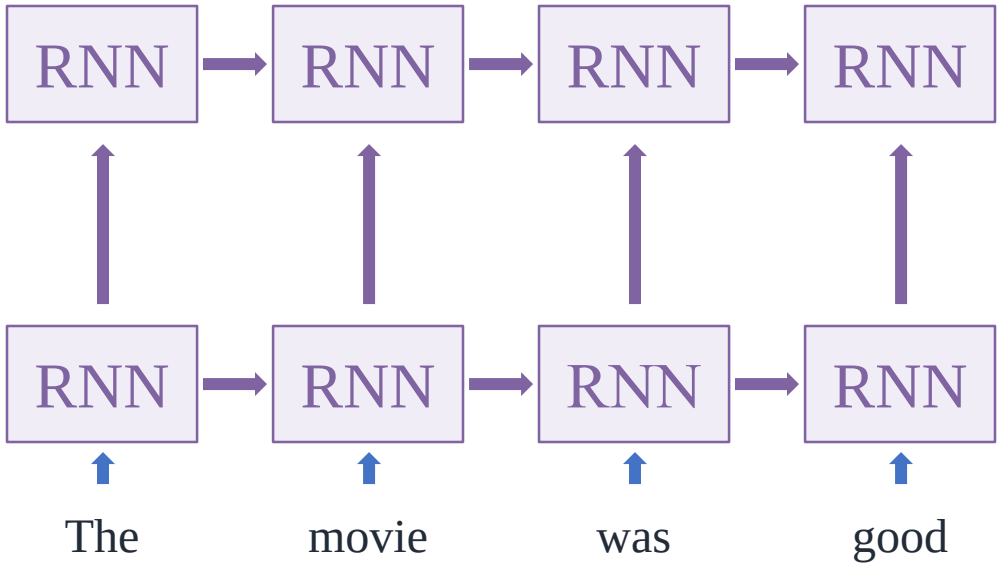
Convention: h new = z × h old + (1 − z) × candidate.

Gers et al. (2000), Learning to Forget: Continual Prediction with LSTM. Cho et al. (2014), Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.

RNN Depth and Direction

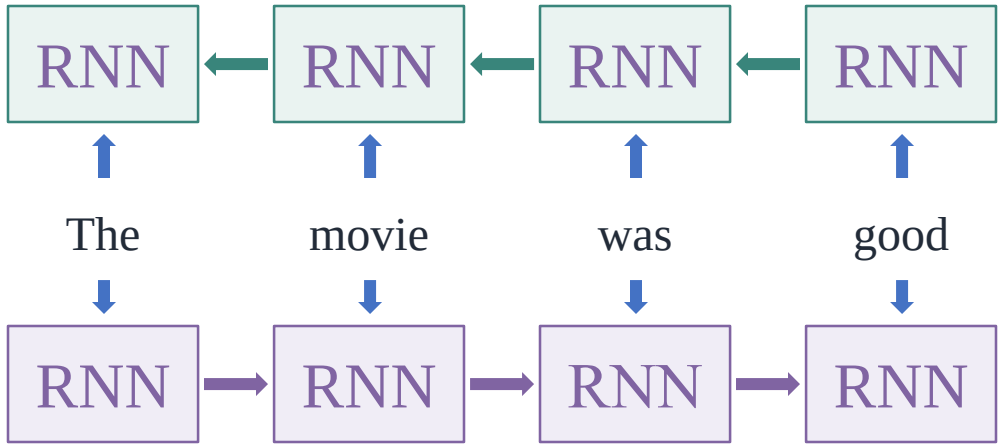
A recurrent cell can be stacked across layers or run in both directions.

Stacked layers



Each layer transforms the sequence.

Bidirectional encoding

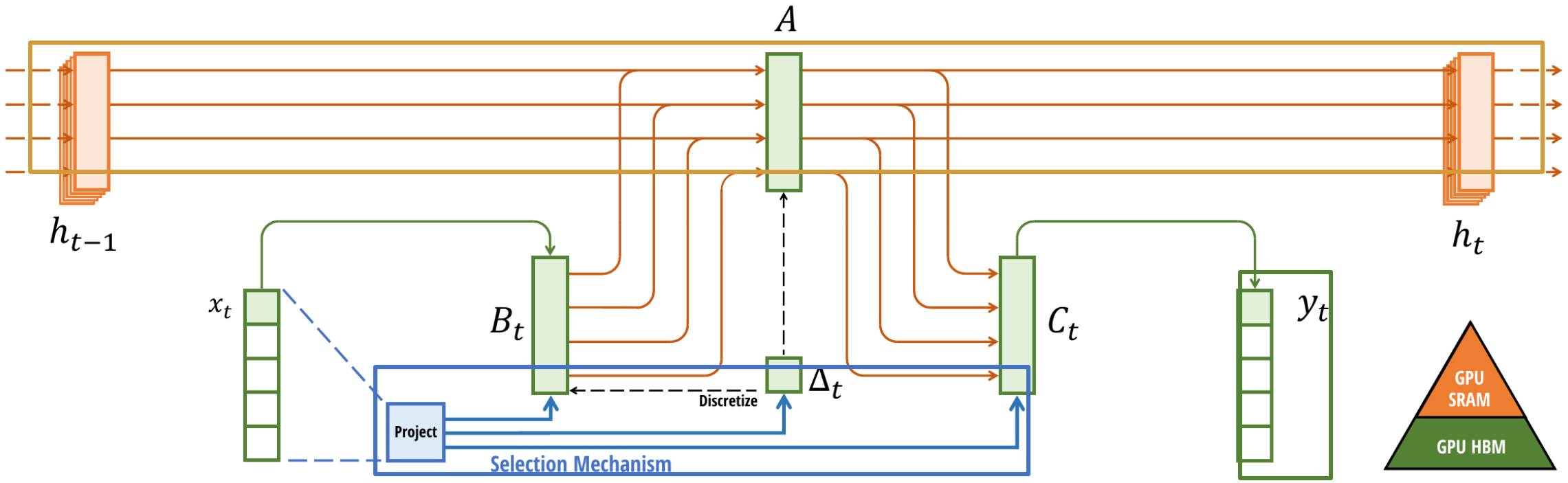


Both directions inform each position.

RNN, LSTM and GRU cells can use these layouts. Bidirectional encoding requires both past and future words.

Mamba: Selective State Updates

Each token selects how the model writes, retains and reads its memory.

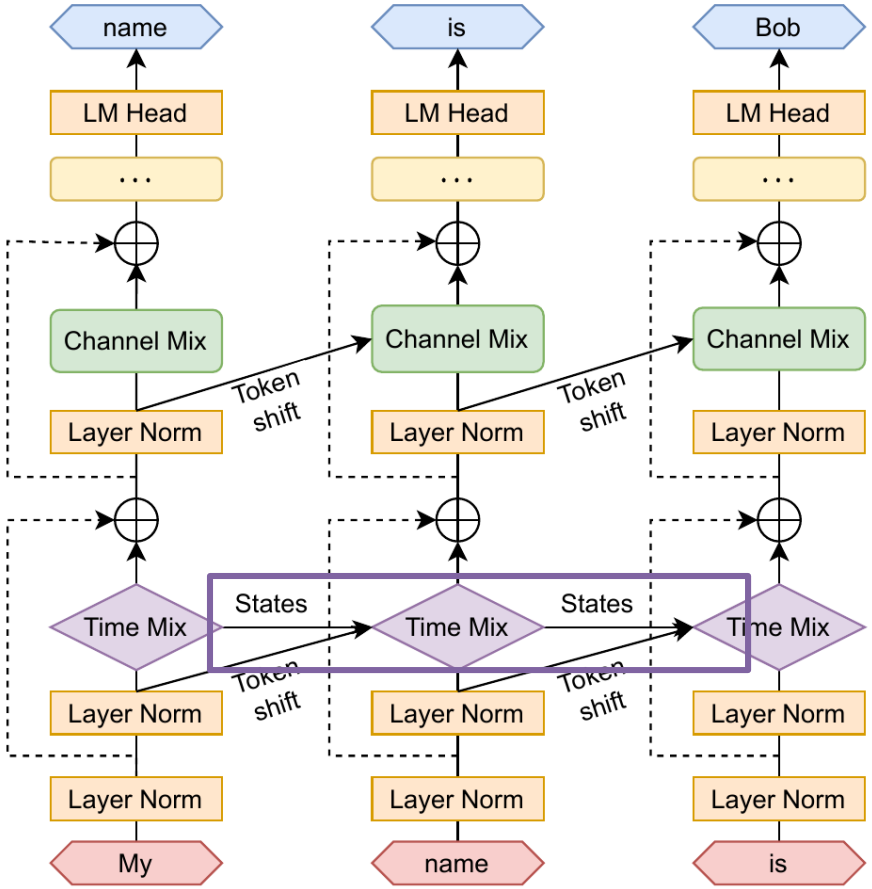


1 Input-dependent selection 2 Update a fixed-size state 3 Read useful features

Gu & Dao (2023). Mamba: Linear-Time Sequence Modeling with Selective State Spaces.
Figure 1. Original paper diagram; teaching highlights added separately.

RWKV: Recurrent Inference

Time mixing carries a compact memory from one token to the next.



1 Pass a recurrent state across tokens.

2 Keys weight values. Learned decay discounts history.

3 Receptance gates the read. Channel mixing transforms features.

Fixed state size in context length; linear work over the sequence.

Peng et al. (2023). RWKV: Reinventing RNNs for the Transformer Era. Findings of EMNLP 2023, Figure 3.

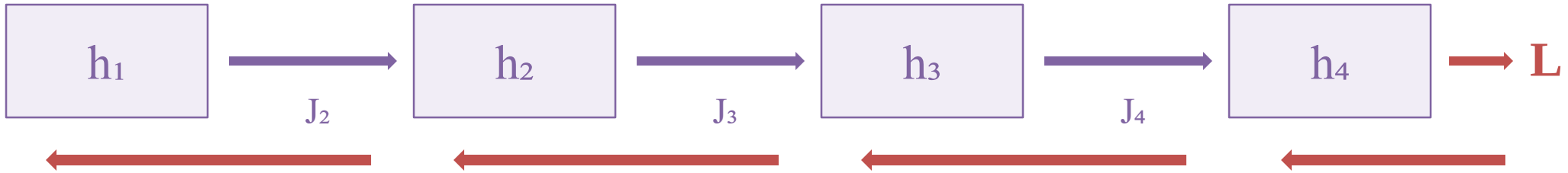
Backpropagation Through Time (BPTT)

The final loss sends a correction through each earlier state transition.

$$h_t = f(W\ h_{t-1} + U\ x_t + b)$$

State-transition Jacobian

$$J_t = \frac{\partial\ h_t}{\partial\ h_{t-1}}$$



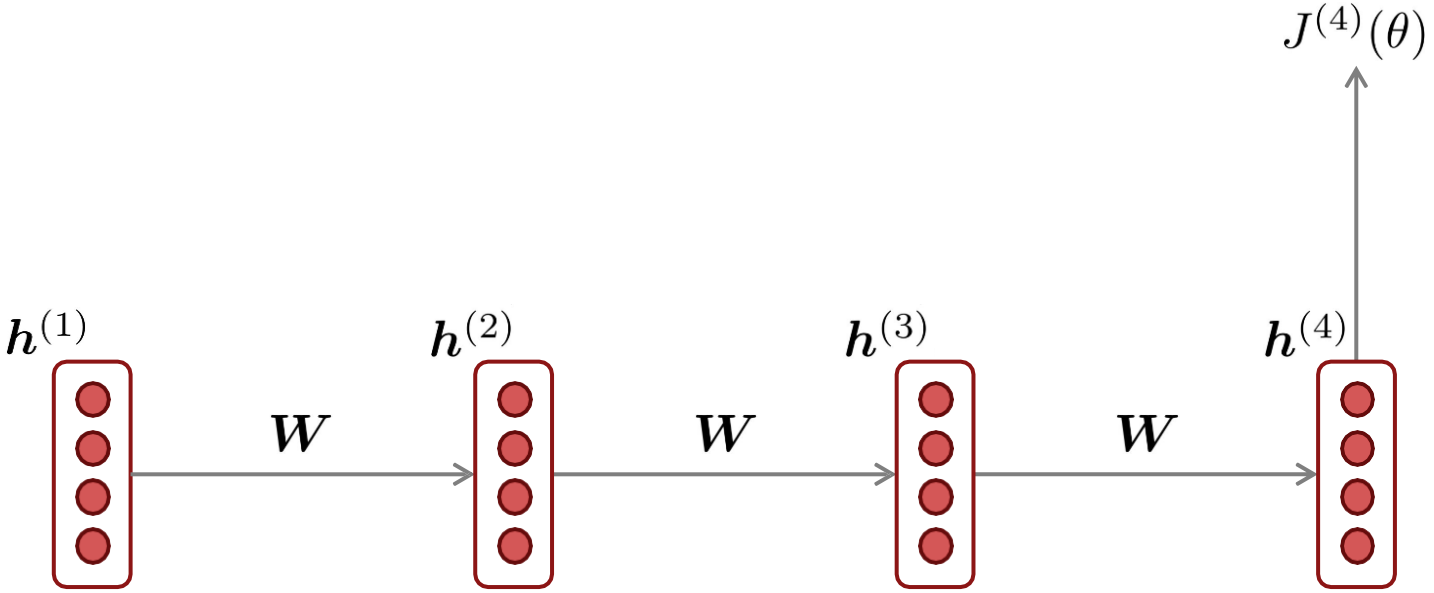
g_t is the gradient of the loss with respect to the hidden vector h_t.

$$g_1 = J_2^T\ g_2 = J_2^T\ J_3^T\ J_4^T\ g_4$$

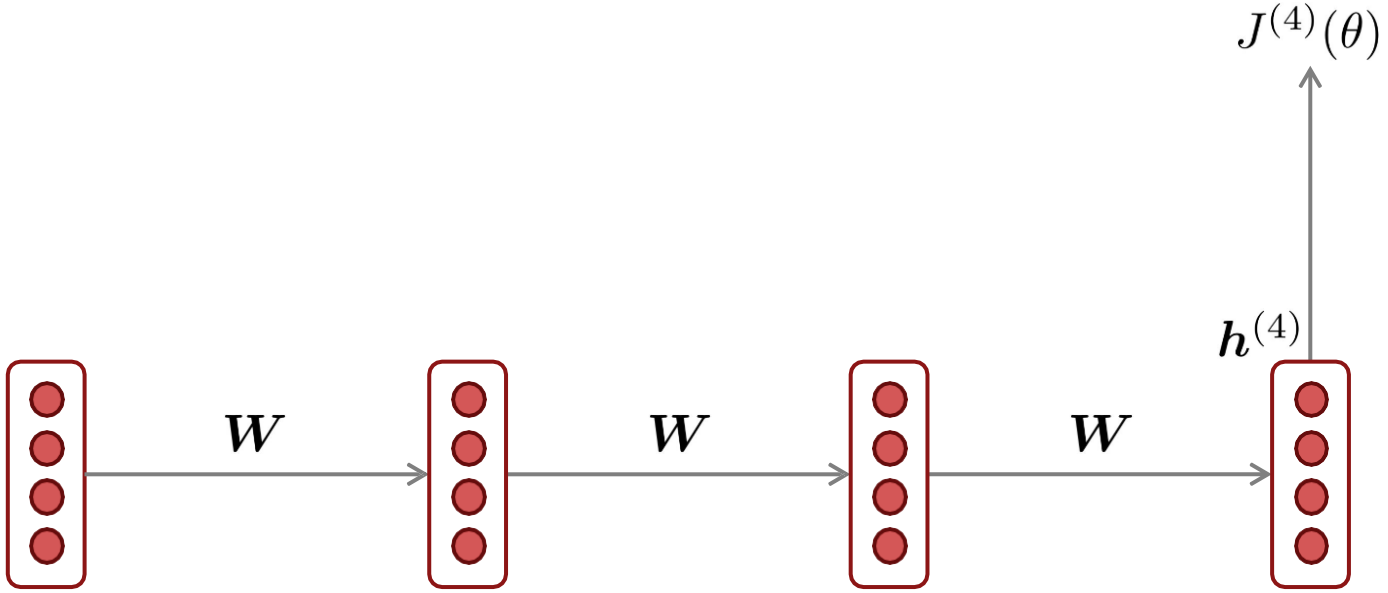
Repeated Jacobian products can contract or amplify gradients; shared-weight contributions add.

Pascanu, Mikolov & Bengio (2013), On the Difficulty of Training Recurrent Neural Networks. ICML.

Problems with RNNs: Vanishing and Exploding Gradients

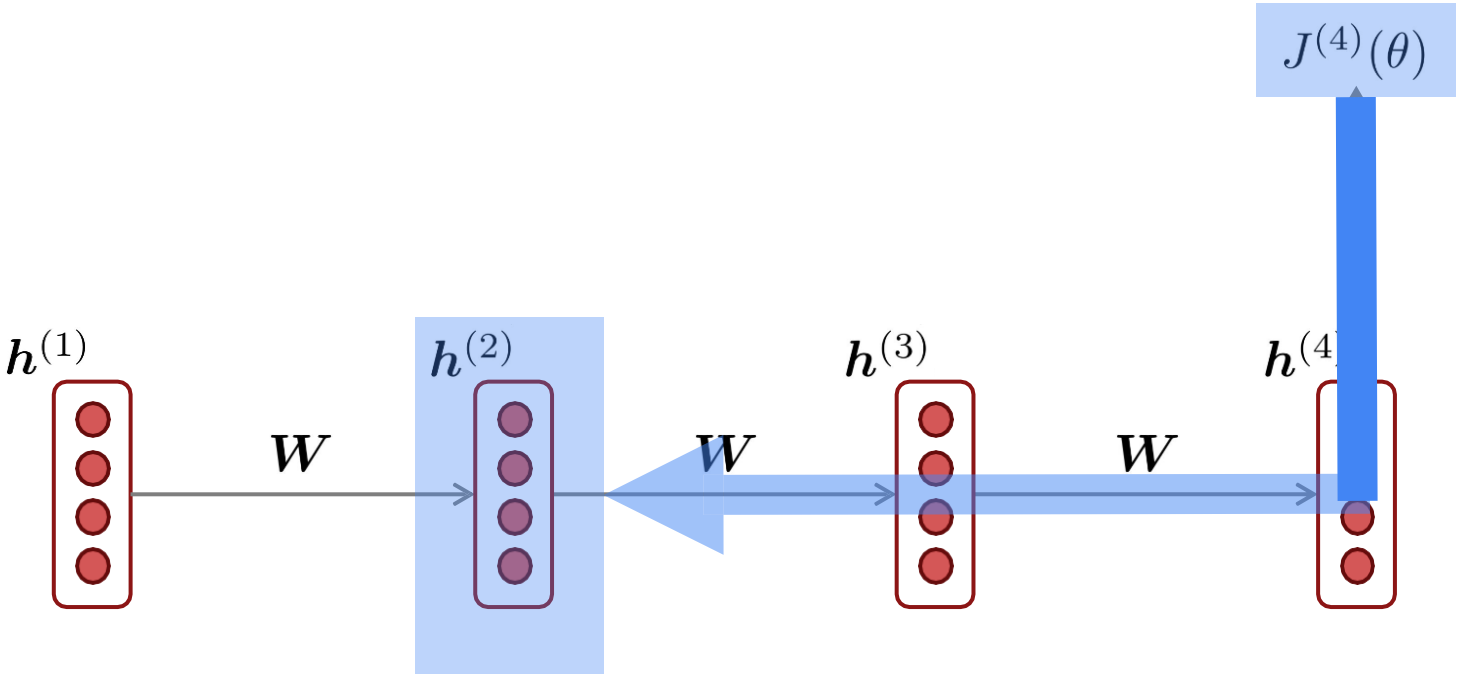


Vanishing gradient intuition



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

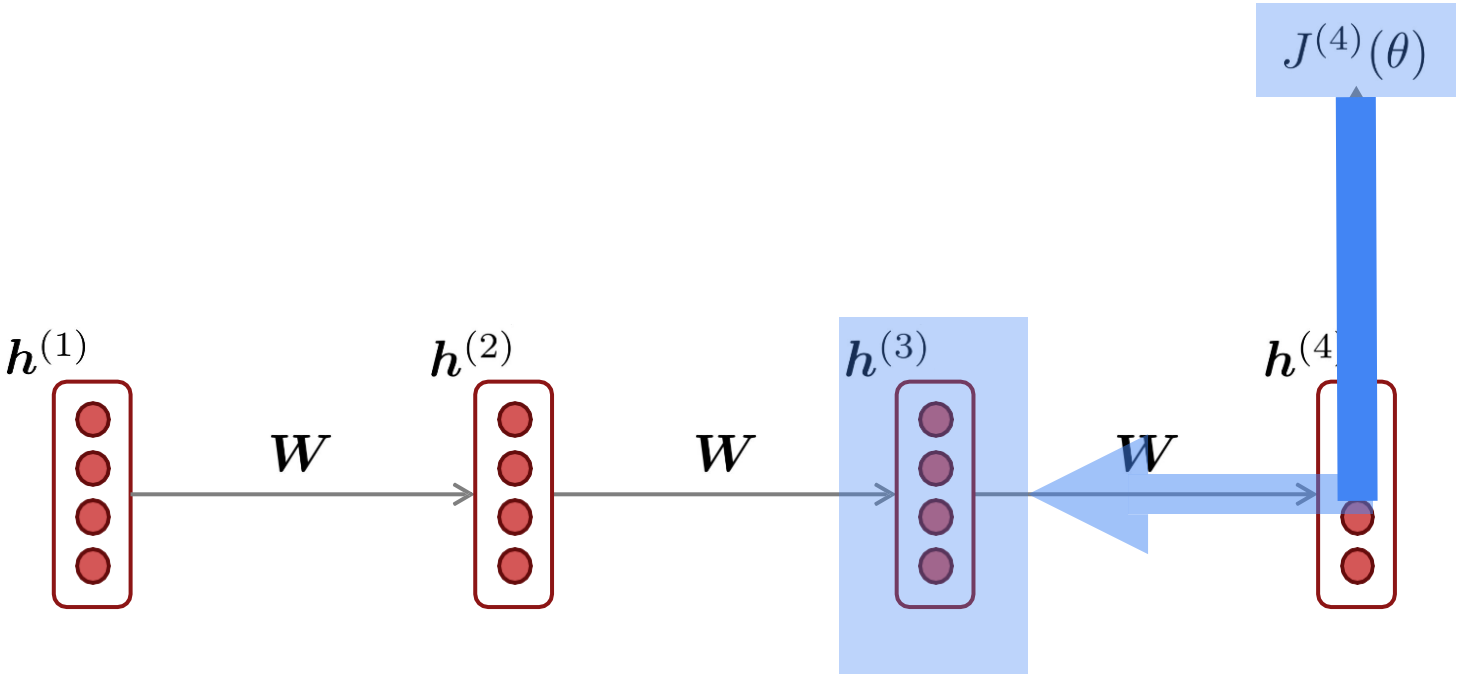
Vanishing gradient intuition



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

chain rule!

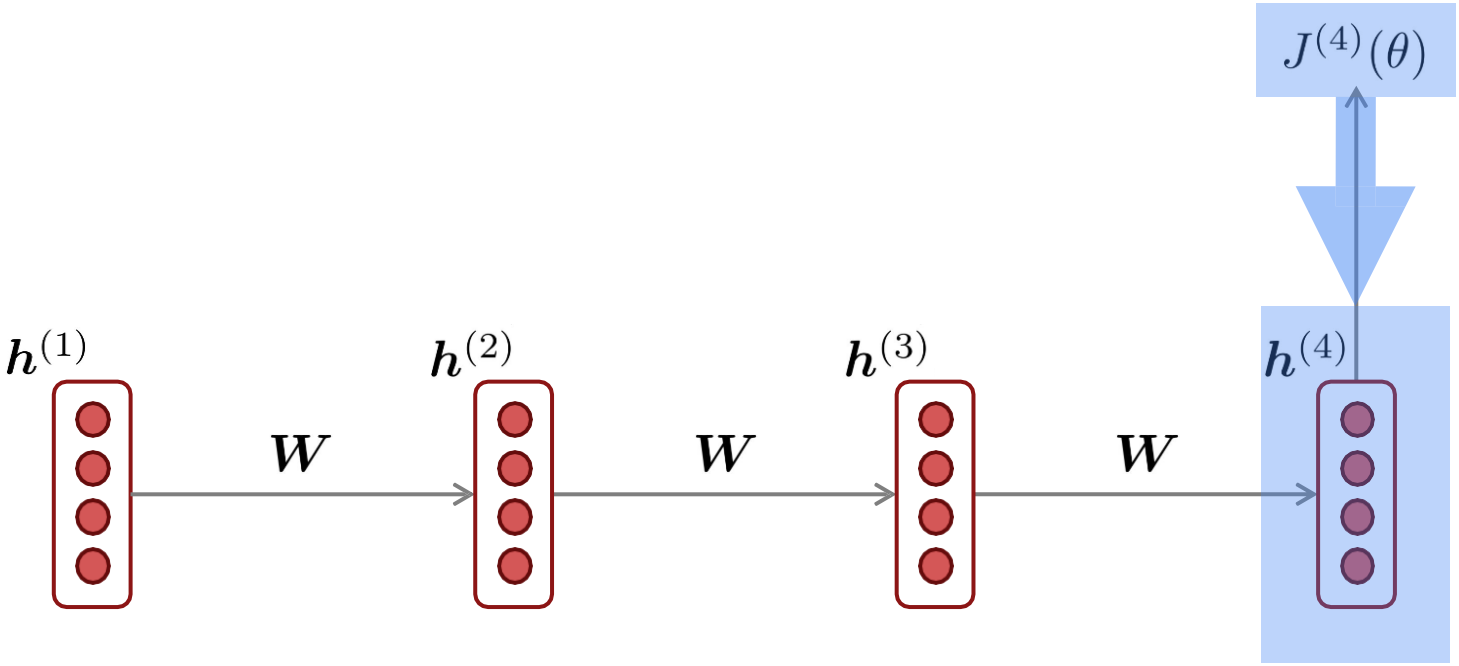
Vanishing gradient intuition



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

chain rule!

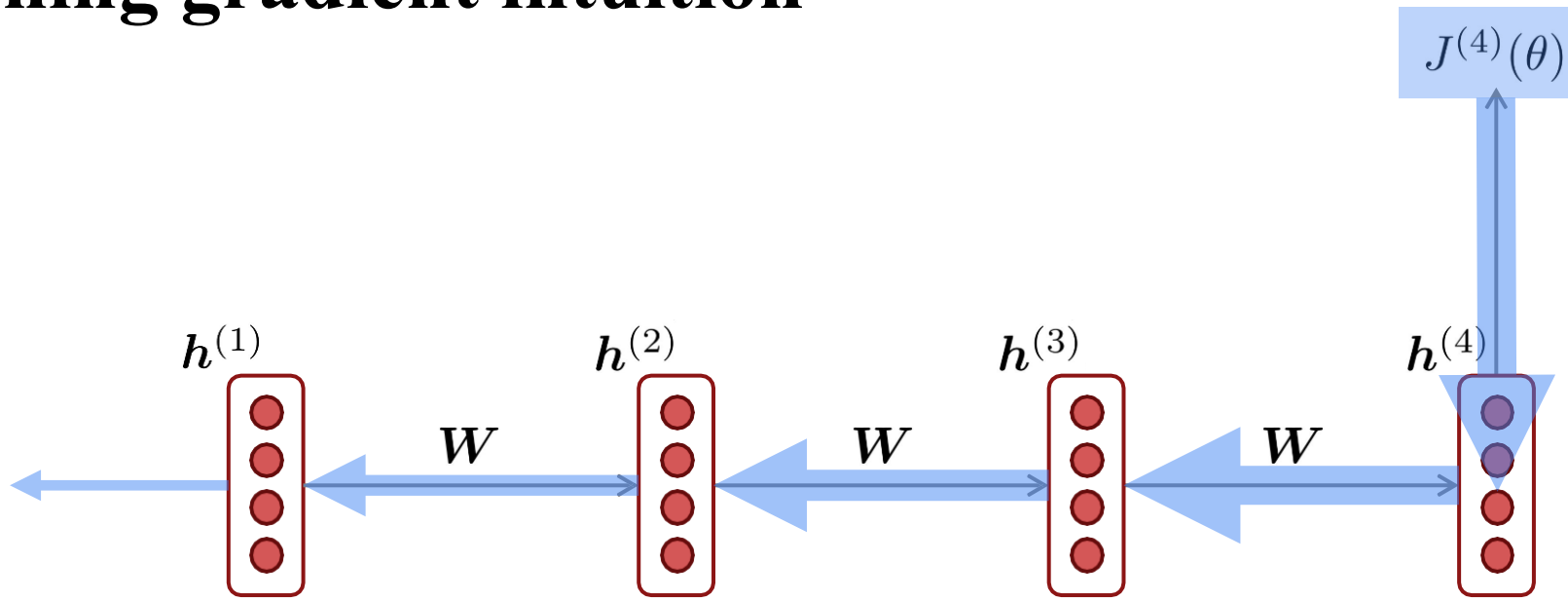
Vanishing gradient intuition



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

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

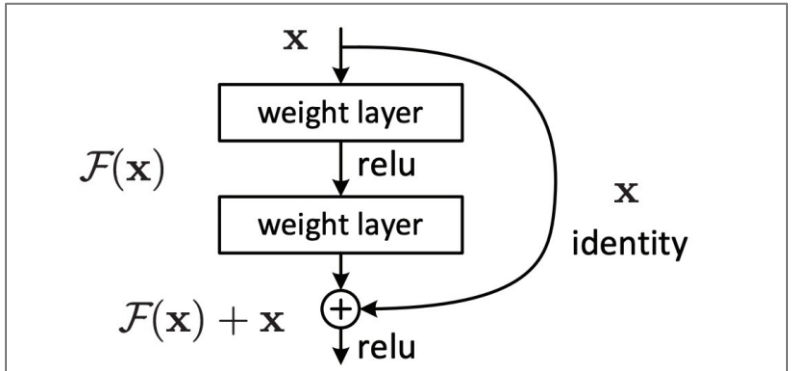


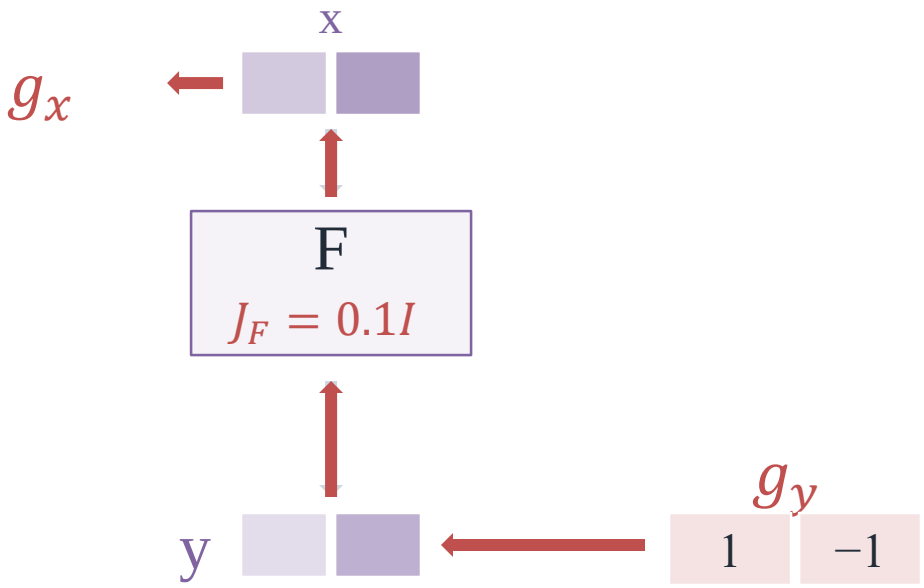
Figure 2. Residual learning: a building block.

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

Gradient Flow With and Without Residuals

The correction can reach earlier layers without passing through F.

Without residual $y = F(x)$

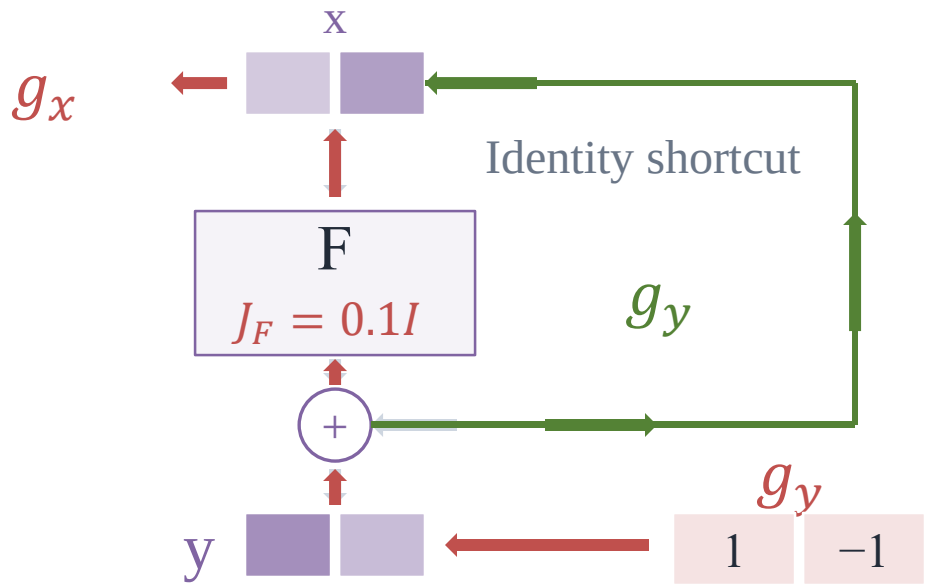


Only the transformed term

$g_x =$ 0.1 −0.1

$g_x = J_F^T g_y$

With residual $y = x + F(x)$



Direct term + transformed term

1 −1 + 0.1 −0.1 = 1.1 −1.1

$g_x = g_y + J_F^T g_y$

He et al. (2016). Deep Residual Learning for Image Recognition.
Pascanu et al. (2013). On the difficulty of training recurrent neural networks.

CNNs and RNNs: Task Comparisons

			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

Yin et al. (2017), Comparative Study of CNN and RNN for Natural Language Processing

Lecture Roadmap

1 NLP tasks and representations

2 Inductive bias: semantic abstraction and composition

3 Architectures for Neural representations

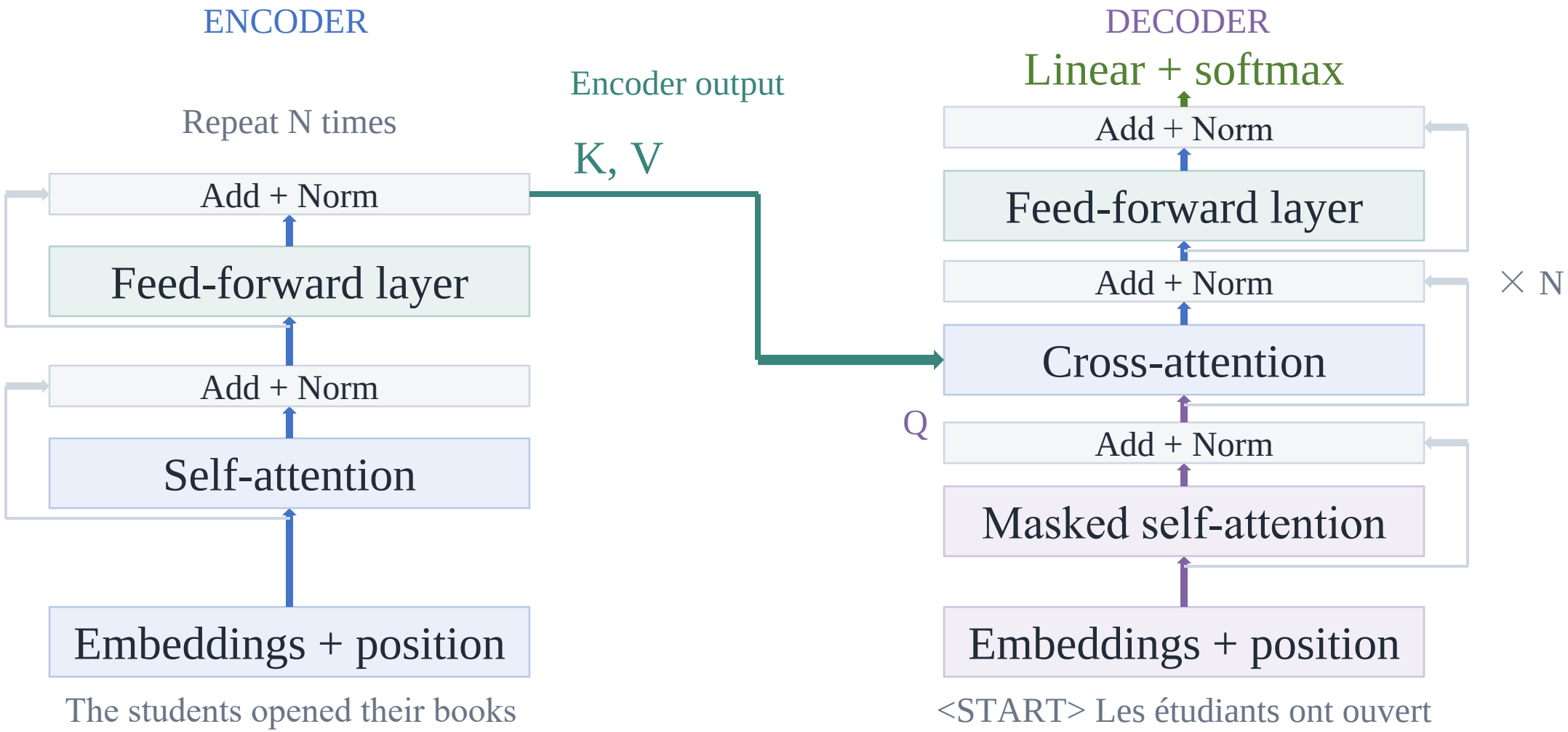
- 3.1 Multilayer perceptrons
- 3.2 Convolutional networks
- 3.3 Recurrent networks
- **3.4 Transformer**

4 Scaling laws and emergence

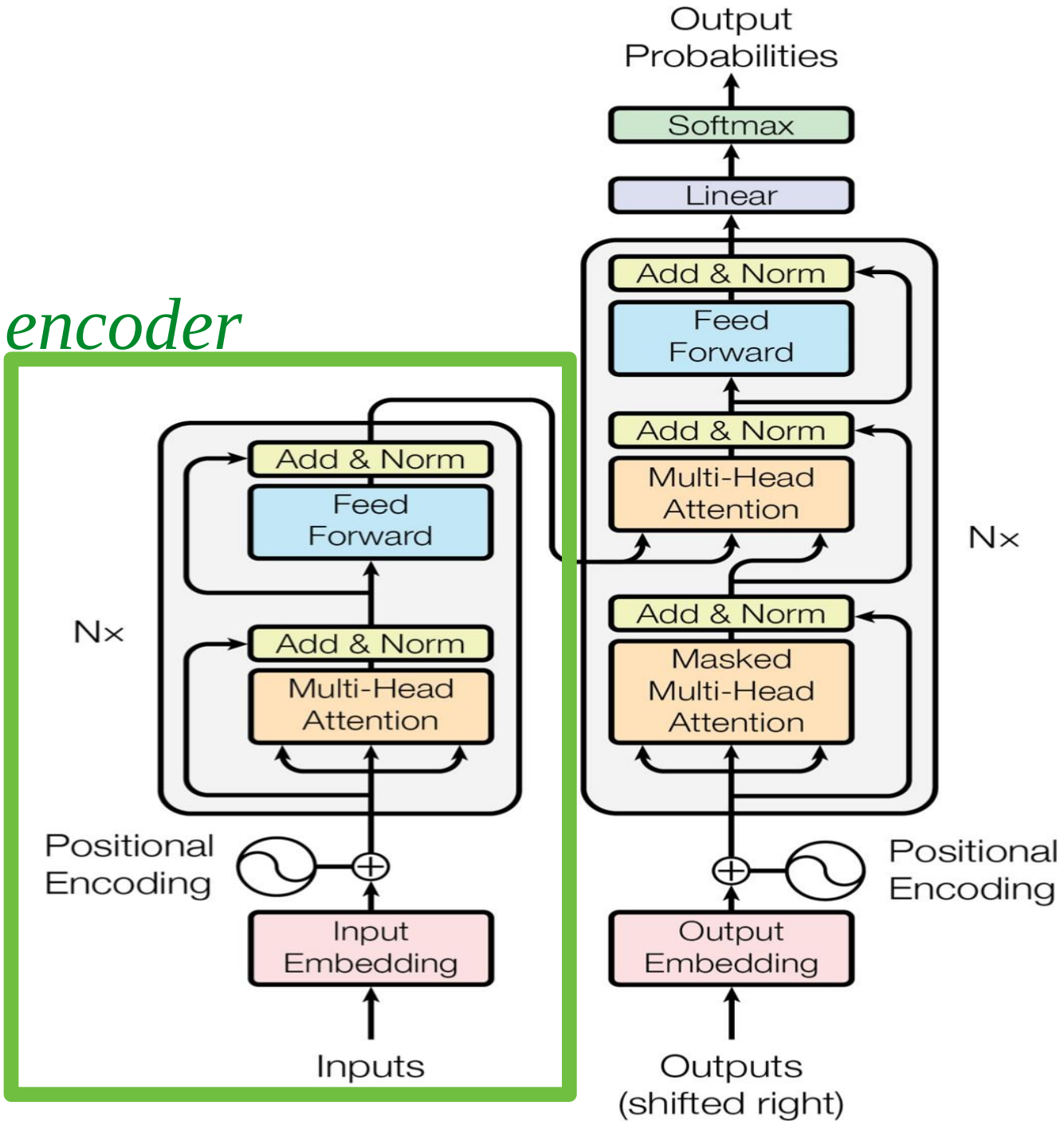
Transformer Components

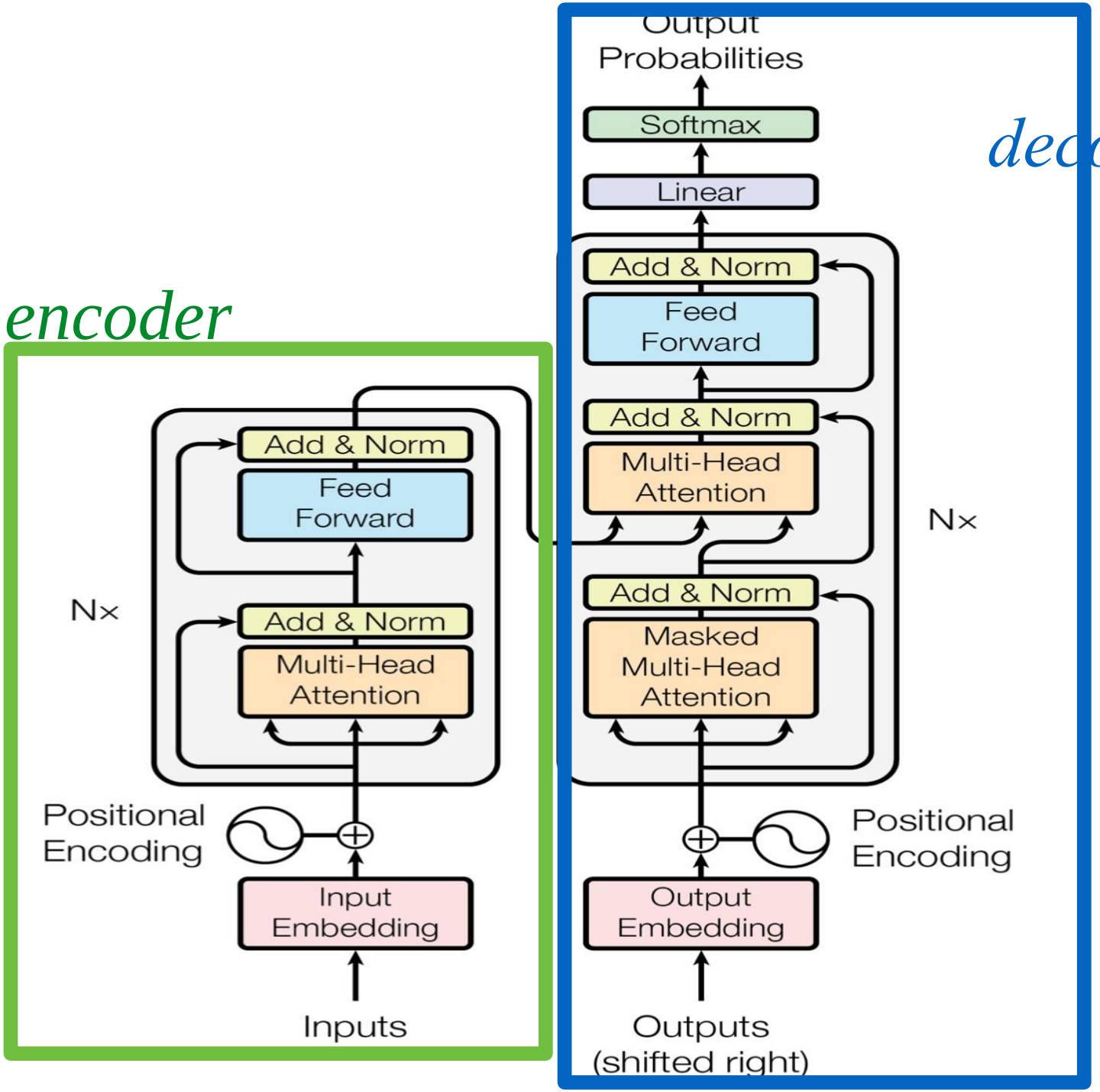
Encoder and decoder
Self-attention and cross-attention
Multiple attention heads
Position information
Residual and feed-forward layers

The Transformer Architecture

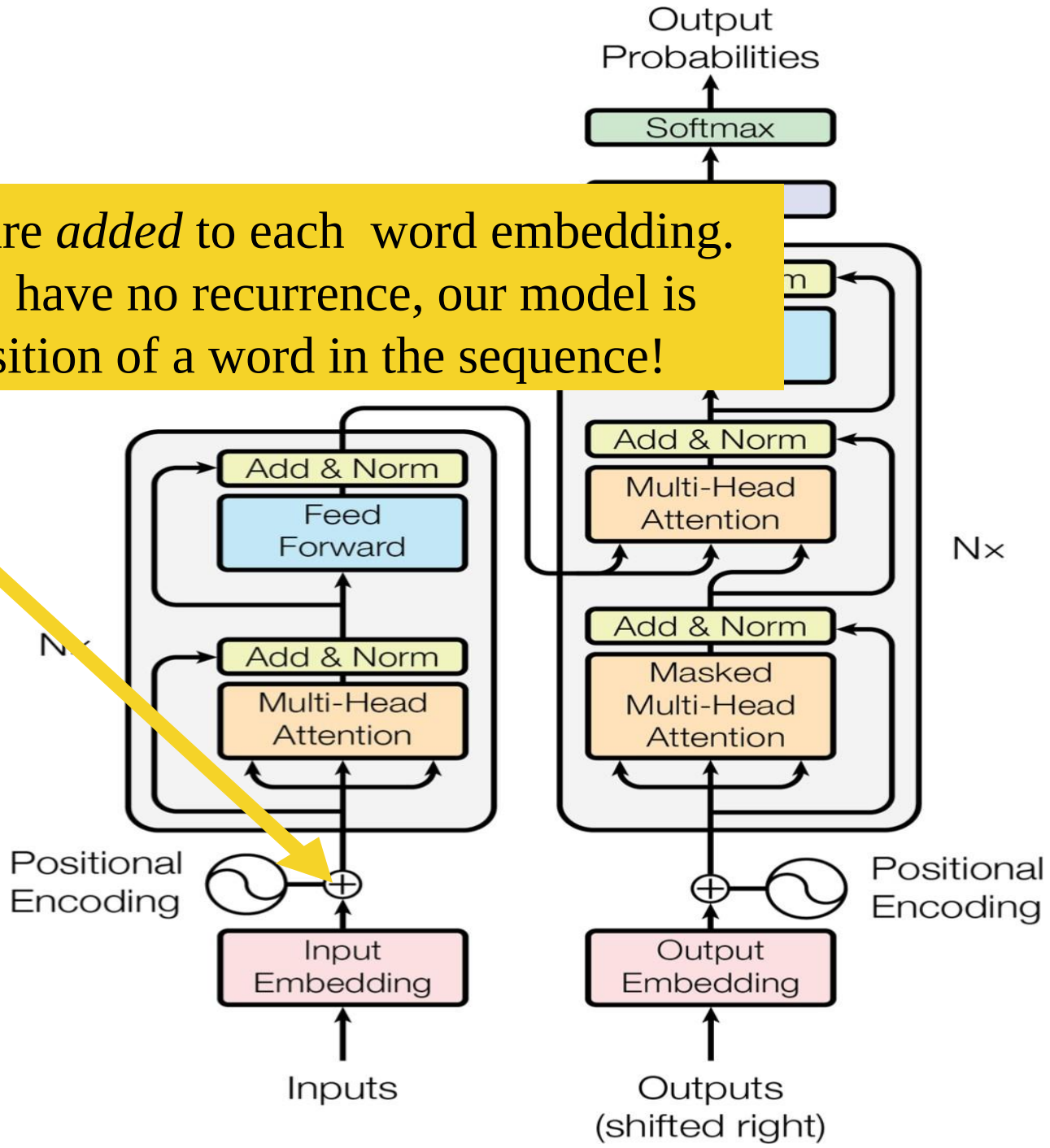


Cross-attention asks with the decoder state and reads from the encoder output.

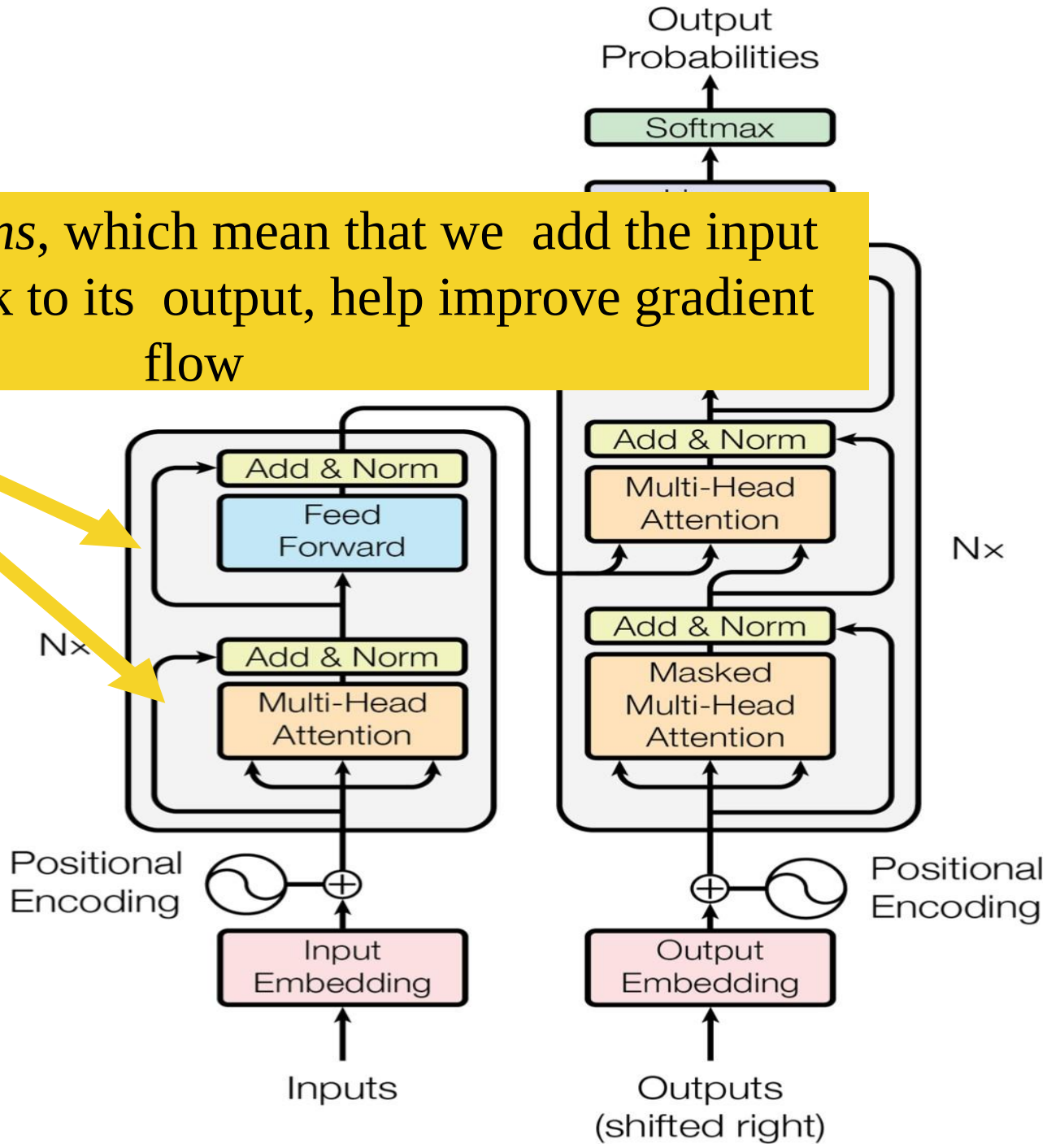

encoder
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


encoder
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

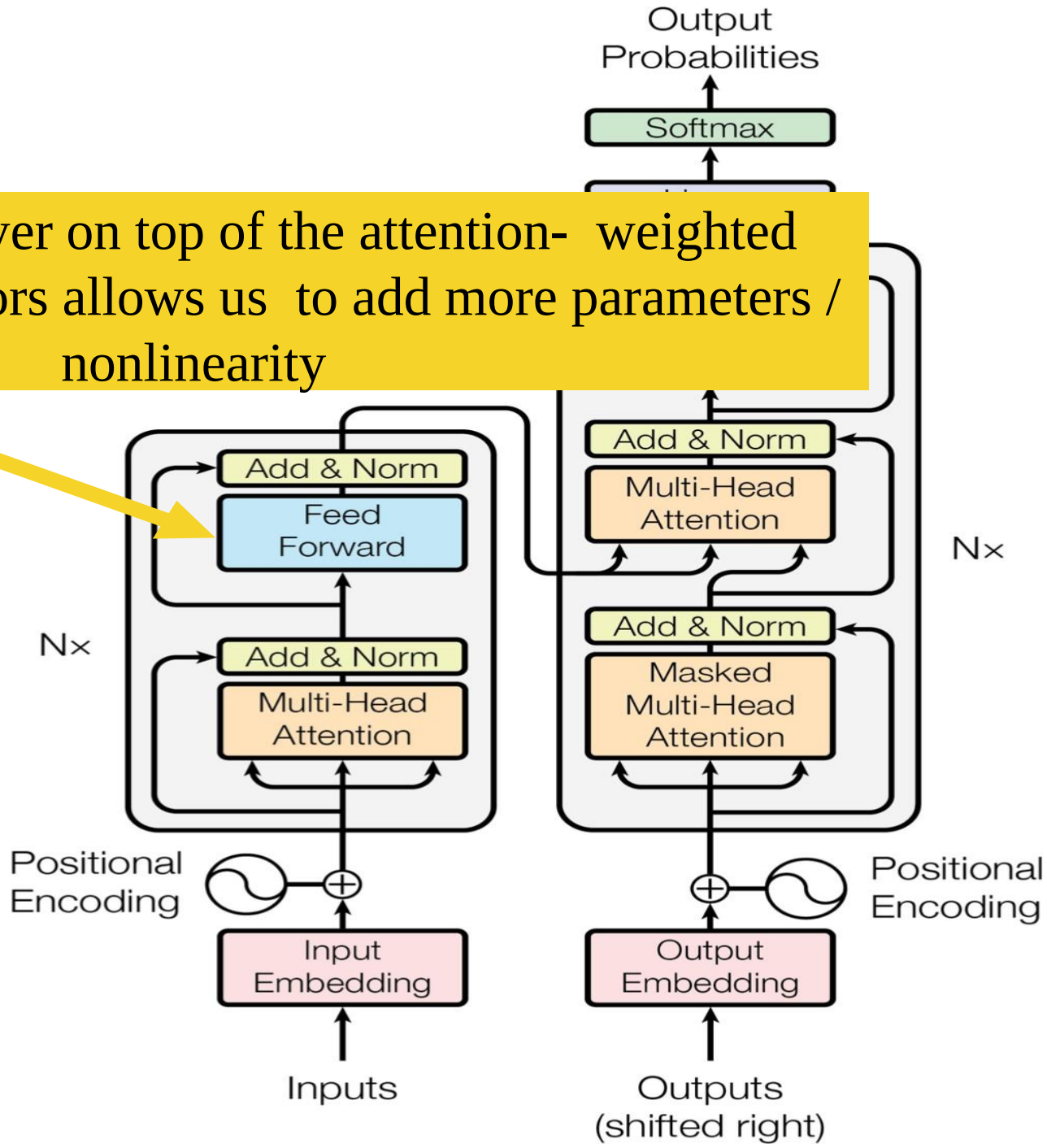
Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!



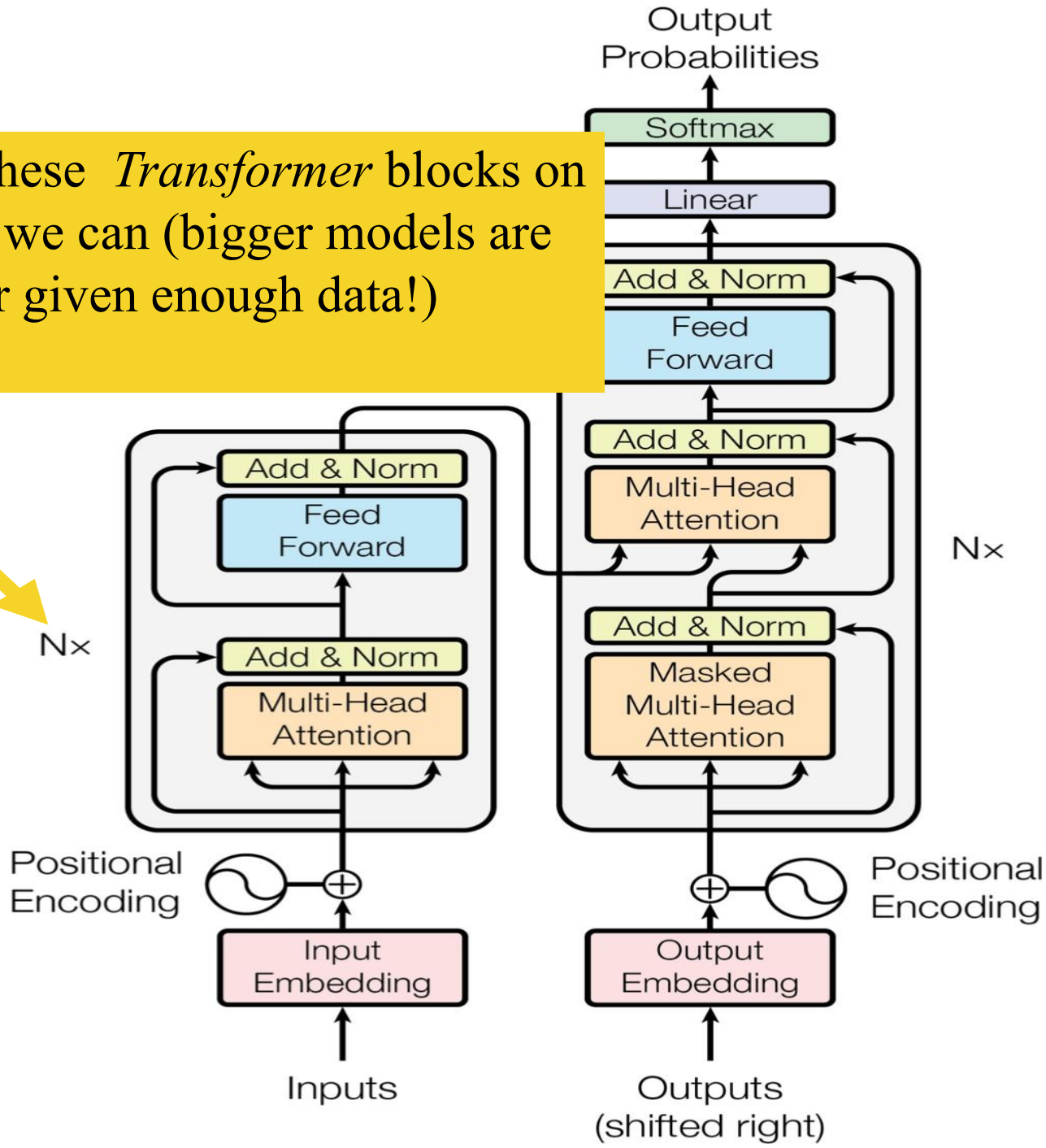
Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow



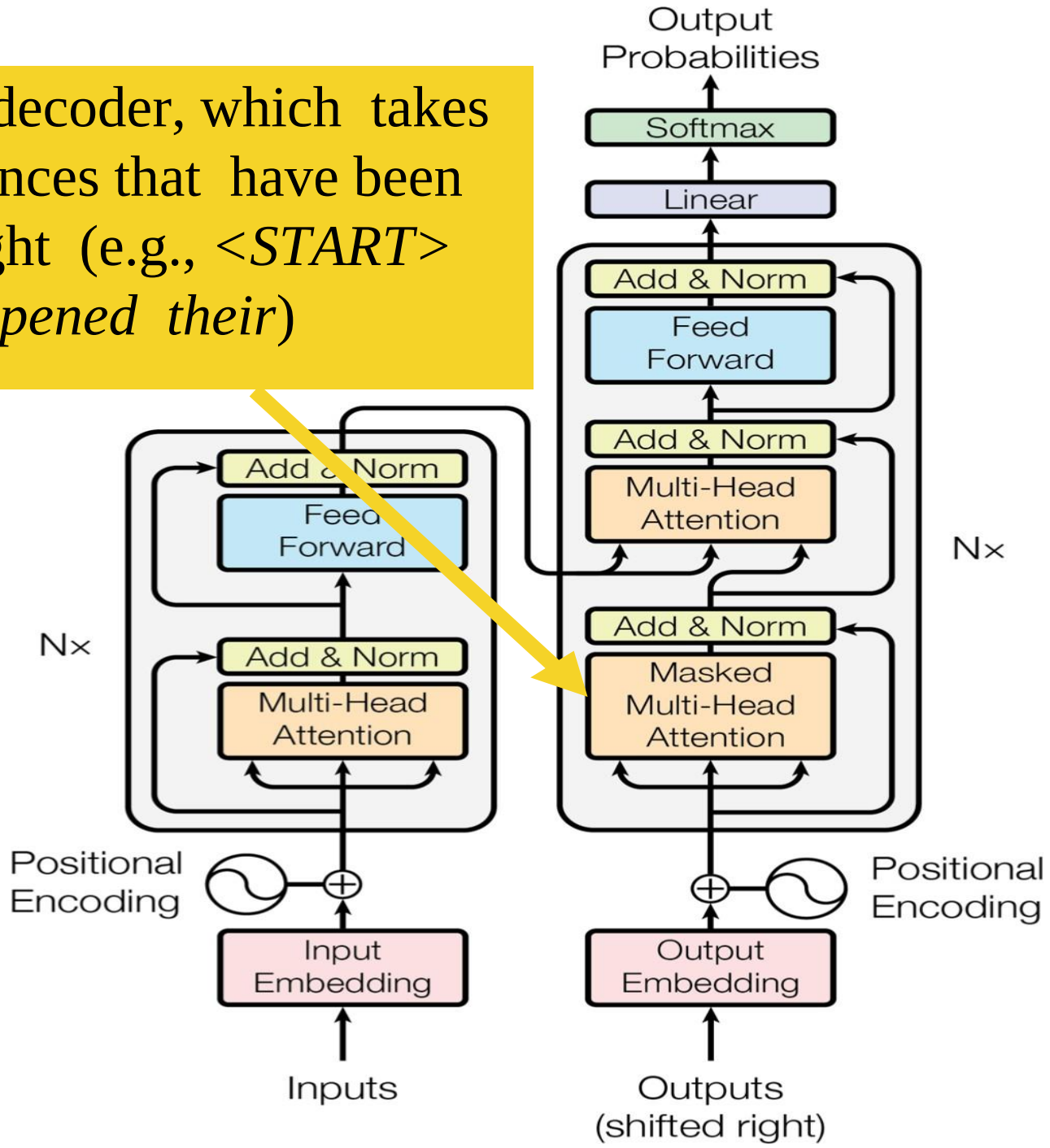
A feed-forward layer on top of the attention- weighted averaged value vectors allows us to add more parameters / nonlinearity



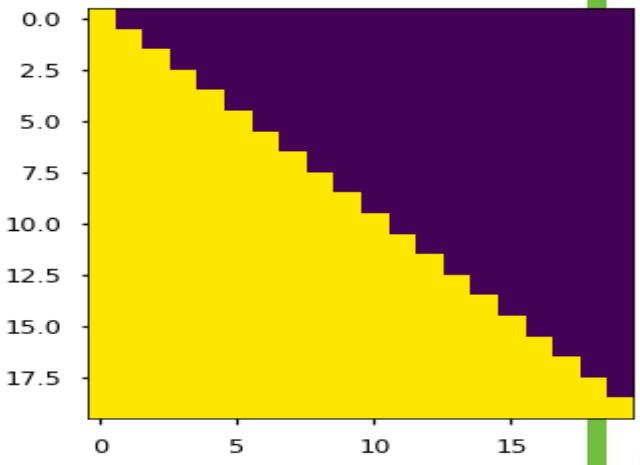
We stack as many of these *Transformer* blocks on top of each other as we can (bigger models are generally better given enough data!)

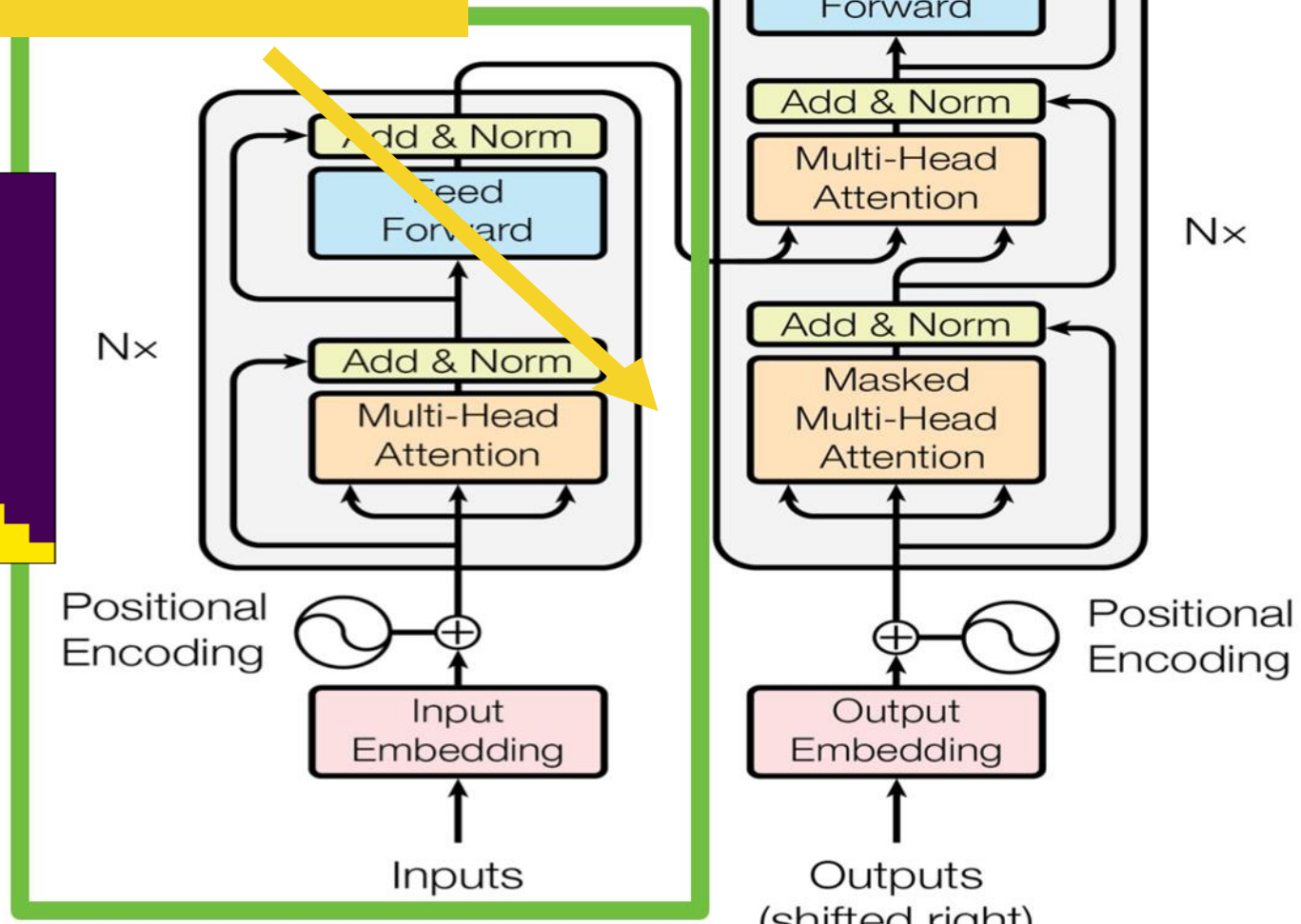


Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)



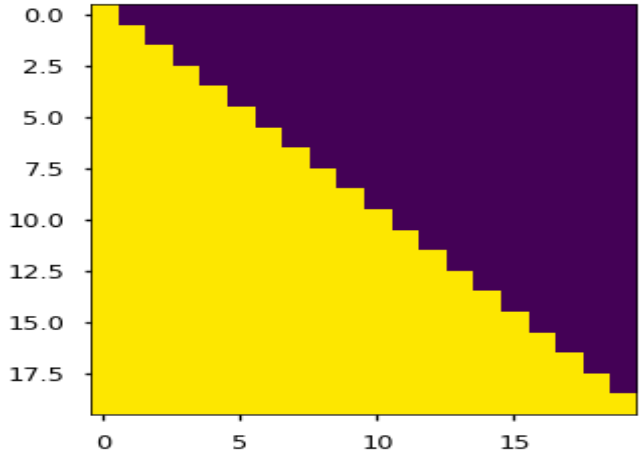
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

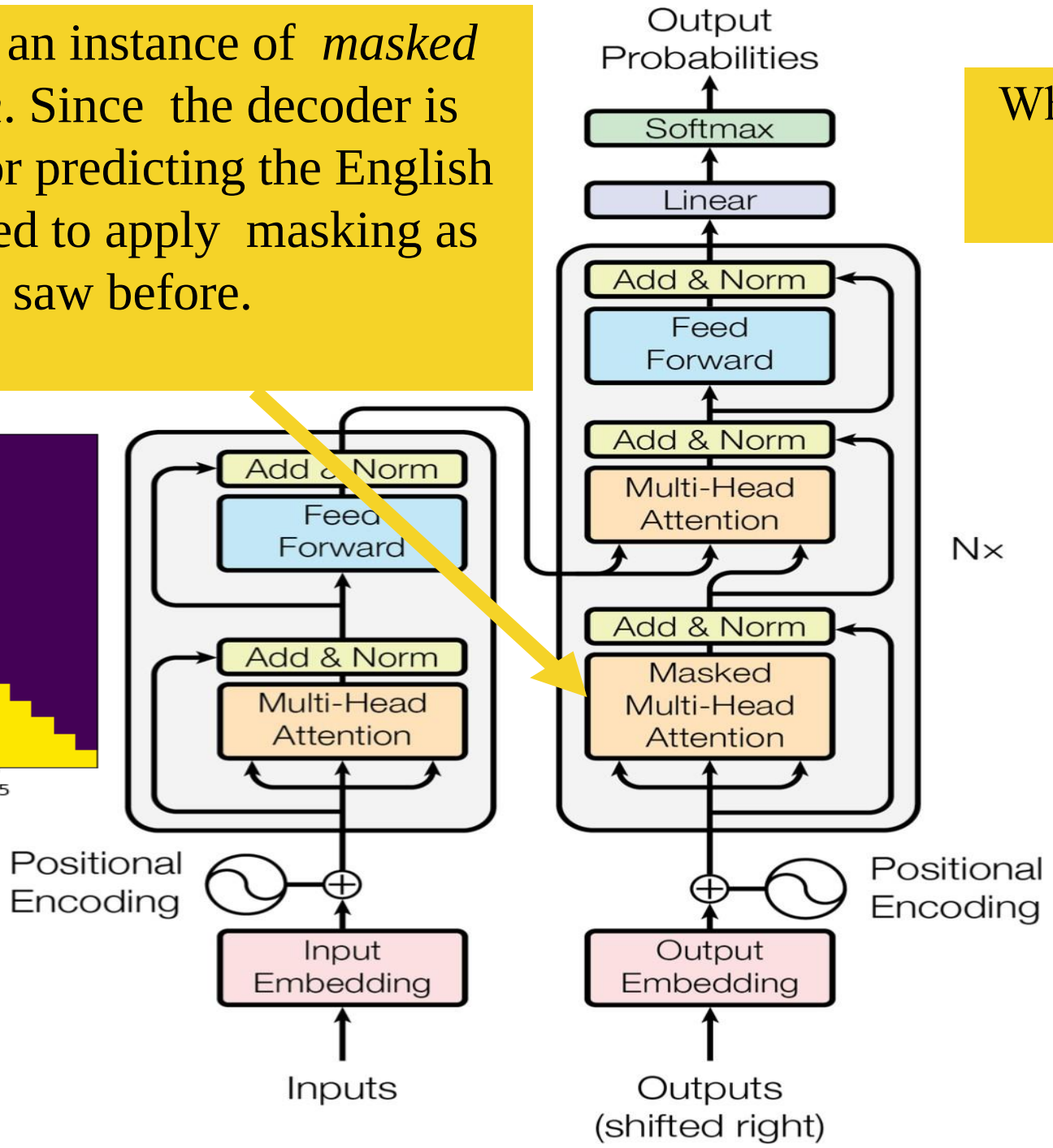




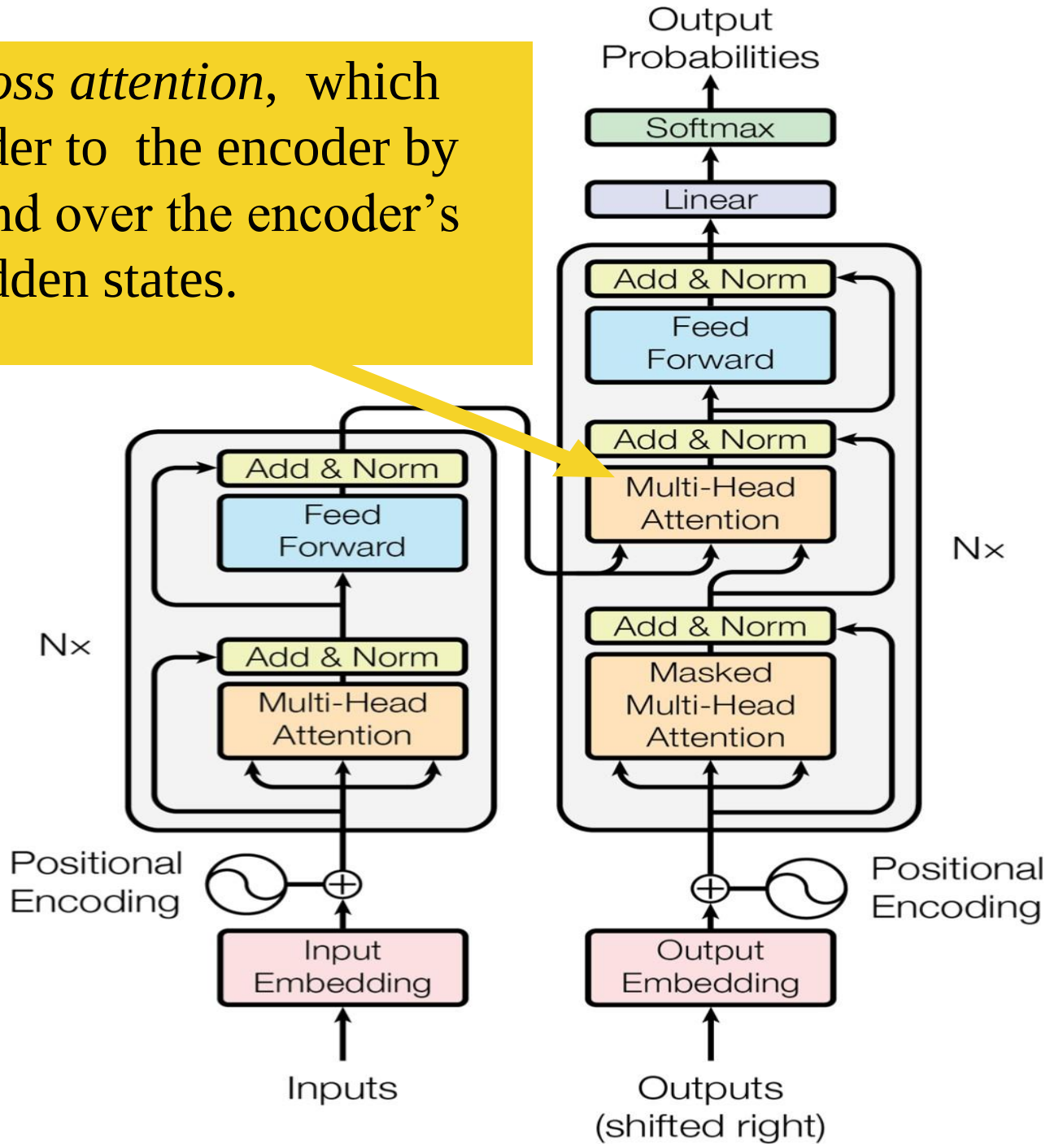
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

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

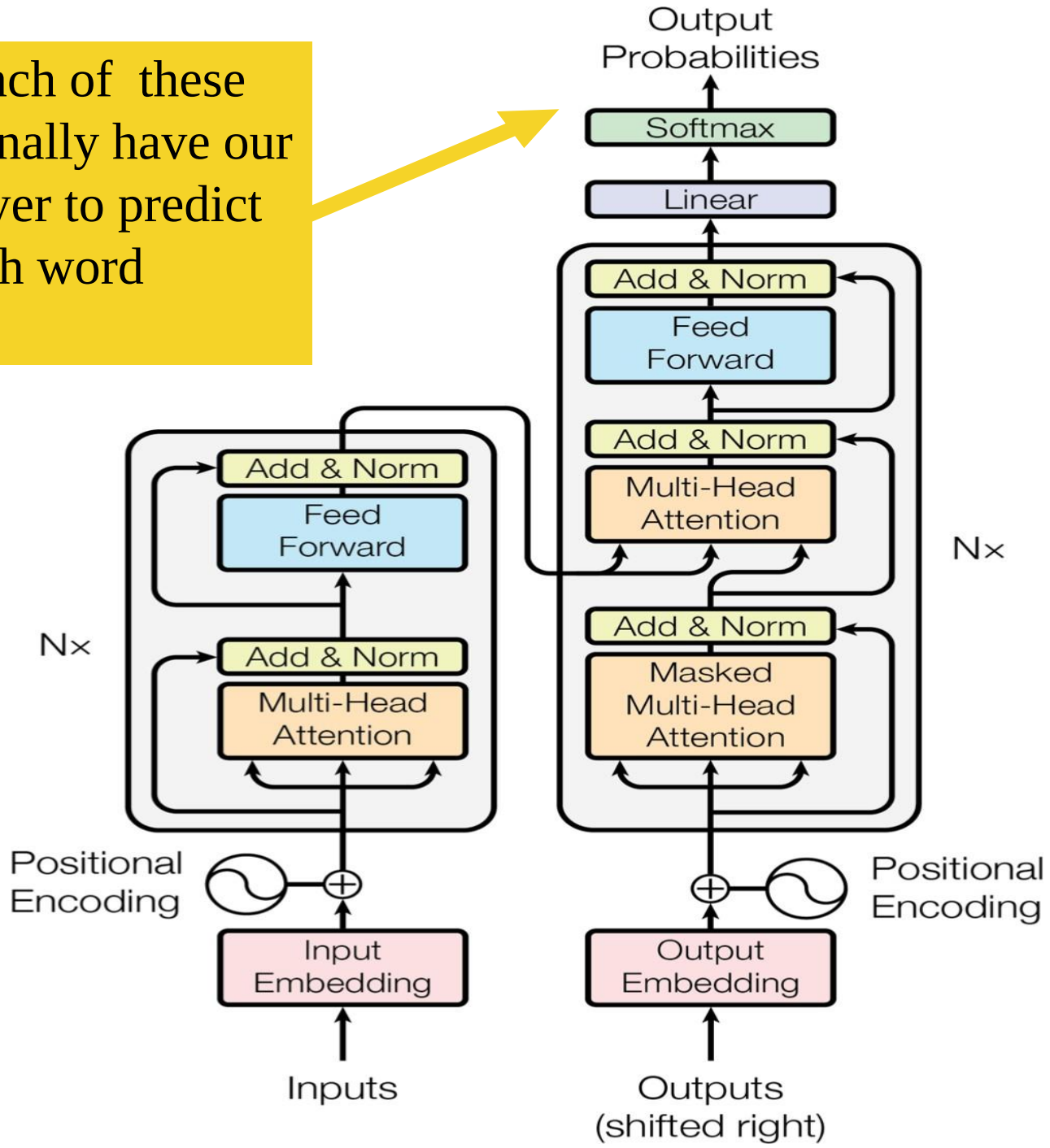


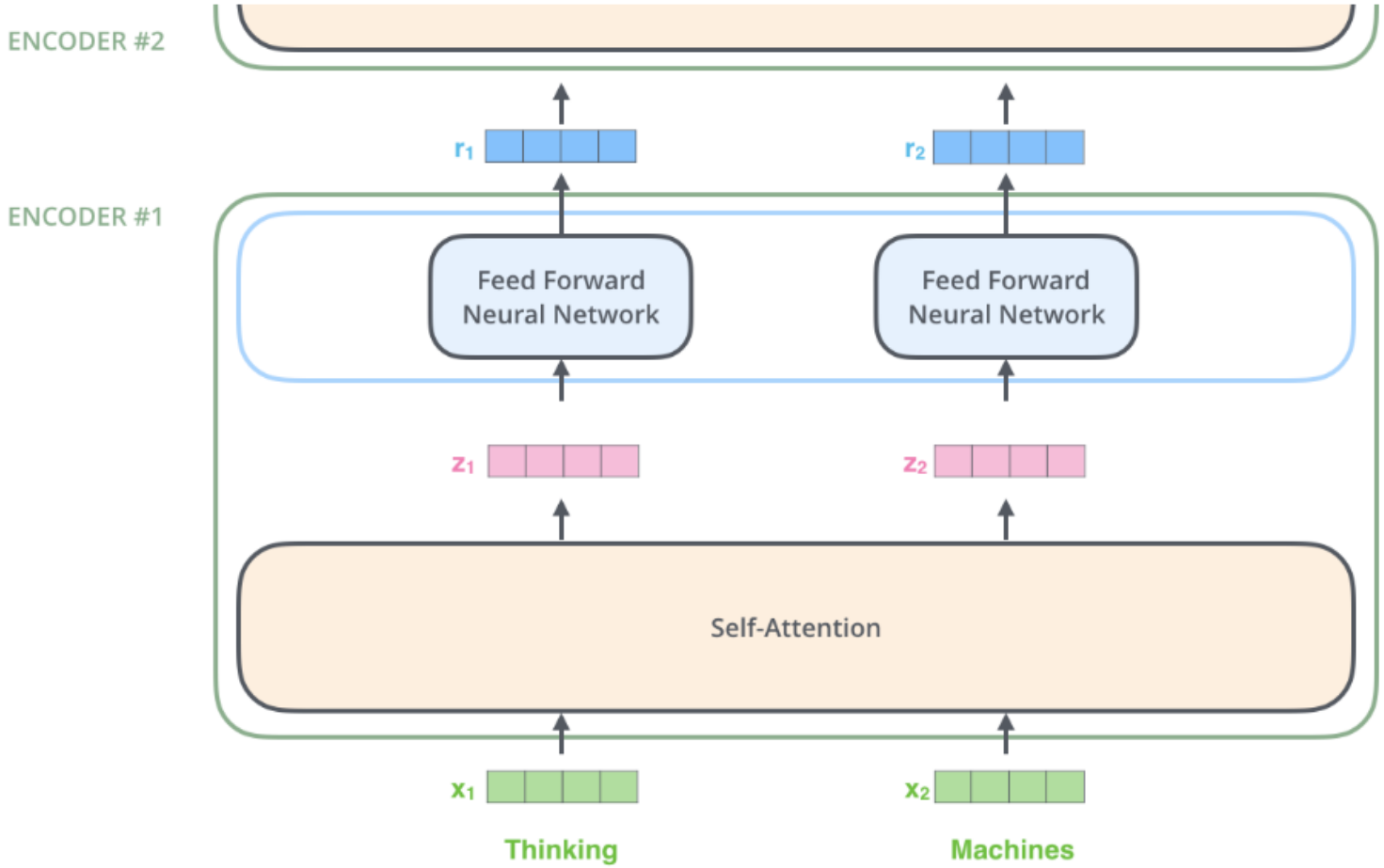


Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



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



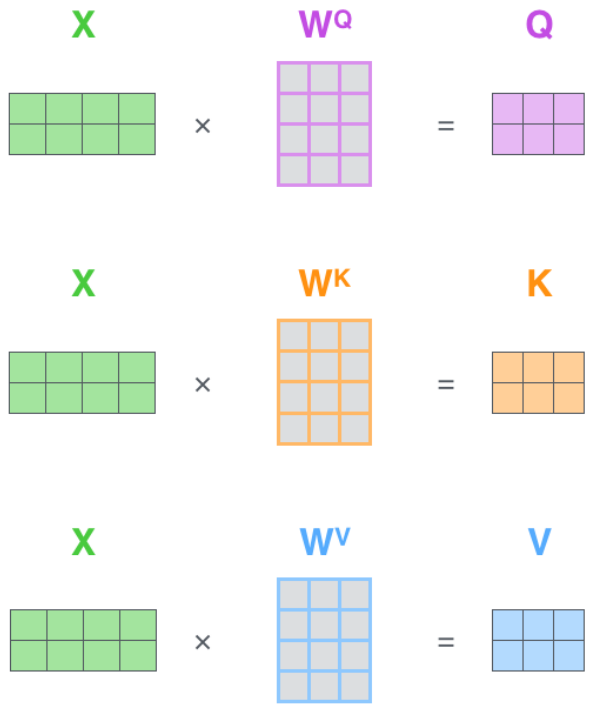


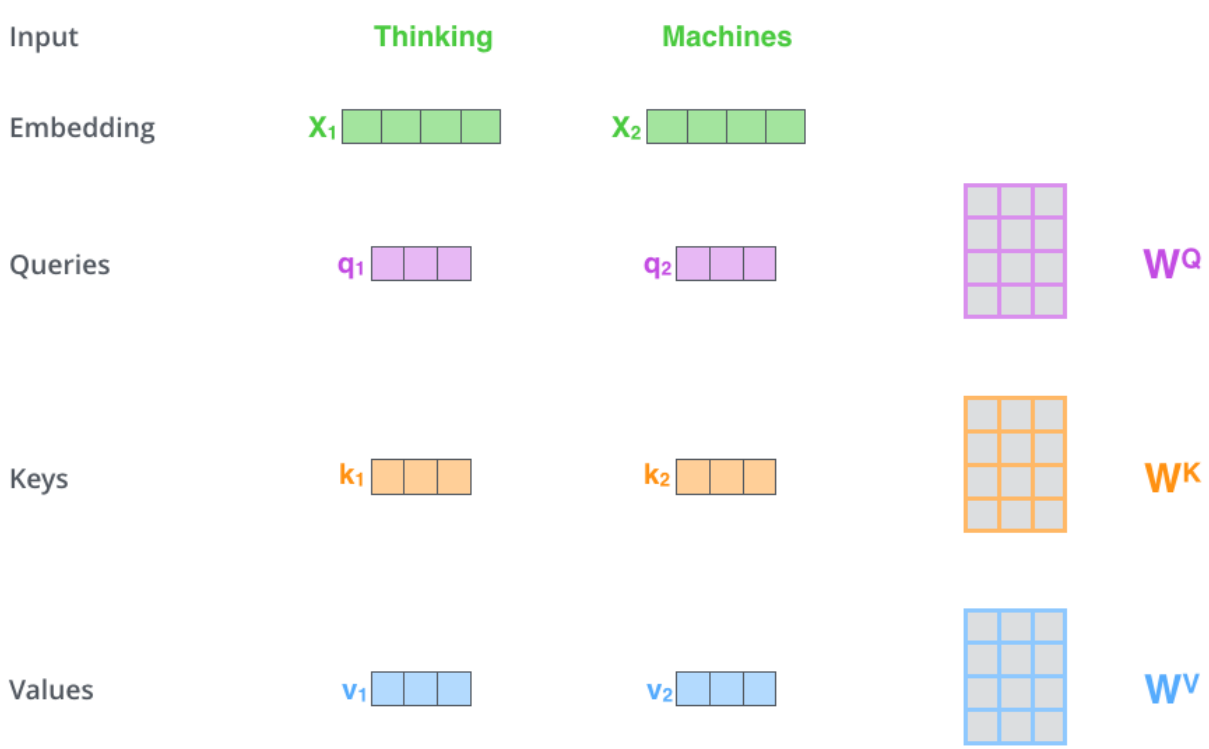
The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

Token abstraction

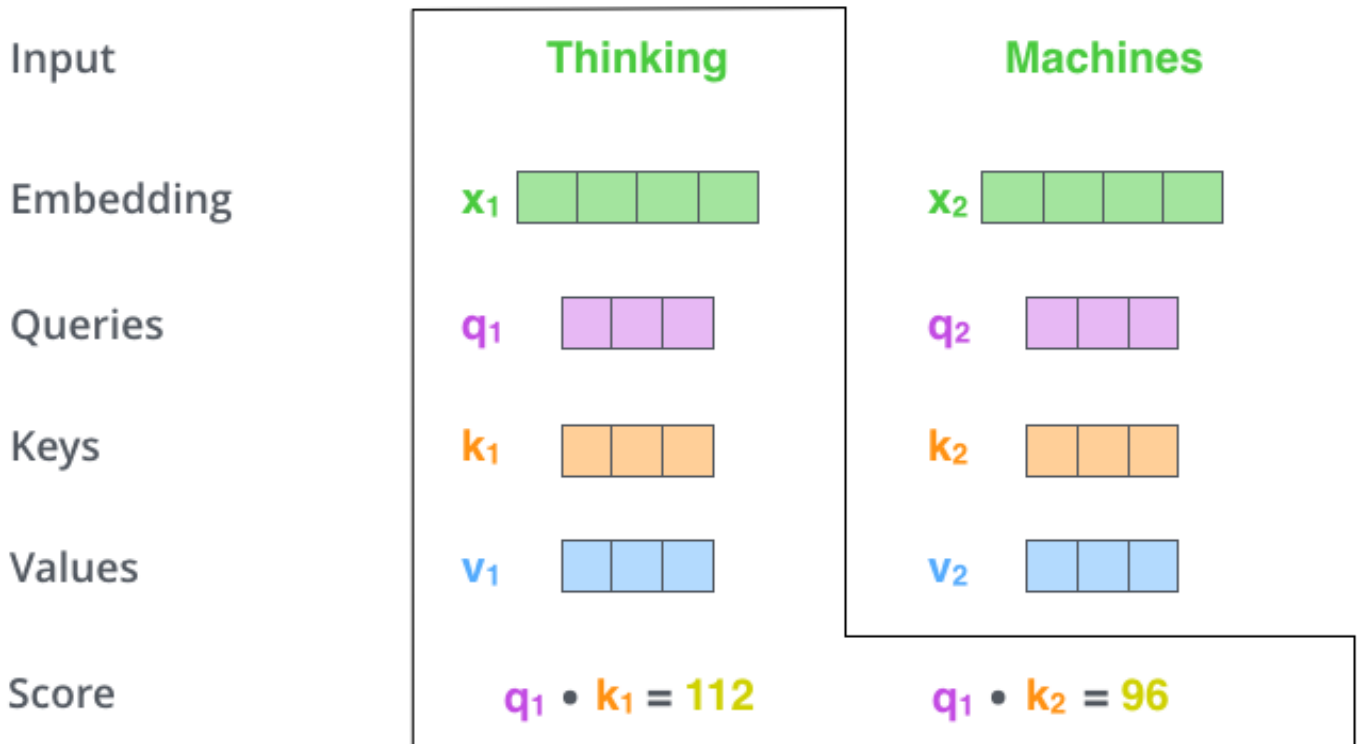
Between-token interaction

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

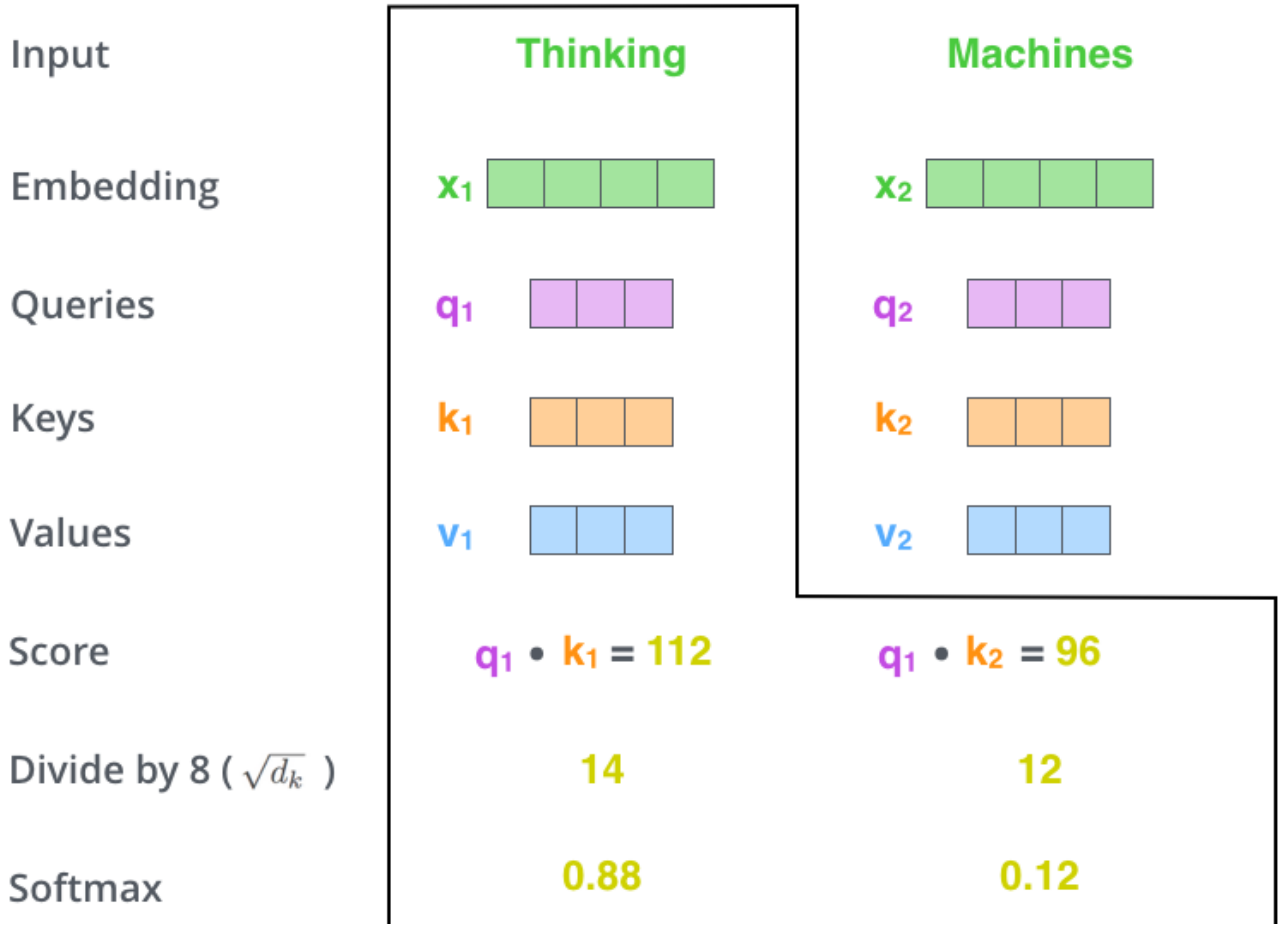




Multi-faced token representation (QKV)

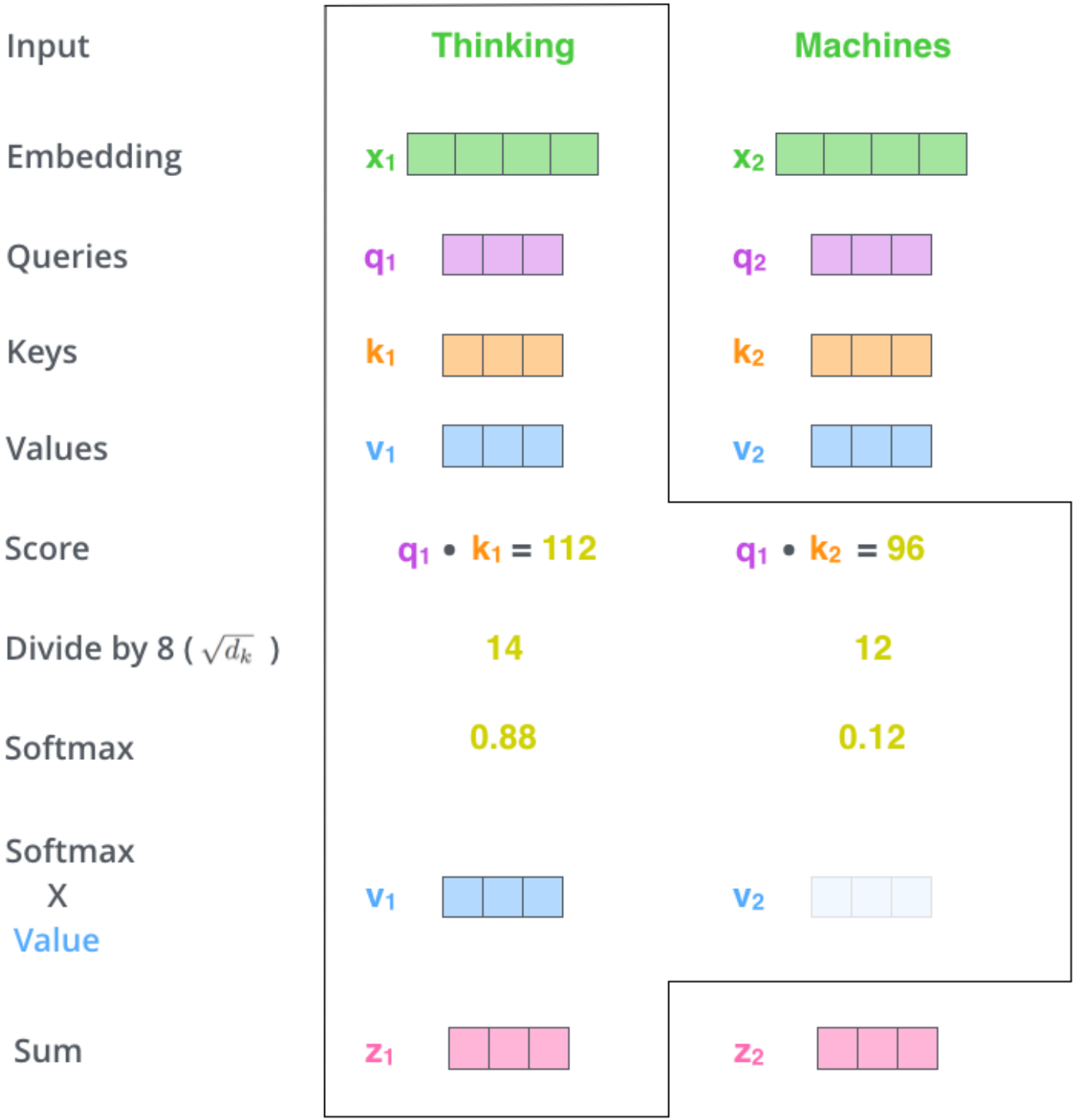


Key-value interaction between tokens



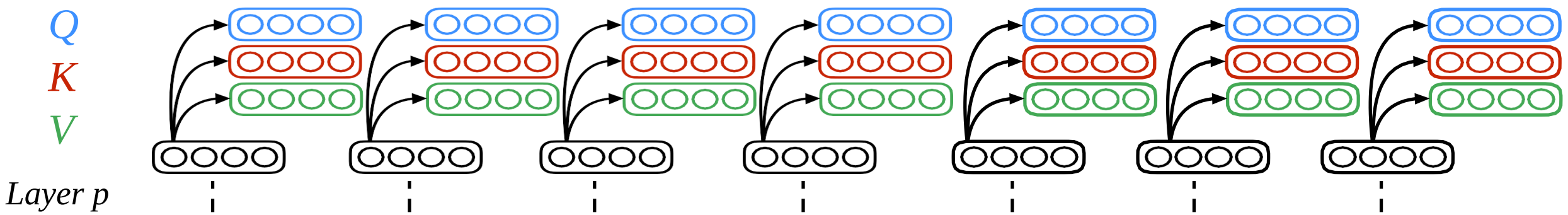
Normalize the **Key-value** interaction as attention (a probability distribution)

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

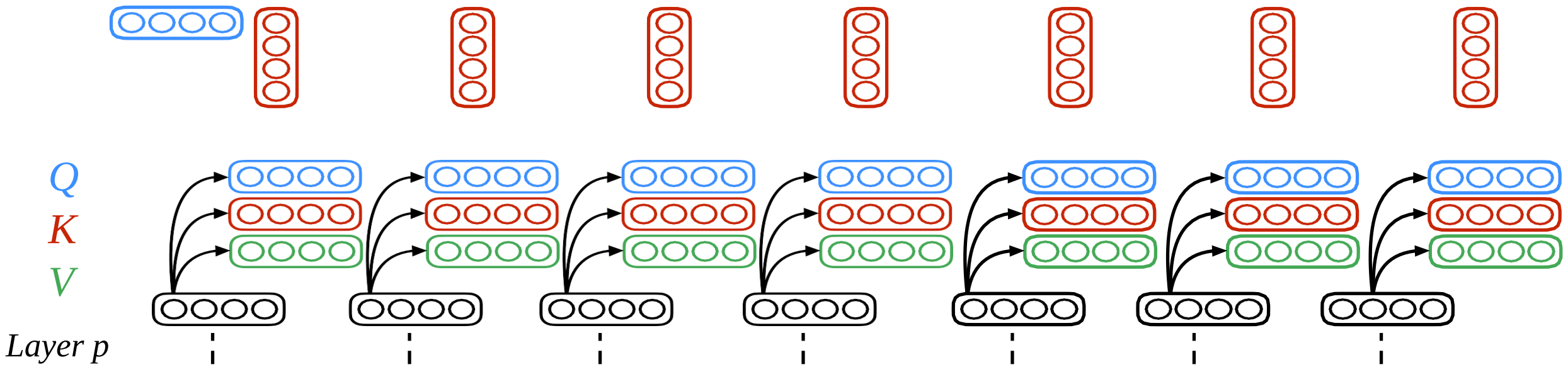
[Vaswani et al. 2017]

Self-attention (in encoder)



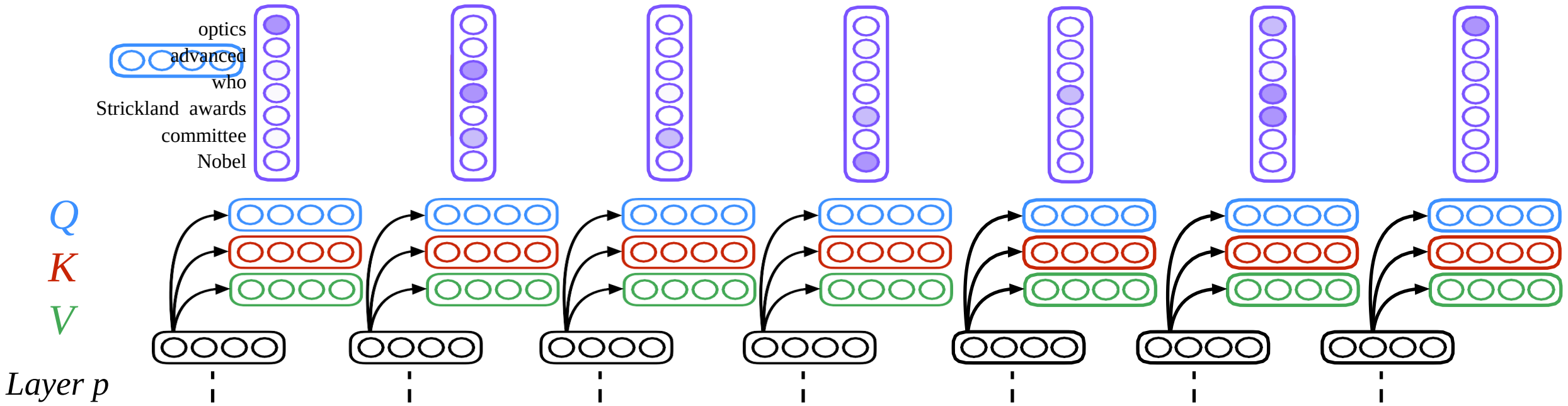
[Vaswani et al. 2017]

Self-attention (in encoder)



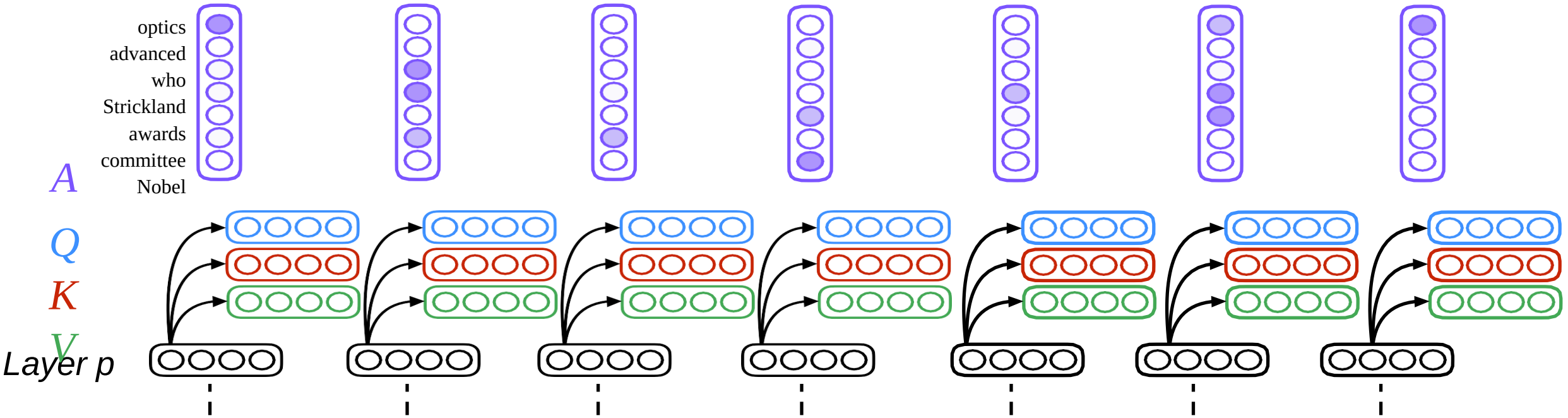
[Vaswani et al. 2017]

Self-attention (in encoder)



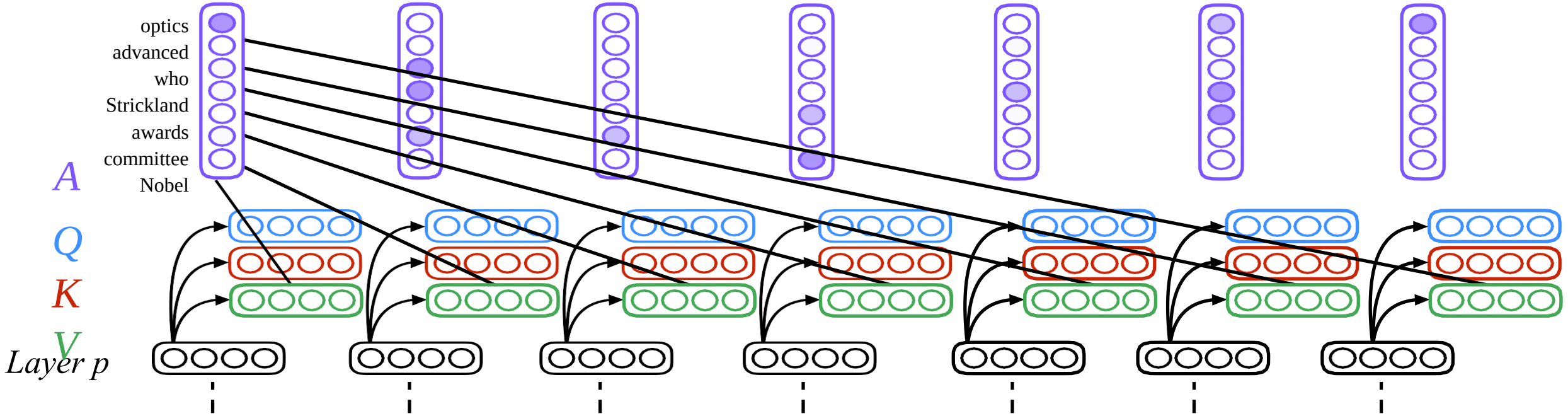
[Vaswani et al. 2017]

Self-attention (in encoder)



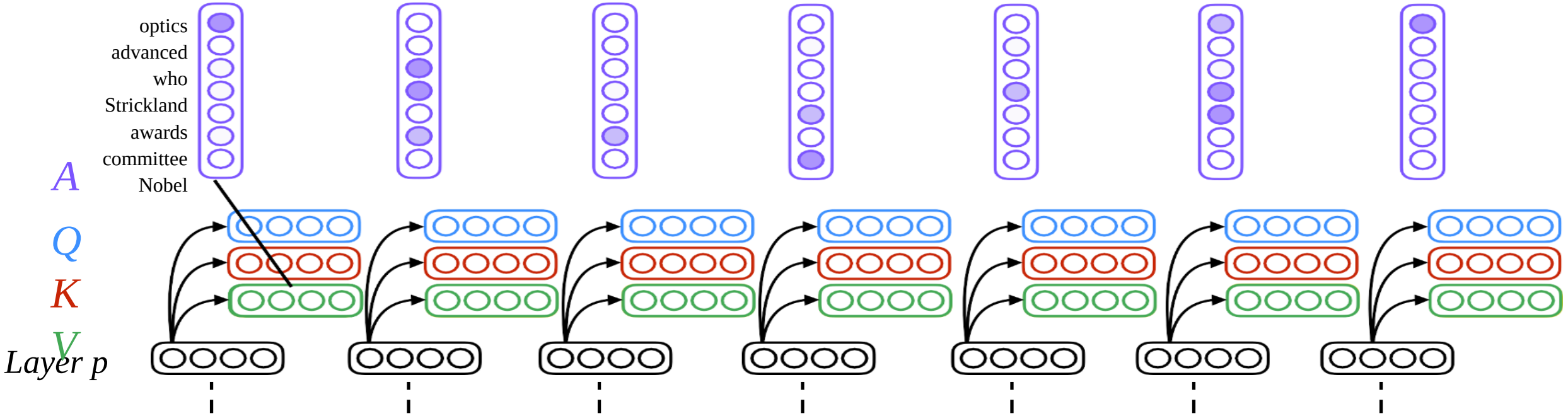
[Vaswani et al. 2017]

Self-attention (in encoder)



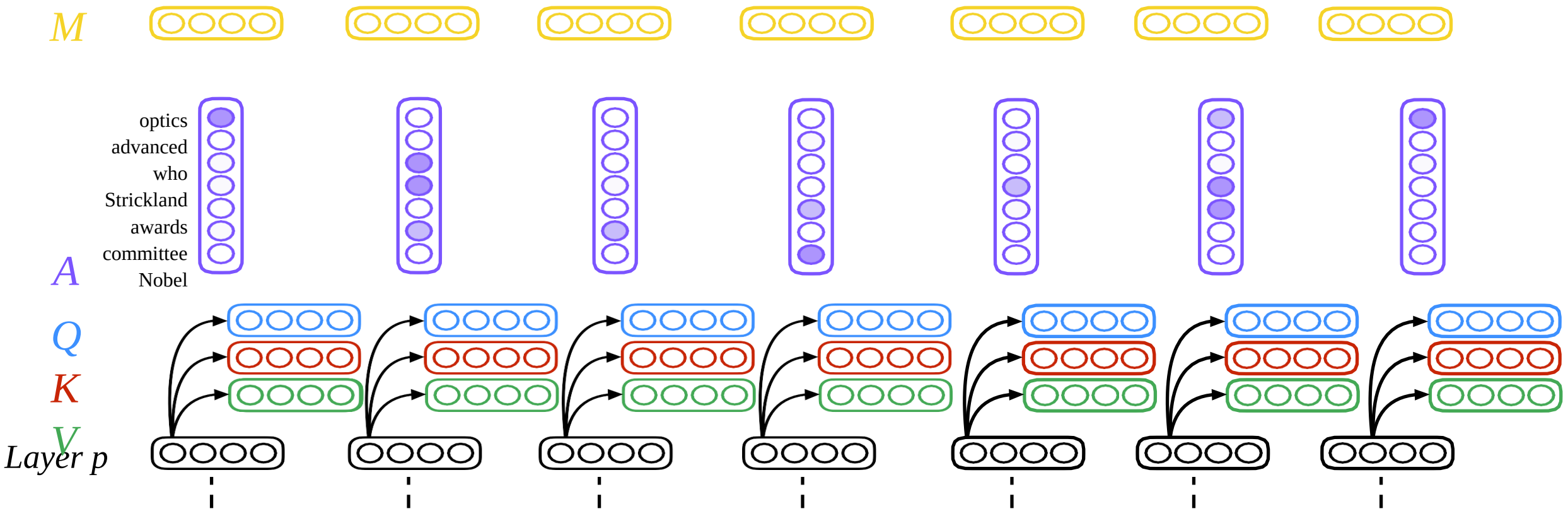
[Vaswani et al. 2017]

Self-attention (in encoder)



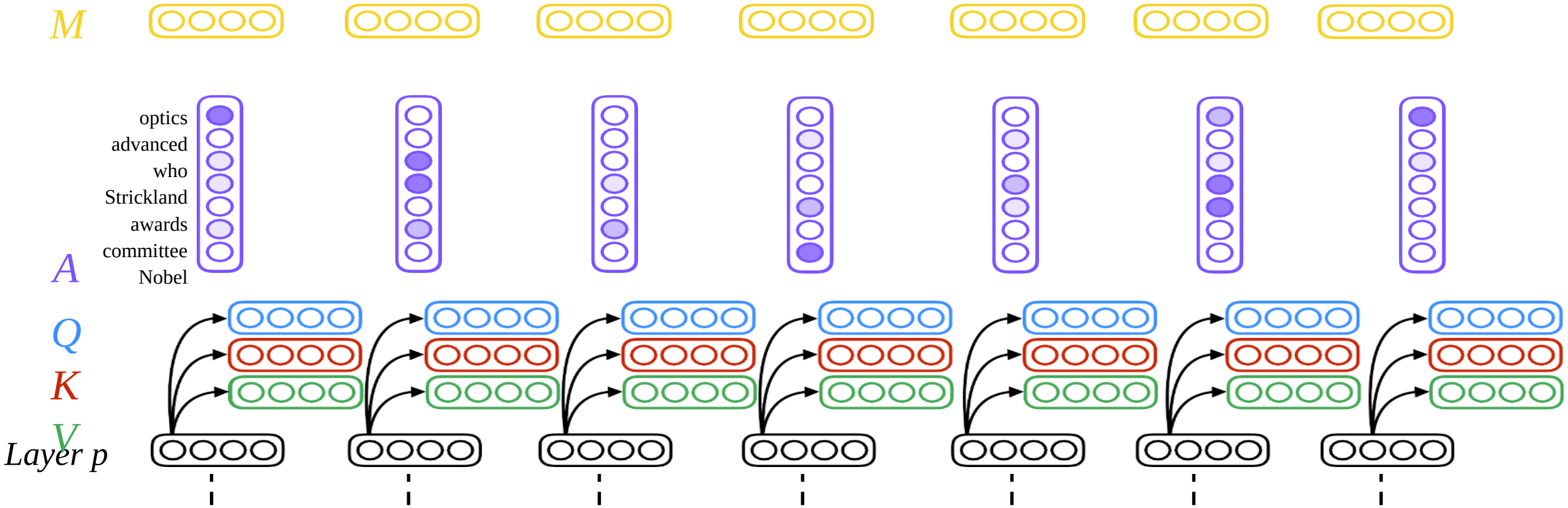
[Vaswani et al. 2017]

Self-attention (in encoder)



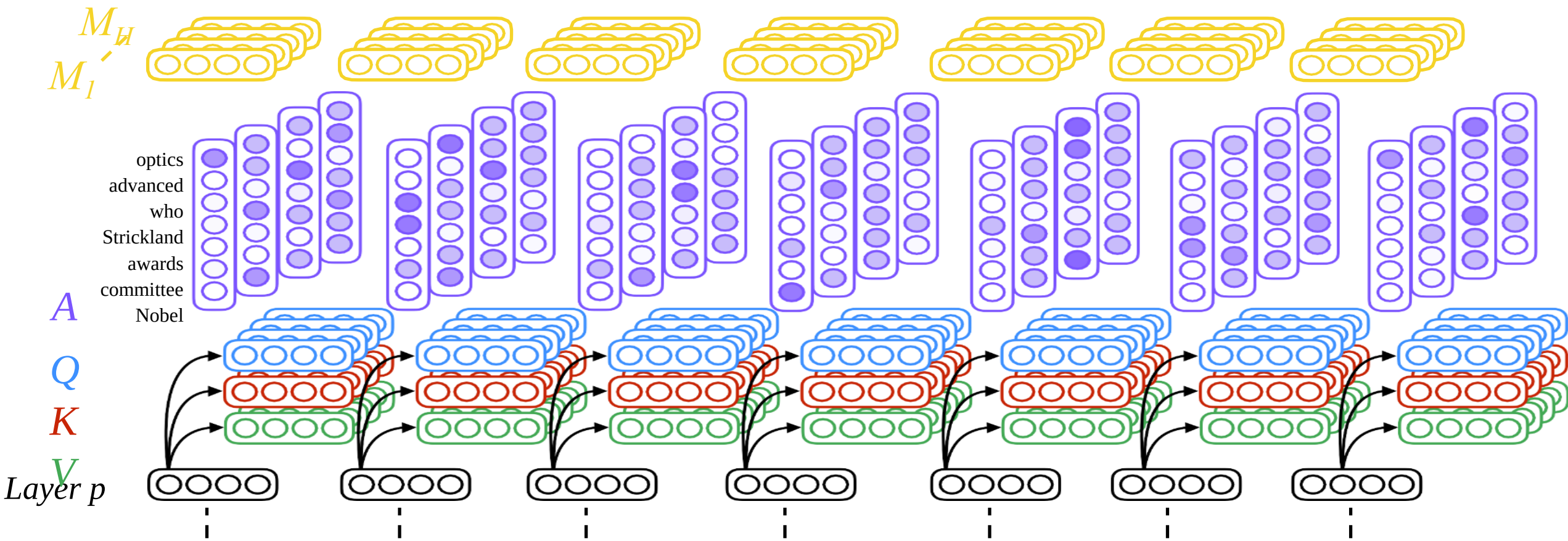
[Vaswani et al. 2017]

Self-attention (in encoder)



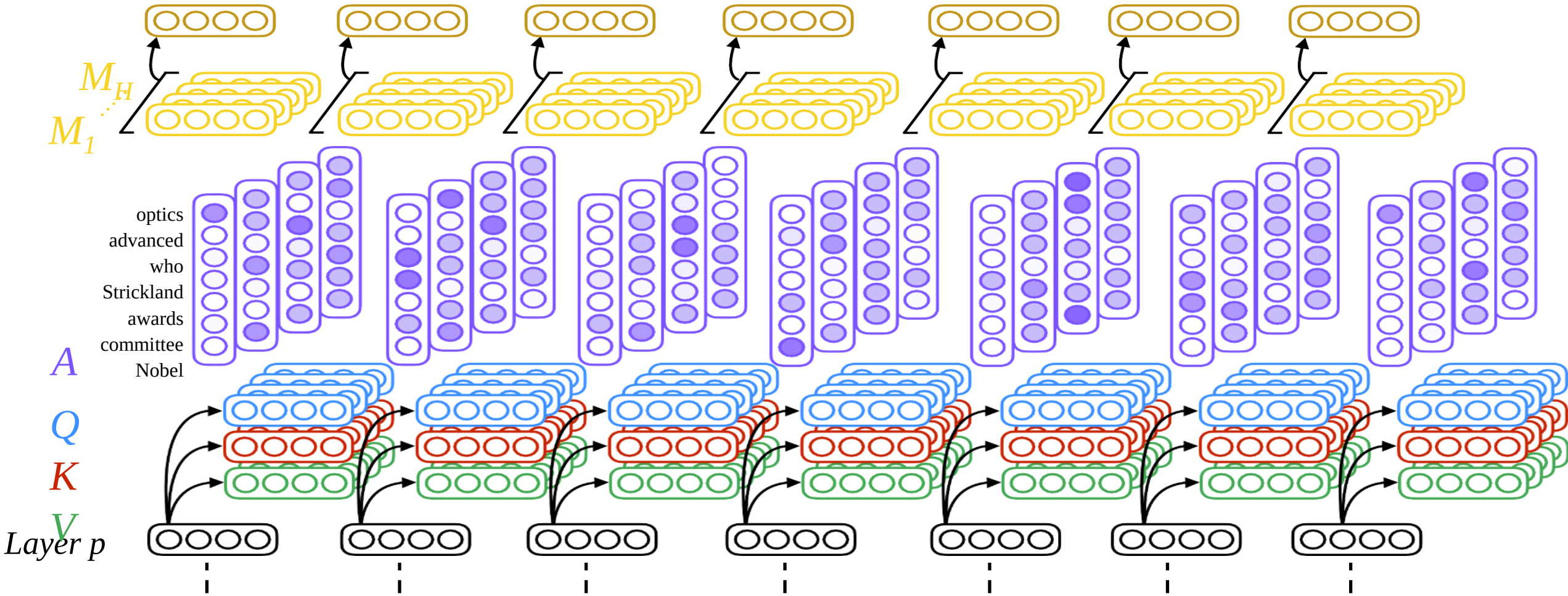
[Vaswani et al. 2017]

Multi-head self-attention



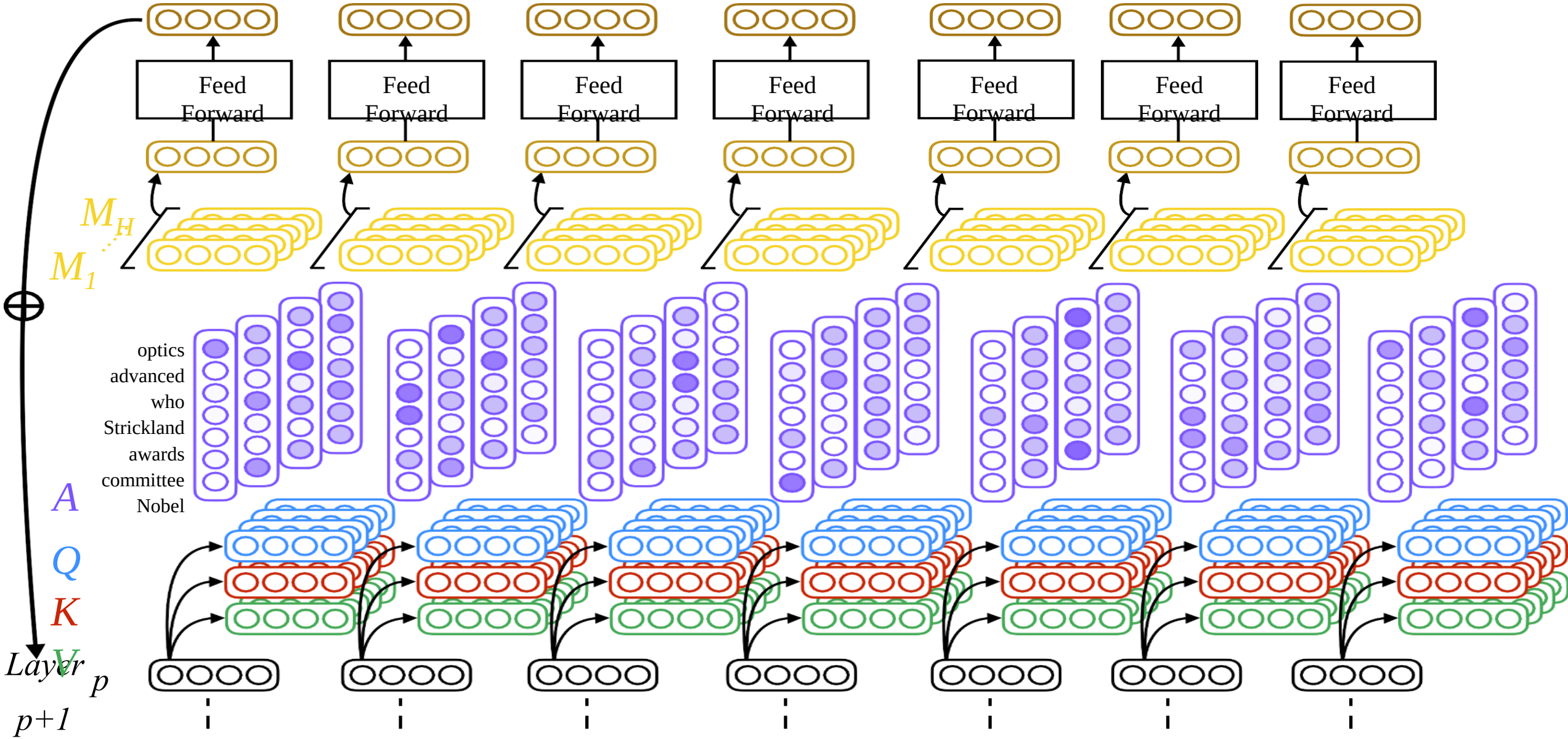
[Vaswani et al. 2017]

Multi-head self-attention



Multi-head self-attention

[Vaswani et al. 2017]



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

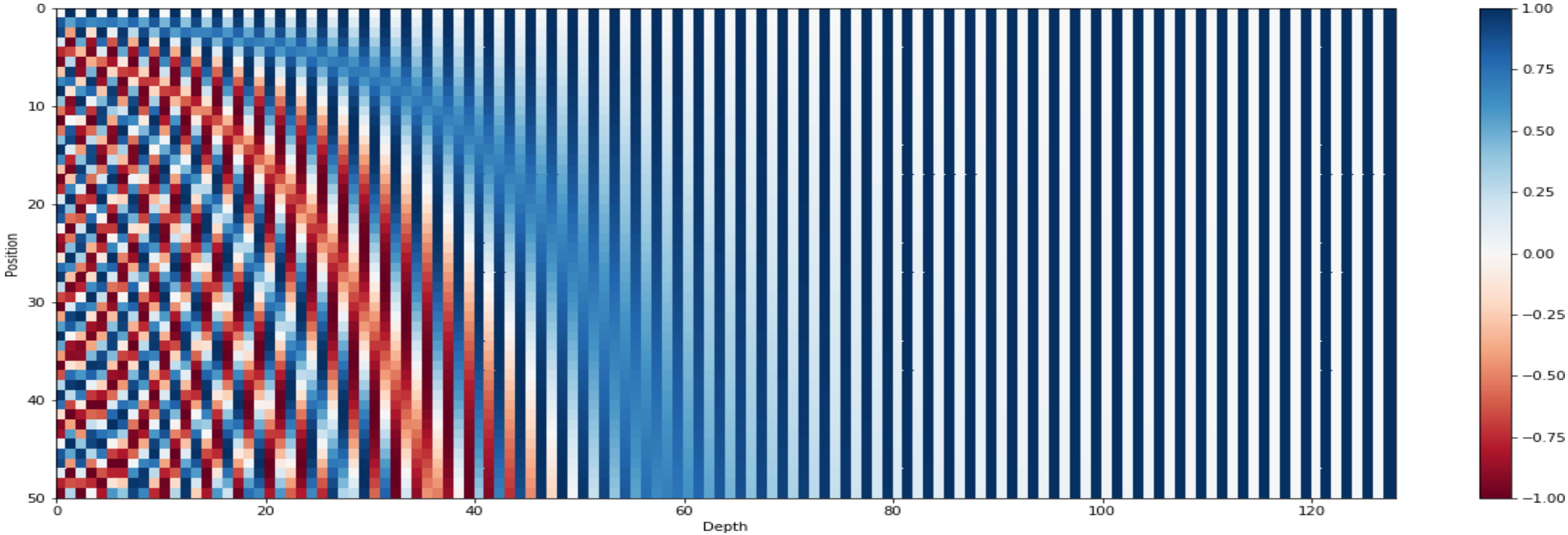
Intuitive example

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/

What does this look like?

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



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

Sinusoidal Position Vectors

The same word receives a different position vector at each location.

Position	Dimension							
	0	1	2	3	4	5	6	7
0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
1	0.8	0.5	0.1	1.0	0.0	1.0	0.0	1.0
2	0.9	-0.4	0.2	1.0	0.0	1.0	0.0	1.0
3	0.1	-1.0	0.3	1.0	0.0	1.0	0.0	1.0
4	-0.8	-0.7	0.4	0.9	0.0	1.0	0.0	1.0
5	-1.0	0.3	0.5	0.9	0.0	1.0	0.0	1.0
6	-0.3	1.0	0.6	0.8	0.1	1.0	0.0	1.0
7	0.7	0.8	0.6	0.8	0.1	1.0	0.0	1.0

Fast changes — Slow changes

Slow position channels change little over eight positions.
Changing the word still changes the combined representation.

Same word: “books”

Word embedding

0.2	-0.3	0.5	0.1	-0.2	0.7	0.1	-0.4

\+ position 7

0.7	0.8	0.6	0.8	0.1	1.0	0.0	1.0

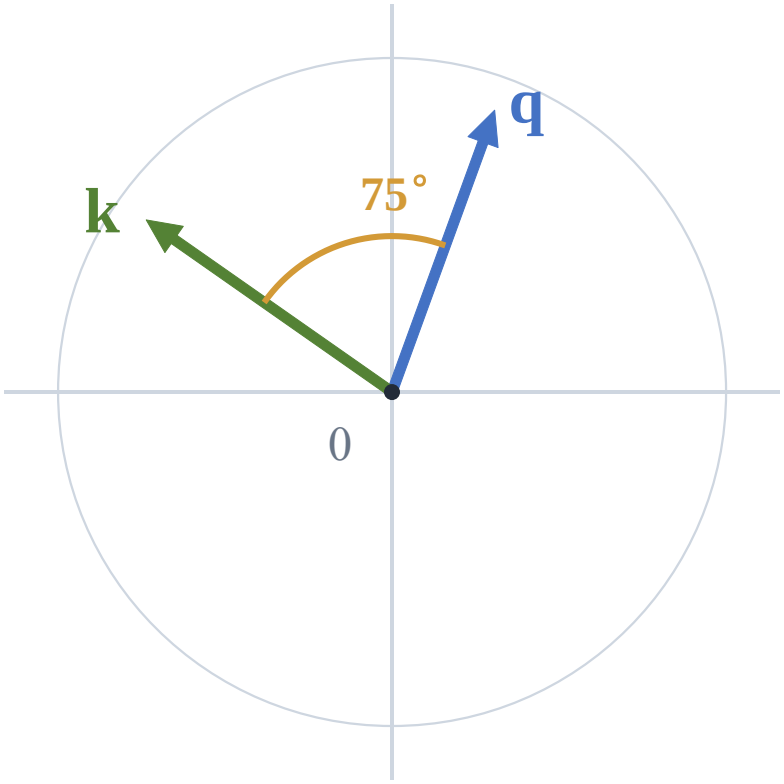
Combined representation

0.9	0.5	1.1	0.9	-0.1	1.7	0.1	0.6

Vaswani et al. (2017). Attention Is All You Need. NeurIPS 2017.
Section 3.5. Eight dimensions shown. Values round to one decimal.

Rotary Position Embedding

Position rotates query and key feature pairs around the same origin.



q at m = 4 k at n = 7
Both shift by two positions
The relative angle stays 75°

Relative offset: n − m = 3

Total angle = 30° content + 45° position

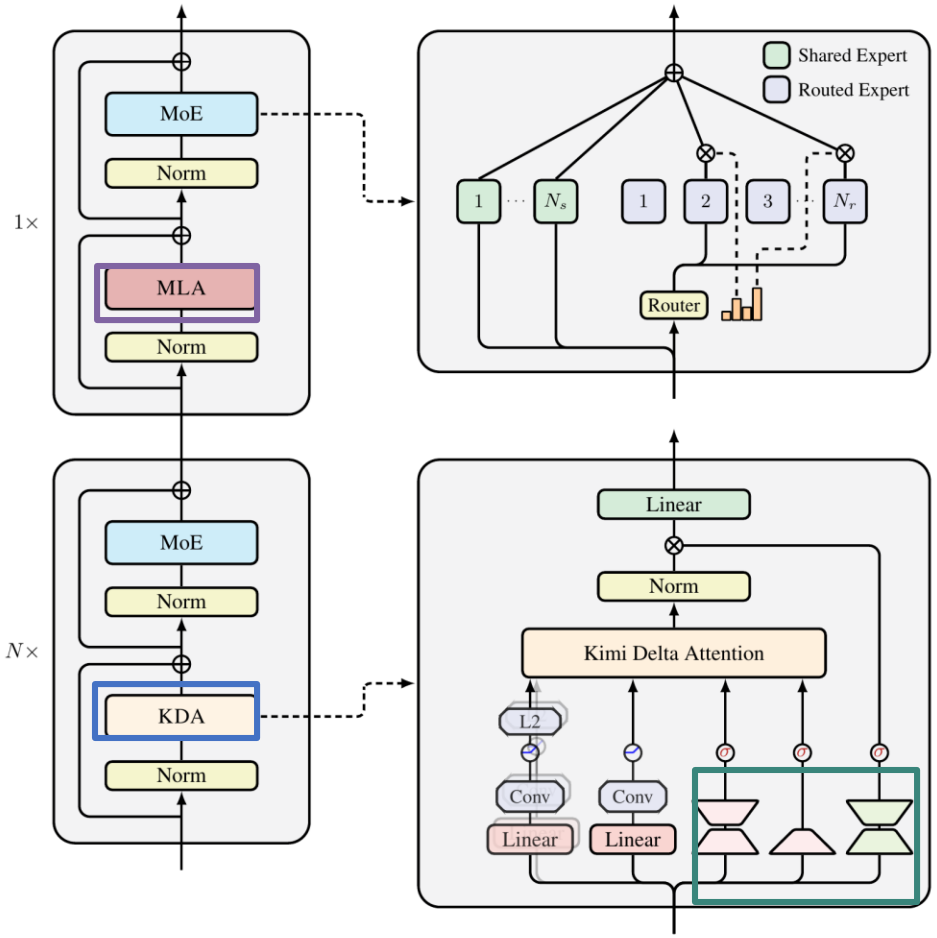
$$(R_m\, q)^T (R_n\, k) = q^T\, R_{n-m}\, k$$

Same content + same offset
gives the same dot product.

Su et al. (2021). RoFormer: Enhanced Transformer with Rotary Position Embedding.
Section 3.2. The angular rate here illustrates one two-dimensional feature pair.

Kimi Linear Without Explicit Position Embeddings

Ordered recurrent updates already carry information about token order.



1 KDA updates memory in order.
Changing token order changes the state.

2 Learned channel-wise decay
controls how far history persists.

3 Three KDA layers precede one MLA.
MLA reads their contextual outputs.

This model supplies order through KDA.
MLA uses no explicit position encoding.

Kimi Team (2025). Kimi Linear: An Expressive, Efficient Attention Architecture.
Figure 3 and Sections 4, 6.1. N = 3 in the paper.

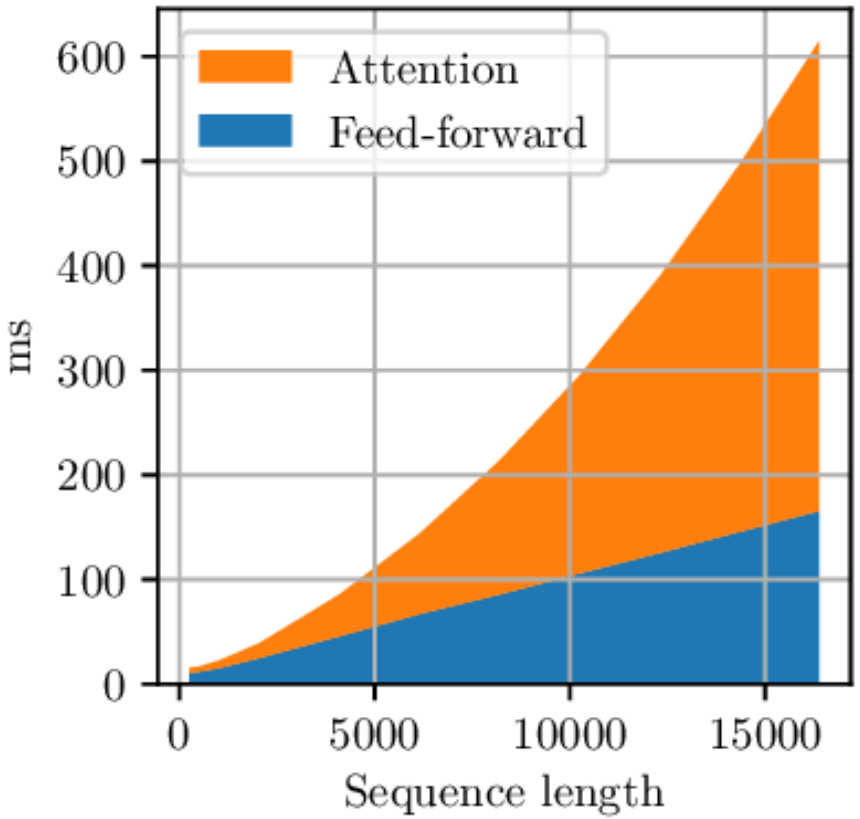
Transformer Variants

The Cost of Full Attention

Full self-attention compares every pair of positions.

Its attention matrix grows quadratically with sequence length.

The cost can dominate for long contexts.



Quadratic Attention Complexity

Self-attention can process positions in parallel. Its pairwise computation grows quadratically with sequence length.

$$QK^T \in \mathbb{R}^{n \times n} \qquad Cost: O(n^2 d)$$

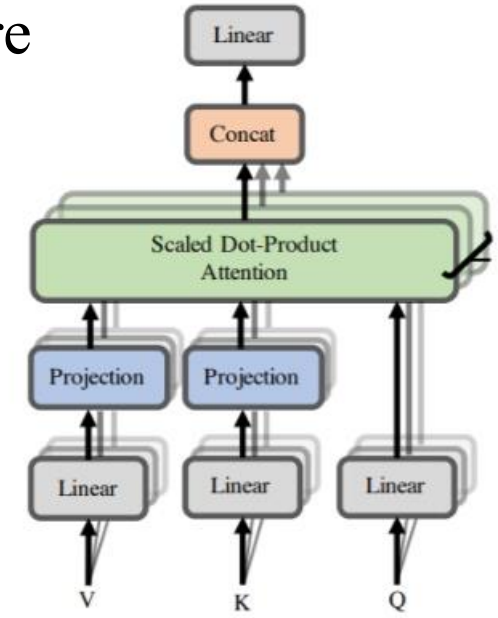
n is the sequence length. d is the representation dimension.

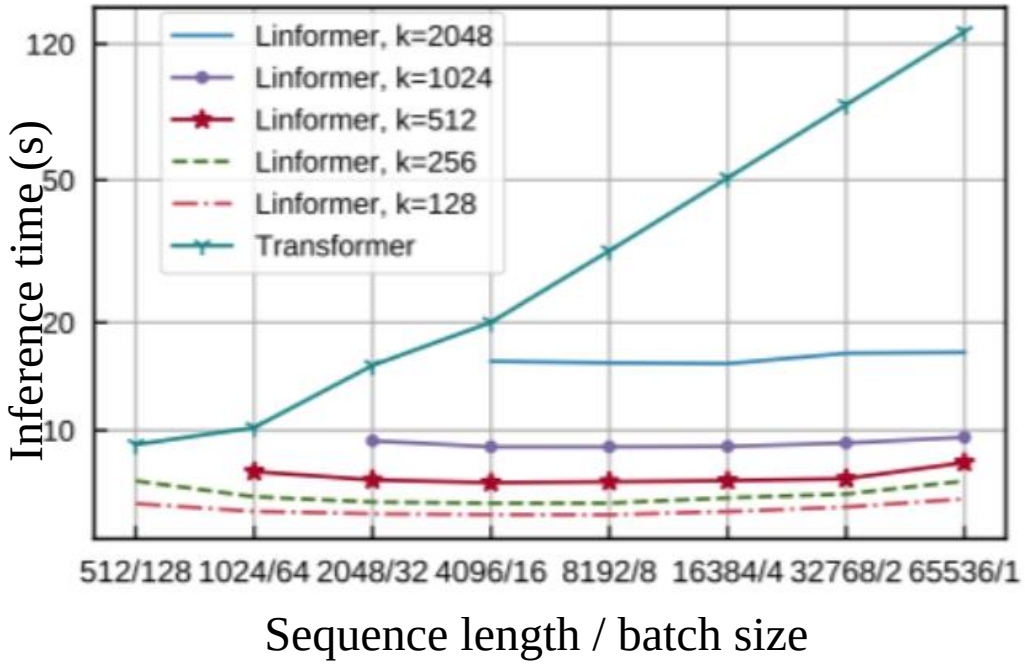
Illustration with d around 1,000
A short sentence with n ≤ 30 has at most 900 token pairs.
A context limit such as n = 512 keeps the matrix manageable.
A long document with n ≥ 50,000 needs far more pairwise work.

Linformer: Compressed Attention

Can we reduce the all-pairs attention cost?

Linformer compresses keys and values along the sequence dimension before attention.

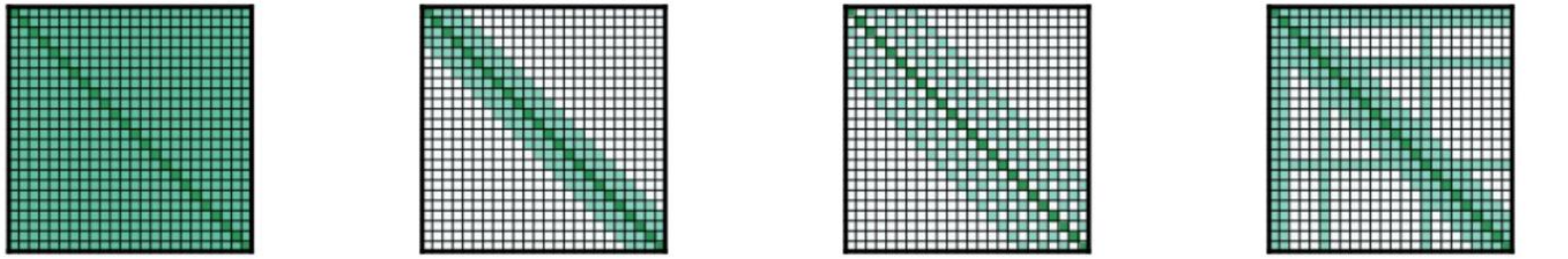




Wang et al. (2020), Linformer: Self-Attention with Linear Complexity

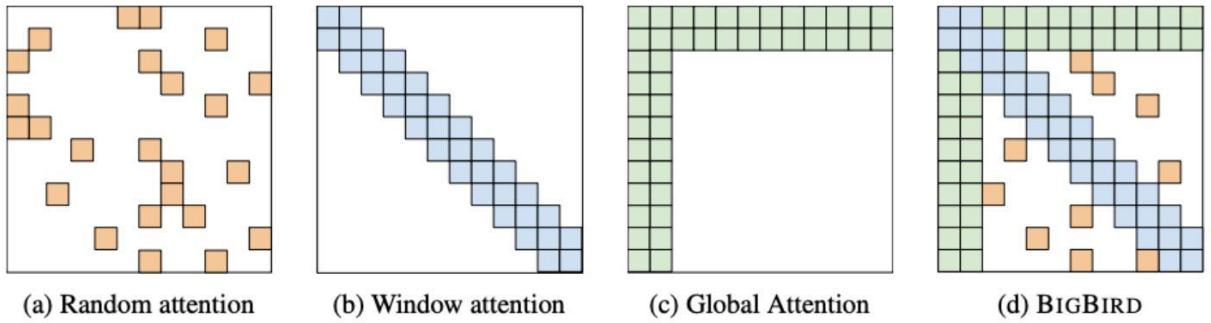
Longformer and Big Bird: Sparse Attention

Sparse patterns reduce the number of token pairs.



(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. (2020), Big Bird: Transformers for Longer Sequences

Attention Cost in the Full Model

Total cost depends on model width, sequence length and implementation.

Attention may not dominate at every scale.

A cheaper attention mechanism must also preserve model quality.

Long contexts make efficient alternatives increasingly relevant.

Do Transformer Modifications Transfer?

Most modifications did not improve performance consistently in the study's evaluated settings.

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	204M	9.2T	3.55	2.250 ± 0.002	1.899	66.57	16.21	**24.07**	**26.66**
Adaptive softmax	204M	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	224M	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

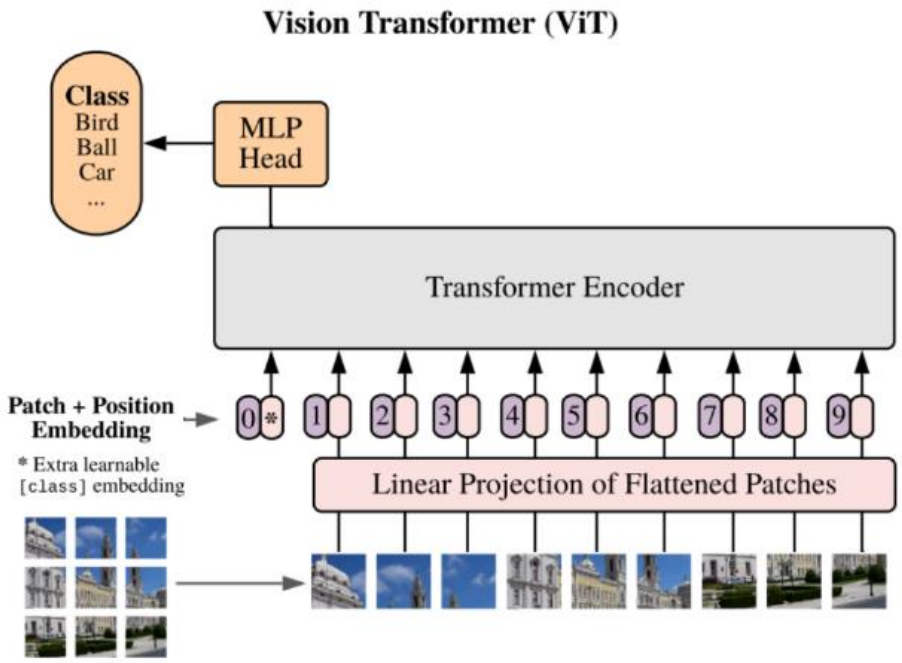
Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel

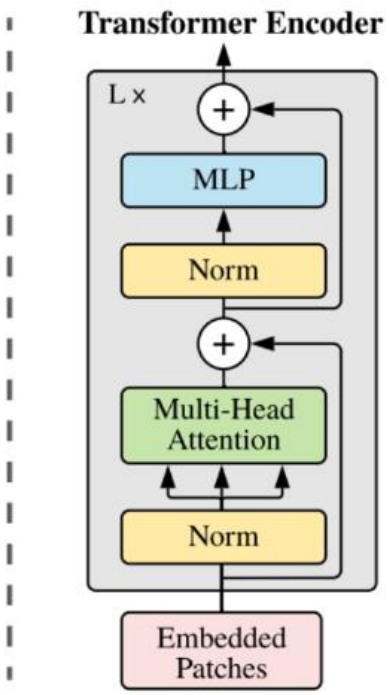
Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li

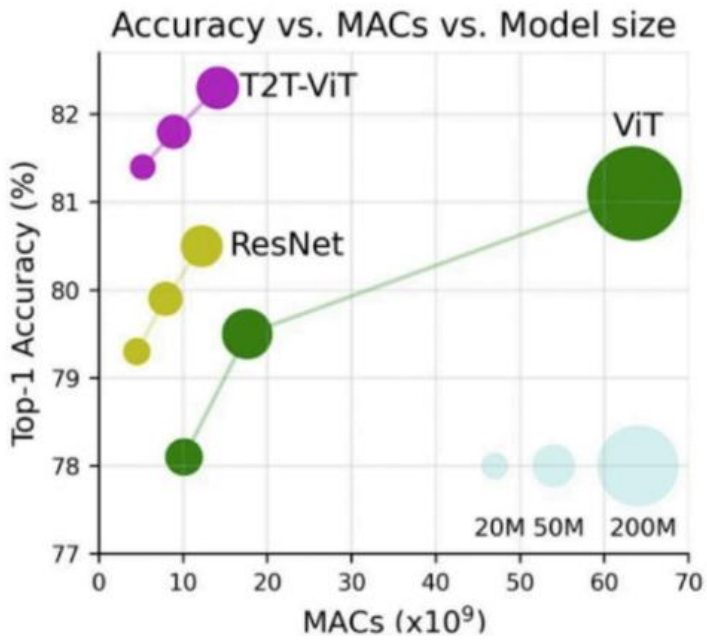
Nan Ding Jake Marcus Adam Roberts Colin Raffel†

Narang et al. (2021), Do Transformer Modifications Transfer Across Implementations and Applications?

Vision Transformer

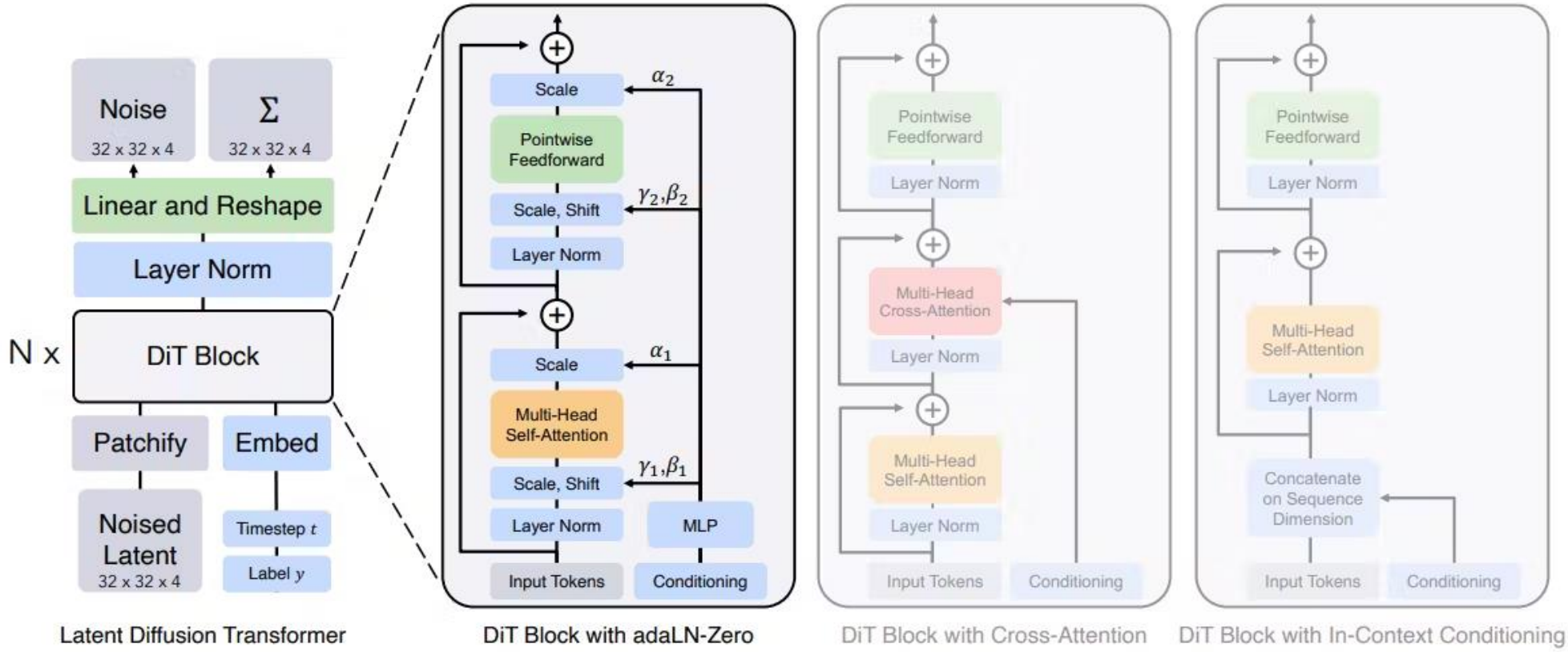






Dosovitskiy et al. (2021), An Image Is Worth 16 × 16 Words: Transformers for Image Recognition at Scale
Yuan et al. (2021), Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet

Diffusion Transformer

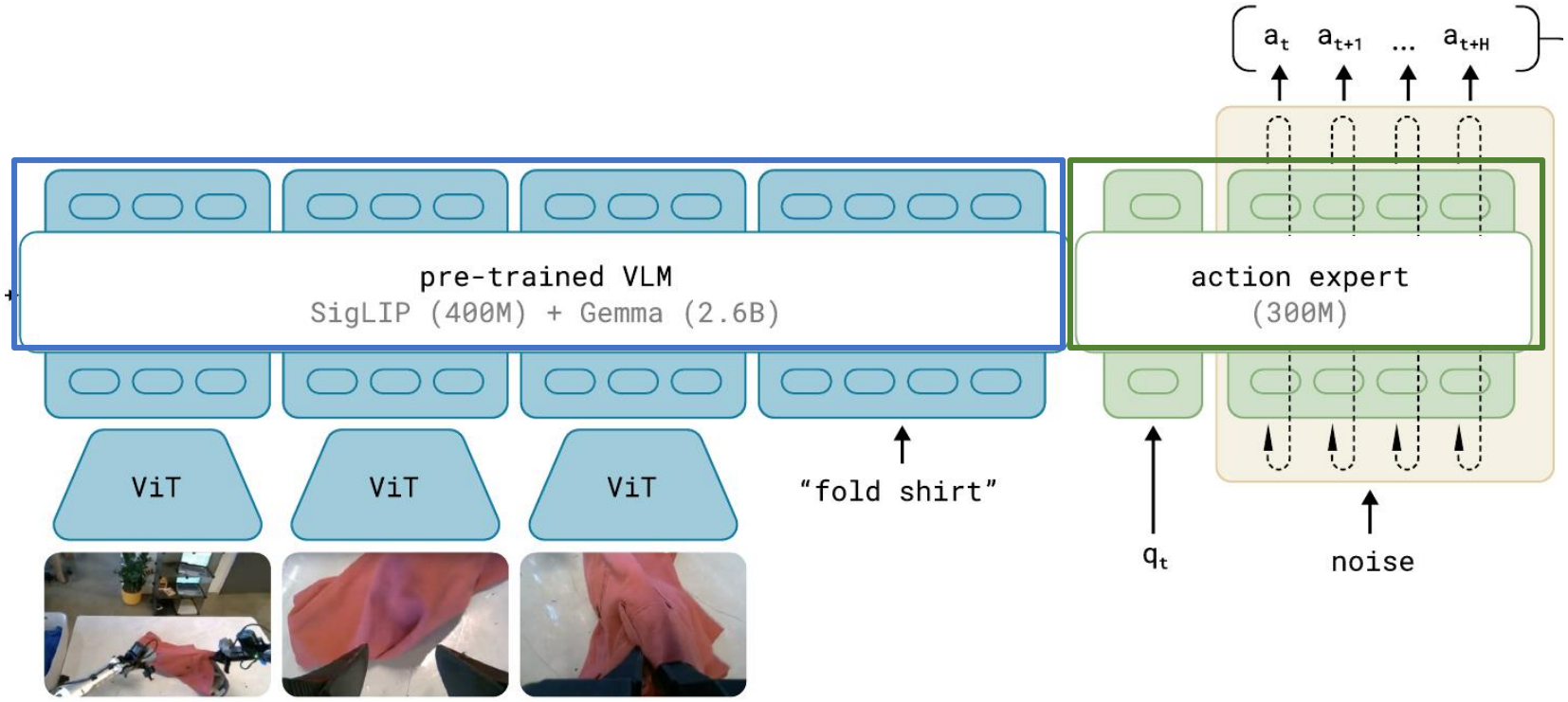


Peebles et al. (2022), Scalable Diffusion Models with Transformers

DiT replaces the U-Net backbone with a Transformer in a diffusion model.

π_0: Continuous Action Generation

The action expert refines an entire future action chunk under visual and language context.



One action coordinate



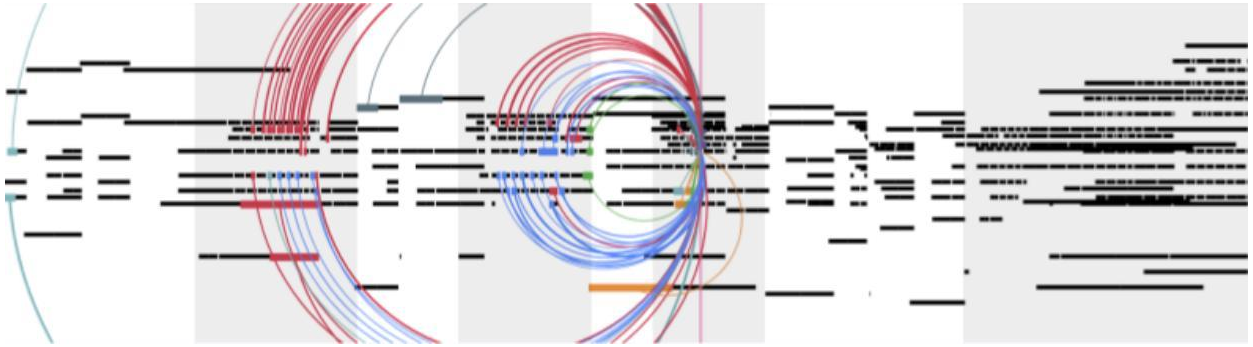
Future action time

Coordinated actions

50 future control vectors; 10 flow steps refine the whole chunk.

Black et al. (2024). π_0: A Vision-Language-Action Flow Model for General Robot Control. Figure 3 and Section IV. The small trajectory is schematic, not a robot recording.

Music Transformer





Huang et al. (2019), Music Transformer: Generating Music with Long-Term Structure

Why Pretraining + Transformers

1. Because transformers are more efficient?

Transformers are slower than LSTMs with the same number of parameters.

2. Because transformers are better at machine translation?

RNNs and CNNs are equally good at machine translation.

3. Because transformers use nothing but attention?

So what?

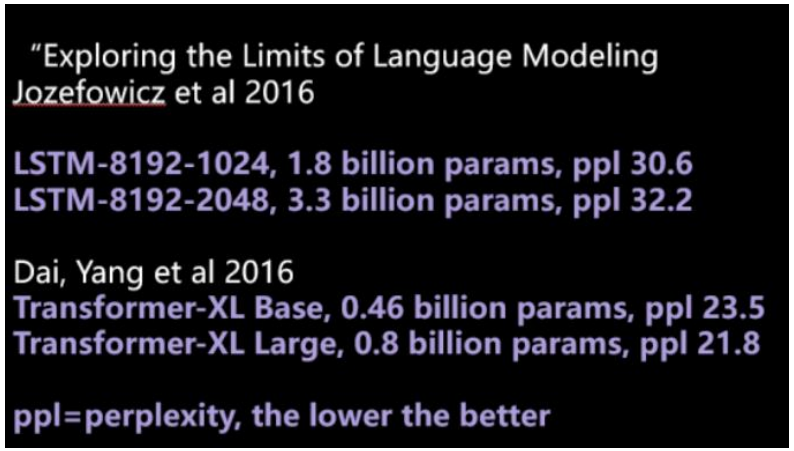
4. Because transformers learn contextualized word embeddings?

RNNs can also learn contextualized word embeddings (see ELMo).

Original lecture discussion · Credit: Zhilin Yang, Recurrent AI

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

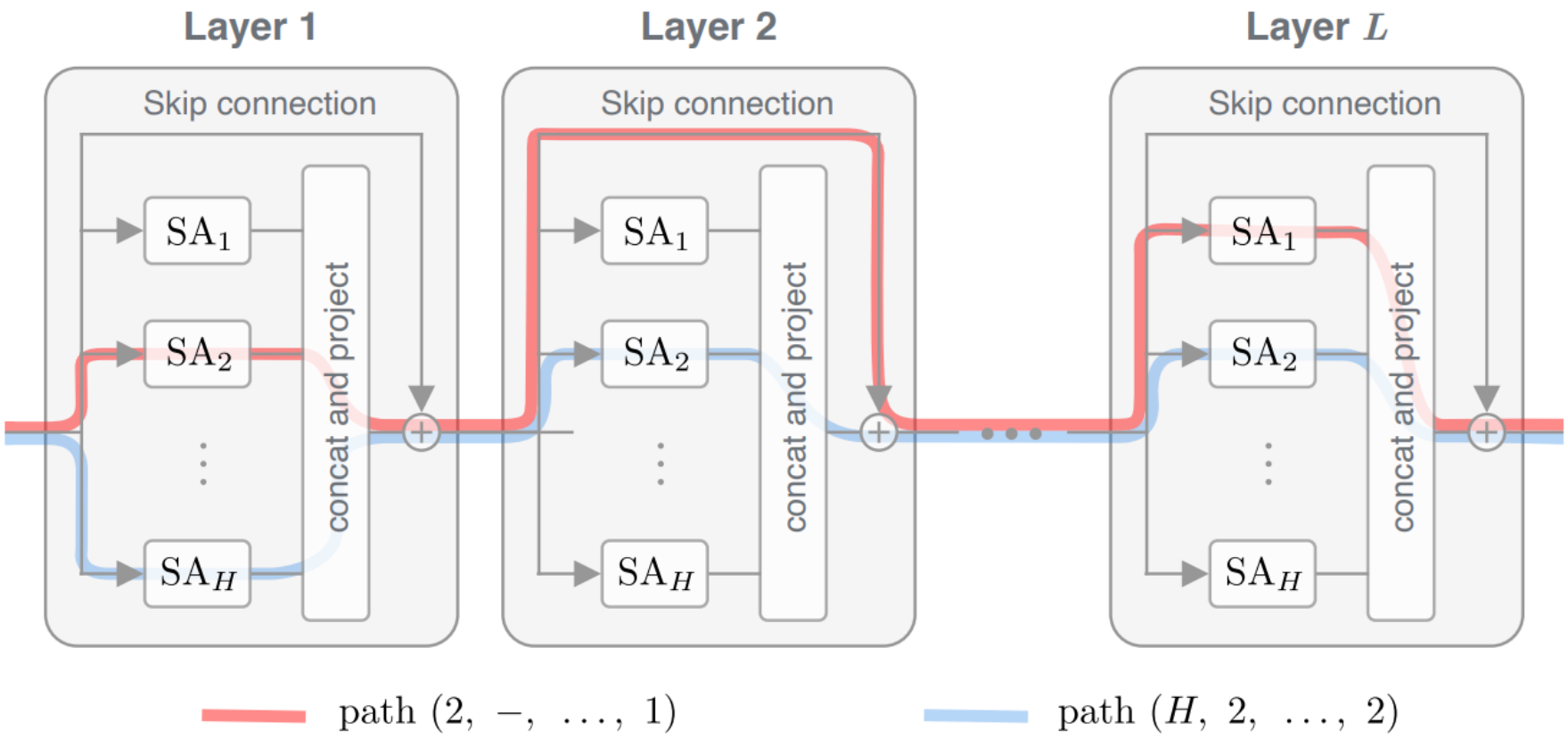
Understanding Transformer Components

Transformer Component Ablations

What changes if we remove or replace:

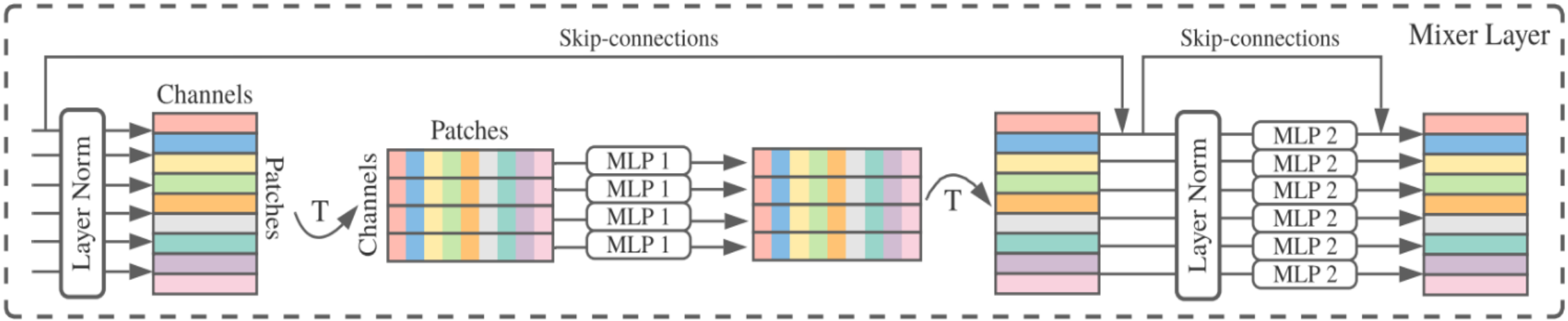
- Self-attention
- Feed-forward layers
- Position embeddings
- Other architectural components

Pure Attention Without Feed-forward Layers



Dong et al. (2021), Attention Is Not All You Need:
Pure Attention Loses Rank Doubly Exponentially with Depth

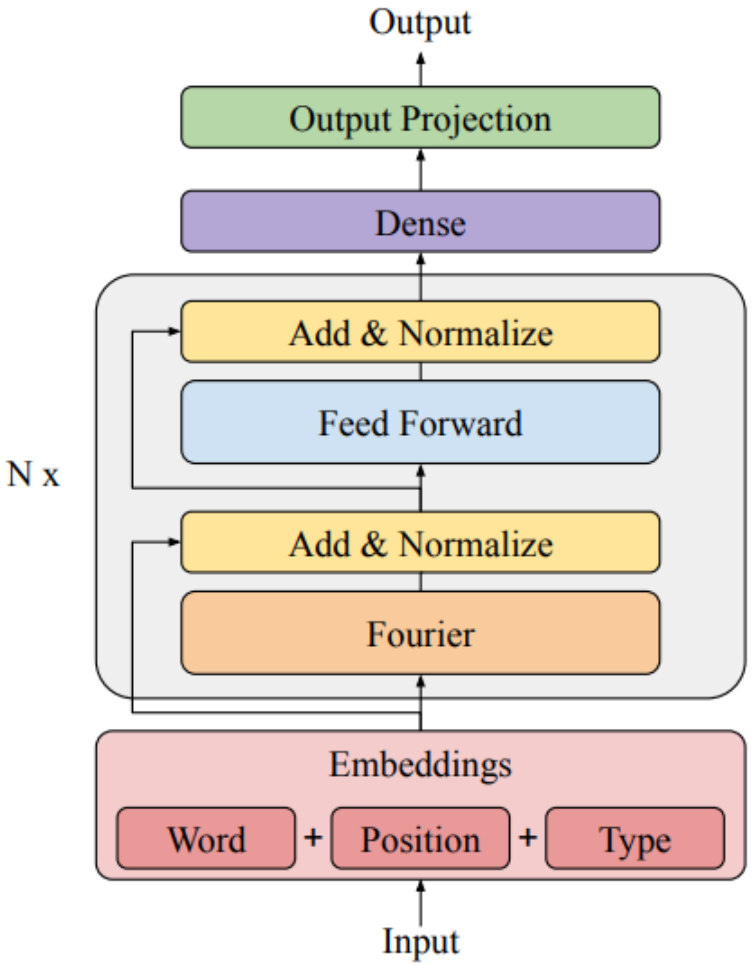
Token Mixing Without Self-attention



At least it works for computer vision.

Tolstikhin et al. (2021), MLP-Mixer: An All-MLP Architecture for Vision

FNet: Fourier Token Mixing



Fourier transforms mix token information using a fixed linear operation.

The reported text-classification results motivate further study across tasks.

Lee-Thorp et al. (2022), FNet: Mixing Tokens with Fourier Transforms

Ordering Attention and Feed-forward Layers

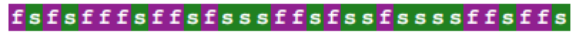
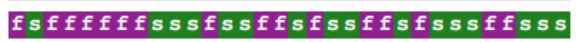
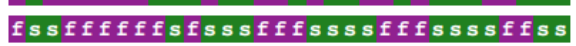
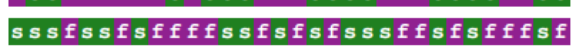
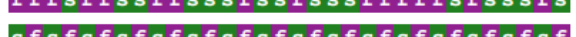
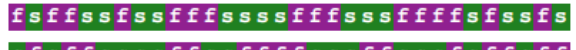
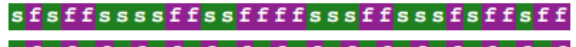
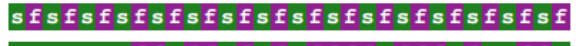
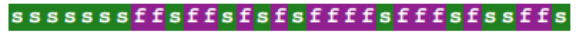
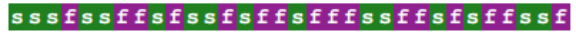
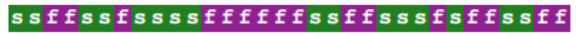
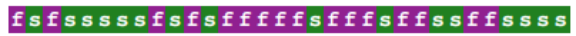
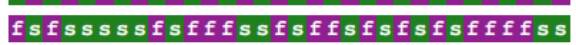
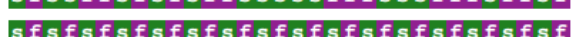


(a) Interleaved Transformer



(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

Press et al. (2020), Improving Transformer Models by Reordering Their Sublayers

Changing Position Embeddings

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

Wang et al. (2021), On Position Embeddings in BERT, ICLR

DeepNorm and Deep Transformers

DeepNet: scaling Transformers to 1,000 layers

DeepNorm rescales the residual connection and uses a corresponding initialization.

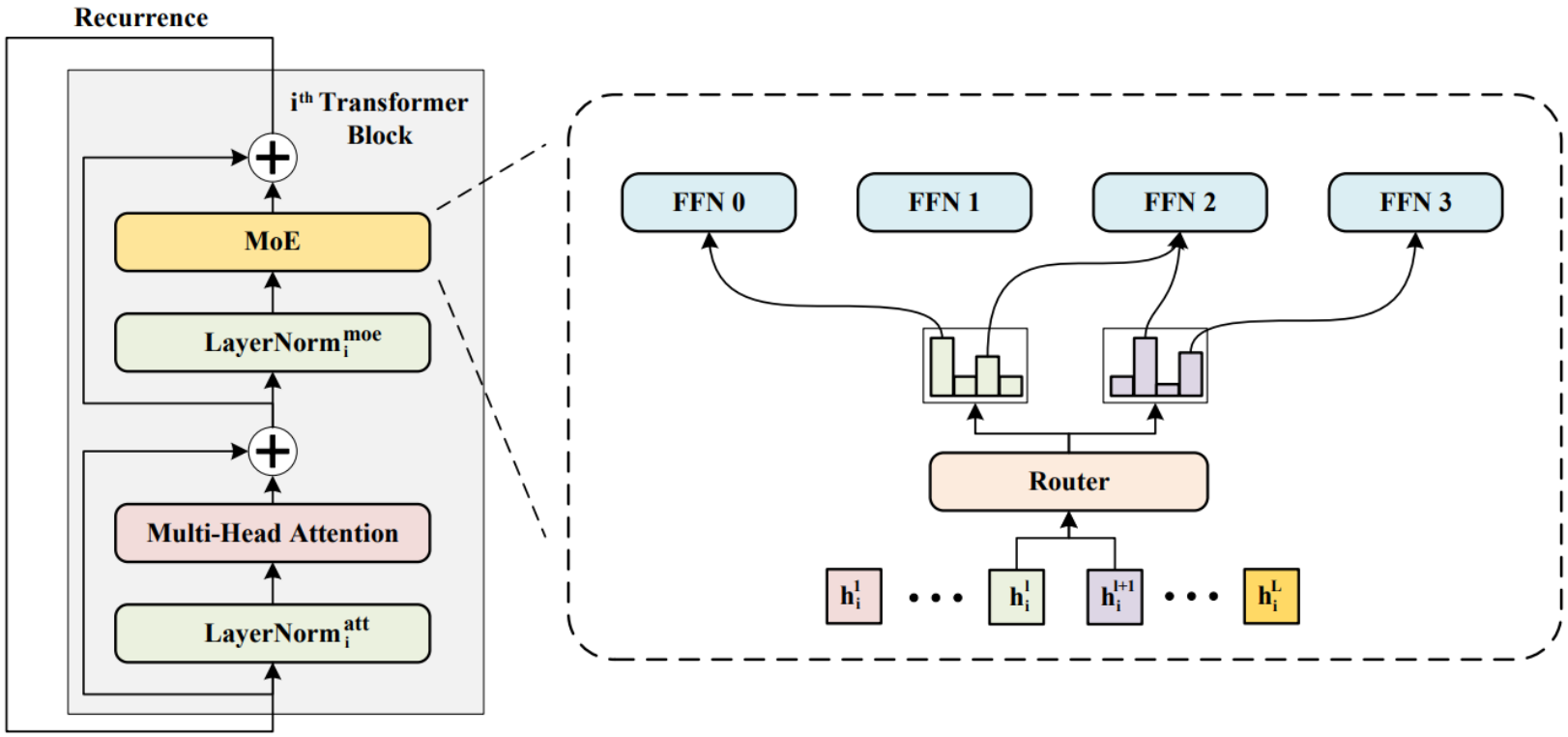
$\text{LayerNorm}(x + F(x))$ $\text{LayerNorm}(\alpha x + F(x))$

The design aims to combine stable optimization with strong deep-model performance.

Wang et al. (2022), DeepNet: Scaling Transformers to 1,000 Layers

Wider or Deeper Models?

Go Wider Instead of Deeper



WideNet shares parameters across blocks and expands capacity with experts.
Each expert needs enough training tokens.

Xue et al. (2021), Go Wider Instead of Deeper

Lecture Roadmap

1 NLP tasks and representations

2 Inductive bias: semantic abstraction and composition

3 Architectures for Neural representations

- 3.1 Multilayer perceptrons
- 3.2 Convolutional networks
- 3.3 Recurrent networks
- 3.4 Transformer

4 Scaling laws and emergence

Scaling Laws

Scaling Model, Data and Compute

Language-model loss often follows predictable trends as model size, data and compute increase.

Scaling Laws for Neural Language Models

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

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

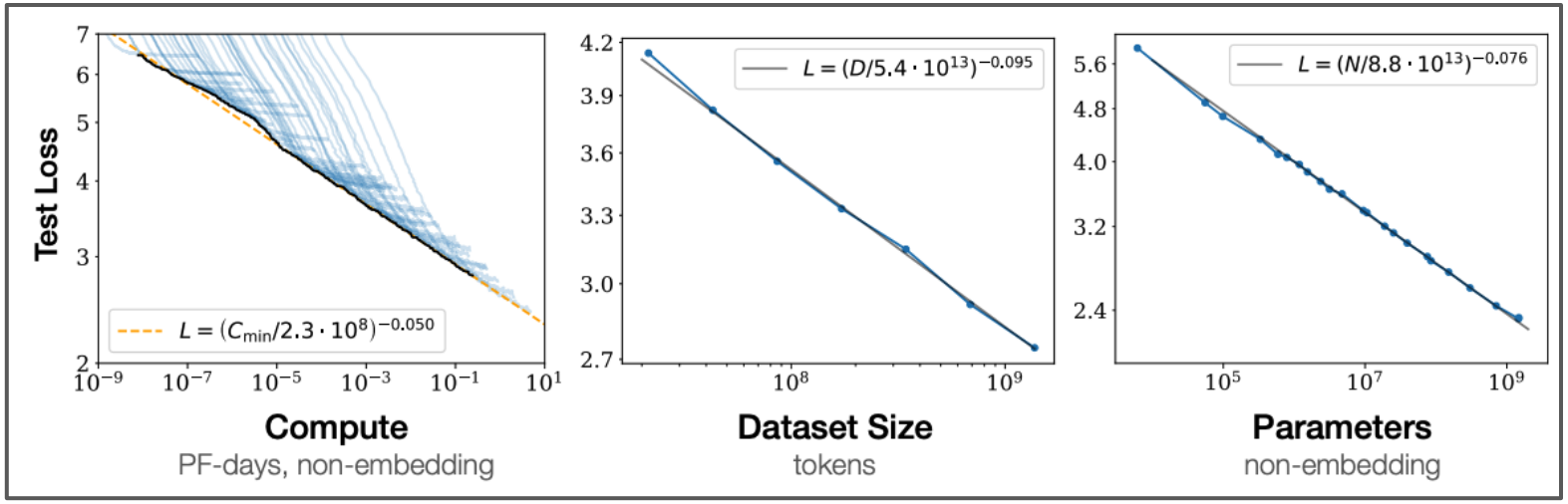
Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

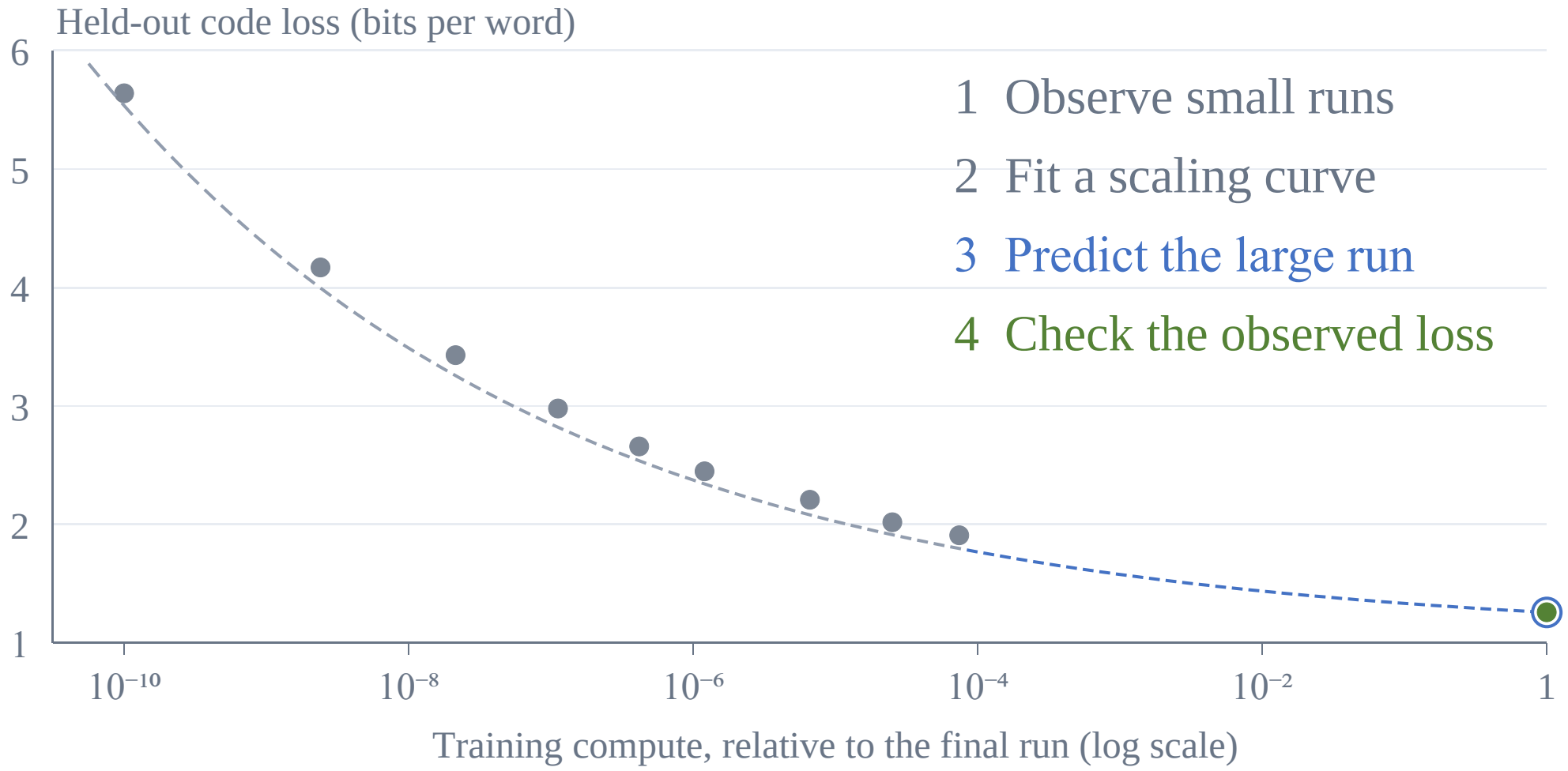
Dario Amodei
OpenAI
damodei@openai.com



Kaplan et al. (2020), Scaling Laws for Neural Language Models

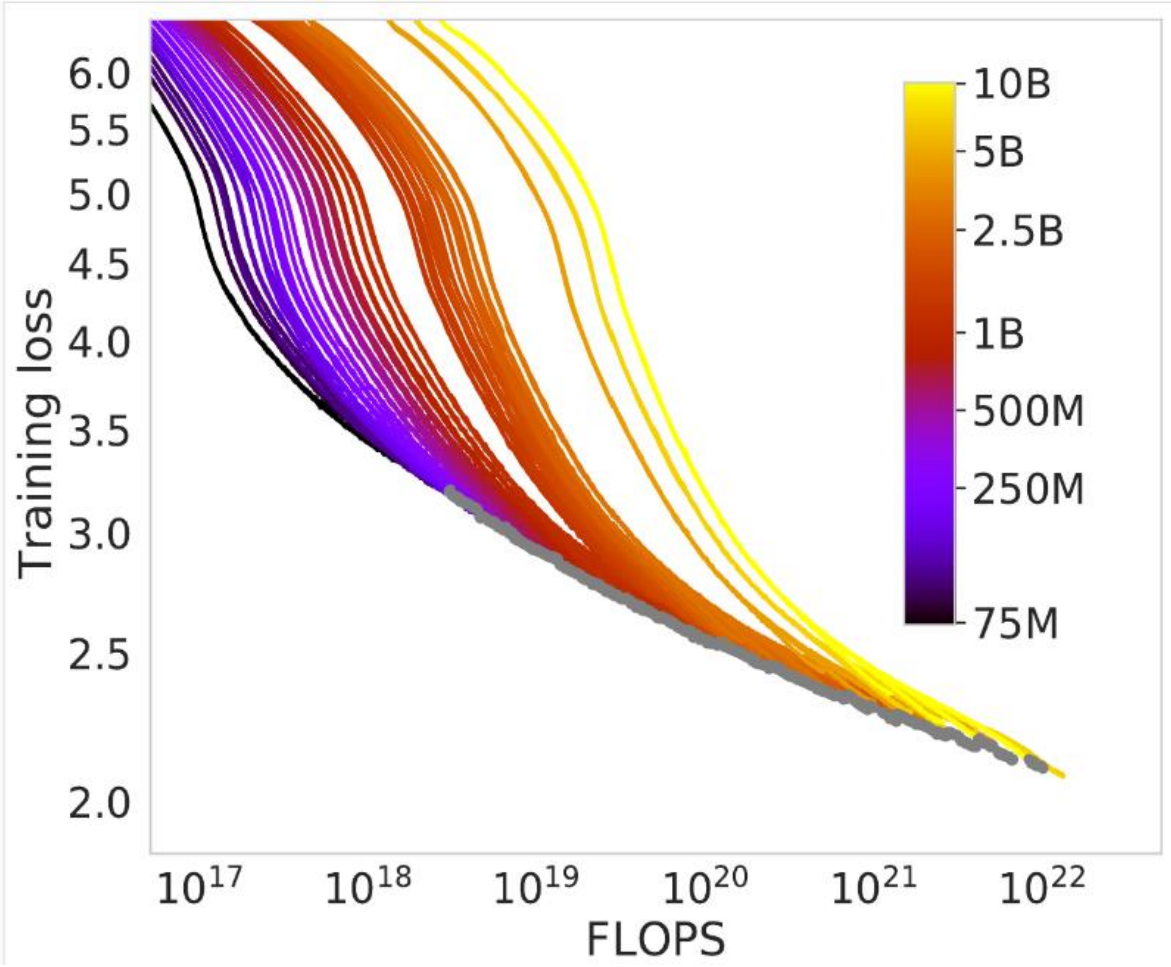
Predicting Performance at Scale

Measure smaller runs, fit a trend, then test a prediction at a larger scale.



OpenAI (2023), GPT-4 Technical Report, Figure 1. Approximate visual reconstruction.

Model Capacity and Training Budget



Small models can learn quickly early in training.

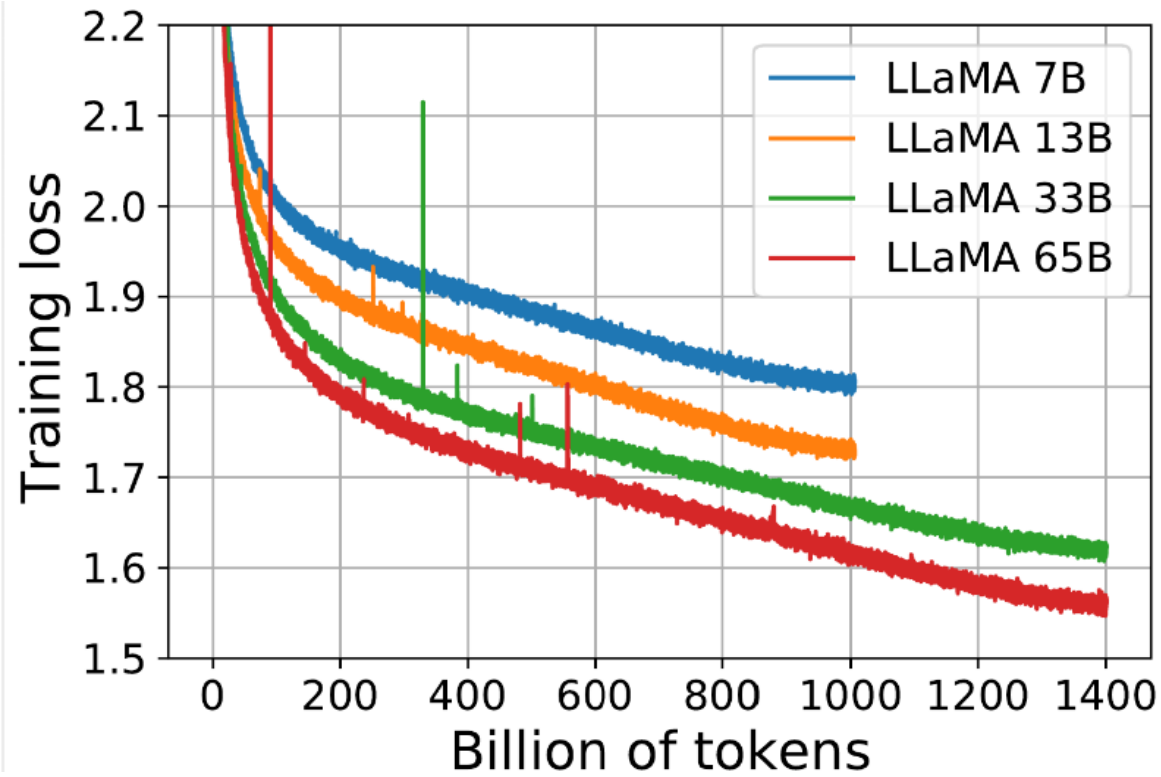
Larger models may achieve lower loss when given more training compute.

The best choice depends on the budget.

Hoffmann et al. (2022), Training Compute-Optimal Large Language Models
Tyl (2023), Chinchilla's Death

Scaling Curves and Learning-rate Schedules

Token budget and learning-rate schedule



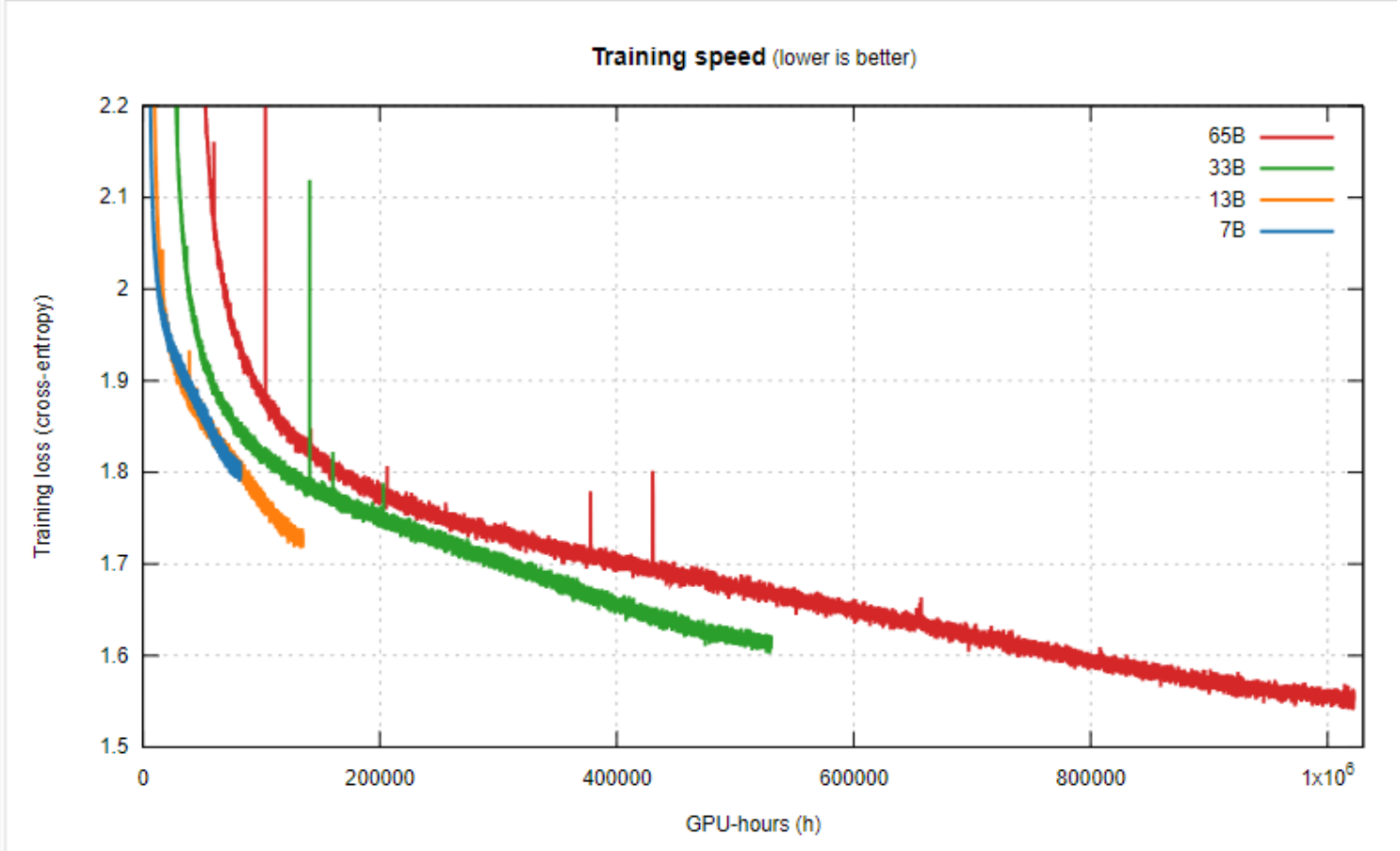
Curves change slope over training.

A flatter endpoint can also reflect the cosine learning-rate schedule.

Interpret capacity limits together with the optimization setup.

Source: Chinchilla's Death, espadrine.github.io

Comparing Models at Fixed Compute



At fixed compute, compare how much each model learns within the same budget.

Longer training may change the ranking; extrapolation remains uncertain.

Source: Chinchilla's Death, espadrine.github.io

New Dimensions of Scaling

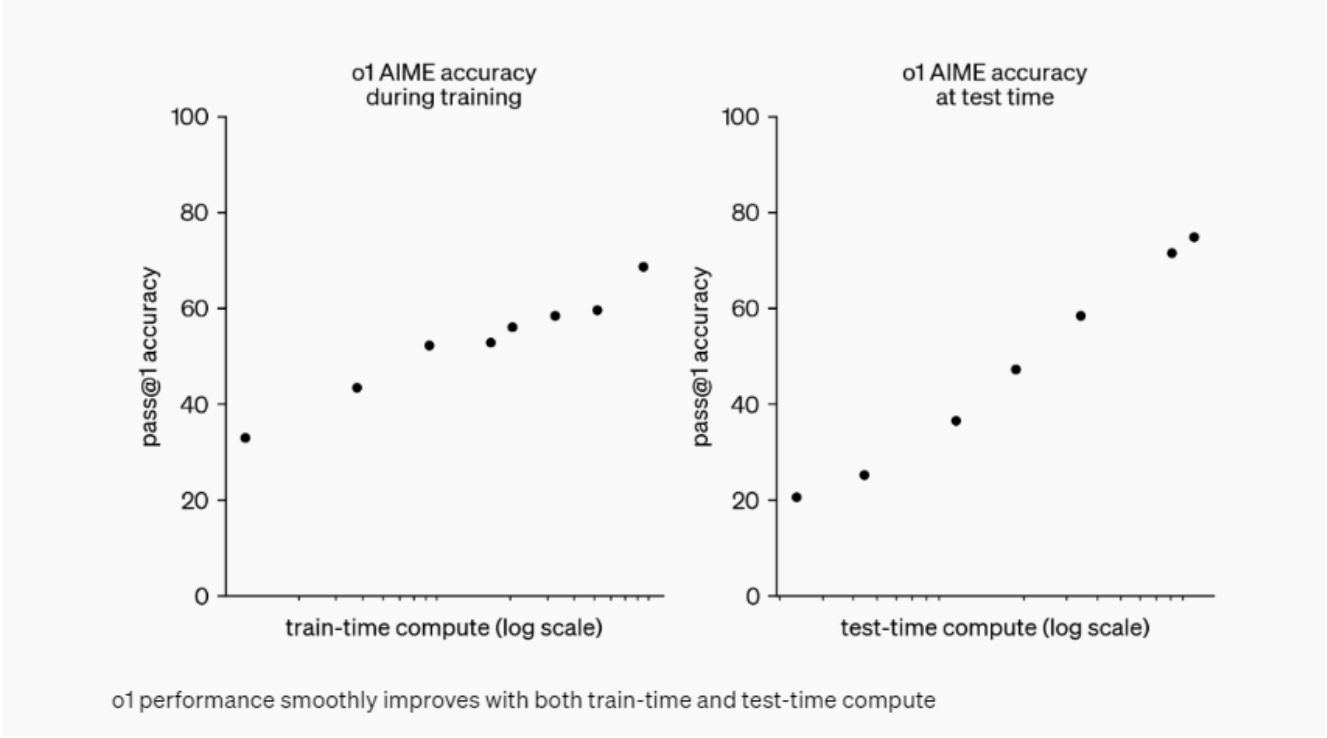
Parallel scaling
Sample multiple candidates and select using a verifier.

Sequential scaling
Generate more intermediate reasoning tokens.
Examples include OpenAI o1 and DeepSeek R1.

Yu, Li and Wang (2025), Scaling Flaws of Verifier-Guided Search in Mathematical Reasoning

Training-time and Test-time Compute

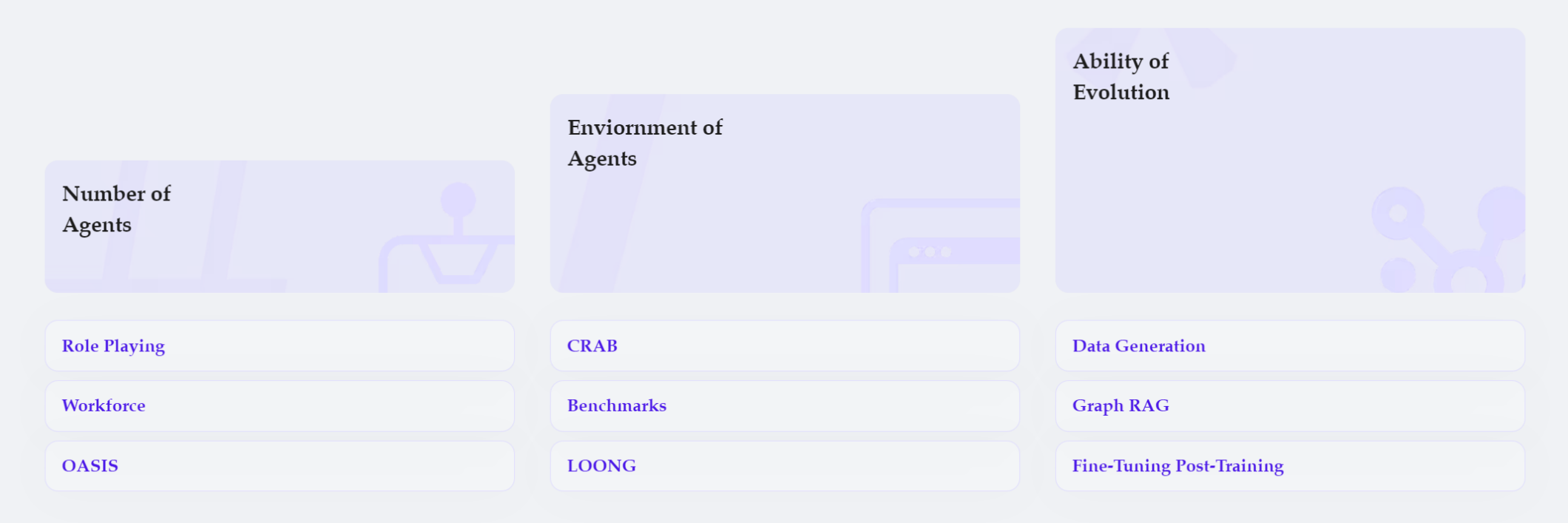
o1 improves with more reinforcement-learning compute during training and more reasoning compute at test time.
These budgets differ from pre-training.



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

OpenAI, Learning to Reason with LLMs

Scaling Agent Systems



More agents alone may offer limited gains.

Environment interaction is a promising direction.

Learning requires useful feedback from the environment.

Source: CAMEL-AI

New Scaling Directions

A useful scaling proposal names both the resource and the measurable return.

What increases?	What should improve?
Data and experience	More coverage, feedback or verified examples?
Reasoning and search	More useful computation before the final answer?
Memory and tools	More relevant knowledge available when needed?
Model structure	More effective capacity per unit of compute?
Coordination	Better division of work, after communication cost?

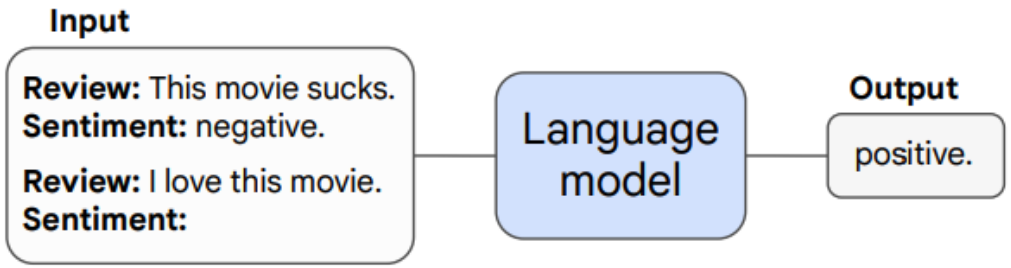
Compare gains at matched cost, latency and evaluation quality.

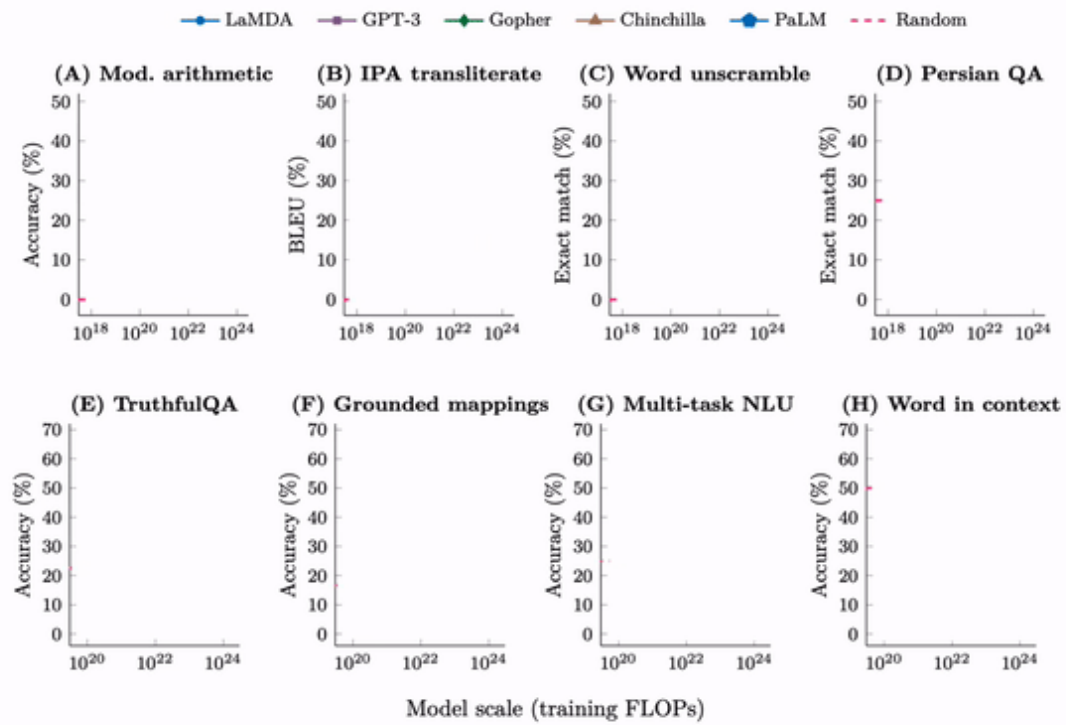
Emergent Abilities

Emergent Abilities in Language Models

An ability may appear near the random baseline in small models and become measurable in larger models.

Example: few-shot prompting

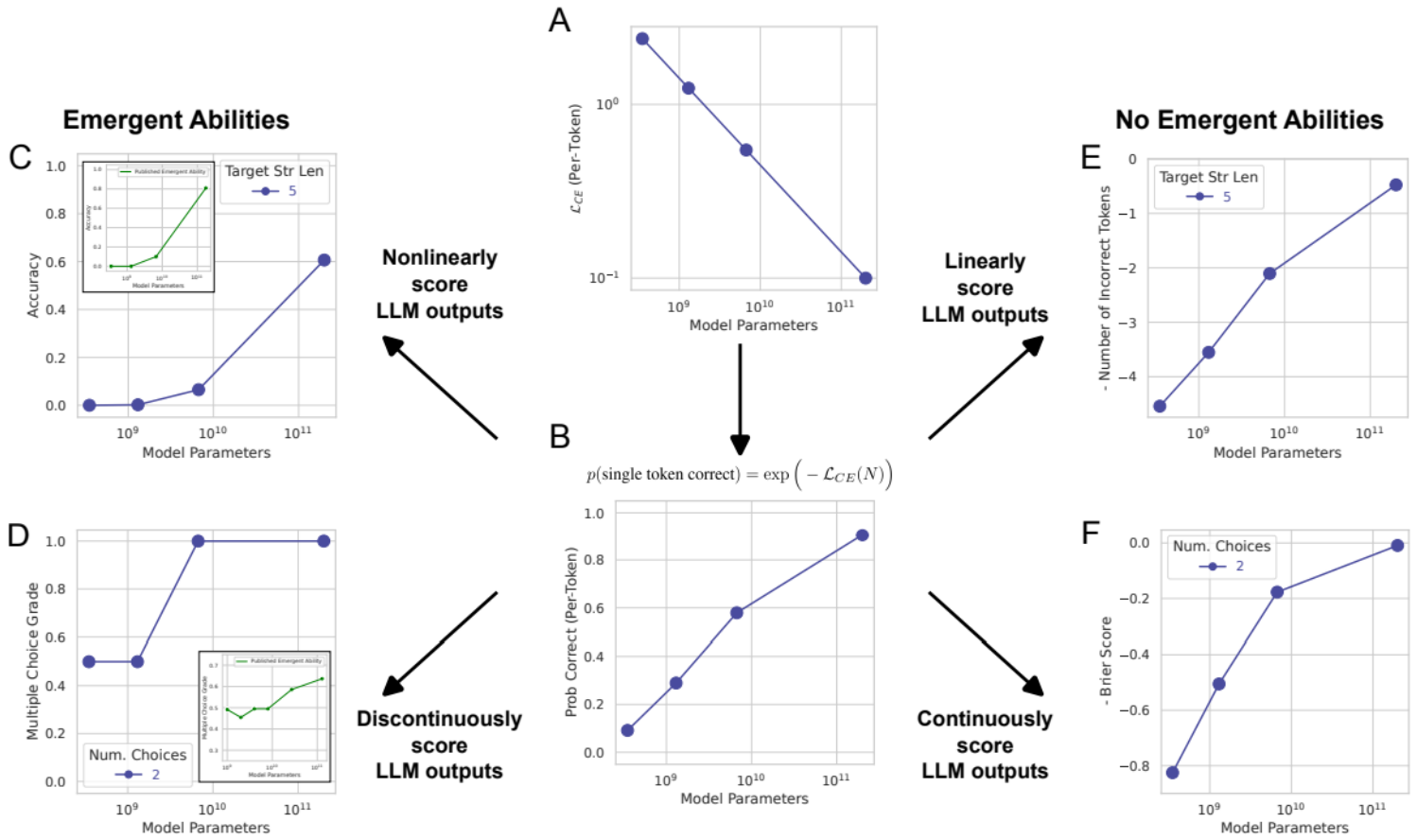




The apparent onset depends on the task, baseline and evaluation metric.

Wei et al. (2022), Emergent Abilities of Large Language Models

Emergence and the Choice of Metric



Nonlinear or discontinuous metrics can make smooth improvements look abrupt.

Schaeffer et al. (2023), Are Emergent Abilities of Large Language Models a Mirage?

Acknowledgements

Contextualized Word Embeddings

Princeton University, COS 484: Natural Language Processing, Fall 2019

Language Models

University of Illinois, CS 447: Natural Language Processing

Deep Learning for Computer Vision

Stanford University, CS231n course and lecture notes

Summary: Seq2Seq Model Using Convolutional Neural Network

Gautam Karmakar, Medium

Transformers and Sequence-to-sequence Learning

Mohit Iyyer, University of Massachusetts Amherst, CS 685, Fall 2021

Course Project Discussion

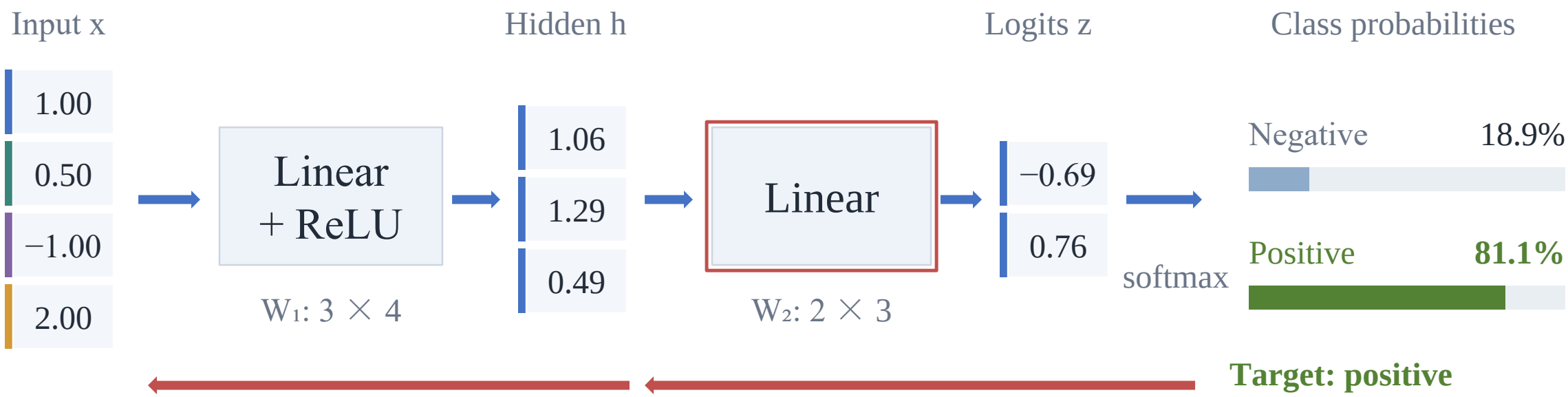
- Final reports
- Research track: sharpen the question and evaluation.
- Application track: define users, needs and evidence of value.

- Assignments
- Discuss current obstacles and feedback.

Backpropagation: Worked Examples

Backpropagation

The loss assigns a local correction to every weight along the same graph.



Forward: make a prediction. Backward: compute responsibility.

Update the weights, then measure the new loss. 0.928 → 0.210

A gradient measures sensitivity. Its sign tells the optimizer which direction reduces local loss.

Backpropagation on a Computation Graph

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

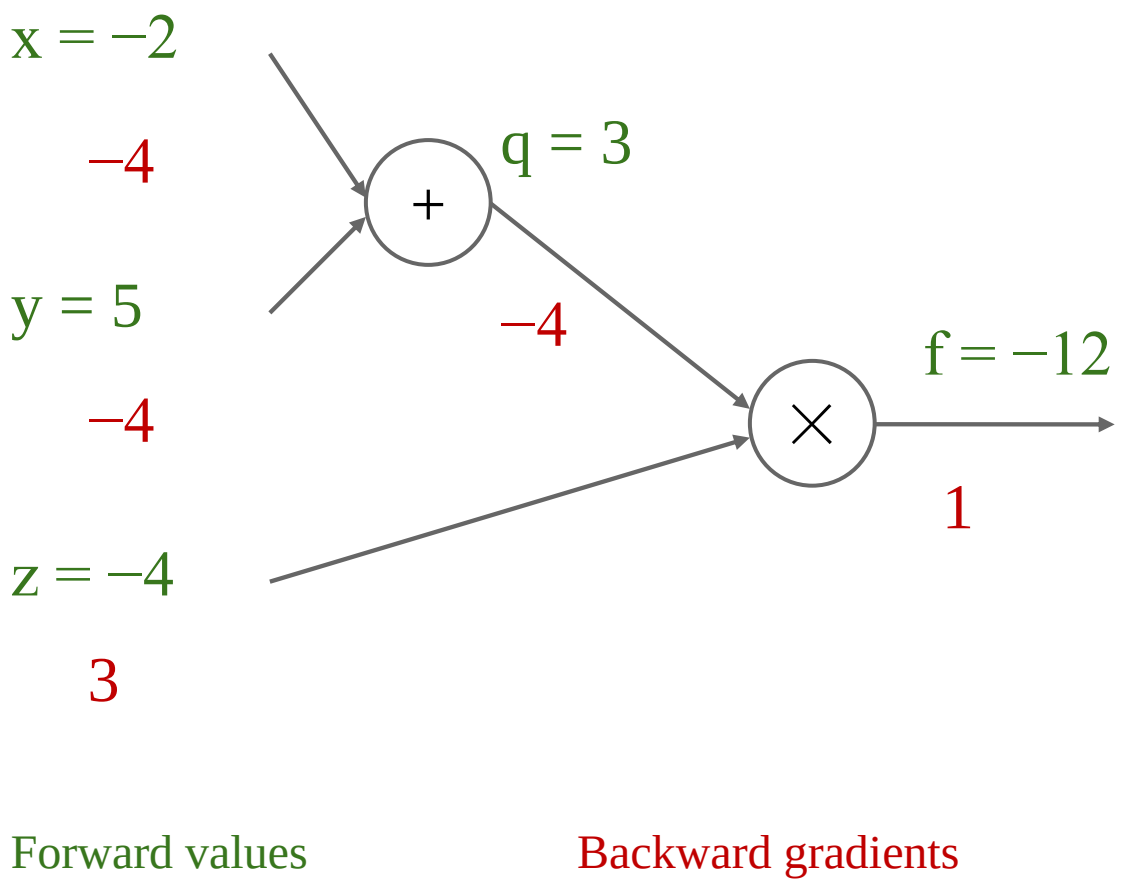
x = −2, y = 5, z = −4

$$q = x + y = 3$$

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

$$f = qz = -12$$

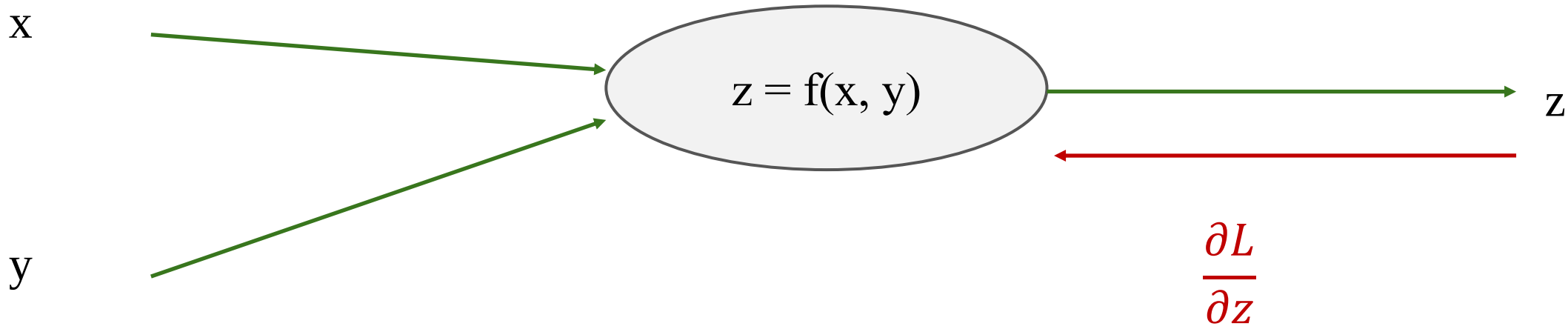
$$\frac{\partial f}{\partial q} = z = -4, \qquad \frac{\partial f}{\partial z} = q = 3$$



∂f/∂x = (∂f/∂q)(∂q/∂x) = −4 × 1 = −4.

The Local Rule for Backpropagation

A node receives the gradient of the loss with respect to its output.



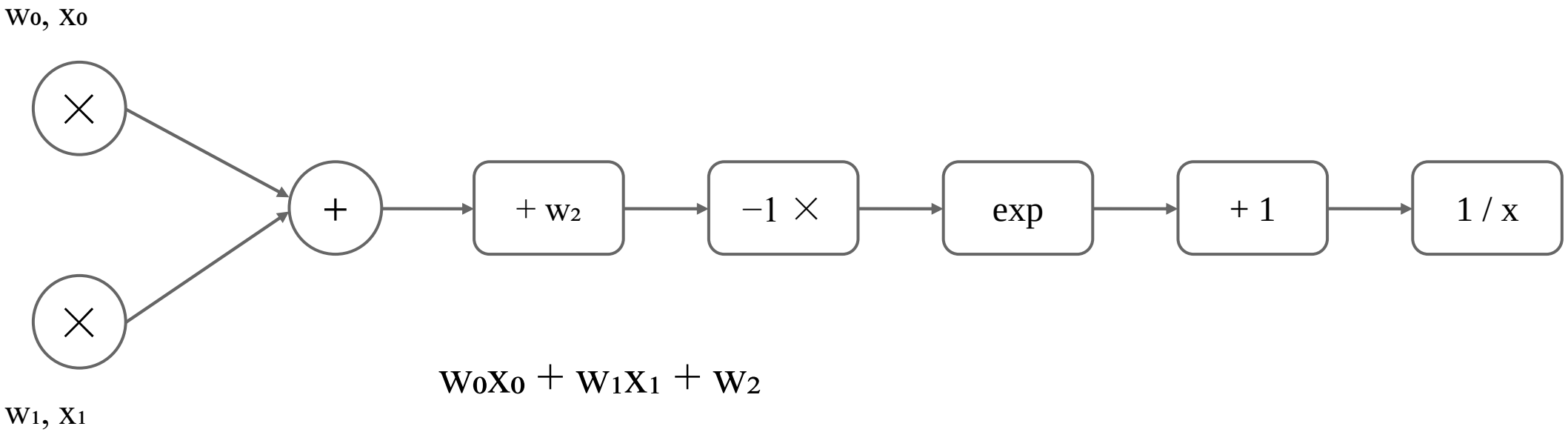
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial x}$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial y}$$

Input gradient = upstream gradient $\times$ local derivative

A Sigmoid Computation Graph

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

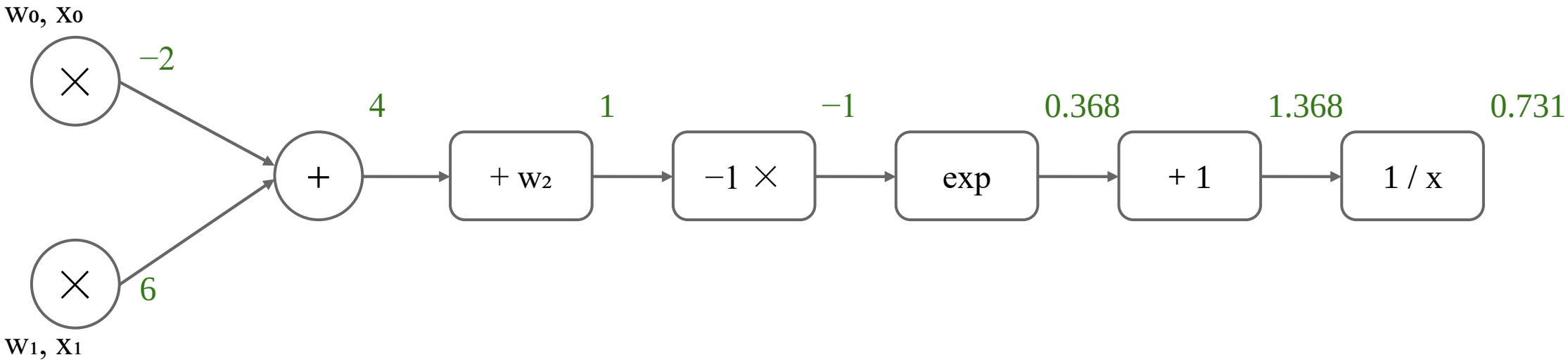


The same sigmoid function can be decomposed into simple operations.

Backpropagation Through Elementary Operations

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

$w_0 = 2$, $x_0 = -1$, $w_1 = -3$, $x_1 = -2$, $w_2 = -3$



$$Upstream \times local: \quad \frac{\partial f}{\partial w_0} \approx 0.20 \times (-1) = -0.20$$

$$\frac{\partial f}{\partial x_0} \approx 0.20 \times 2 = 0.40$$

Local rules: d(exp x)/dx = exp x
d(1/x)/dx = −1/x²; d(ax)/dx = a; d(c+x)/dx = 1

The Sigmoid Module and Its Derivative

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

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



$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x))$$

Upstream gradient $\times$ local derivative

$1.00 \times 0.731 \times (1 - 0.731) \approx 0.197$

A computation graph is not unique. Choose modules with simple local gradients.

Forward and Backward Module Interfaces

A custom multiplication operation caches the inputs during the forward pass.

```
class Multiply(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y):
        ctx.save_for_backward(x, y)
        return x * y

    @staticmethod
    def backward(ctx, grad_z):
        x, y = ctx.saved_tensors
        grad_x = y * grad_z
        grad_y = x * grad_z
        return grad_x, grad_y
```

Forward

$z = x \times y$

Backward

Each input receives the upstream gradient multiplied by the other input.

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot y$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \cdot x$$

Sigmoid in PyTorch

Forward implementation

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

```
return 1 / (1 + std::exp(-a));
```

Backward implementation

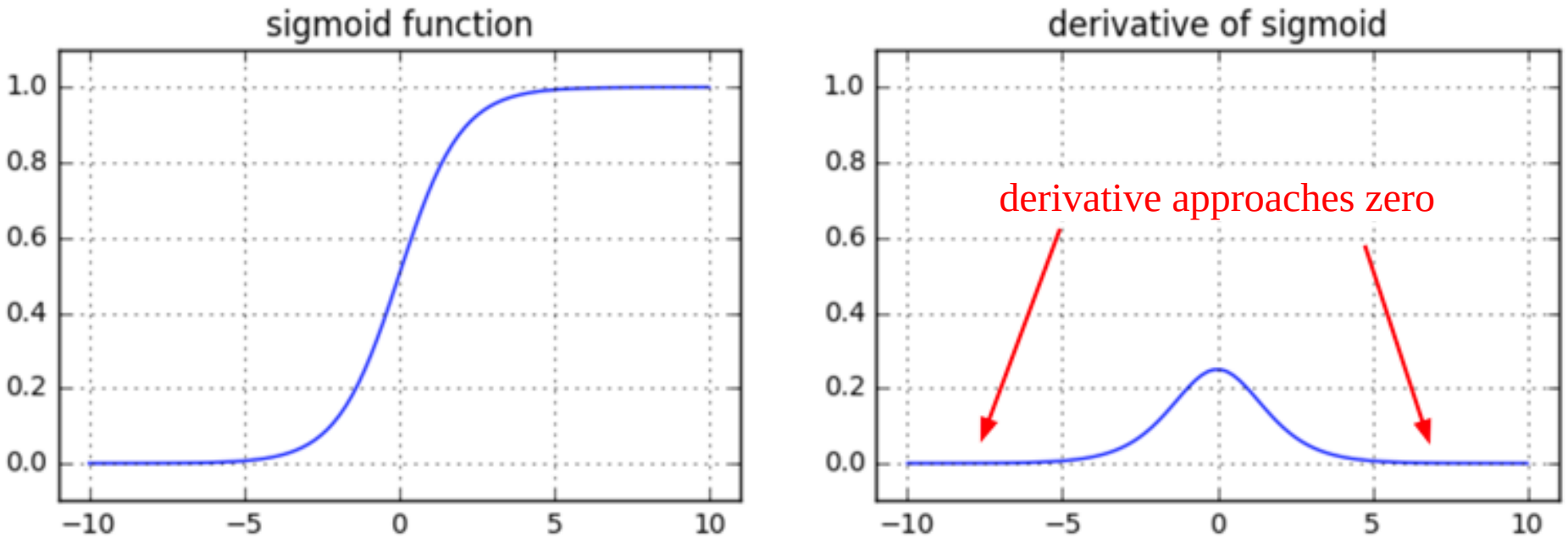
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

```
z = output;
grad_input = grad_output * (1 - z) * z;
```

The backward pass reuses the forward output z.
Both implementations express the same sigmoid and its derivative.

Historical PyTorch example: aten/src/THNN/generic/Sigmoid.c

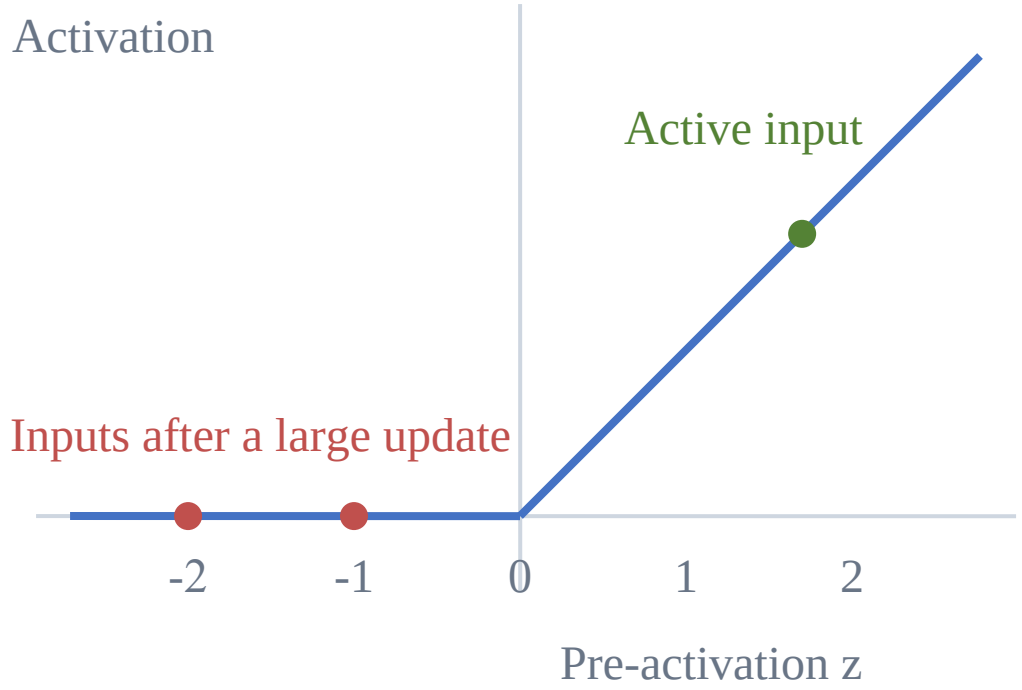
Vanishing Gradients with Sigmoid



Saturated sigmoid or tanh units have very small derivatives.
Initialization should keep activations in a region with useful gradients.

Dying ReLU Units

A unit can stop learning when all training inputs fall on its flat side.



$$a = \max(0, z)$$

1 Positive input

The unit passes a signal and a gradient.

2 Every example has z < 0

The output is zero for every training example.

3 The gradient is blocked

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot 0 \cdot x = 0$$

Smaller updates and suitable initialization reduce the risk. Leaky ReLU keeps a negative-side slope.

Maas et al. (2013), Rectifier Nonlinearities Improve Neural Network Acoustic Models

Exploding Gradients in RNNs

Repeated state transformations can amplify or shrink gradients.

```
H = 5; T = 50
Whh = np.random.randn(H, H)
hs = {-1: np.random.randn(H)}
ss = {}
for t in range(T):
    ss[t] = Whh @ hs[t - 1]
    hs[t] = np.maximum(0, ss[t])

dhs = {T - 1: np.random.randn(H)}
for t in reversed(range(T)):
    dss = (hs[t] > 0) * dhs[t]
    dhs[t - 1] = Whh.T @ dss
```

Forward
Apply the shared matrix, then ReLU.

Backward
Apply the ReLU mask, then the transposed matrix.
Large products of local Jacobians amplify gradients. Small products weaken them. ReLU masks also matter.

Gradient clipping controls gradient magnitude. Gates regulate information flow.