CS11-711 Advanced NLP

Attention and Transformers

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

https://phontron.com/class/anlp-fall2024/

Attention

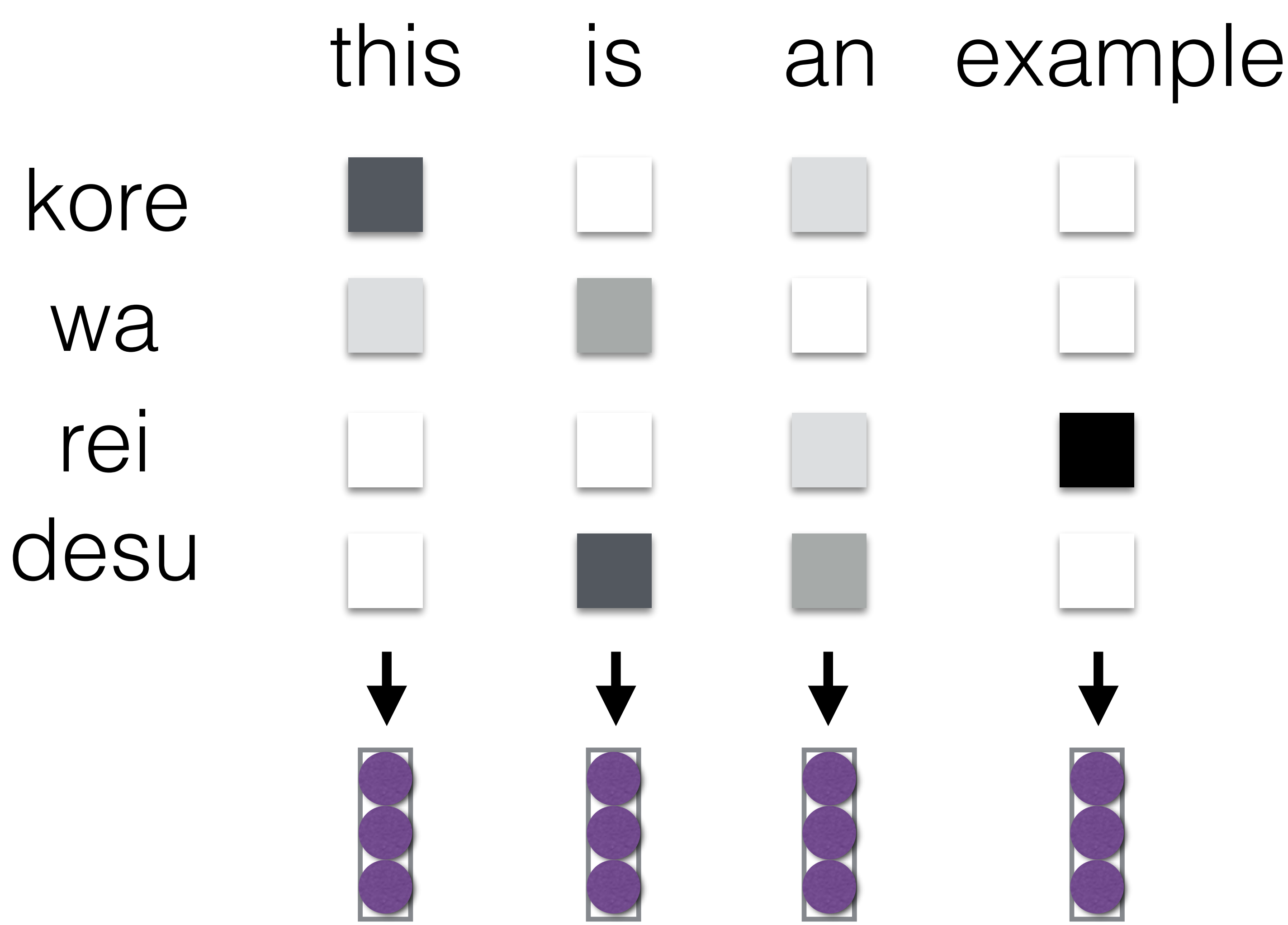
Basic Idea

(Bahdanau et al. 2015)

- Encode each token in the sequence into a vector
- When decoding, perform a linear combination of these vectors, weighted by “attention weights”

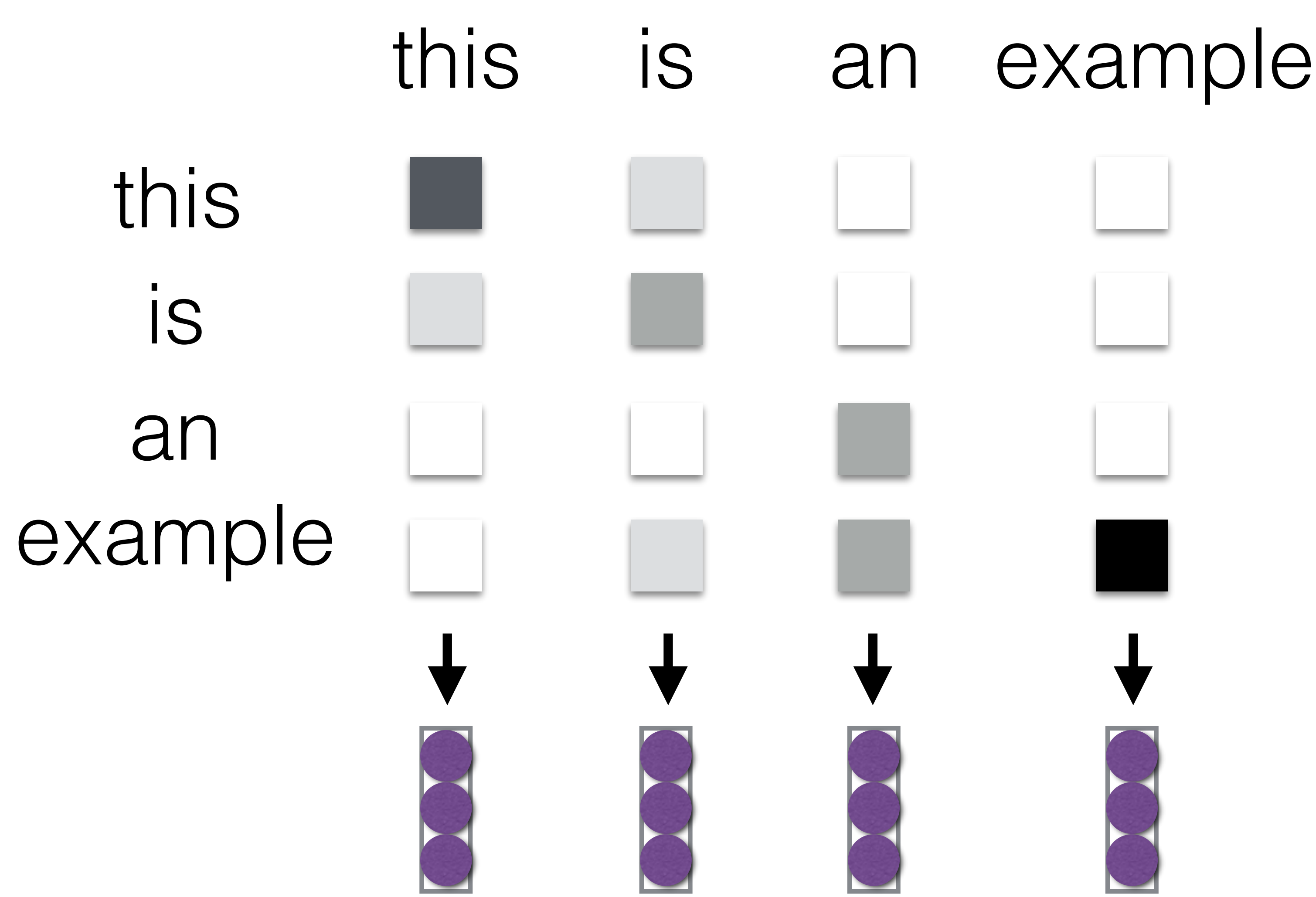
Cross Attention
(Bahdanau et al. 2015)

- Each element in a sequence attends to elements of another sequence



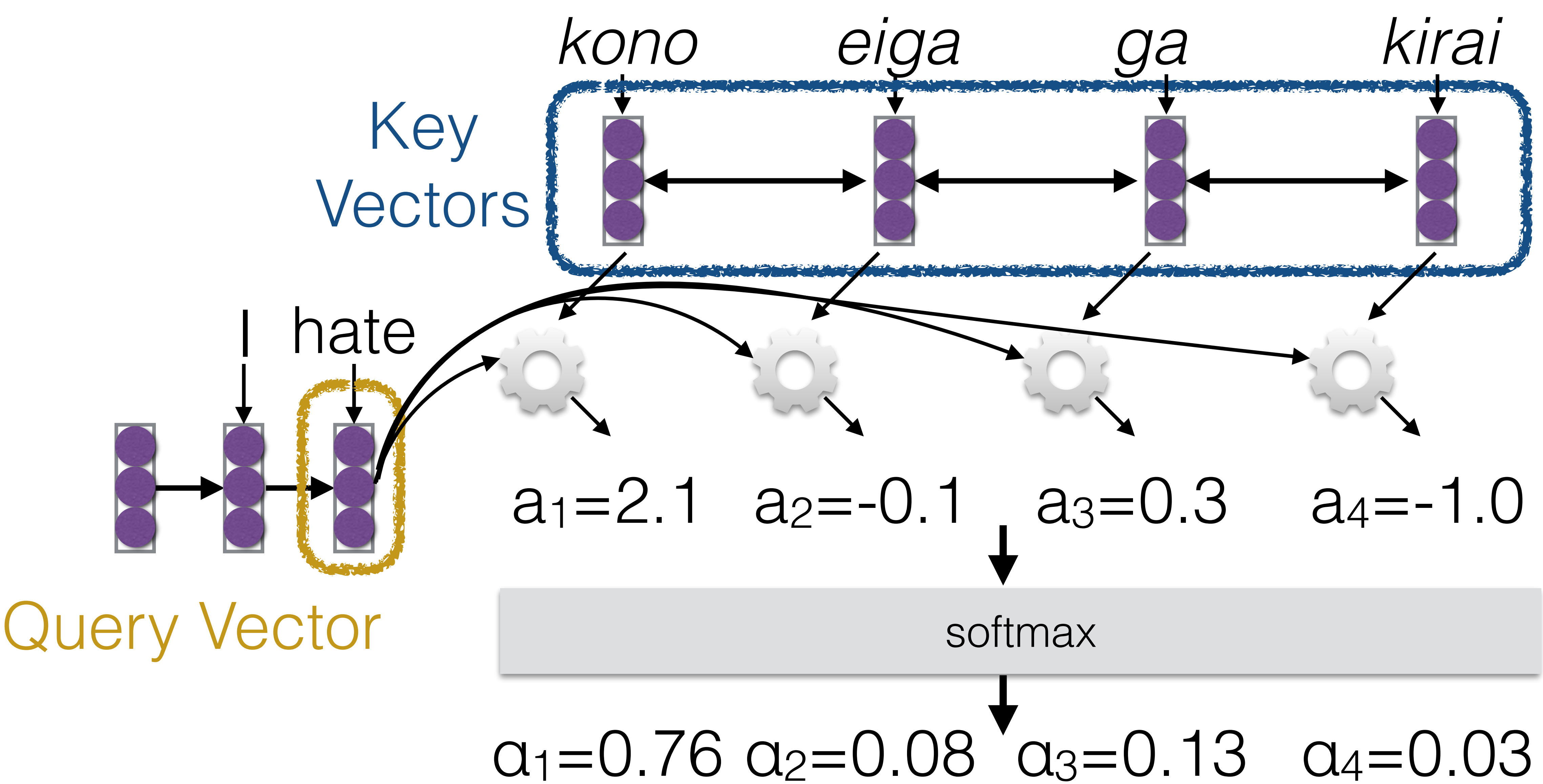
Self Attention
(Cheng et al. 2016, Vaswani et al. 2017)

- Each element in the sequence attends to elements of that sequence → context sensitive encodings!



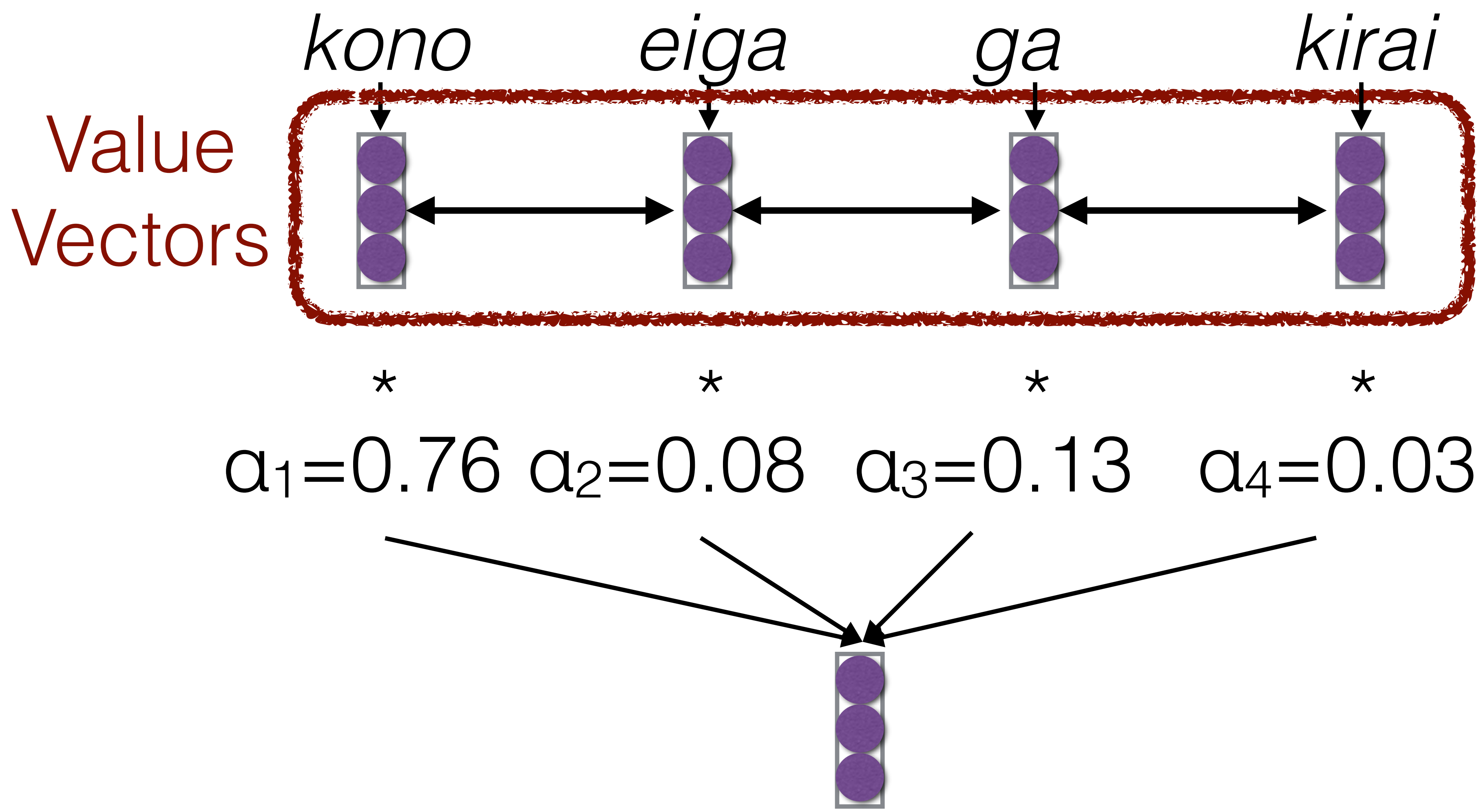
Calculating Attention (1)

- Use "query" vector (decoder state) and "key" vectors (all encoder states)
- For each query-key pair, calculate weight
- Normalize to add to one using softmax



Calculating Attention (2)

- Combine together value vectors (usually encoder states, like key vectors) by taking the weighted sum



- Use this in any part of the model you like

A Graphical Example

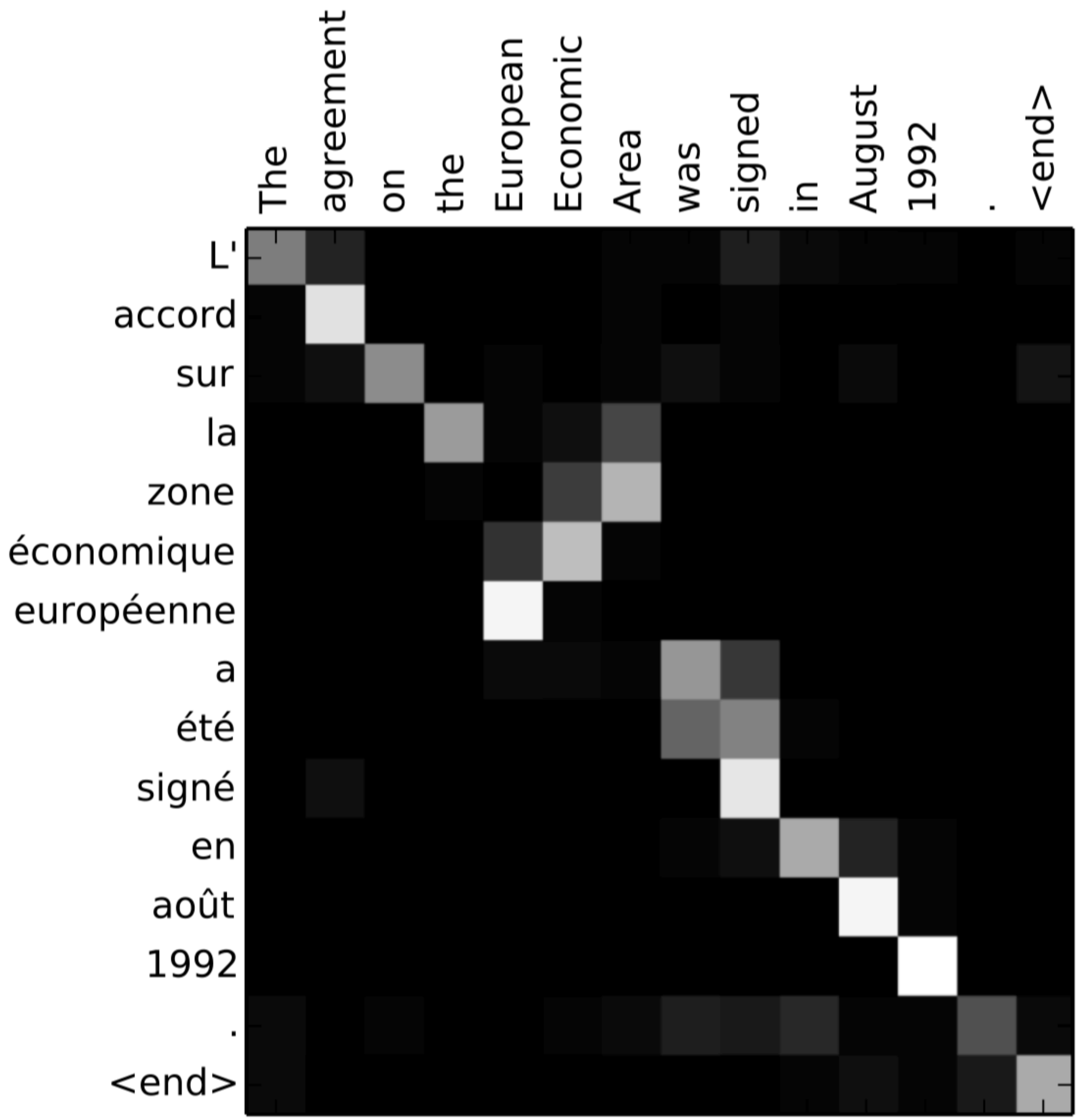


Image from Bahdanau et al. (2015)

Attention Score Functions (1)

- ***q*** is the query and ***k*** is the key
- **Multi-layer Perceptron** (Bahdanau et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^\intercal \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$$

  - Flexible, often very good with large data
- **Bilinear** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal W \boldsymbol{k}$$

Attention Score Functions (2)

- **Dot Product** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal \boldsymbol{k}$$

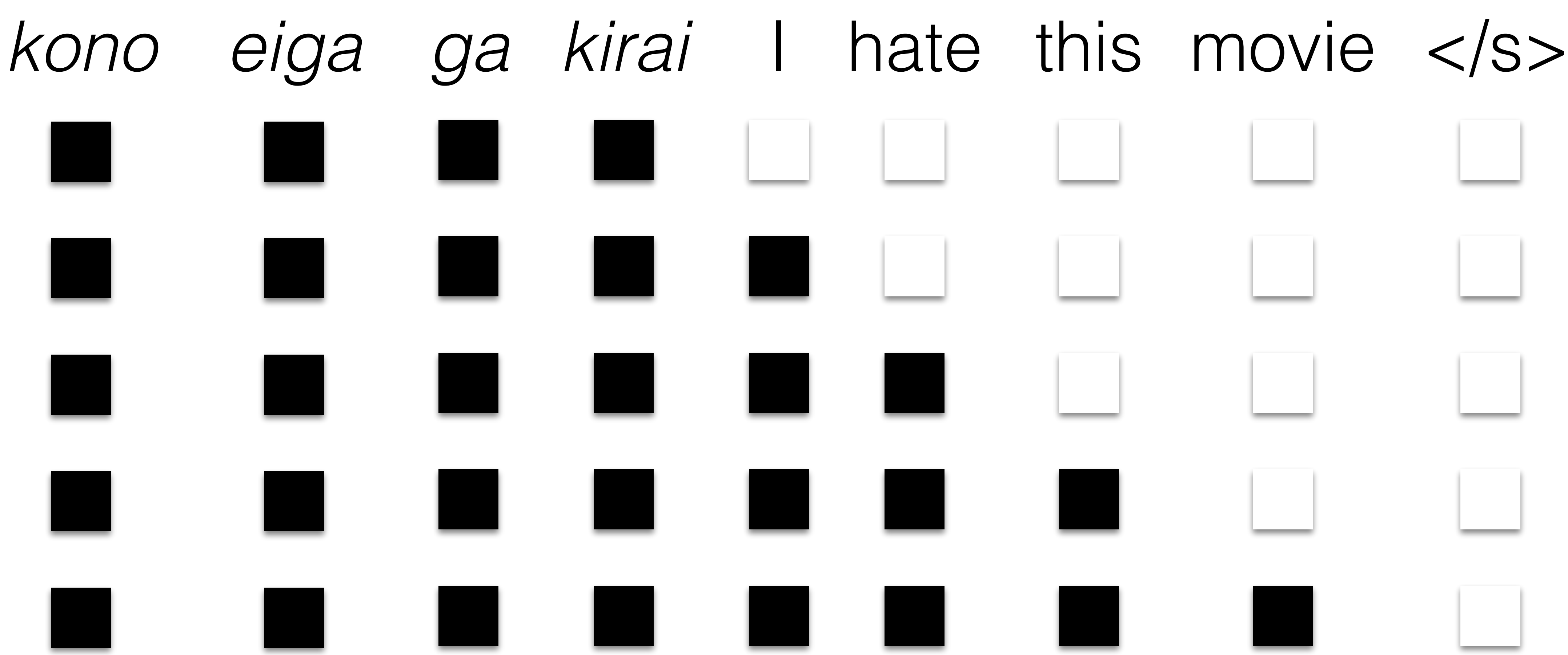
  - No parameters! But requires sizes to be the same.

- **Scaled Dot Product** (Vaswani et al. 2017)
  - *Problem:* scale of dot product increases as dimensions get larger
  - *Fix:* scale by size of the vector

$$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^\intercal \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

Masking for Training

- We want to perform training in as few operations as possible using big matrix multiplies
- We can do so by “masking” the results for the output

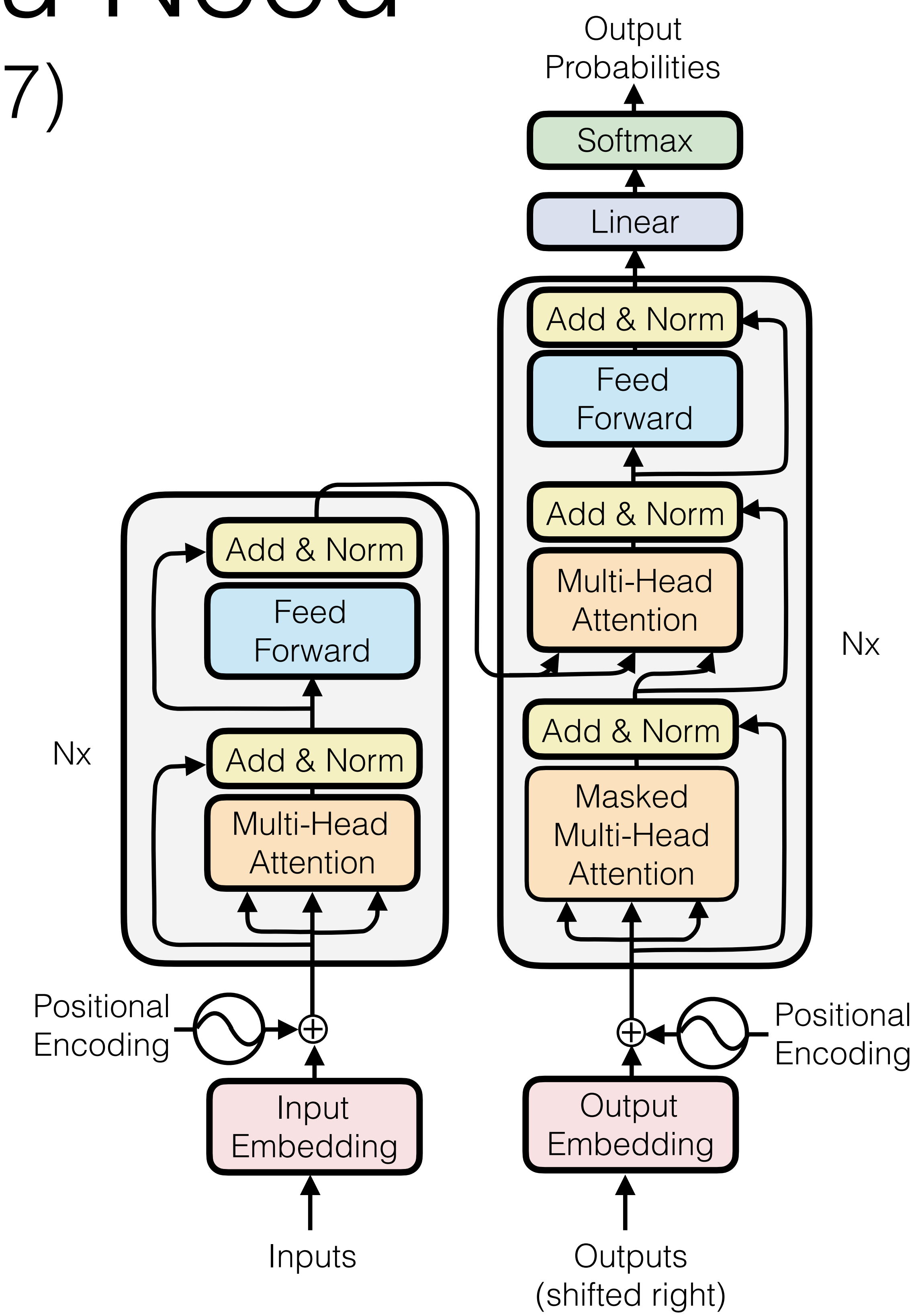


Transformers

“Attention is All You Need”

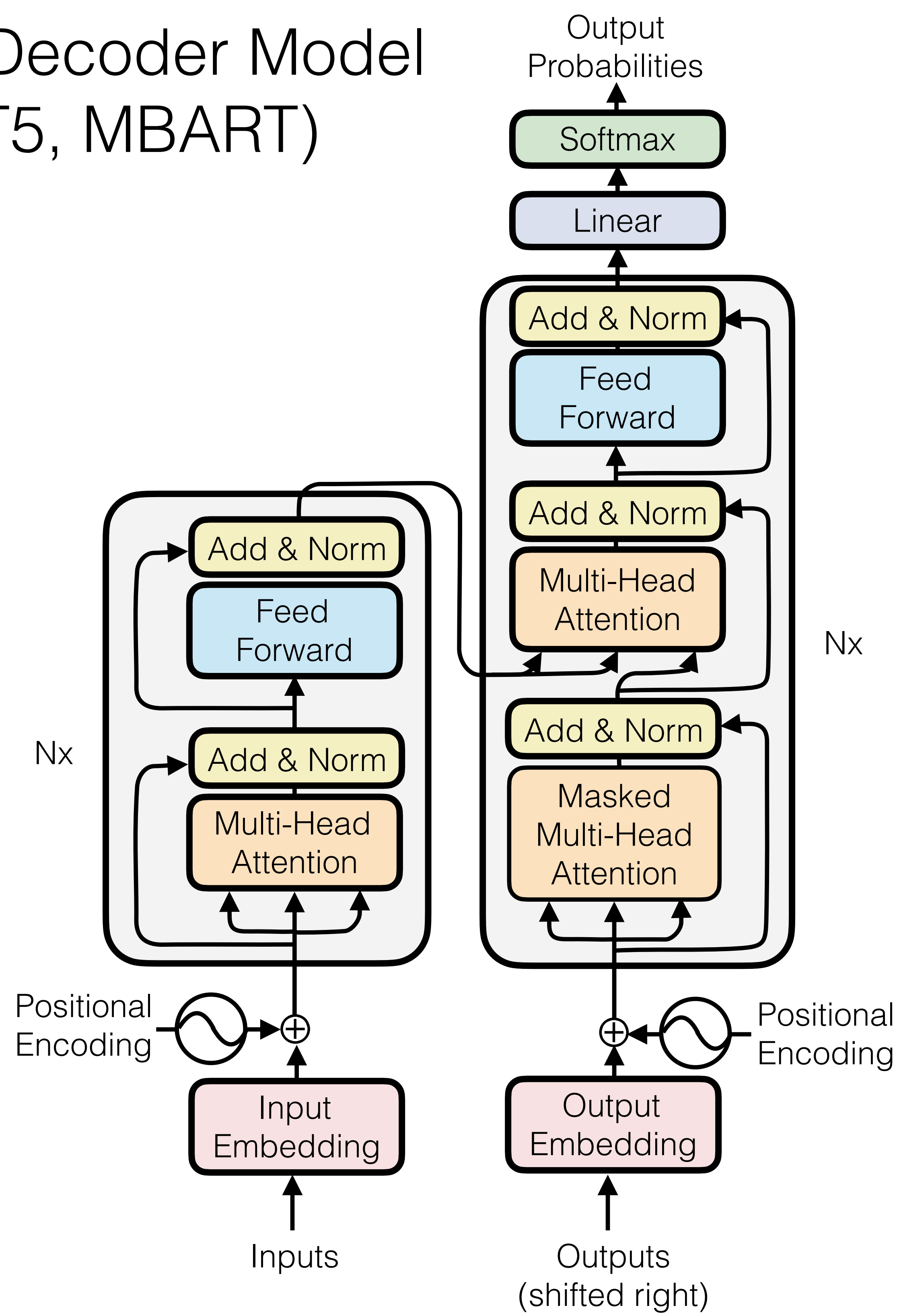
(Vaswani et al. 2017)

- A sequence-to-sequence model based entirely on attention
- Strong results on machine translation
- Fast: only matrix multiplications

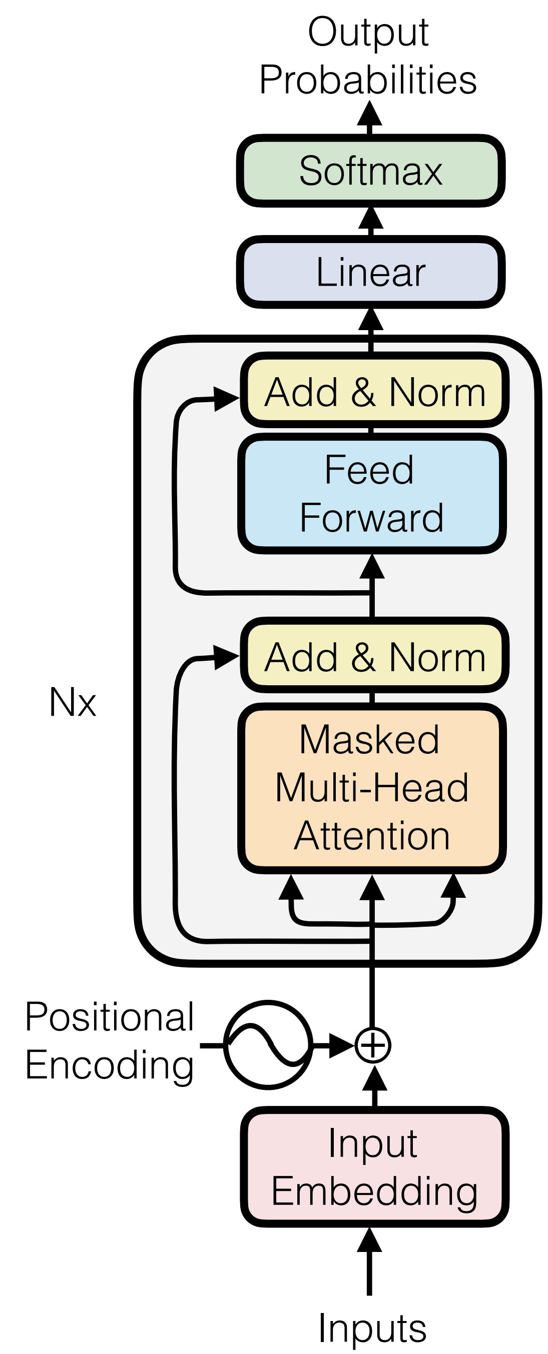


Two Types of Transformers

Encoder-Decoder Model
(e.g. T5, MBART)



Decoder Only Model
(e.g. GPT, LLaMa)



Core Transformer Concepts

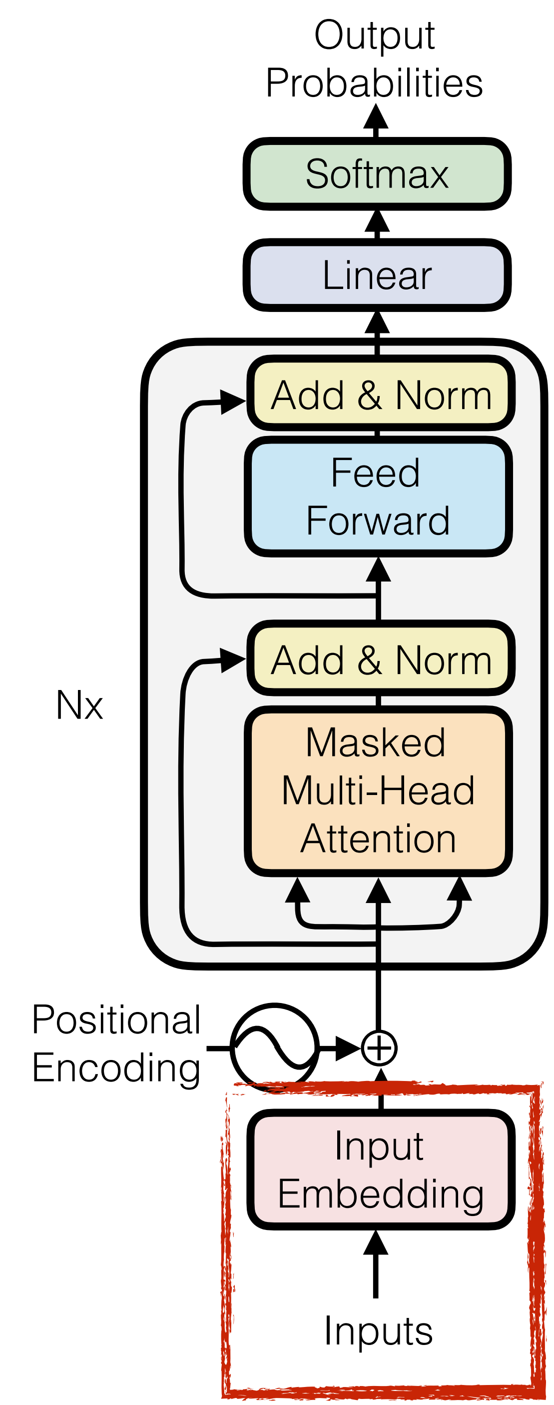
- Positional encodings
- Multi-headed attention
- Masked attention
- Residual + layer normalization
- Feed-forward layer

(Review)
Inputs and Embeddings

- **Inputs:** Generally split using subwords

the books were improved
the book _s were **improv _ed**

- **Input Embedding:** Looked up, like in previously discussed models



Multi-head Attention

Intuition for Multi-heads

- **Intuition:** Information from different parts of the sentence can be useful to disambiguate in different ways

I **run** a small business

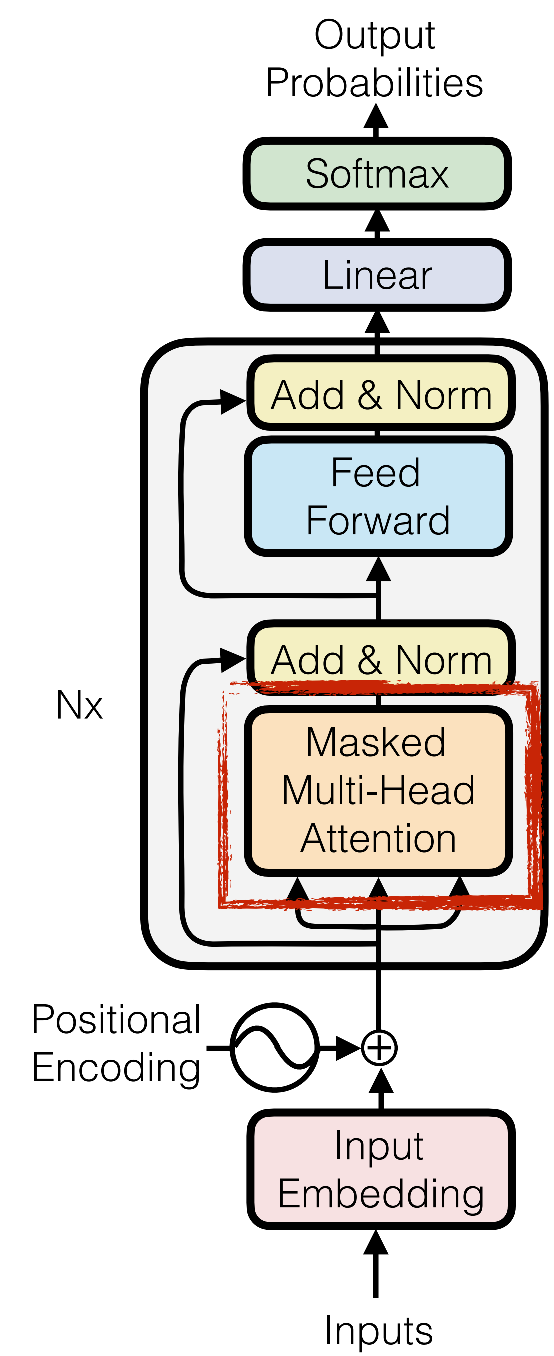
I **run** a mile in 10 minutes

The robber made a **run** for it

The stocking had a **run**

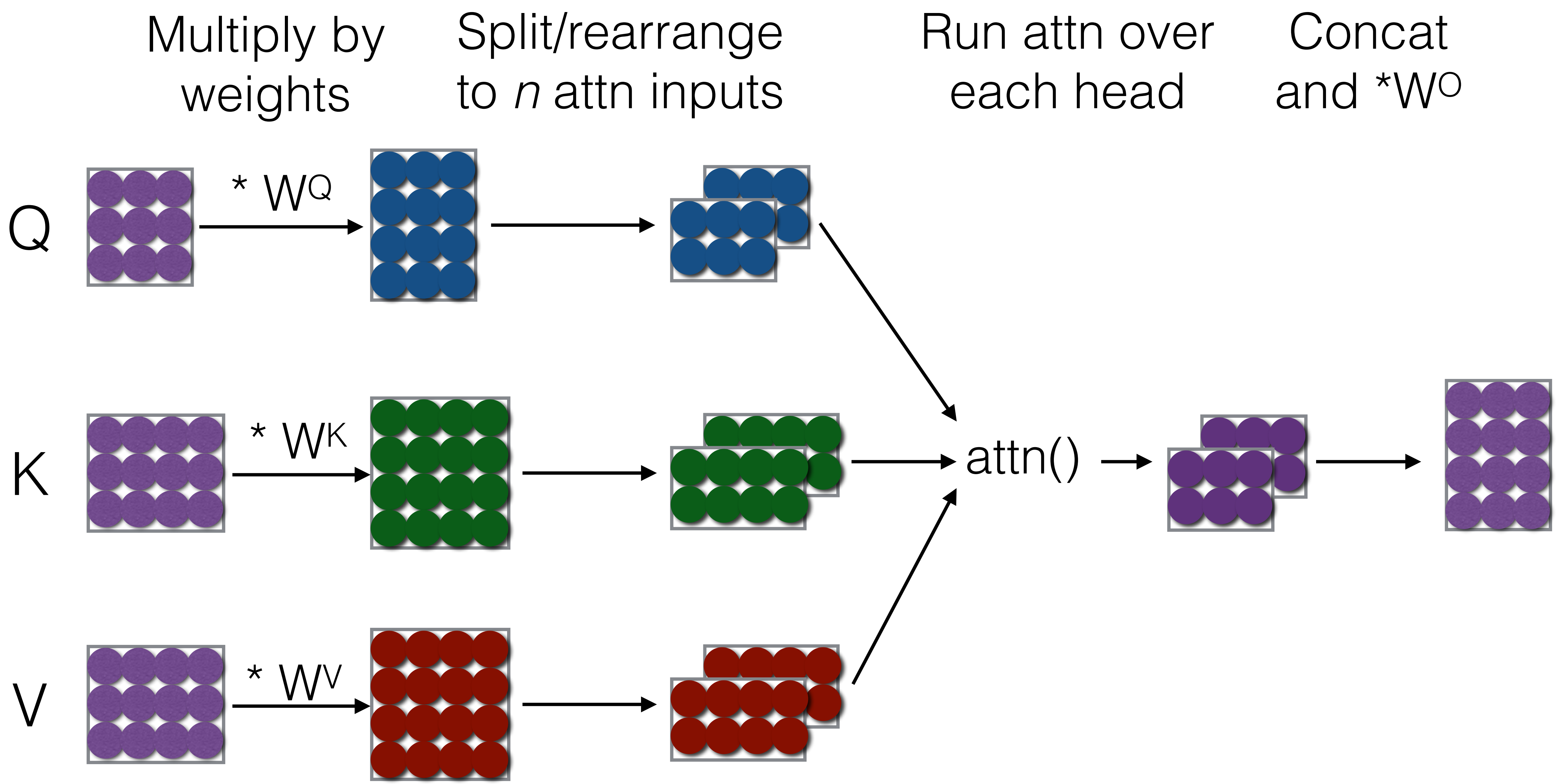
syntax
(nearby context)

semantics
(farther context)



Multi-head Attention Concept

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



Code Example

```
def forward(self, query, key, value, mask=None):
    nbatches = query.size(0)

    # 1) Do all the linear projections
    query = self.W_q(query)
    key = self.W_k(key)
    value = self.W_v(value)

    # 2) Reshape to get h heads
    query = query.view(nbatches, -1, self.heads, self.d_k).transpose(1, 2)
    key = key.view(nbatches, -1, self.heads, self.d_k).transpose(1, 2)
    value = value.view(nbatches, -1, self.heads, self.d_k).transpose(1, 2)

    # 3) Apply attention on all the projected vectors in batch.
    x, self.attn = attention(query, key, value)

    # 4) "Concat" using a view and apply a final linear.
    x = (
        x.transpose(1, 2)
        .contiguous()
        .view(nbatches, -1, self.h * self.d_k)
    )
    return self.W_o(x)
```

What Happens w/ Multi-heads?

- Example from Vaswani et al.

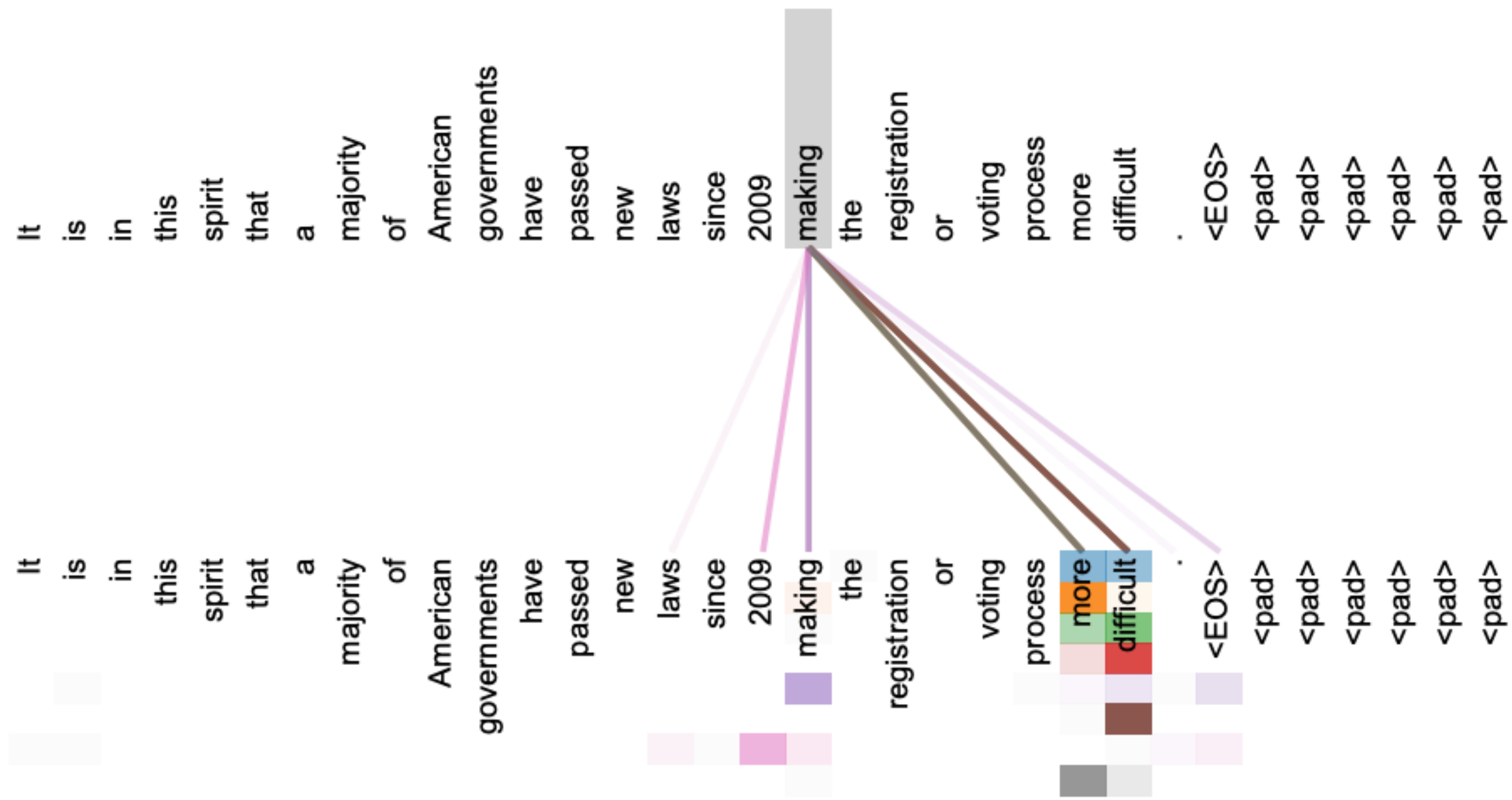


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

- See also BertVis: https://github.com/jessevig/bertviz

Positional Encoding

Positional Encoding

- The transformer model is *purely* attentional
- If embeddings were used, there would be no way to distinguish between *identical words*

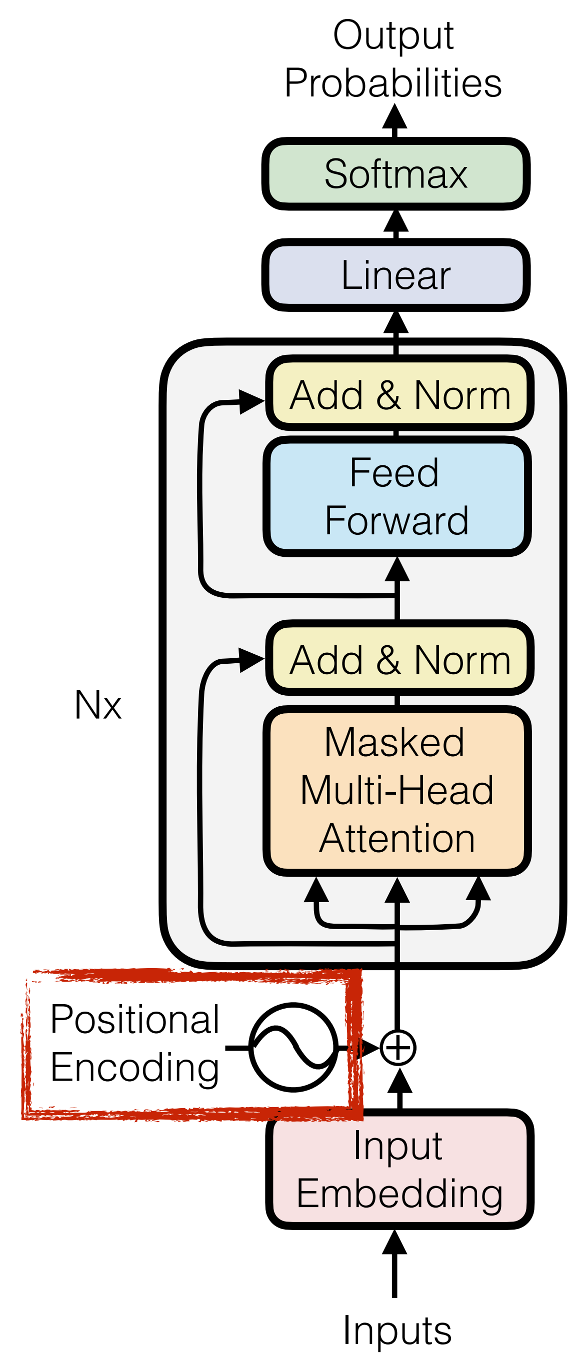
A big dog and a big cat

would be identical!

A big dog and a big cat

- Positional encodings add an embedding based on the word position

$W_{big} + W_{pos2}$ $W_{big} + W_{pos6}$

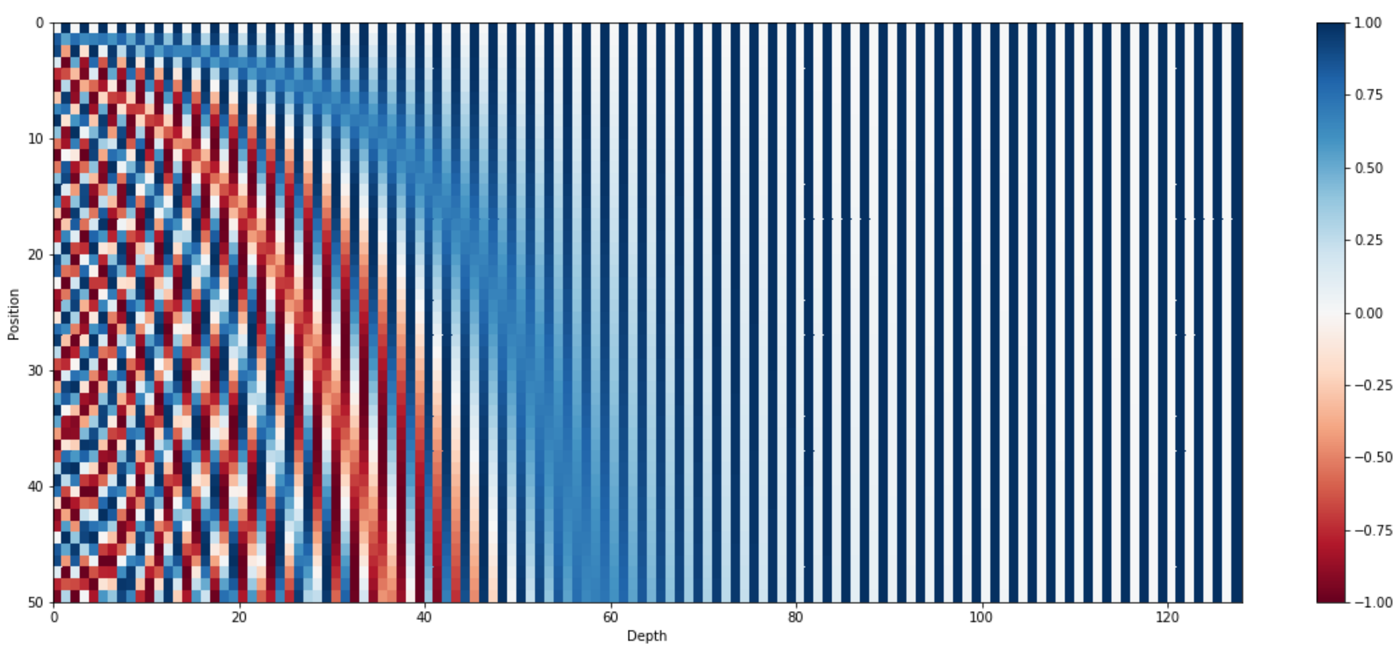


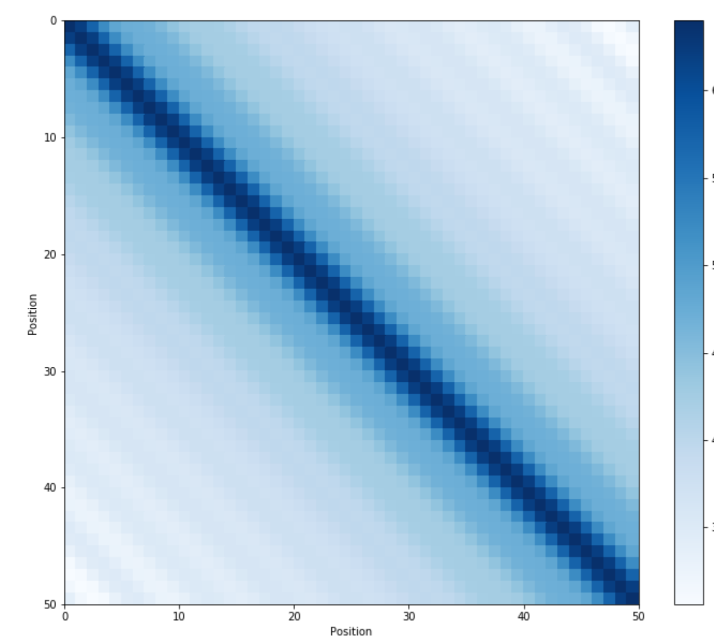
Sinusoidal Encoding

(Vaswani+ 2017, Kazemnejad 2019)

- Calculate each dimension with a sinusoidal function

$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$





- Why? So the dot product between two embeddings becomes higher relatively.

Learned Encoding
(Shaw+ 2018)

- More simply, just create a learnable embedding
- **Advantages:** flexibility
- **Disadvantages:** impossible to extrapolate to longer sequences

Absolute vs. Relative Encodings
(Shaw+ 2018)

- **Absolute** positional encodings add an encoding to the input in *hope* that relative position will be captured

- **Relative** positional encodings *explicitly* encode relative position

Rotary Positional Encodings (RoPE)

(Su+ 2021)

- **Fundamental idea:** we want the dot product of embeddings to result in a function of relative position

$$f_q(\mathbf{x}_m, m) \cdot f_k(\mathbf{x}_n, n) = g(\mathbf{x}_m, \mathbf{x}_n, m - n)$$

- In summary, RoPE uses trigonometry and imaginary numbers to come up with a function that satisfies this property

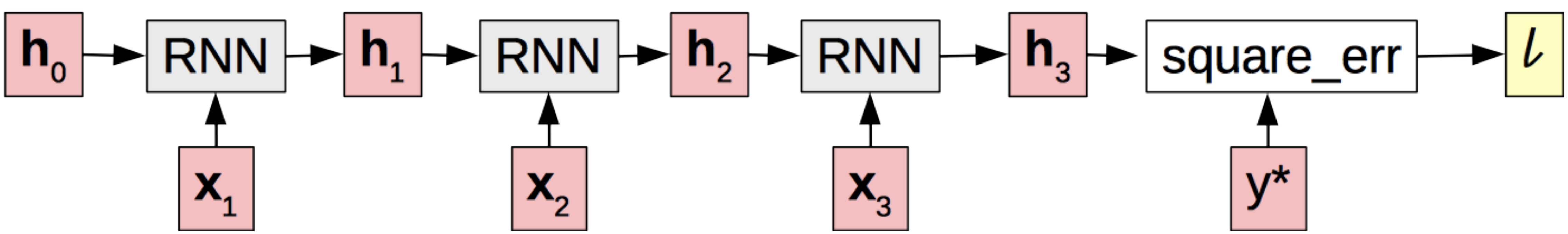
$$R^d_{\Theta,m}\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{\frac{d}{2}} \\ \cos m\theta_{\frac{d}{2}} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{\frac{d}{2}} \\ \sin m\theta_{\frac{d}{2}} \end{pmatrix}$$

Layer Normalization and Residual Connections

Reminder: Gradients and Training Instability

- In RNNs, we asked about how backdrop through a network causes gradients can vanish or explode

$$\frac{dl}{d_{h_0}} = \text{tiny} \qquad \frac{dl}{d_{h_1}} = \text{small} \qquad \frac{dl}{d_{h_2}} = \text{med.} \qquad \frac{dl}{d_{h_3}} = \text{large}$$



- The same issue occurs in multi-layer transformers!

Layer Normalization

(Ba et al. 2016)

- Normalizes the outputs to be within a consistent range, preventing too much variance in scale of outputs

gain

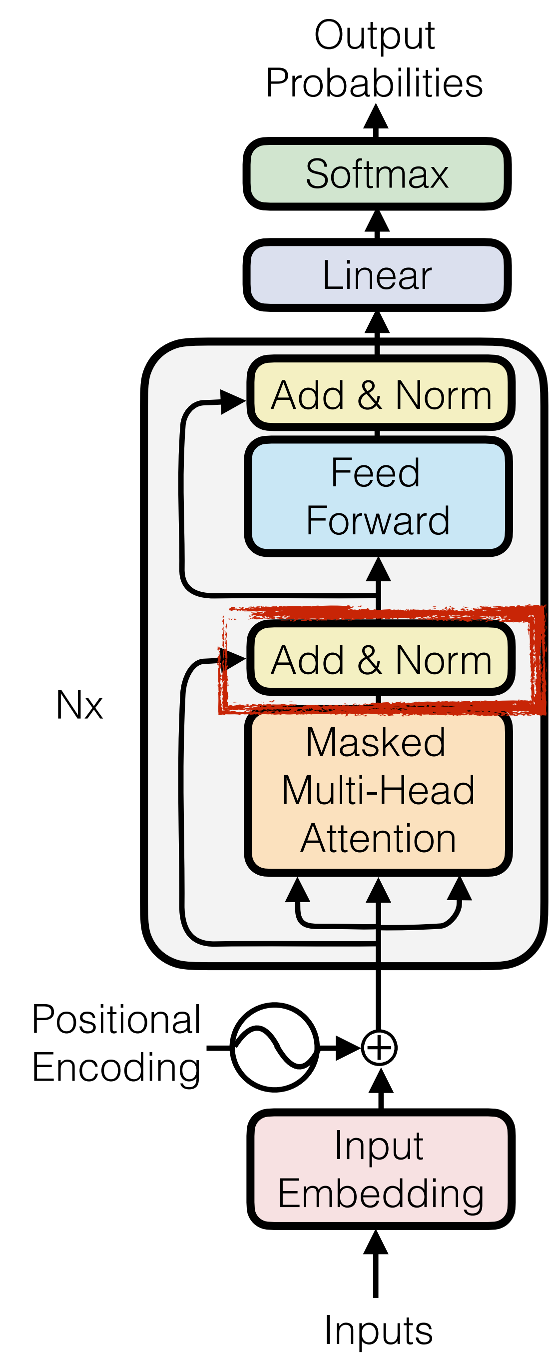
bias

$$\text{LayerNorm}(\mathbf{x}; \mathbf{g}, \mathbf{b}) = \frac{\boxed{\mathbf{g}}}{\boxed{\sigma(\mathbf{x})}} \odot (\mathbf{x} - \boxed{\mu(\mathbf{x})}) + \boxed{\mathbf{b}}$$

vector stddev

vector mean

$$\mu(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} x_i \qquad \sigma(\mathbf{x}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2}$$



RMSNorm
(Zhang and Sennrich 2019)

- Simplifies LayerNorm by removing the mean and bias terms

$$\mathrm{RMS}(\mathbf{x}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

$$\mathrm{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\mathrm{RMS}(\mathbf{x})} \cdot \mathbf{g}$$

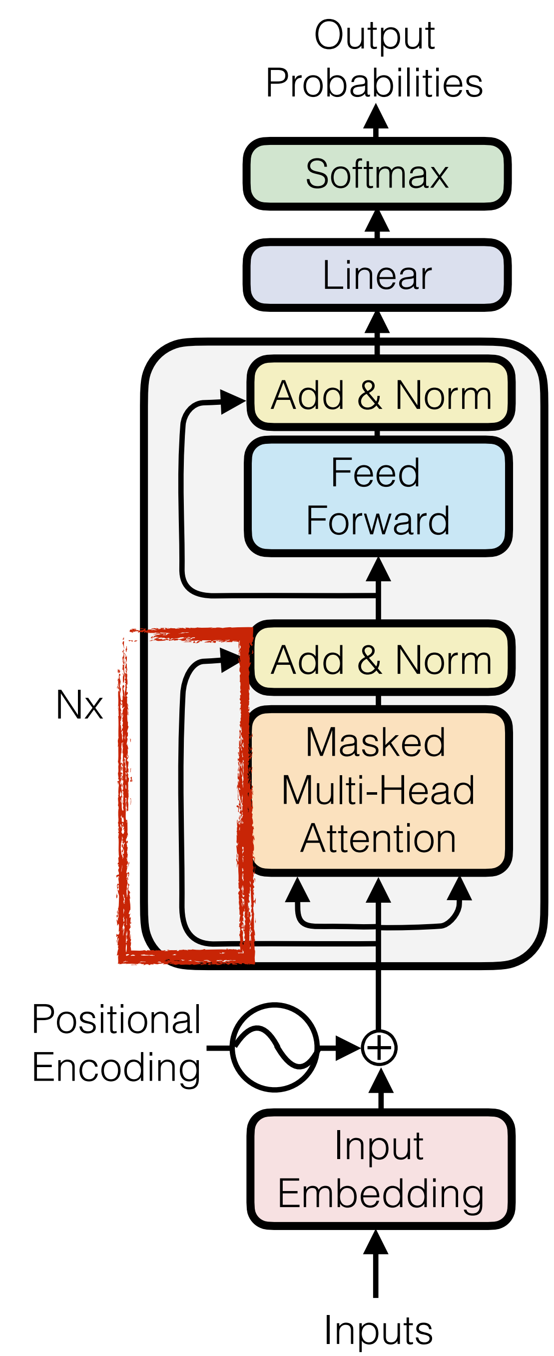
Residual Connections

- Add an additive connection between the input and output

$$\text{Residual}(\mathbf{x}, f) = f(\mathbf{x}) + \mathbf{x}$$

- Prevents vanishing gradients and allows f to learn the *difference* from the input

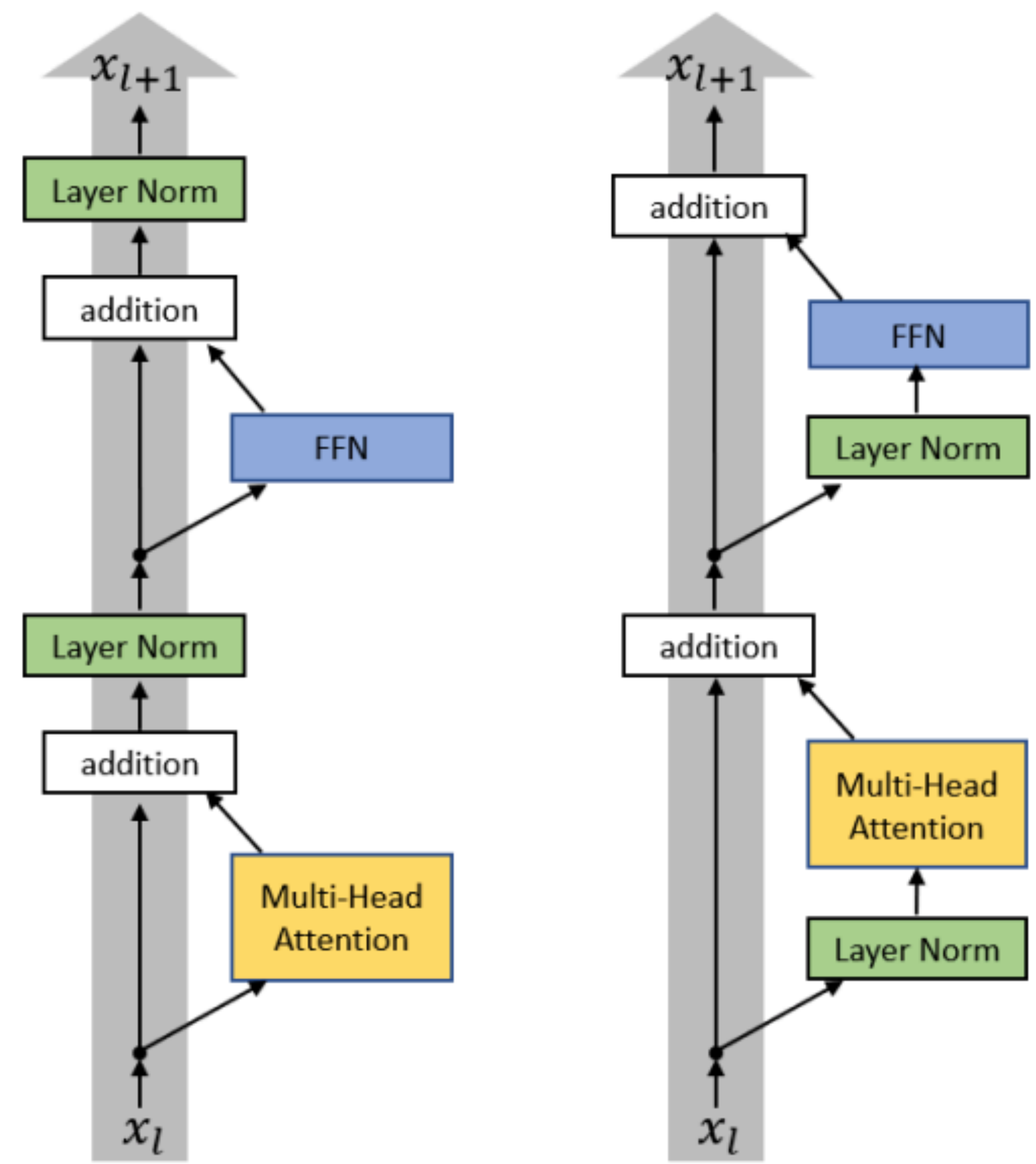
- *Quiz:* what are the implications for self-attention w/ and w/o residual connections?



Post- vs. Pre- Layer Norm

(e.g. Xiong et al. 2020)

- Where should LayerNorm be applied? Before or after?
- Pre-layer-norm is better for gradient propagation



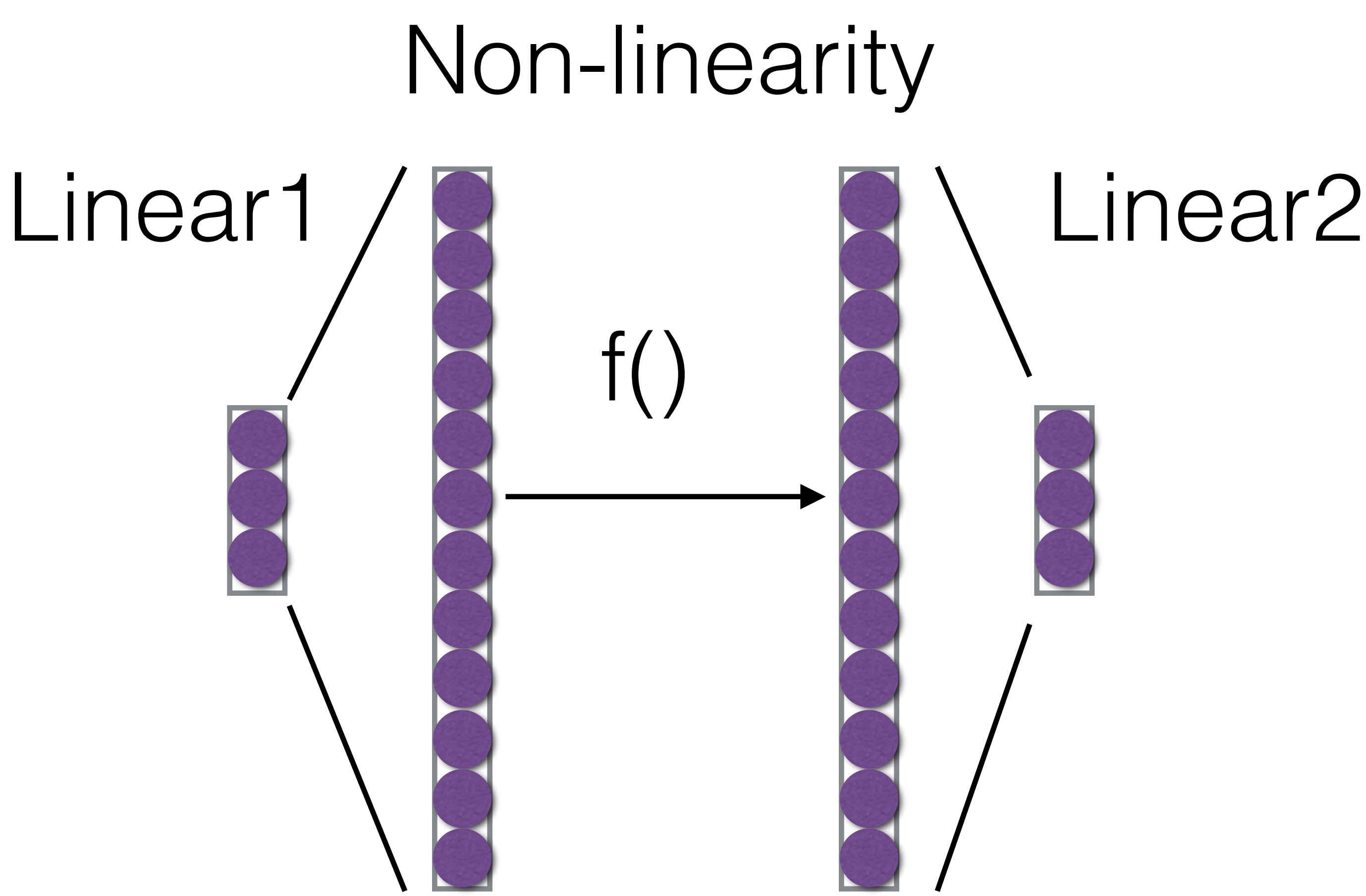
post-LayerNorm pre-LayerNorm

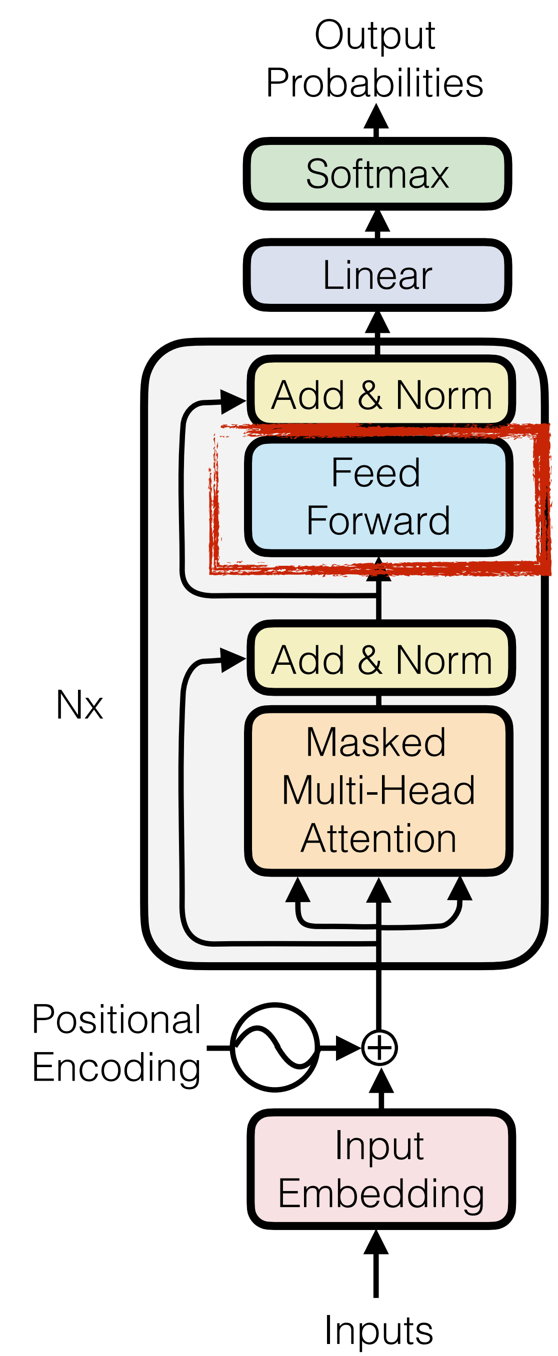
Feed Forward Layers

Feed Forward Layers

- Extract combination features from the attended outputs

$$\mathrm{FFN}(x; W_1, \mathbf{b}_1, W_2, \mathbf{b}_2) = f(\mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$

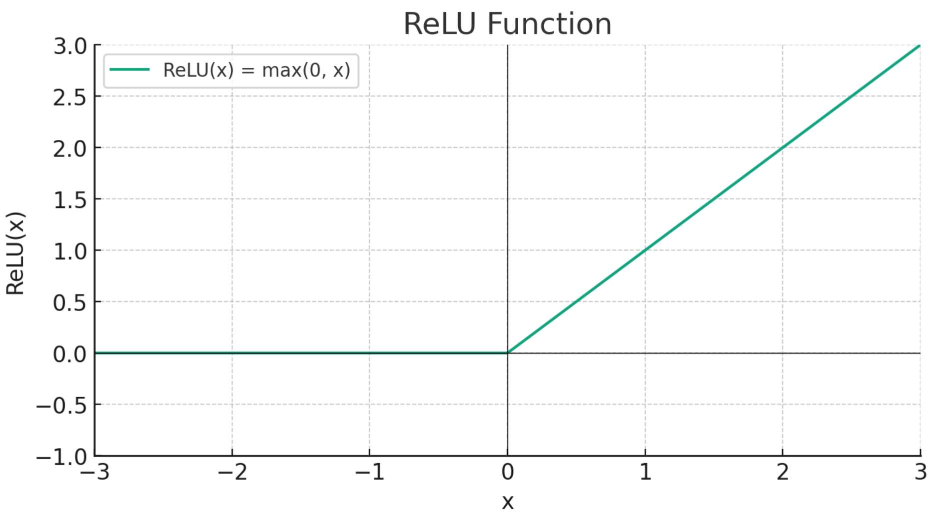




Some Activation Functions in Transformers

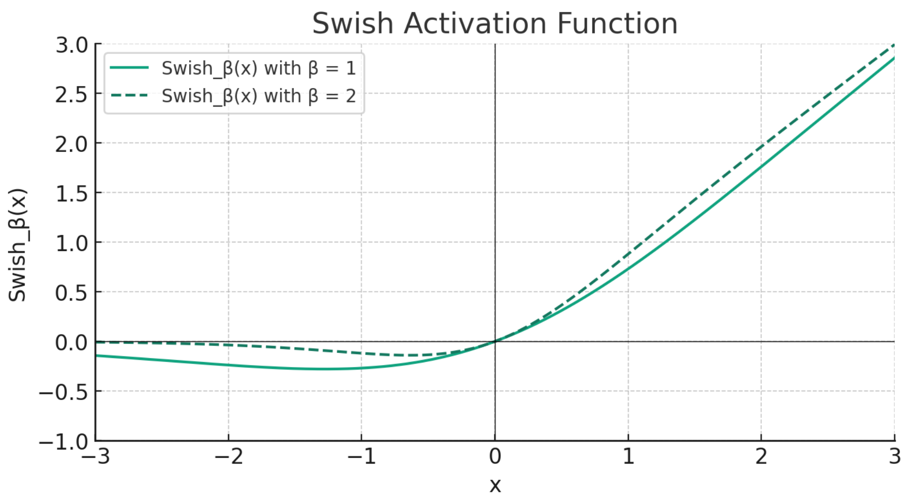
- Vaswani et al.: ReLU

$$\mathrm{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$$



- LLaMa: Swish/SiLU (Hendricks and Gimpel 2016)

$$\mathrm{Swish}(\mathbf{x}; \beta) = \mathbf{x} \odot \sigma(\beta \mathbf{x})$$



Optimization Tricks for Transformers

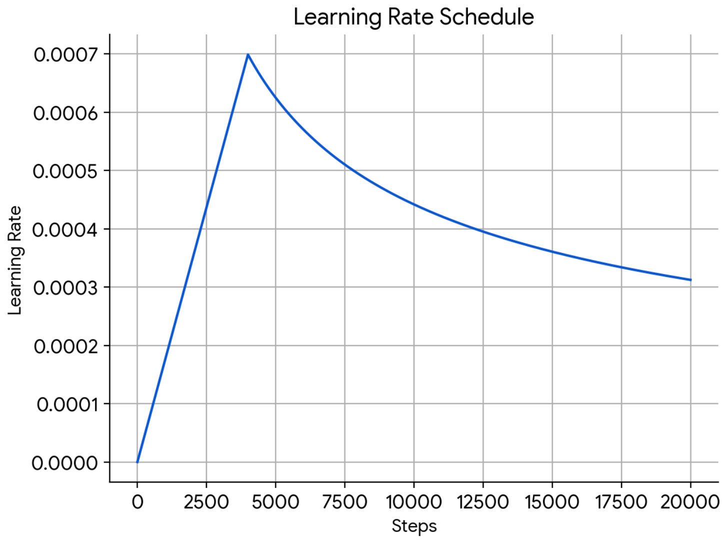
Transformers are Powerful but Fickle

- Optimization of models can be difficult, and transformers are more difficult than others!
- e.g. OPT-175 training logbook https://github.com/facebookresearch/metaseq/blob/main/projects/OPT/chronicles/OPT175B_Logbook.pdf

Optimizers for Transformers

- SGD: Update in the direction of reducing loss
- Adam: Add momentum turn and normalize by stddev of the outputs
- Adam w/ learning rate schedule (Vaswani et al. 2017): Adds a learning rate increase and decrease

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step^{-0.5}, step * warmup\_steps^{-1.5})$$



- AdamW (Loshchilov and Hutter 2017): properly applies weight decay for regularization to Adam

Low-Precision Training

- Training at full 32-bit precision can be costly
- Low-precision alternatives

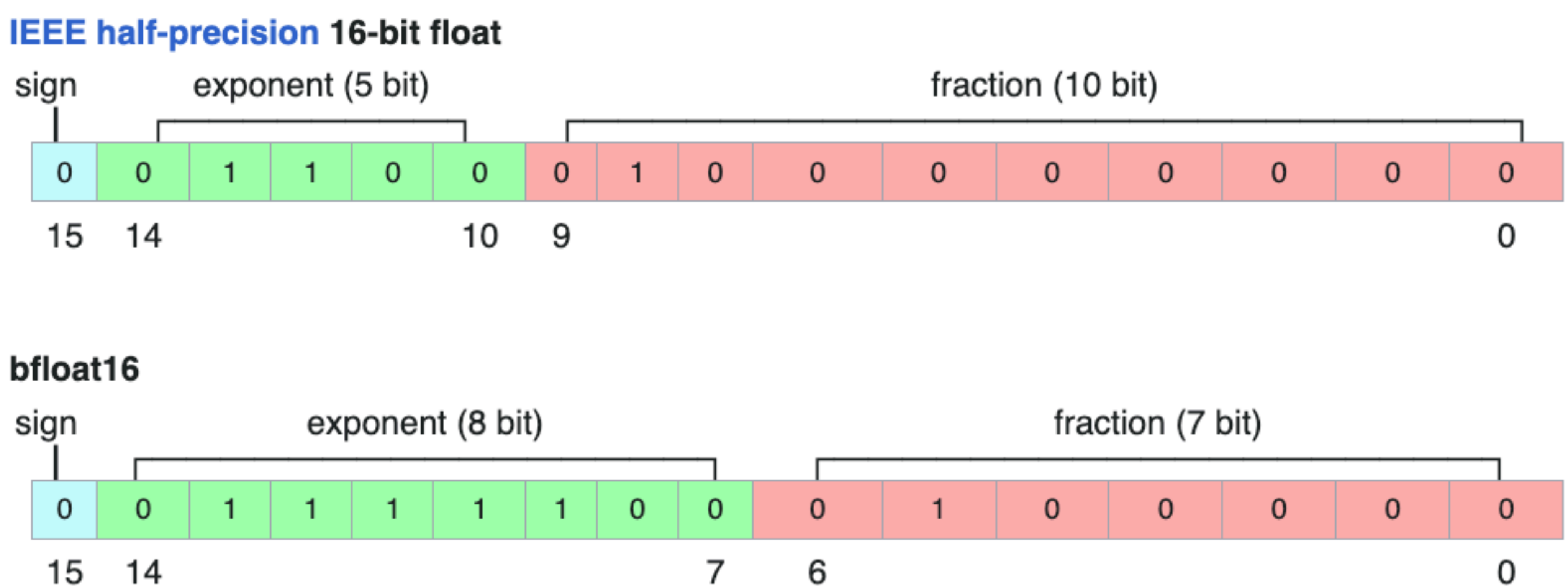
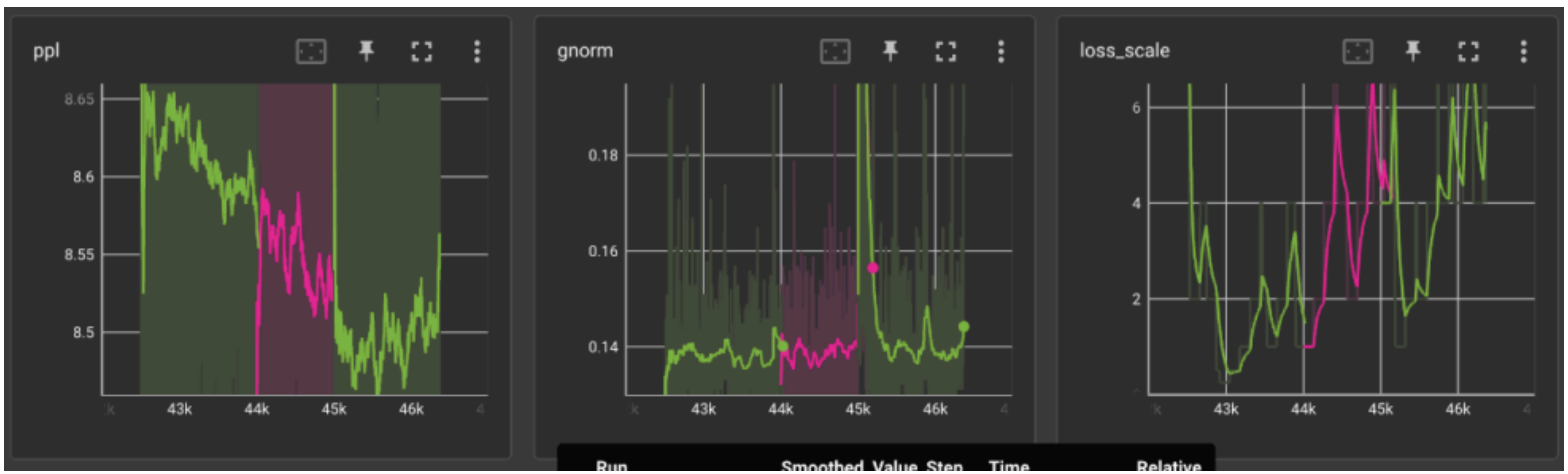


Image: Wikipedia

Checkpointing/Restarts

- Even through best efforts, training can go south — what to do?
- Monitor possible issues, e.g. through monitoring the norm of gradients



- If training crashes, roll back to previous checkpoint, shuffle data, and resume
- (Also, check your code)

Image: OPT Log

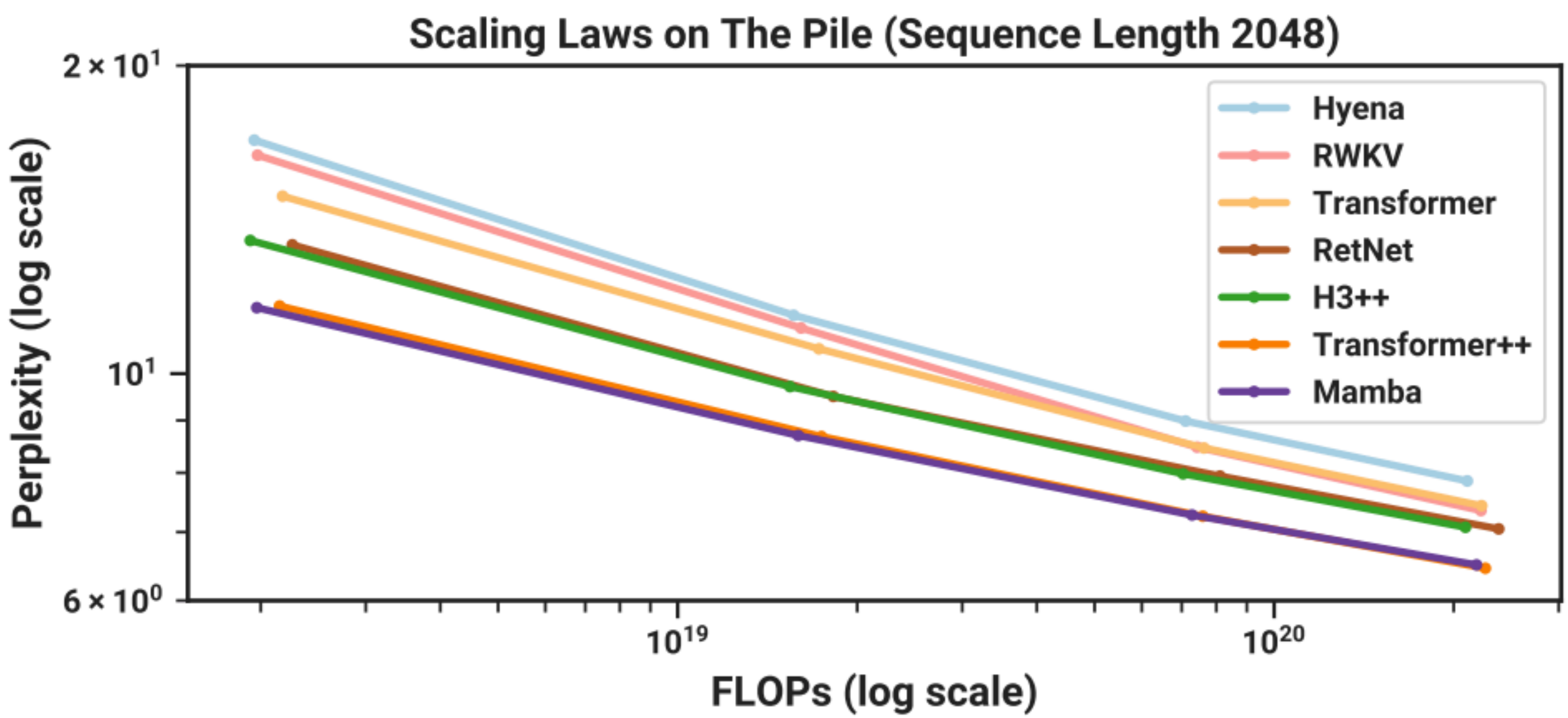
Comparing Transformer Architectures

Original Transformer vs. LLaMa

	Vaswani et al.	LLaMA
Norm Position	Post	Pre
Norm Type	LayerNorm	RMSNorm
Non-linearity	ReLU	SiLU
Positional Encoding	Sinusoidal	RoPE

How Important is It?

- “Transformer” is Vaswani et al., “Transformer++” is (basically) LLaMA

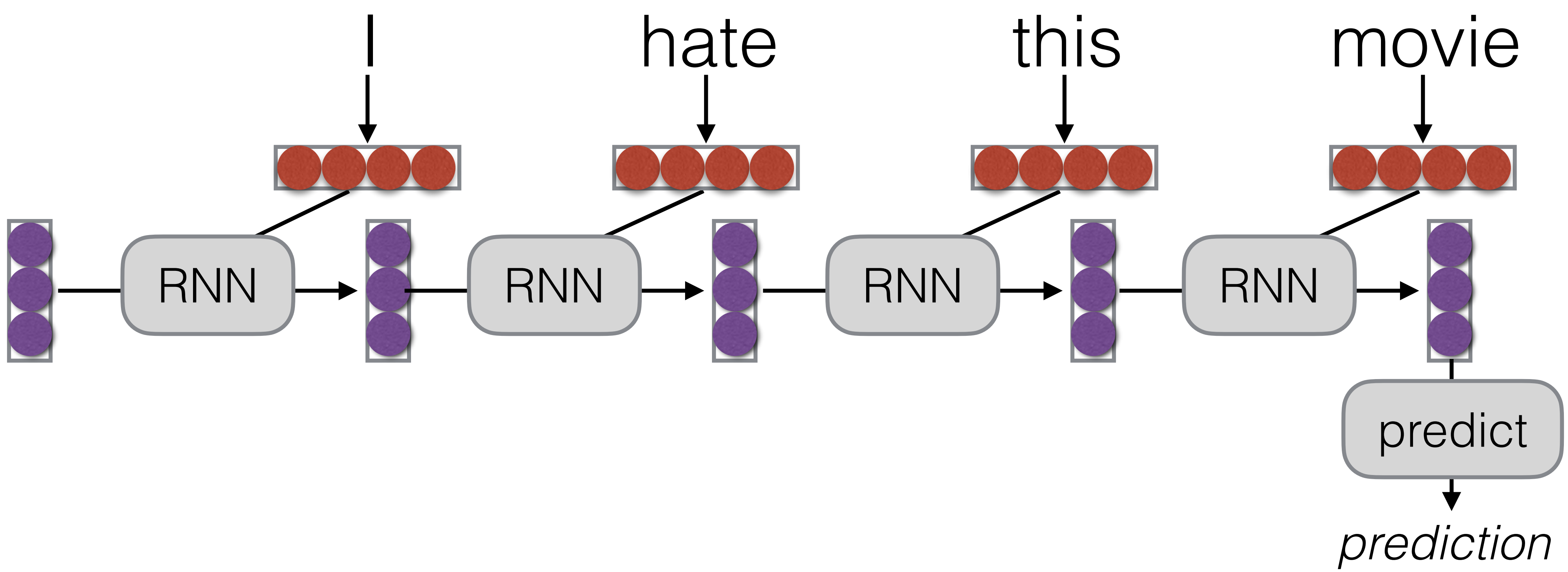


- Stronger architecture is ≈10x more efficient!

Image: Gu and Dao (2023)

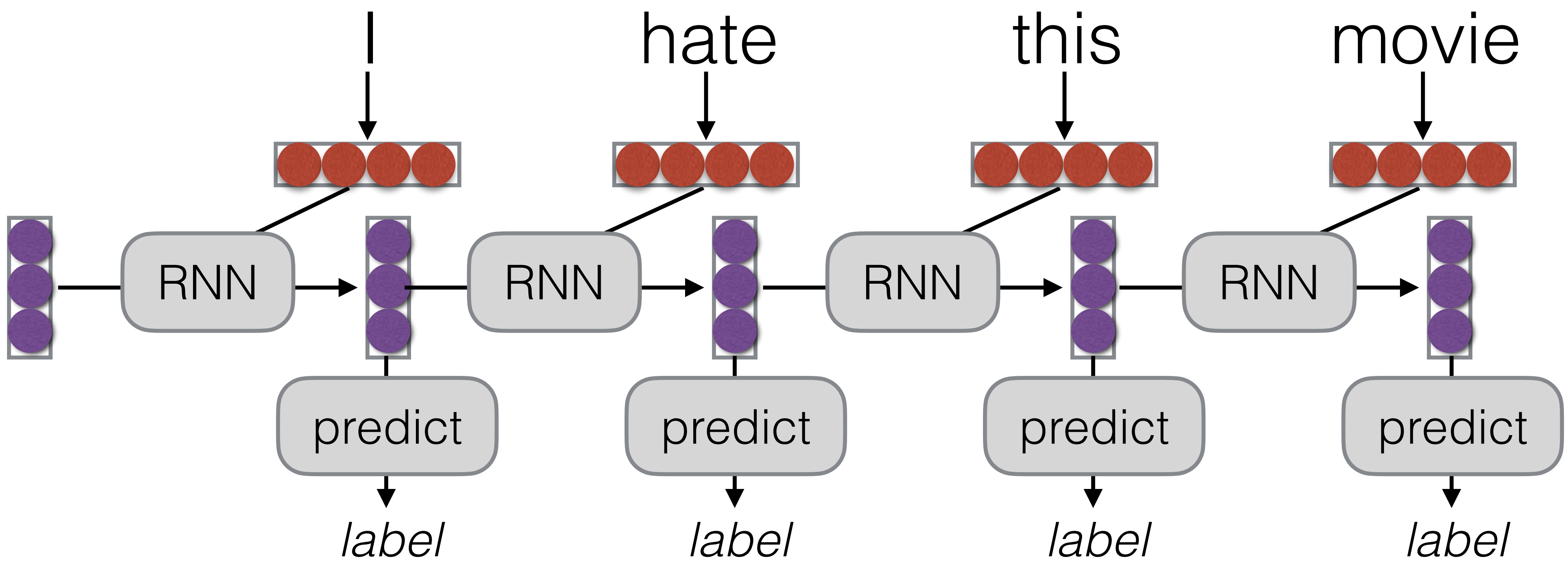
Applications of Sequence Models

Encoding Sequences



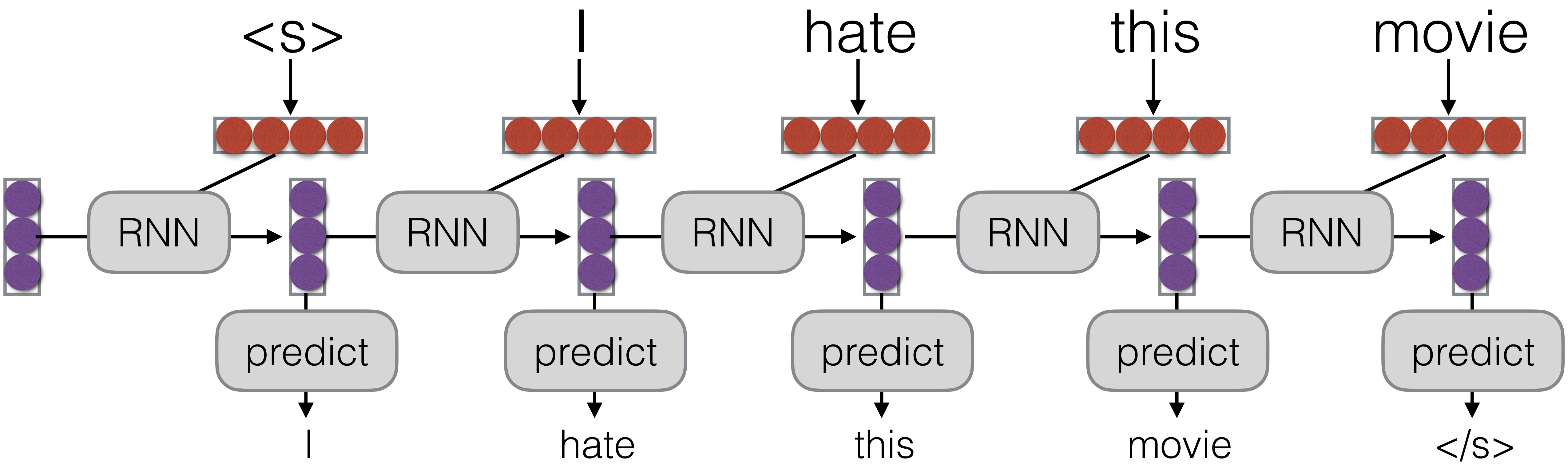
- Binary or multi-class prediction
- Sentence representation for retrieval, sentence comparison, etc.

Encoding Tokens



- Sequence labeling
- Language Modeling

e.g. Language Modeling



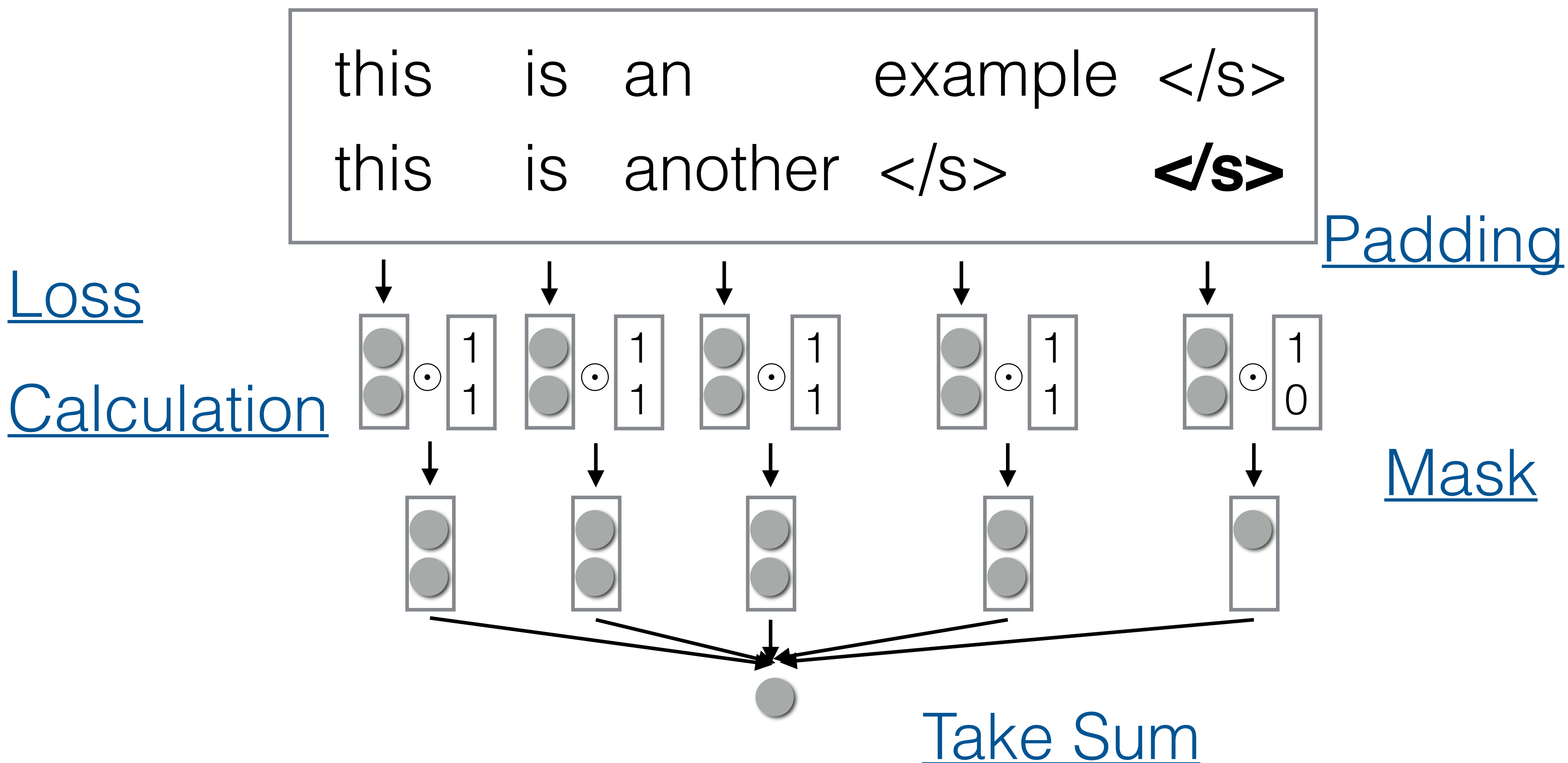
- Language modeling is like a tagging task, where each tag is the next word!
- Note: this is an autoregressive model

Efficiency Tricks for Sequence Modeling

Handling Mini-batching

- Mini-batching makes things much faster!
- But mini-batching in sequence modeling is harder than in feed-forward networks
    - Each word depends on the previous word
    - Sequences are of various length

Mini-batching Method

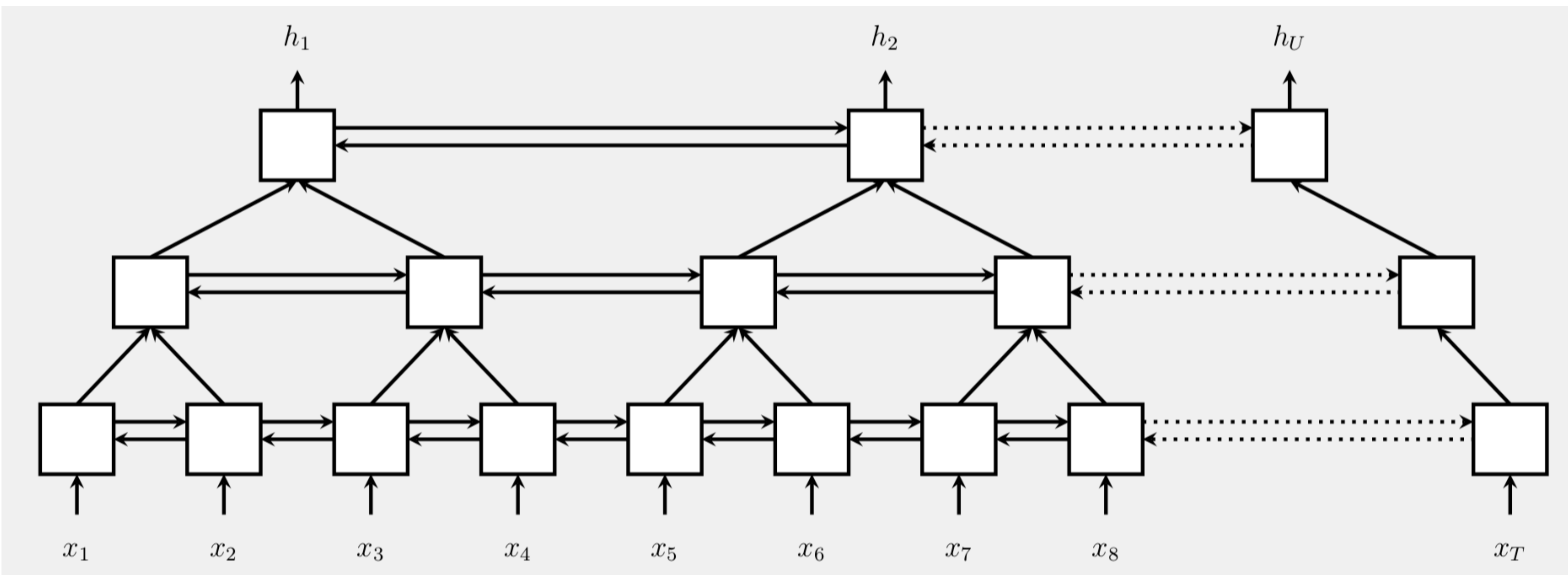


Bucketing/Sorting

- If we use sentences of different lengths, too much padding and sorting can **result in decreased performance**
- To remedy this: **sort sentences** so similarly-lengthed sentences are in the same batch

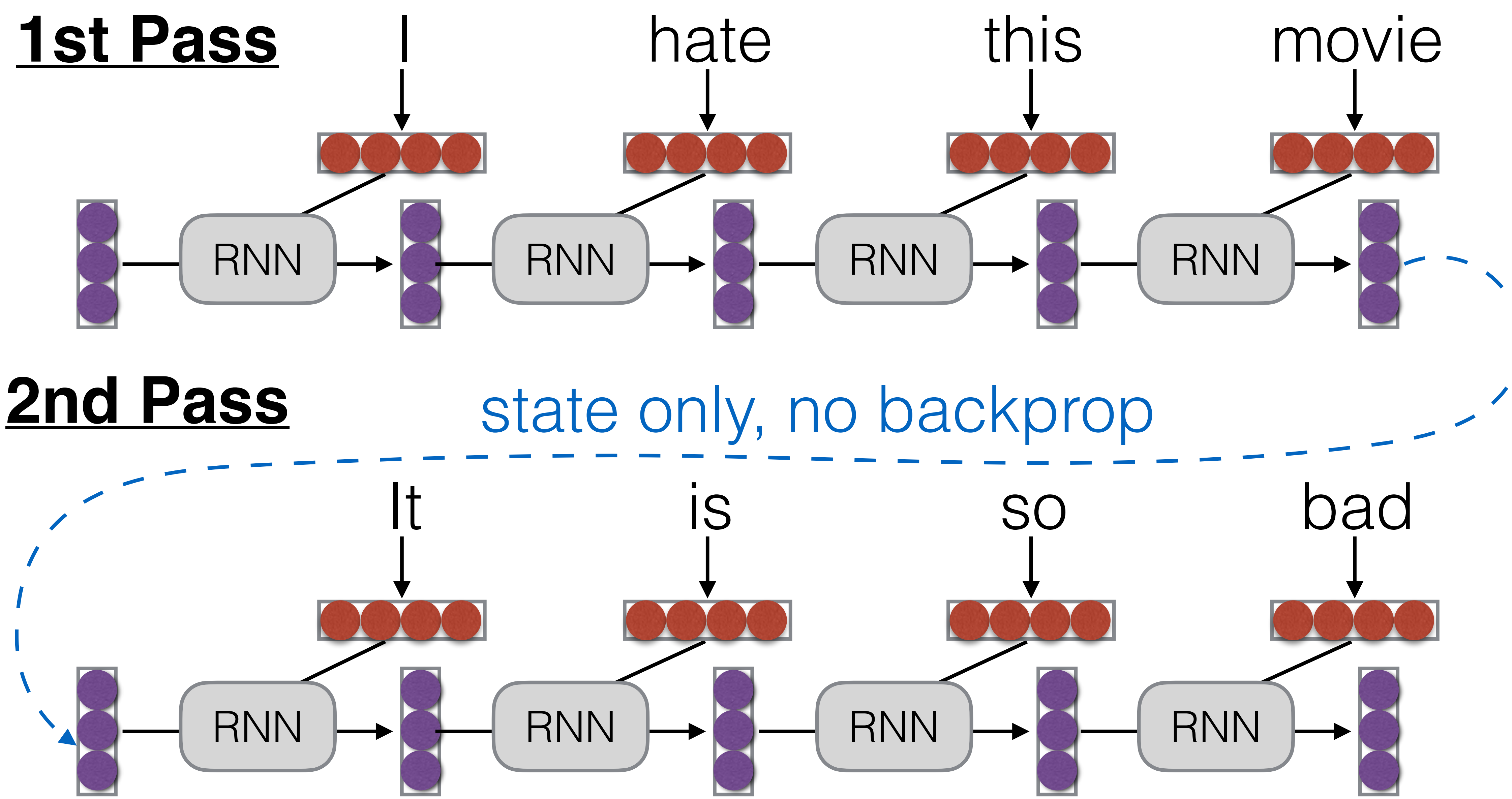
Strided Architectures
(e.g. Chan et al. 2015)

- Downscale between layers



Truncated BPTT

- Backprop over shorter segments, initialize w/ the state from the previous segment



Questions?