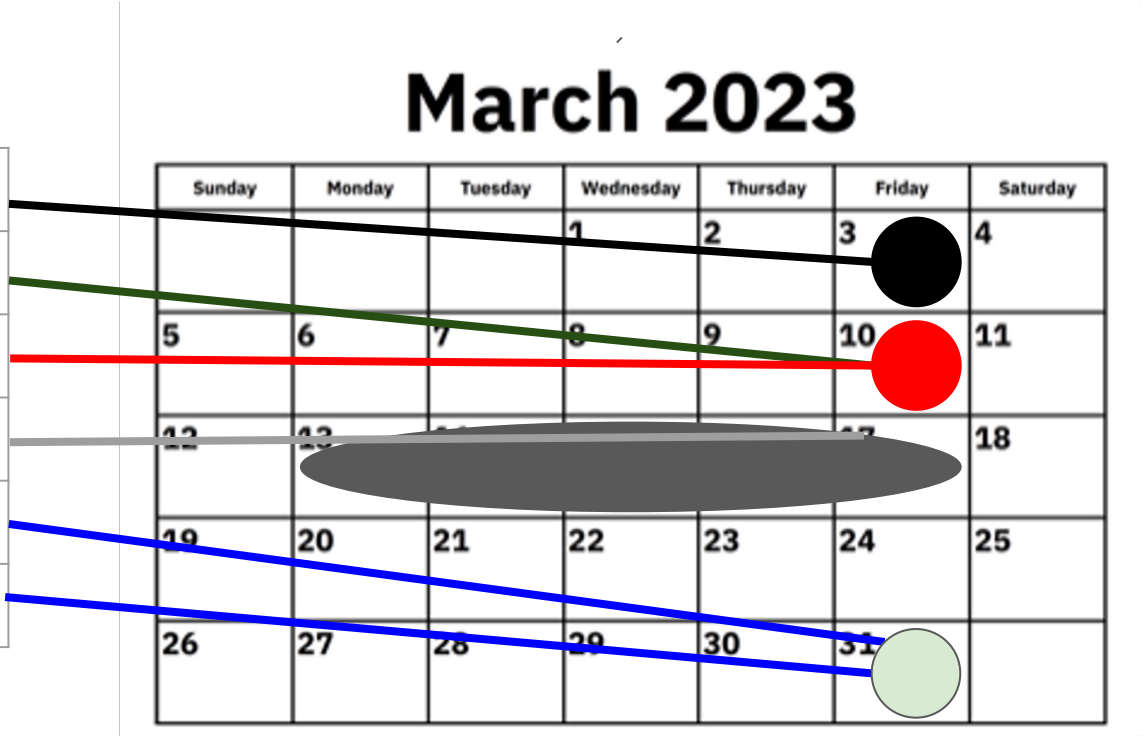
CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 7: Transformers and DL for Vision

[Mar 10] March Deliverables

Project Pitch	Mar 3
Assignment 2 OUT	Mar 10
Project Survey Report	Mar 10
NO CLASS	Mar 17
Assignment 2 Due	*Mar 31*
Project Midterm Report	Mar 31



Survey Reports Due Today, 11:59pm

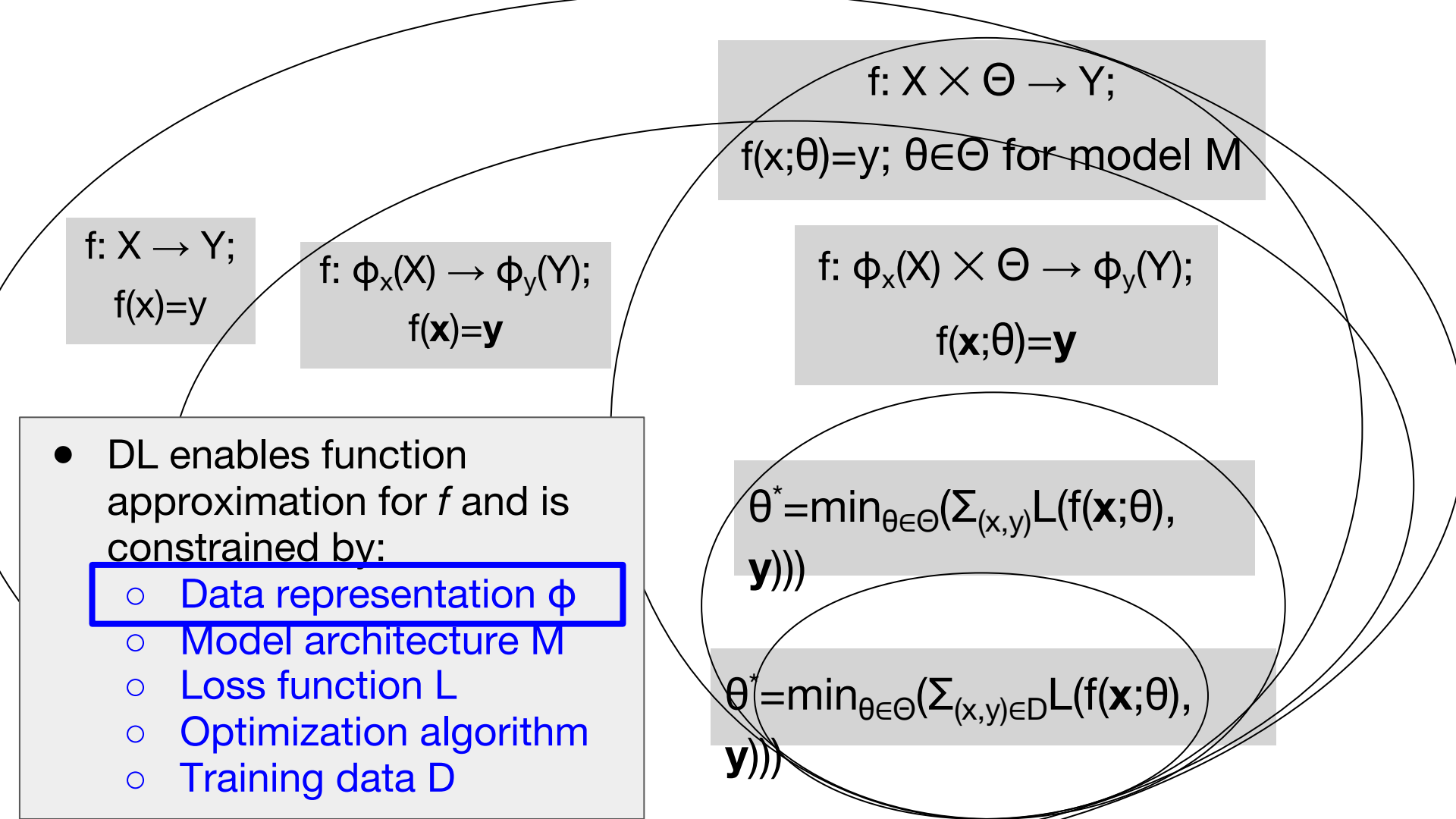
- A literature survey (e.g., "Related Work" section in a conference paper)
- (1) What has been done related to your proposal?
  - This may comprise multiple paragraphs/sections of different related areas and efforts
- (2) What are the limitations or challenges remaining to be solved?
  - What are the open questions? At least one of these should be what your project aims to address, though you should highlight others too.
- Format: a 1.5-2 page (double-column) literature review write-up.
- Additionally, a single page summary of contributions from each team member on your project.
  - Spell out your individual contributions clearly on that final page.

Overview of Today's Plan

- ~~Course organization and deliverables~~
  - Any questions before we move on?
- Transformers
- DL for Computer Vision
- Midterm Debrief

NLP FUNDAMENTALS: Language Models

- A *language model* is an estimate of the likelihood of a string
- Tokenizing the string into component parts is necessary
- So we can consider a string *s* as a sequence of tokens $w_1 \ldots w_n$
- Then a language model estimates:
  - $p(s) = p(w_1 \ldots w_n)$
- Term is overloaded; also used these days to mean:
  - $p(w_k | w_1 \ldots w_{k-1})$ — (e.g., *auto-regressive*)
  - $p(w_k | w_1 \ldots w_{k-1} w_{k+1} \ldots w_n)$ — (e.g., *cloze / MLM*)


f: X → Y;
f(x)=y
f: $\phi_x(X) \rightarrow \phi_y(Y)$;
f(**x**)=**y**
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: $\phi_x(X) \times \Theta \rightarrow \phi_y(Y)$;
f(**x**;θ)=**y**
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y}))$
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y}))$
• DL enables function approximation for *f* and is constrained by:
○ Data representation ϕ
○ Model architecture M
○ Loss function L
○ Optimization algorithm
○ Training data D

NLP FUNDAMENTALS: Word Embeddings

- A word embedding is a *vector representation* of a word
- Word embeddings can be learned for specific tasks (e.g., our book classification task) or from self-supervised objectives
- **“You shall know a word by the company it keeps” - Firth**
- The guiding principle of many learned word embeddings is that two words who share similar *context* should have embeddings that are close together in vector space

J.R. Firth. *A synopsis of linguistic theory*. Studies in linguistic analysis, Blackwell, Oxford (1957)

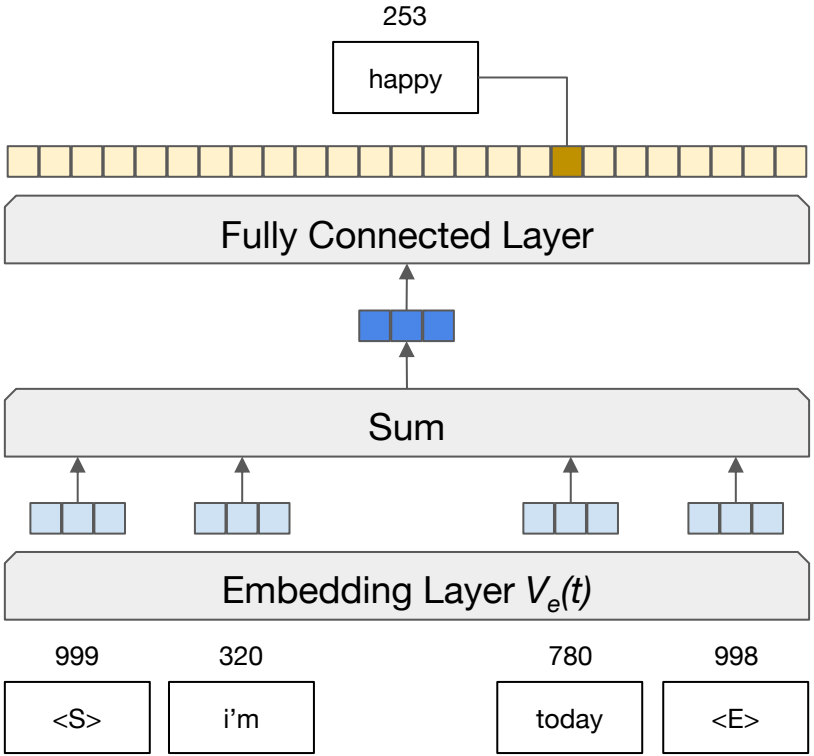
N-gram language models

- How could we *efficiently* estimate $p(w_n|w_1 \ldots w_{n-1})$?
  - Independence assumptions!
- Does a word *k* away in the history really affect what word comes next that much? What if we assume:
  - $p(w_n|w_1 \ldots w_{n-1}) \approx p(w_n|w_{n-k+1} \ldots w_{n-1})$
- Then if we say “a word 4 away in the history does not affect me”, then $p(w_n|w_1 \ldots w_{n-1}) \approx p(w_n|w_{n-3}w_{n-2}w_{n-1})$
- How big on disk?
  - $|V|*|V|*|V|*|V| = |V|^4$

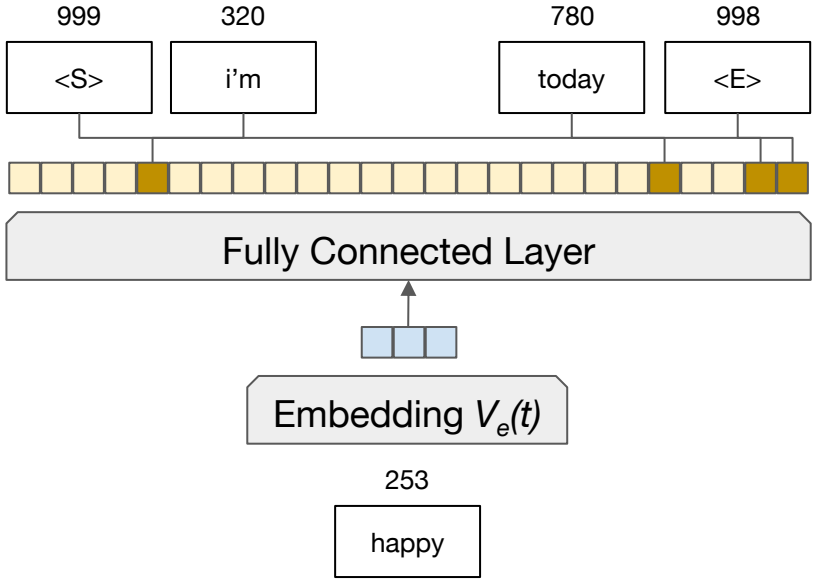
Bag-of-Words Model

- Estimate $p(w_i|w_{i-1}w_{i-2}\ldots w_{i-n+1})$ as distribution of w_i given that $w_{i-1}w_{i-2}\ldots w_{i-n+1}$ occur *in any order* left of w_i
  - Pros:
    - Space to store this model is $|V|^2$
    - Model will be considerably less sparse
  - Cons:
    - Loss of order information will hurt POS guessing / syntax generally
  - Still too sparse! Need to learn a *compressed representation*

Word2Vec: Recap

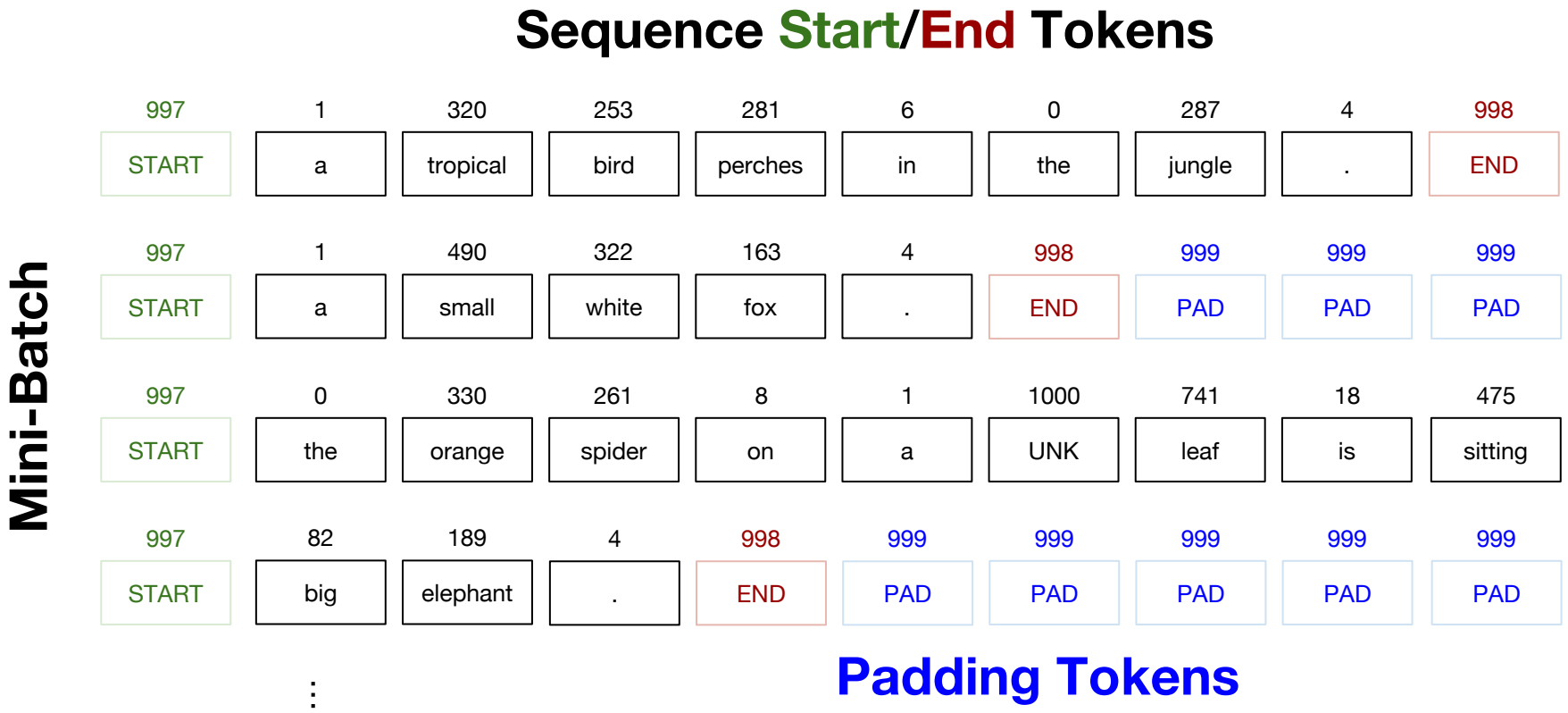


CBOW; $p(w_i|\{w_{i-2}w_{i-1}w_{i+1}w_{i+2}\})$

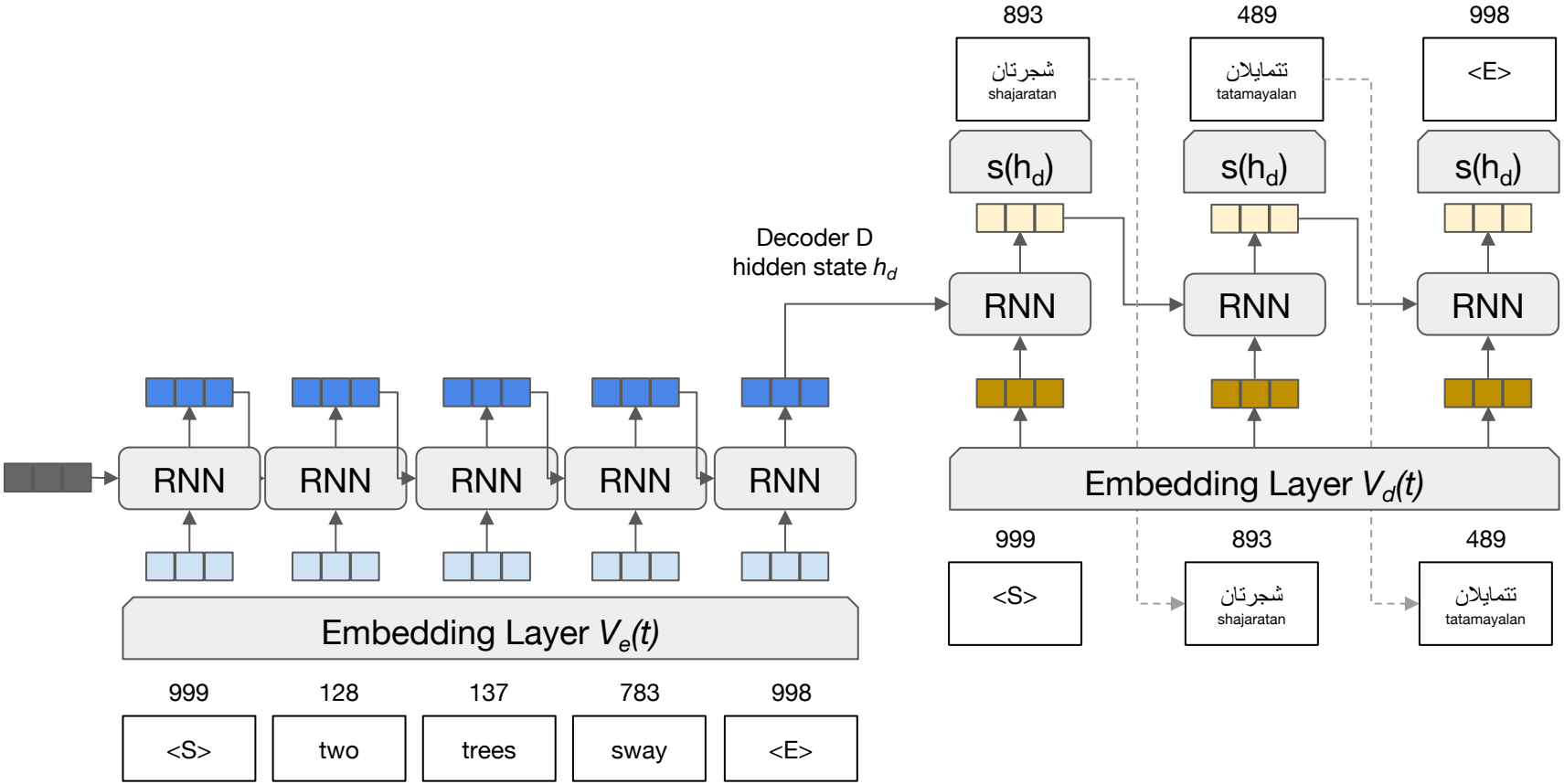


Skip-Gram;
$p(w_{i-2}|w_i)p(w_{i-1}|w_i)p(w_{i+1}|w_i)p(w_{i+2}|w_i)$

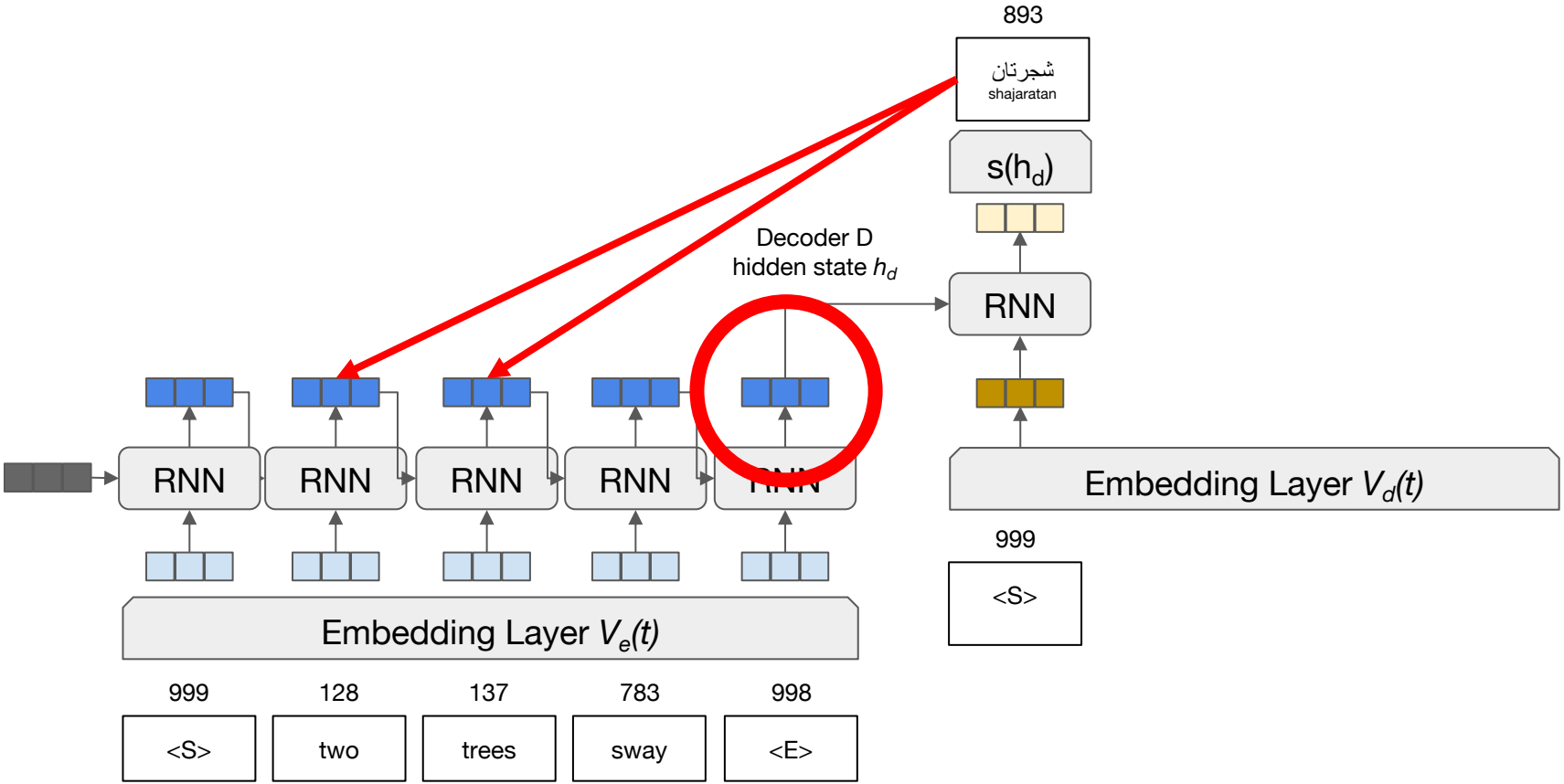
Tokenization and Batching



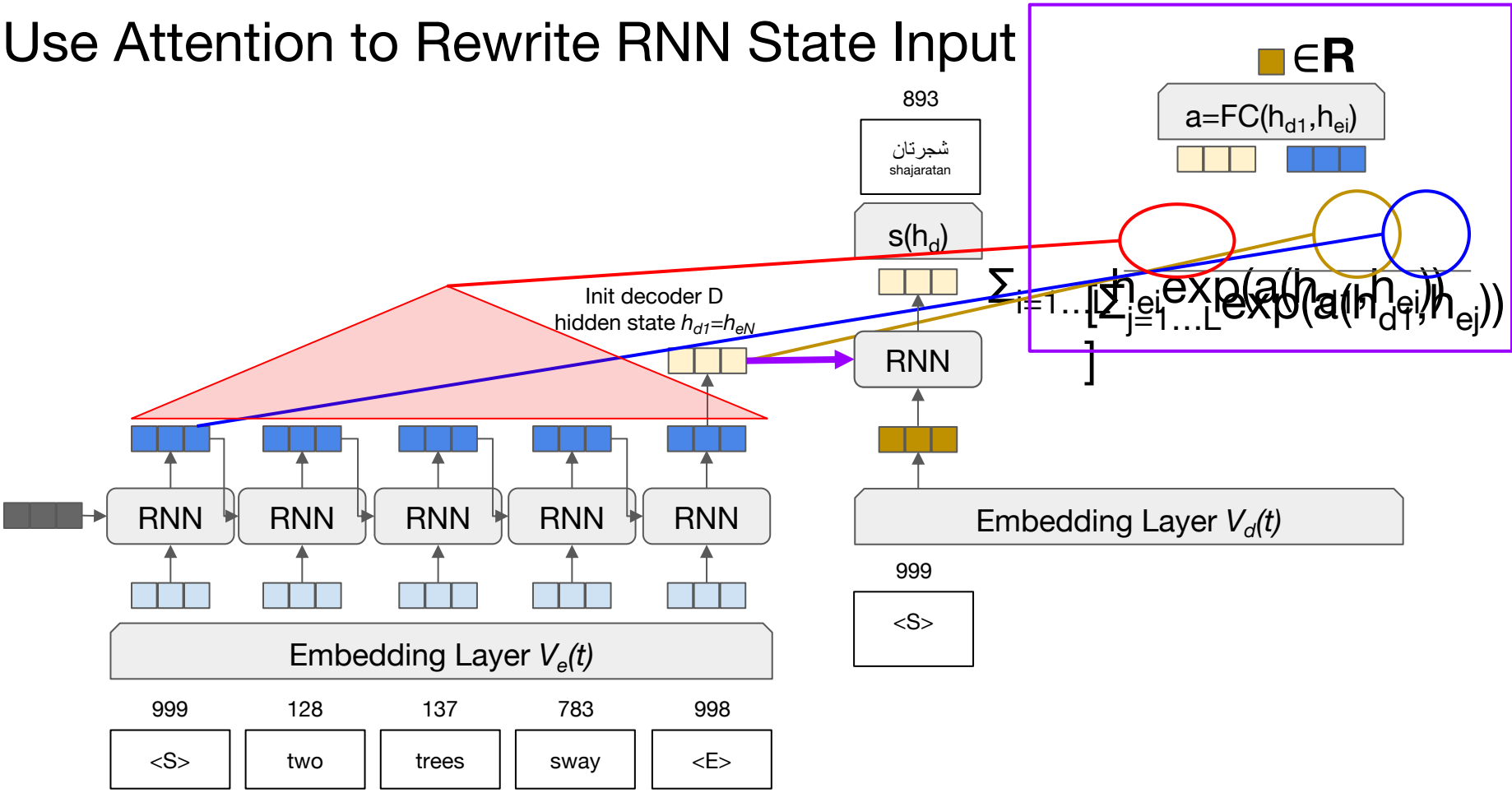
Machine Translation and The Case for Contextual Info



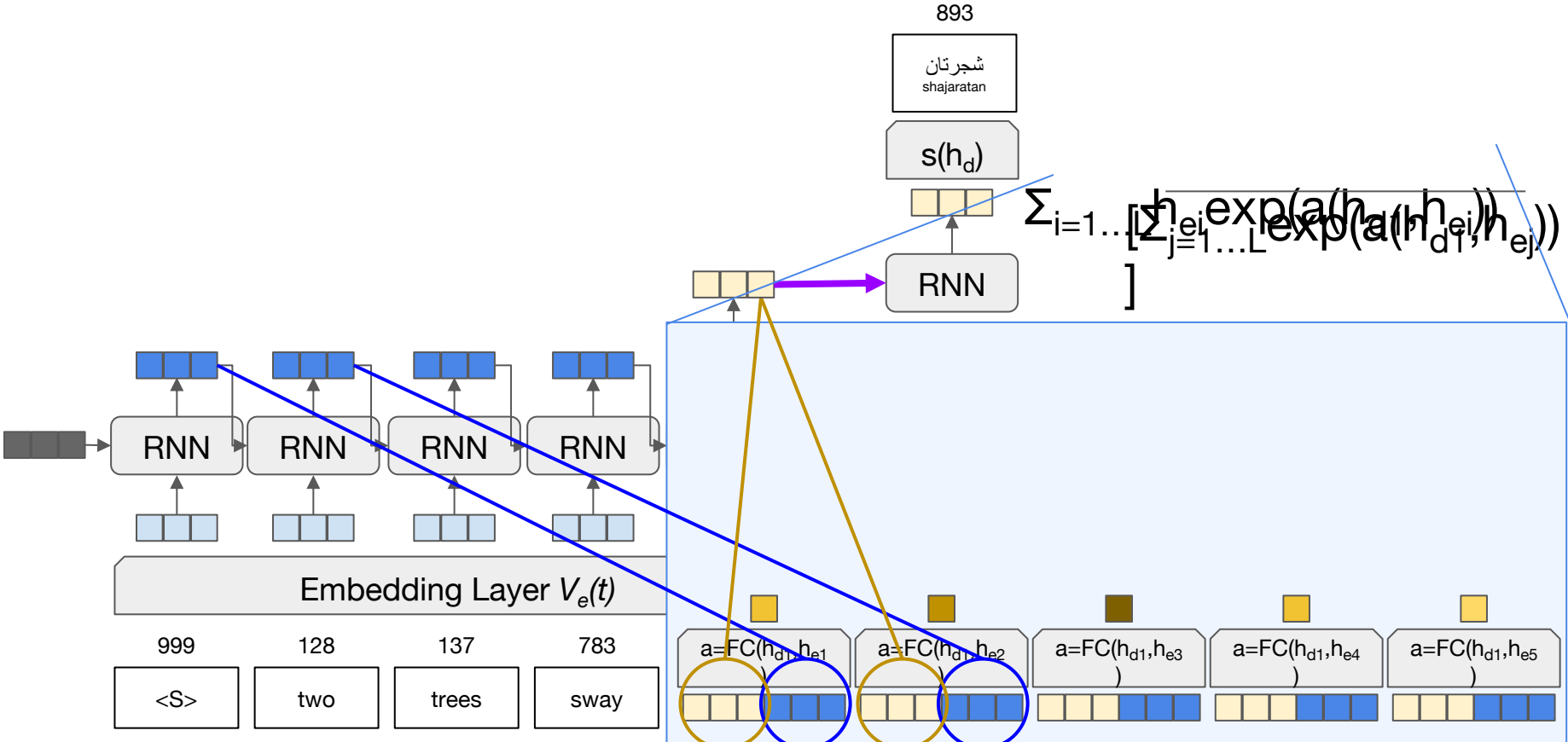
Machine Translation and The Case for Contextual Info



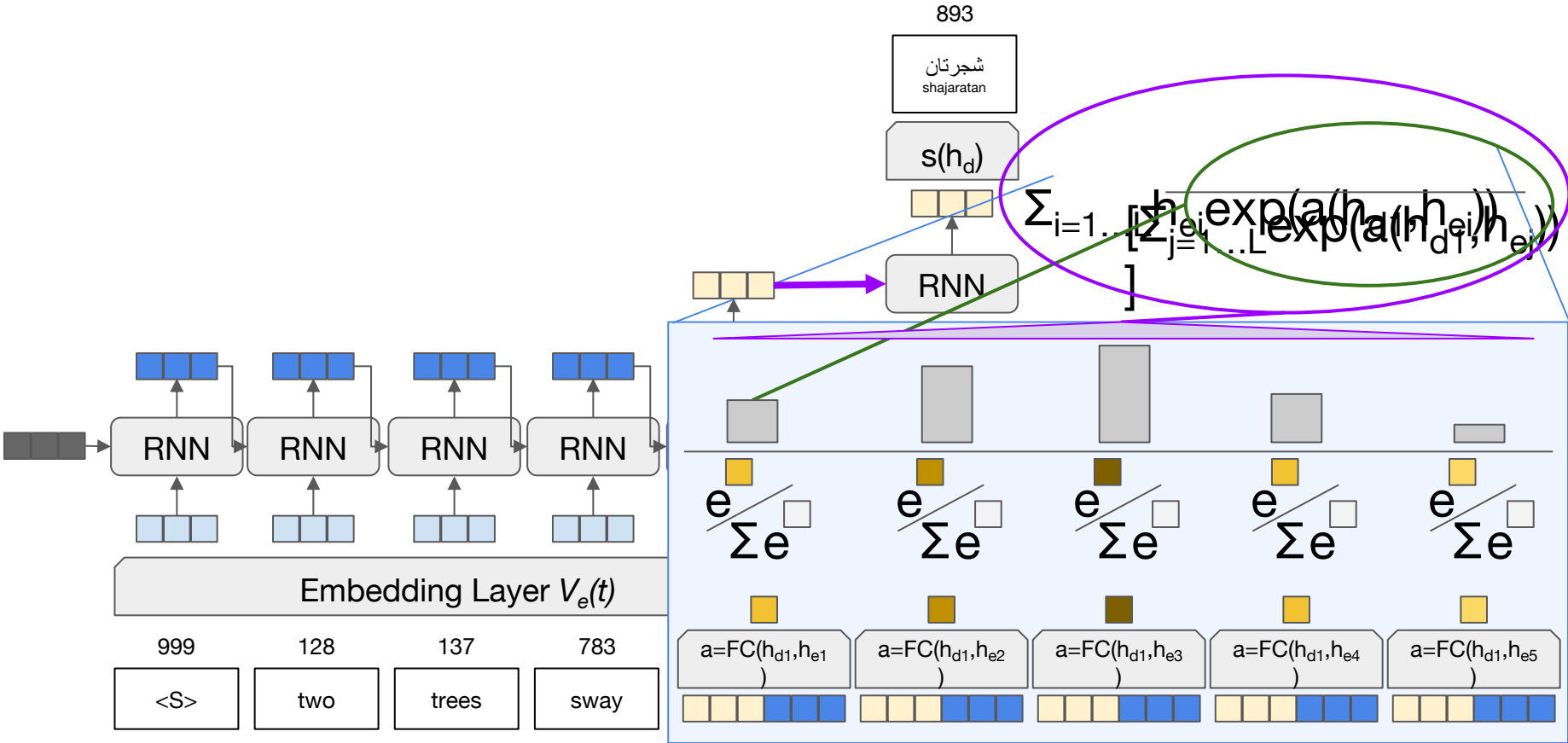
Use Attention to Rewrite RNN State Input



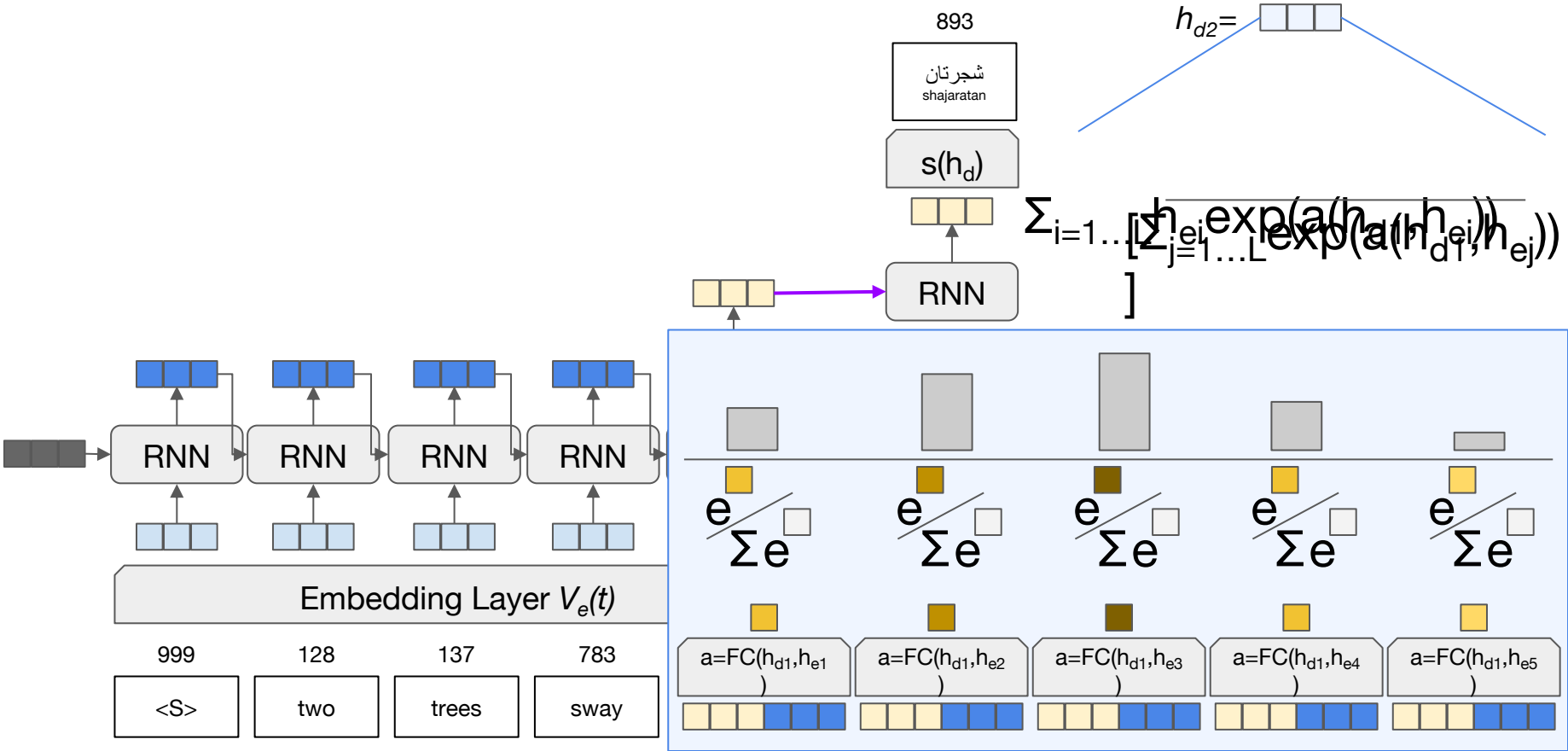
Attention


893
شجرتان
shajaratan
s(h_d)
RNN
Σ_i=1…L [exp(a(h_d1, h_ei)) / Σ_j=1…L exp(a(h_d1, h_ej))]
RNN
RNN
RNN
RNN
Embedding Layer V_e(t)
999
128
137
783
<S>
two
trees
sway
a=FC(h_d1,h_e1)
a=FC(h_d1,h_e2)
a=FC(h_d1,h_e3)
a=FC(h_d1,h_e4)
a=FC(h_d1,h_e5)

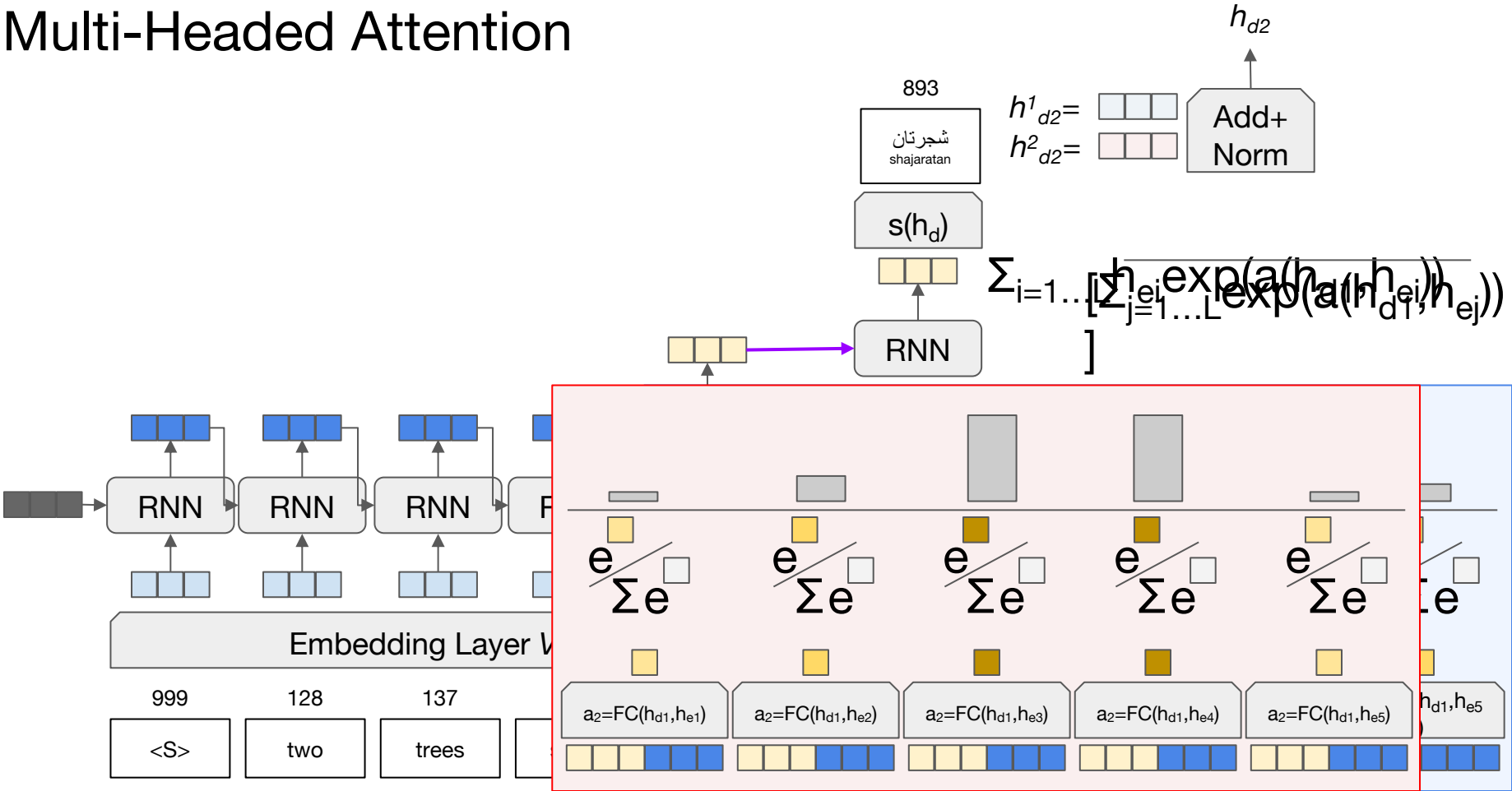
Attention


893
شجرتان
shajaratan
s(h_d)
RNN
RNN
RNN
RNN
RNN
Embedding Layer V_e(t)
999
128
137
783
<S>
two
trees
sway
a=FC(h_d1,h_e1)
a=FC(h_d1,h_e2)
a=FC(h_d1,h_e3)
a=FC(h_d1,h_e4)
a=FC(h_d1,h_e5)

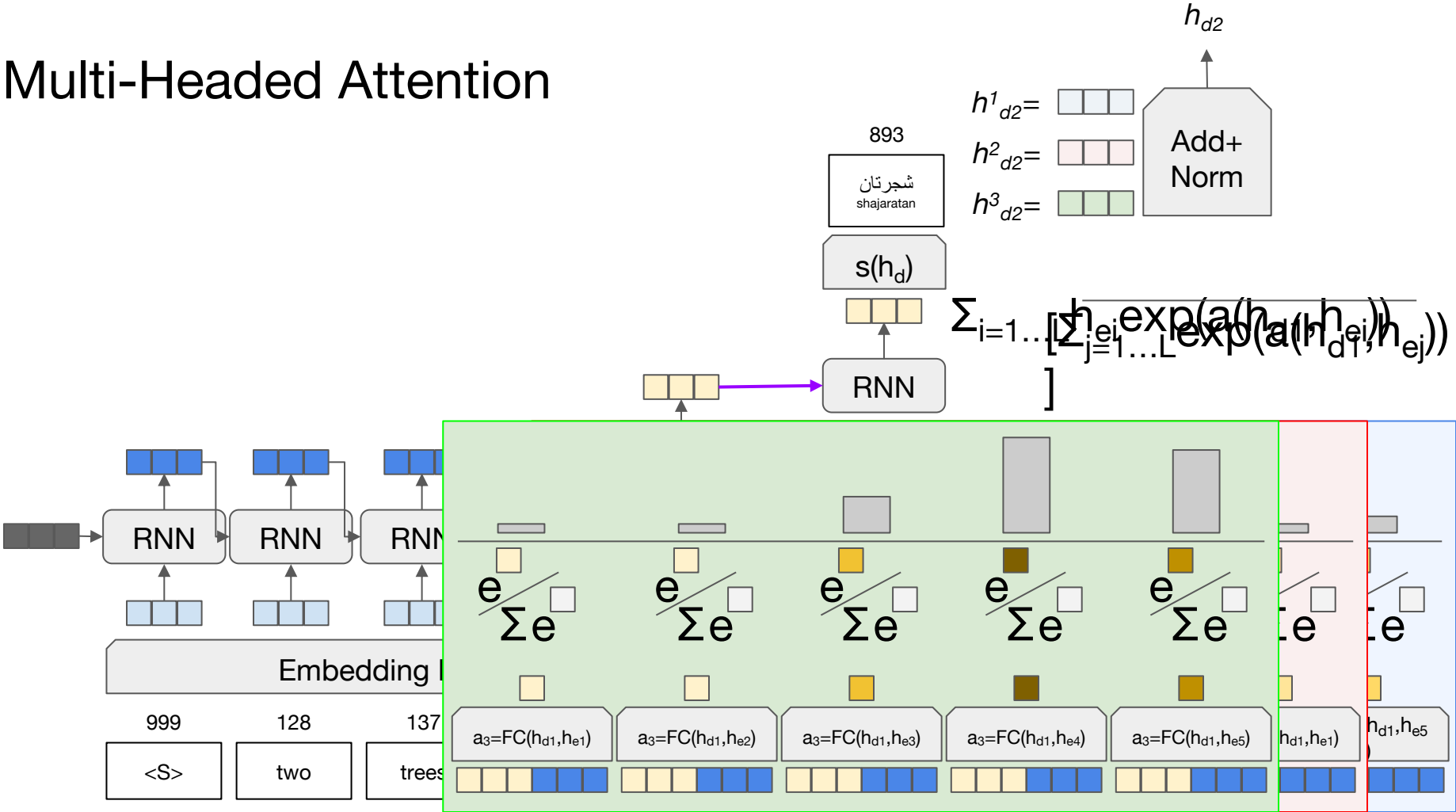
Multi-Headed Attention



Multi-Headed Attention



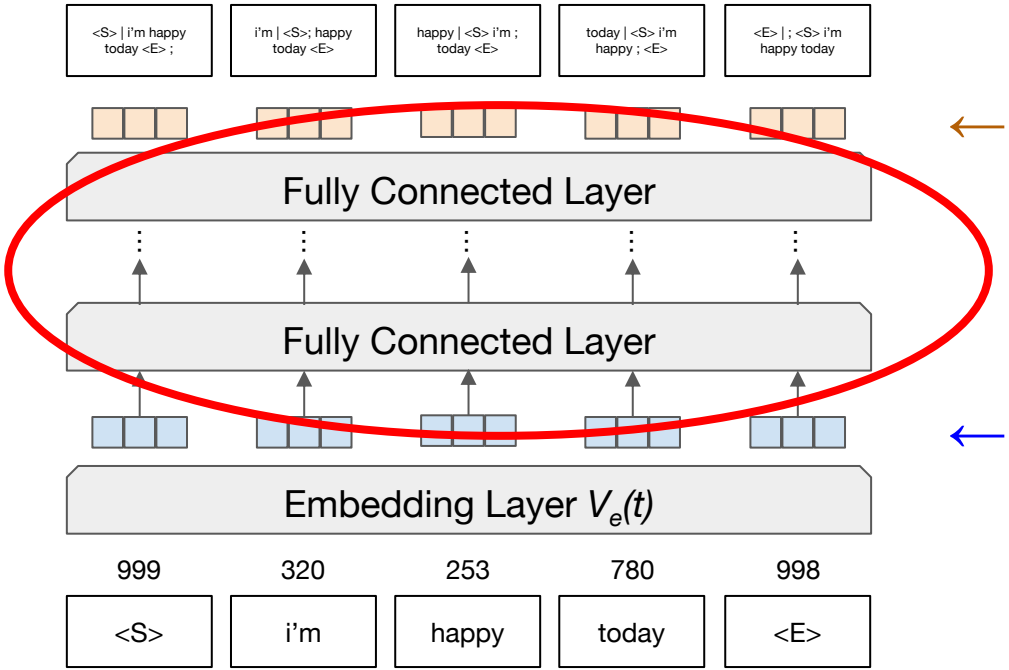
Multi-Headed Attention



Contextual Word Embeddings

- Consider the word vectors in the following sentences:
  - **X** = {“she launched into a bass solo”,
    “the line went taut as the bass pulled”;
    “the power of the bass broke the string”}
- Many words have multiple *senses*, or meanings; even worse for BPE tokens!
- Polysemy is a key weakness of so-called *lexical* embeddings
- We can instead learn a function that combines *input* lexical embeddings from each word to produce *output* **contextual** embeddings that consider surrounding words

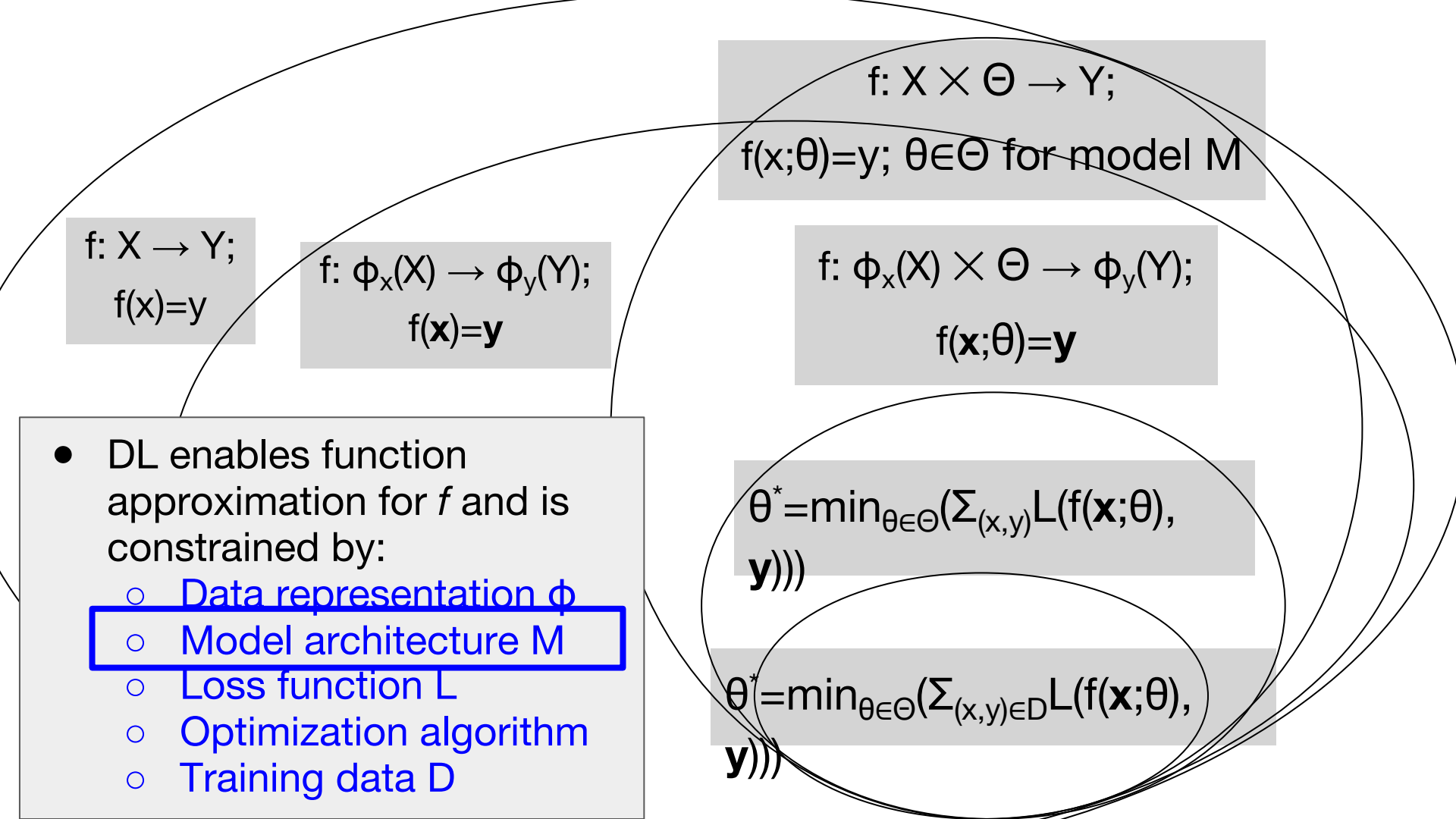
Contextual Word Embeddings



← **Contextual Embeddings**

← **FCs Intractable!!**

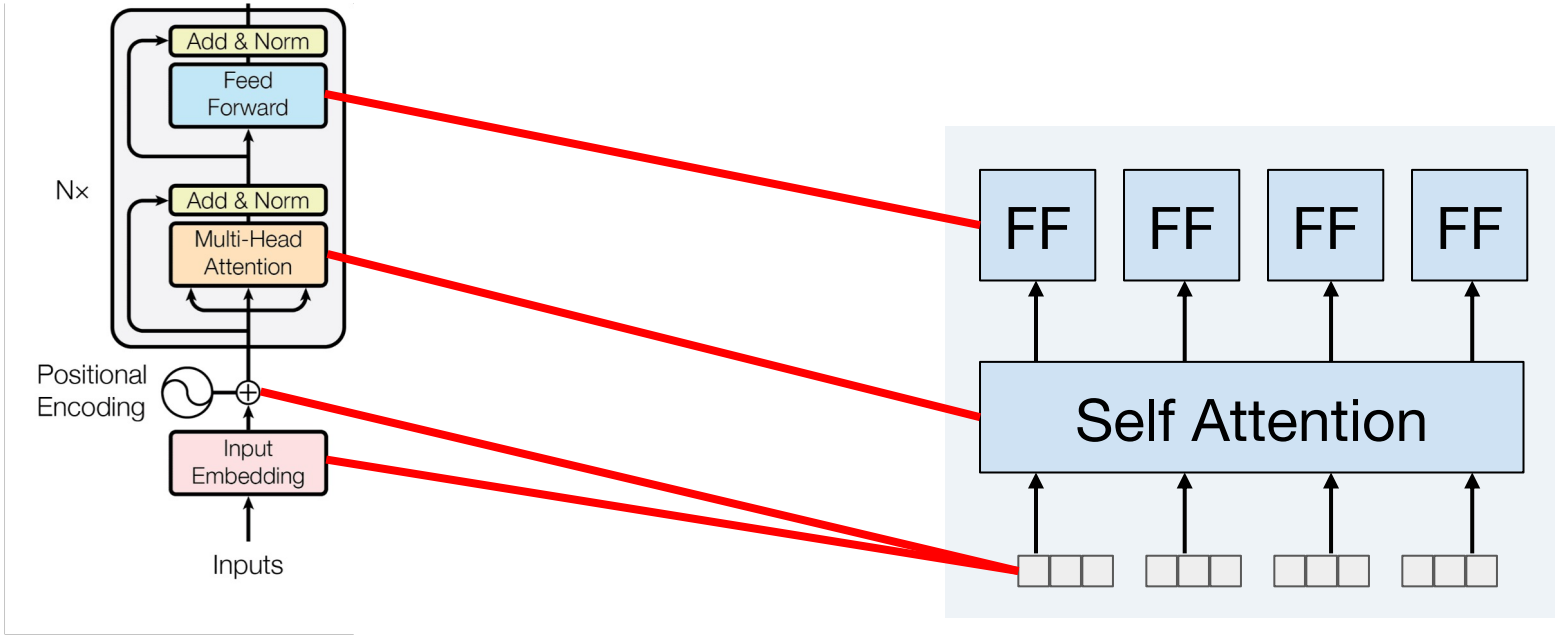
← **Lexical Embeddings**


f: X → Y;
f(x)=y
f: $\phi_x(X) \rightarrow \phi_y(Y)$;
f(**x**)=**y**
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: $\phi_x(X) \times \Theta \rightarrow \phi_y(Y)$;
f(**x**;θ)=**y**
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y}))$
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y}))$
• DL enables function approximation for *f* and is constrained by:
○ Data representation ϕ
○ Model architecture M
○ Loss function L
○ Optimization algorithm
○ Training data D

Transformers

- Transformer $T : S_a \longrightarrow S_b$ is a function from…
  - A set a in S_a to a set b in S_b
- Simple transformer consists of a single layer encoder E and a single layer decoder D with single-headed attention

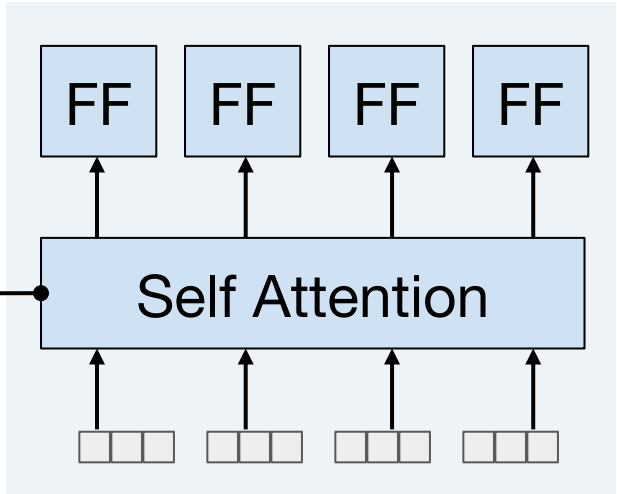
Transformers: Self-attention



Attention Is All You Need

Scaled dot-product attention

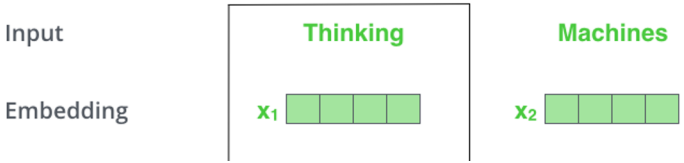
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{T}}}{\sqrt{d_k}}\right)V$$



Our input for *self-attention* is the matrix **X** which is the embeddings of the input sequence

So we call Attention(**X**, **X**, **X**)

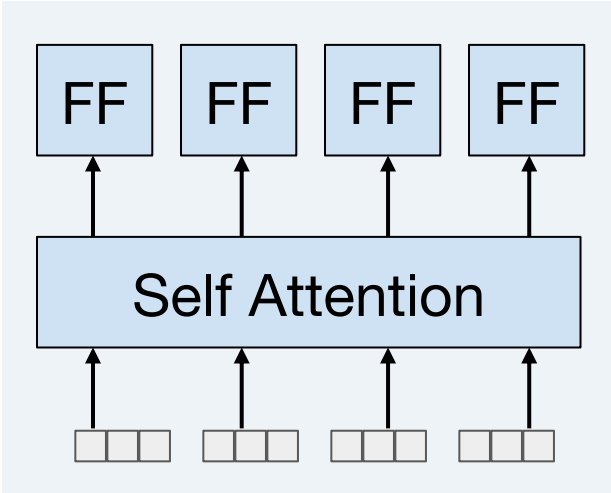
Self-Attention



Attention(**X**, **X**, **X**)

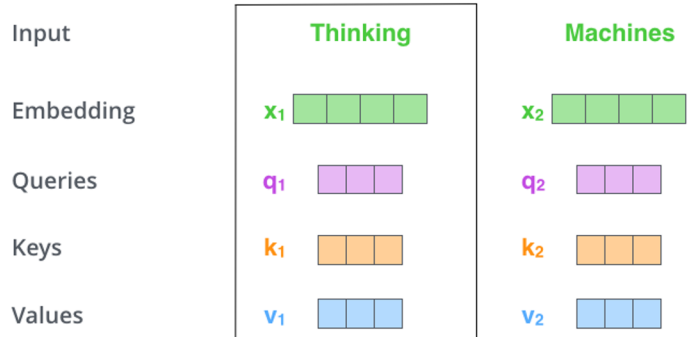
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$

$q_1 = x_1 W_Q \ldots$



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

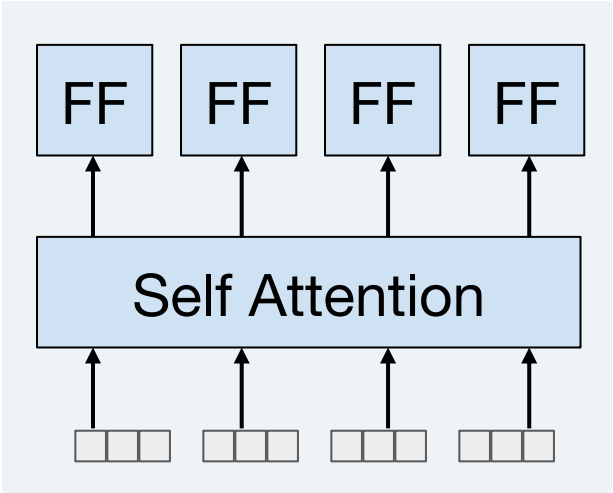
Self-Attention



$q_1 = x_1 W_Q \ldots$
$k_1 = x_1 W_K \ldots$
$v_1 = x_1 W_V \ldots$

Attention(**X**, **X**, **X**)

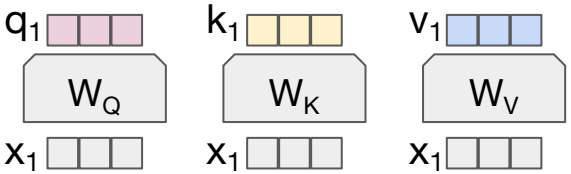
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



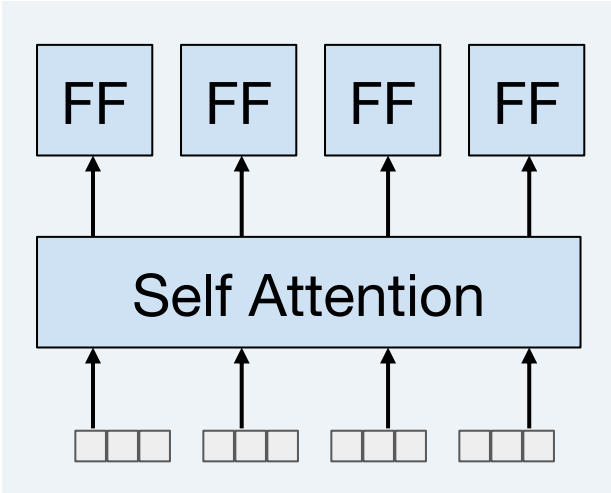
Learnable weights W_Q, W_K, W_V for each attention head.

Self-Attention

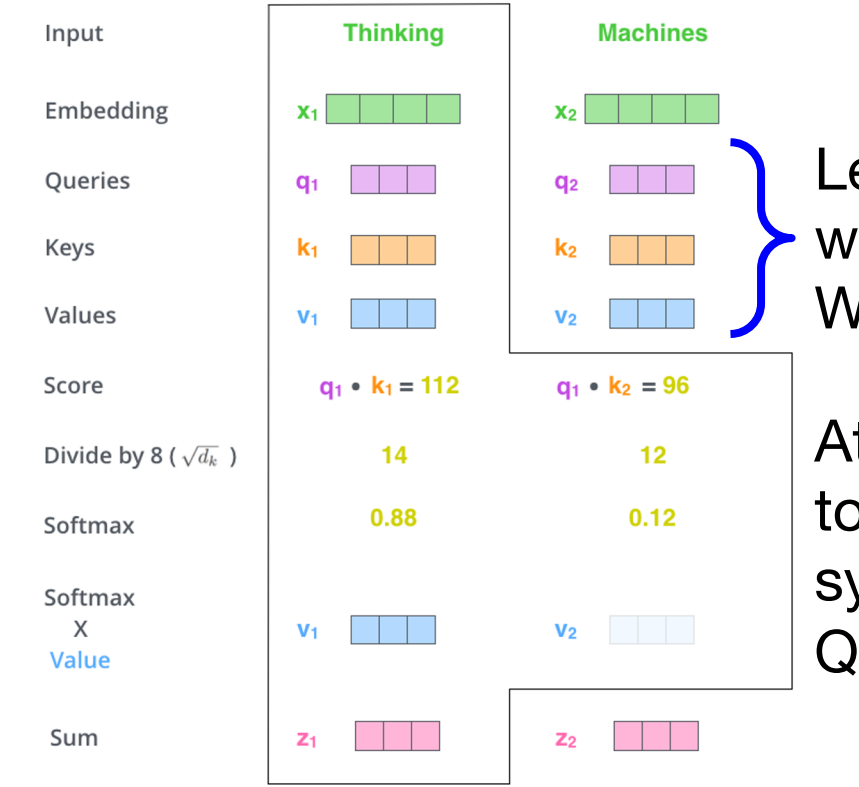
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{T}}}{\sqrt{d_k}}\right)V$$



x



Self-Attention

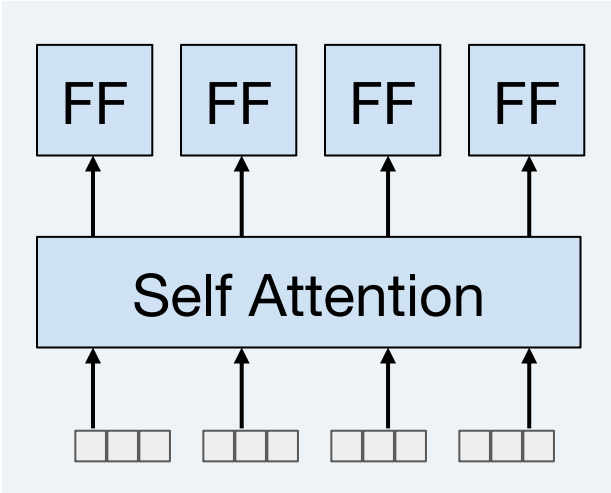


Learnable weights W_Q, W_K, W_V

Attention of x_1 to x_2 is non-symmetric via Q, K matrices

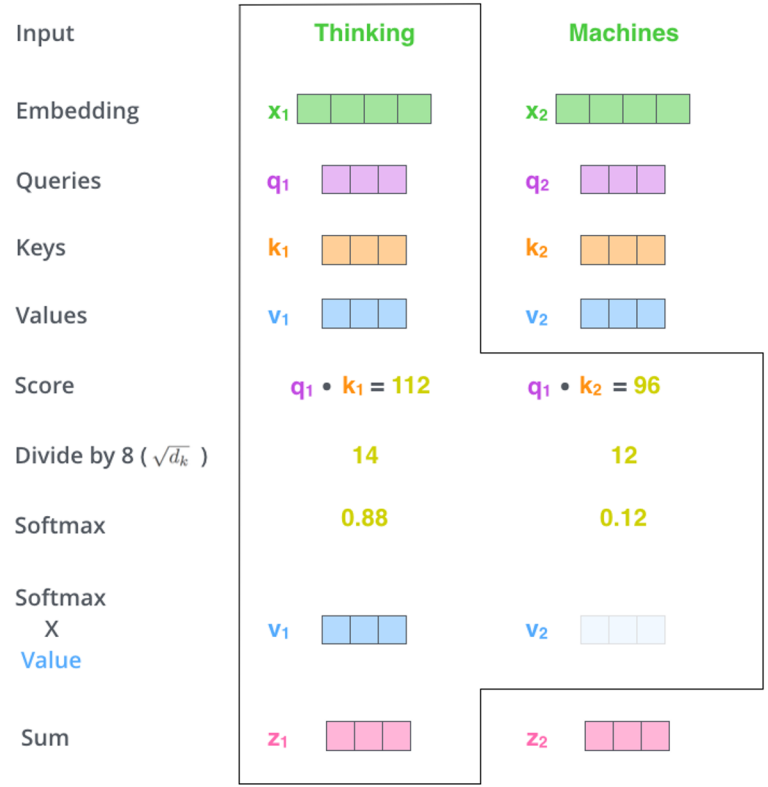
Attention(**X**, **X**, **X**)

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



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

Self-Attention

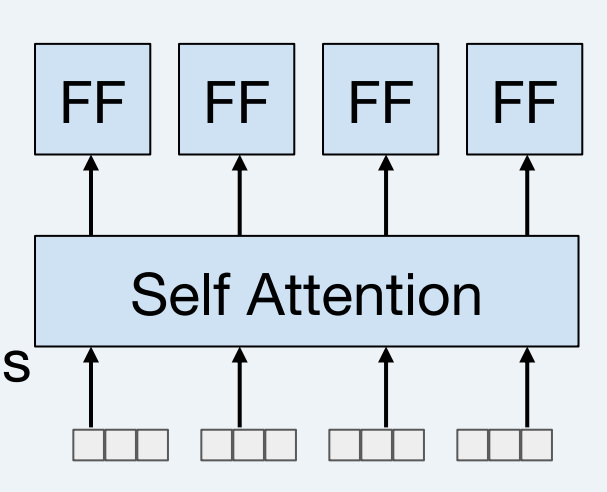


Attention(**X**, **X**, **X**)

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

Attention of x_1 to x_2 is non-symmetric via Q, K matrices

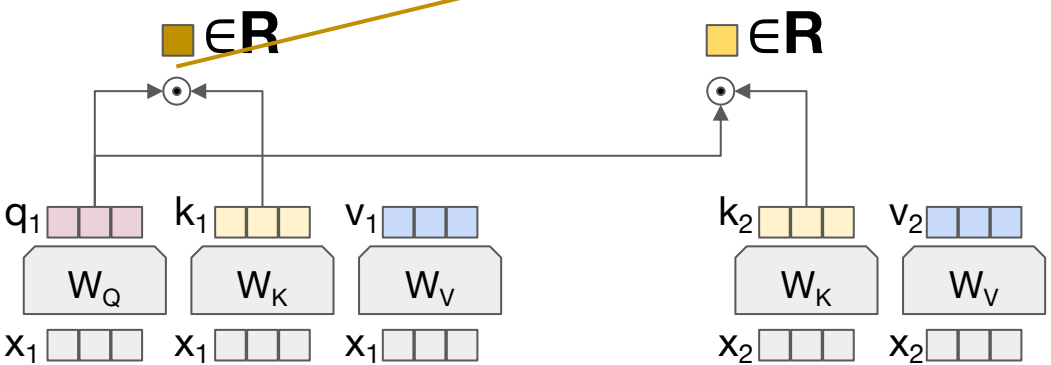
W_V matrix learns a “useful” output for each word’s context



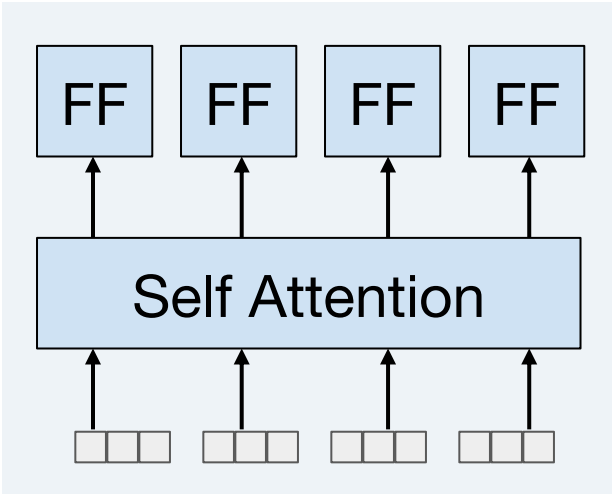
https://jalammar.github.io/illustrated-transformer/

Self-Attention

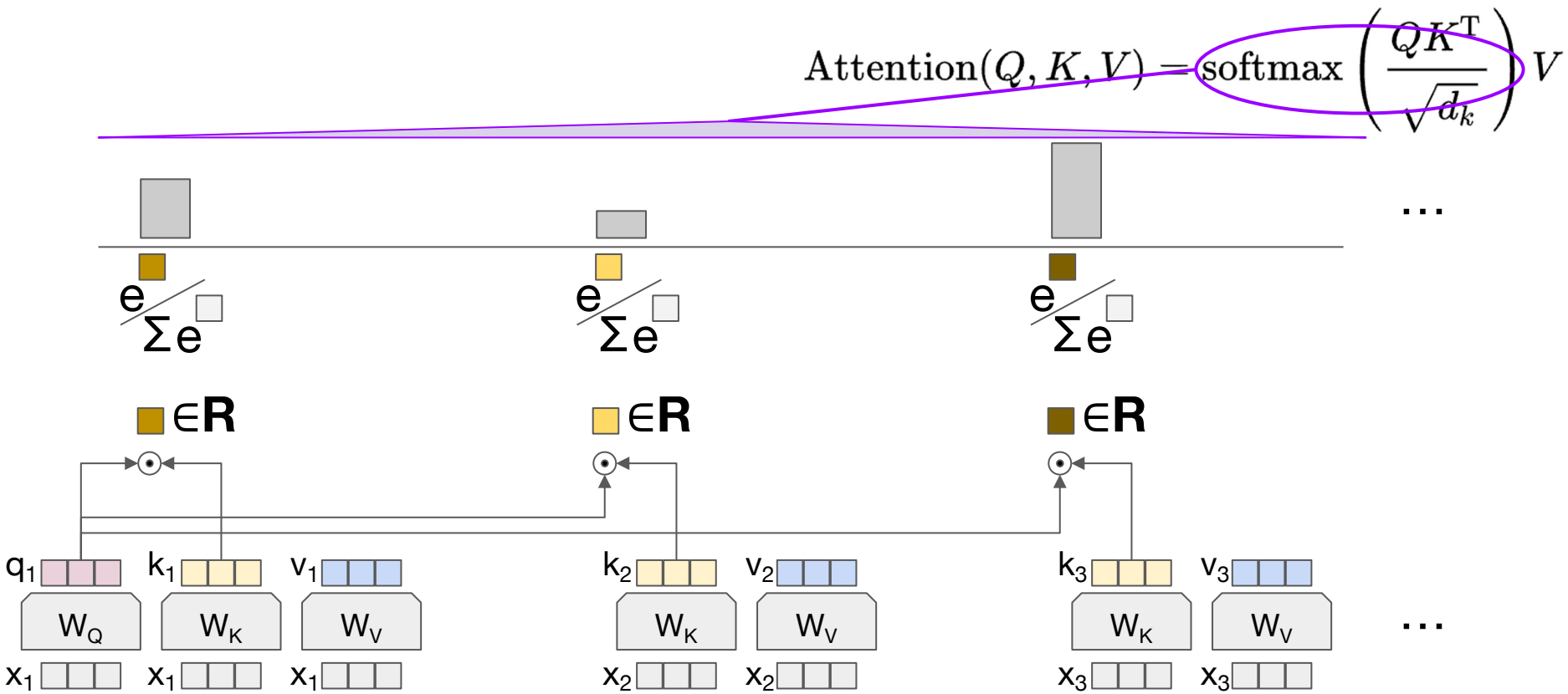
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



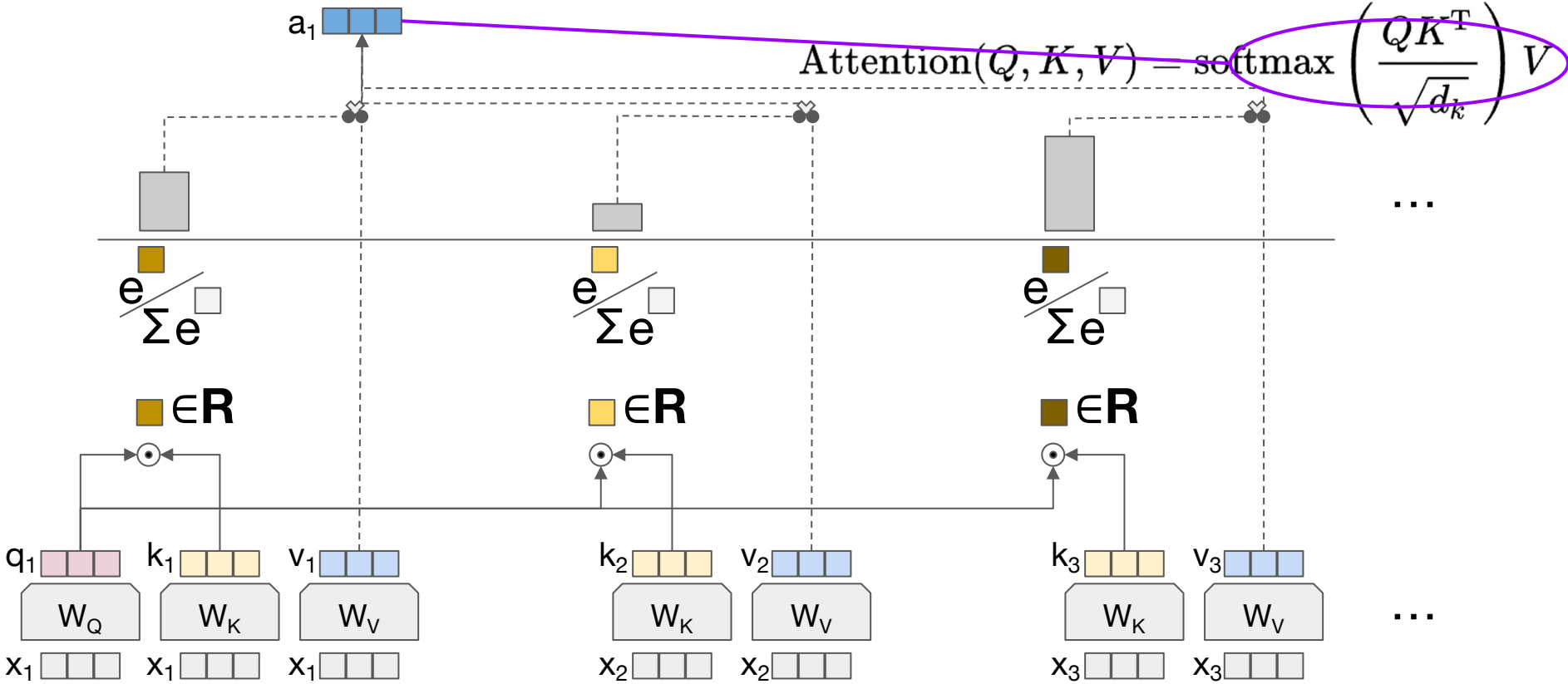
x



Self-Attention

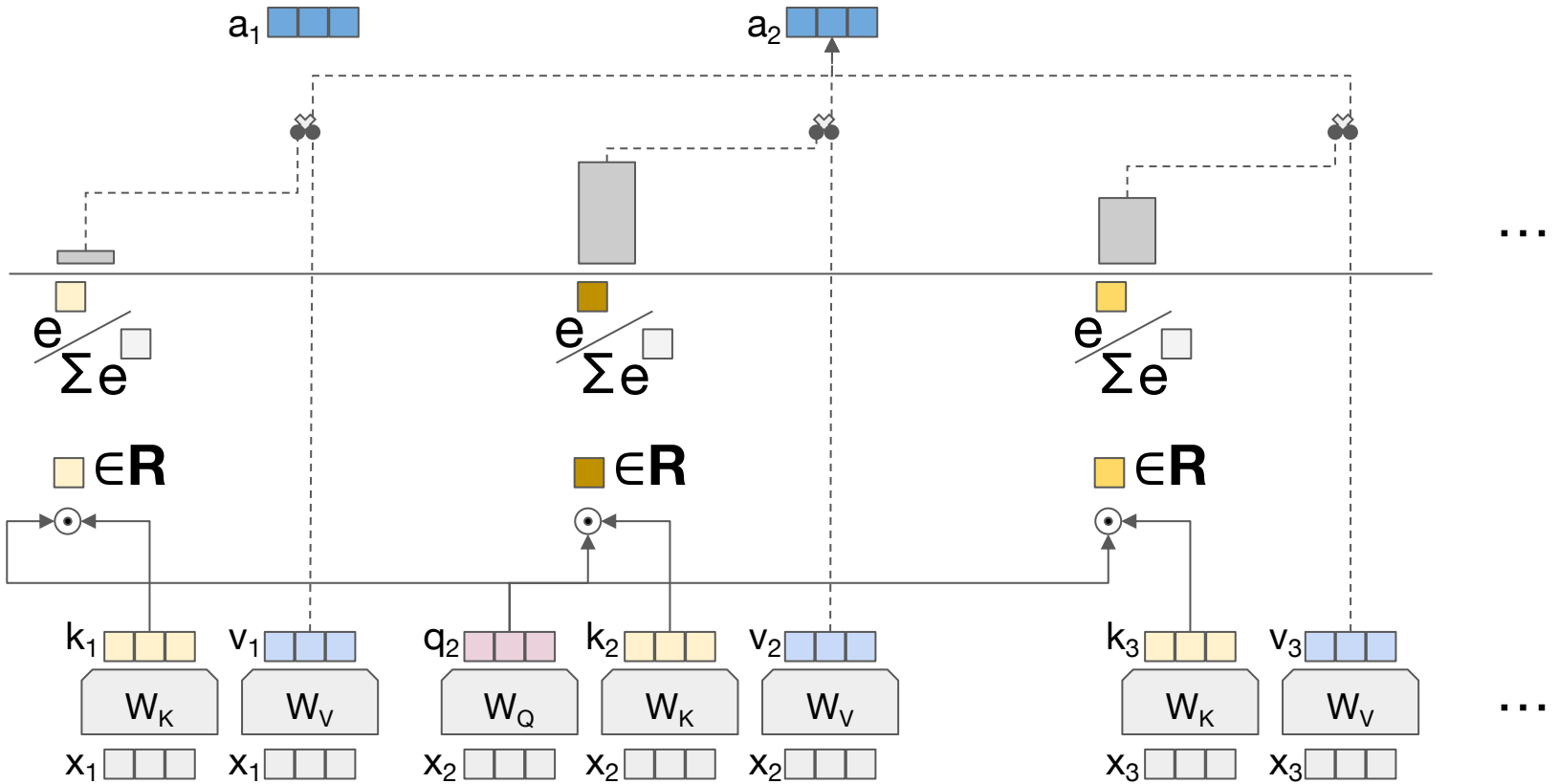


Self-Attention



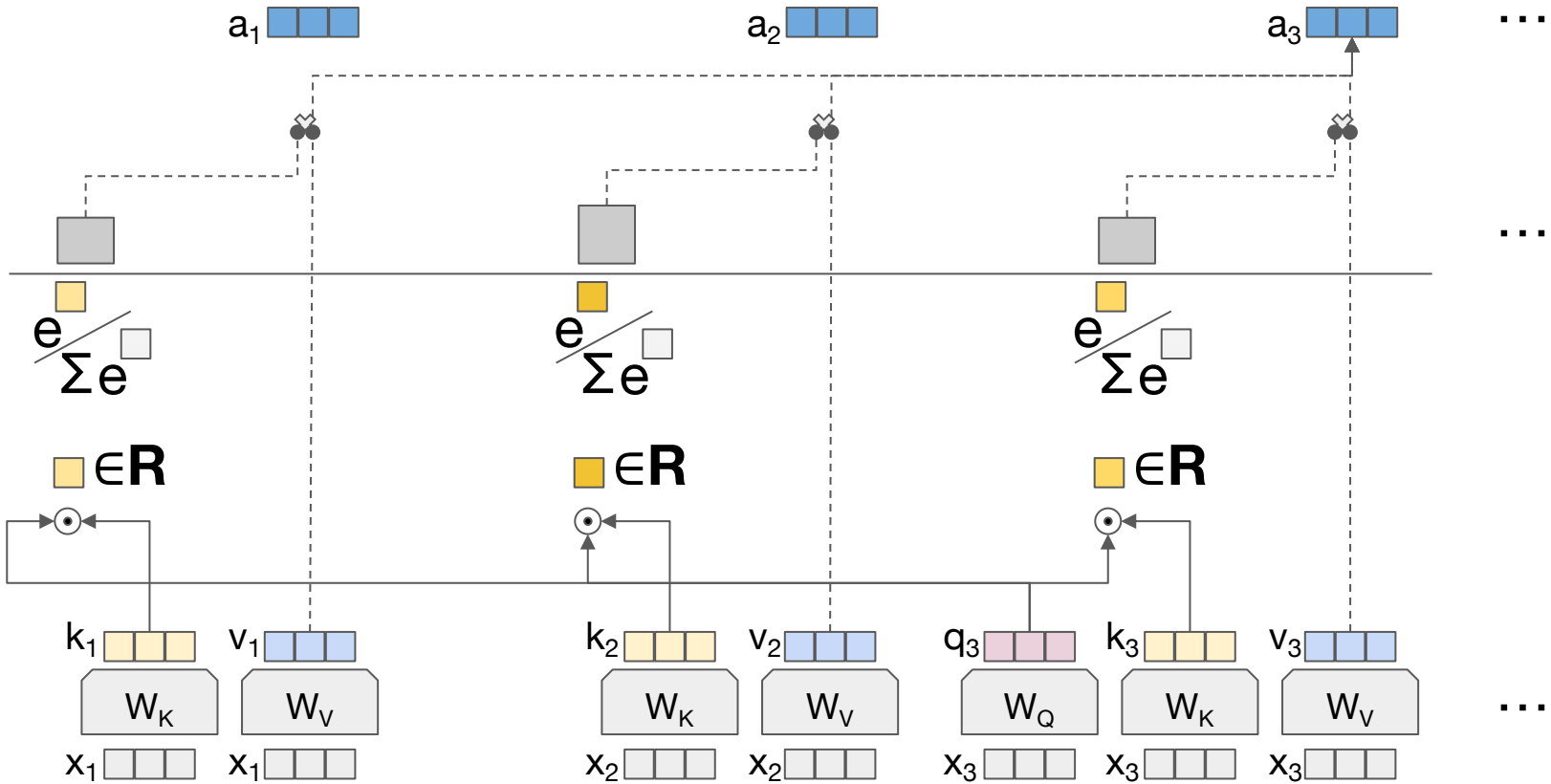
Self-Attention

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



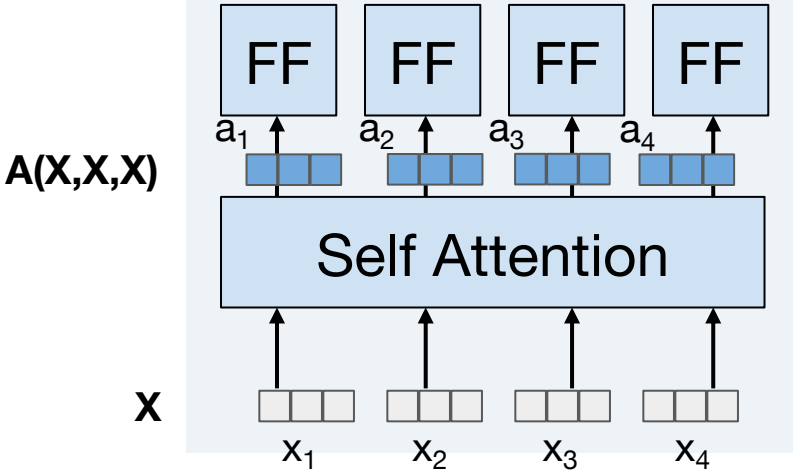
Self-Attention

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

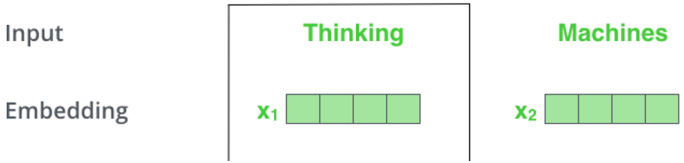


Self-Attention

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



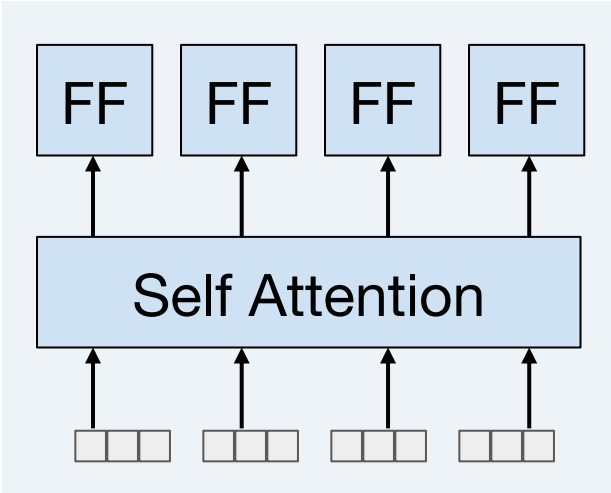
Self-Attention



Attention(**X**, **X**, **X**)

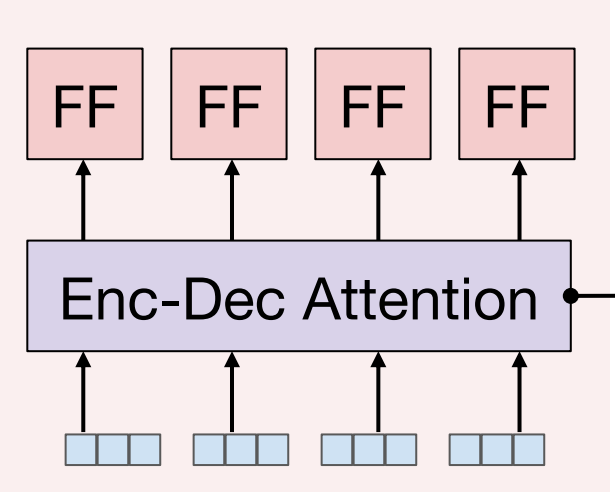
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$

$q_1 = x_1 W_Q \ldots$



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

Encoder-Decoder Attention



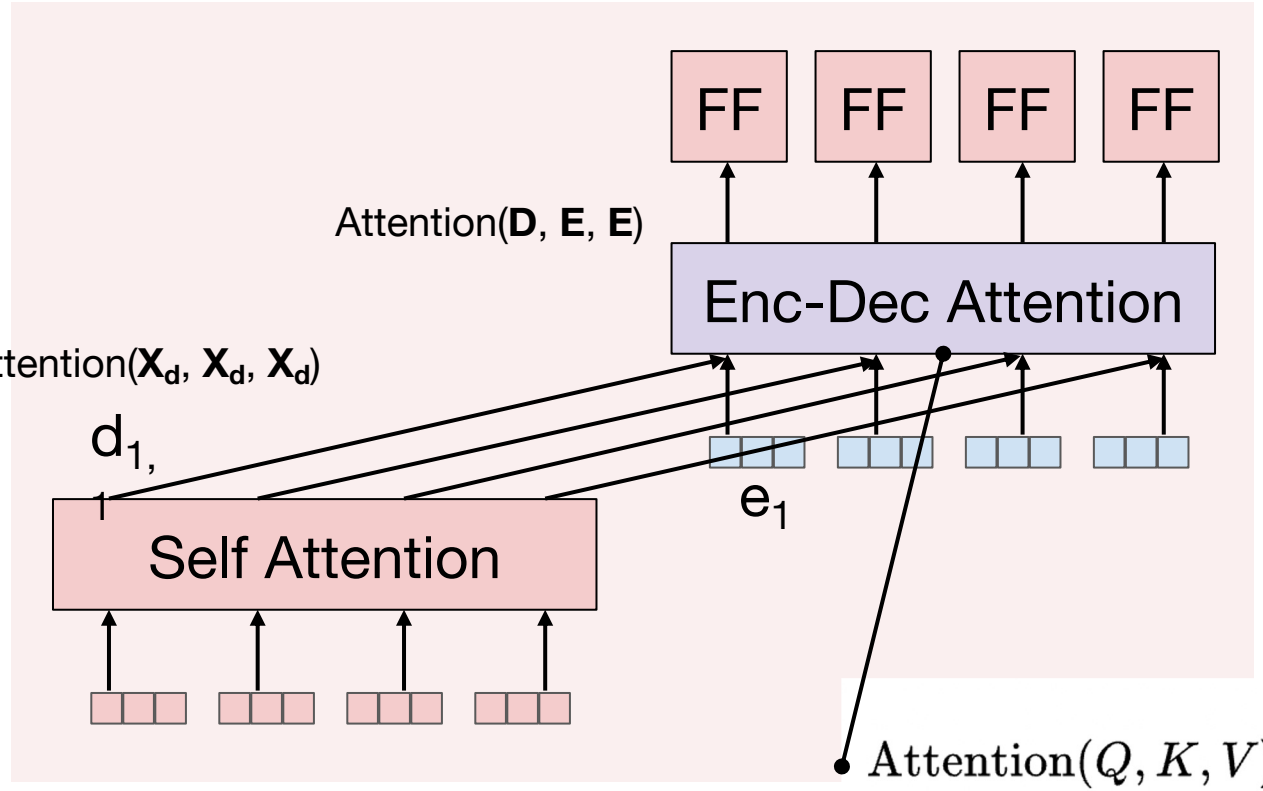
Call outputs from encoder self-attention **E**, decoder self-attention **D**

So we call Attention(**D**, **E**, **E**)

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

- Learnable weights W_Q, W_K, W_V for every attention layer in both encoder and decoder.
- Multi-headed attention: multiple learnable **W** matrices per encoder/decoder layer, all added/softmax before FF layers.

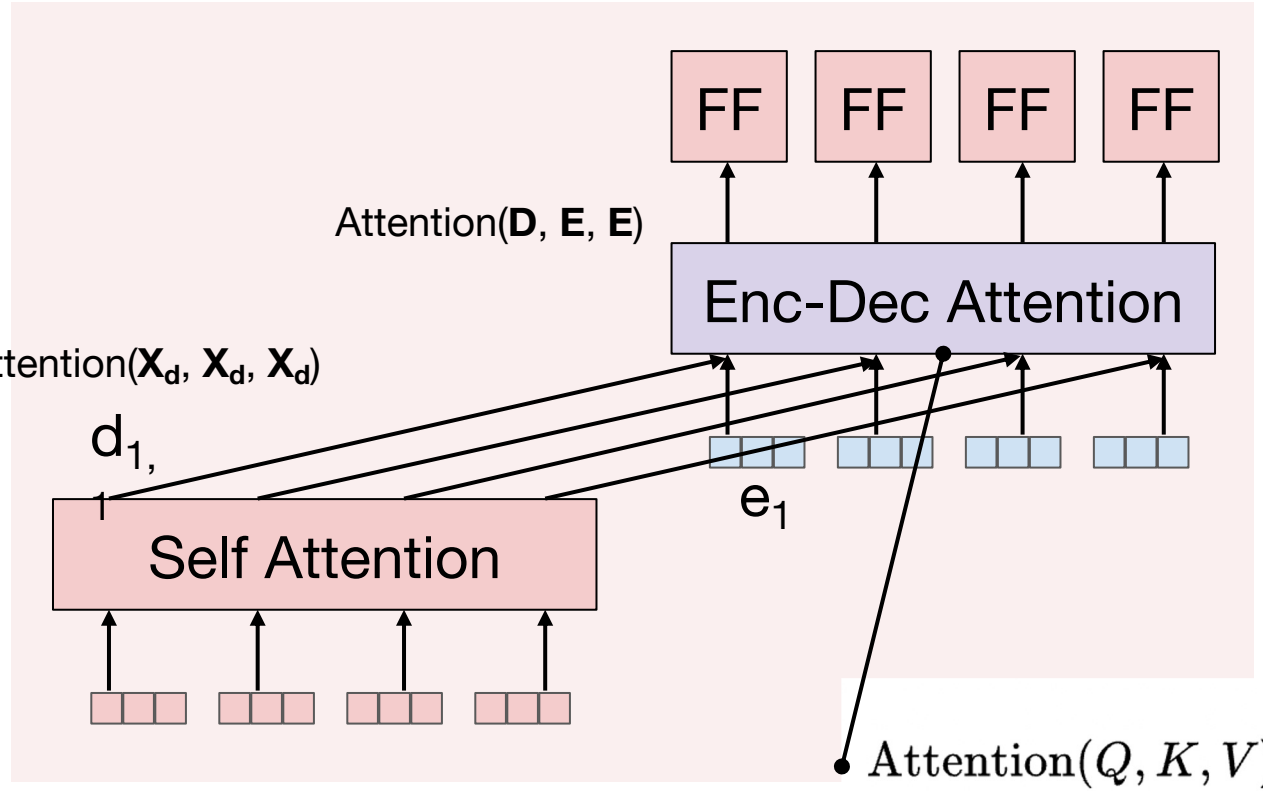
Encoder-Decoder Attention



- Commonly: encoder's outputs form the keys and values, and decoder self-attn outputs are queries.
- e.g., $q_1=d_{1,1}W_Q$; $k_1=e_1W_K$;$v_1=e_1W_V$

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

Encoder-Decoder Attention

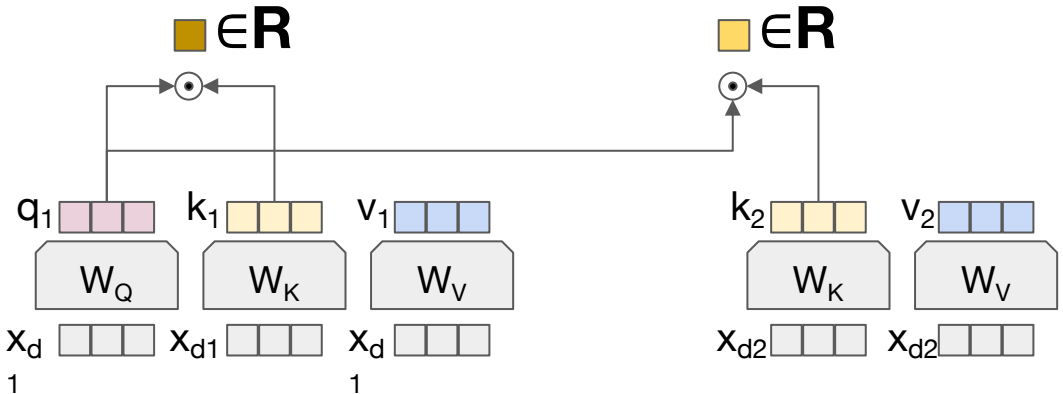


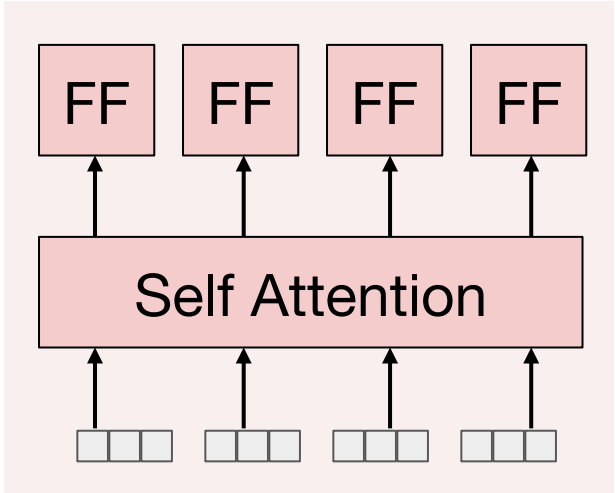
- So Q queries allow relations between dec target and enc rep
- Q from dec run against enc keys K and values V

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

Encoder-Decoder Attention

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

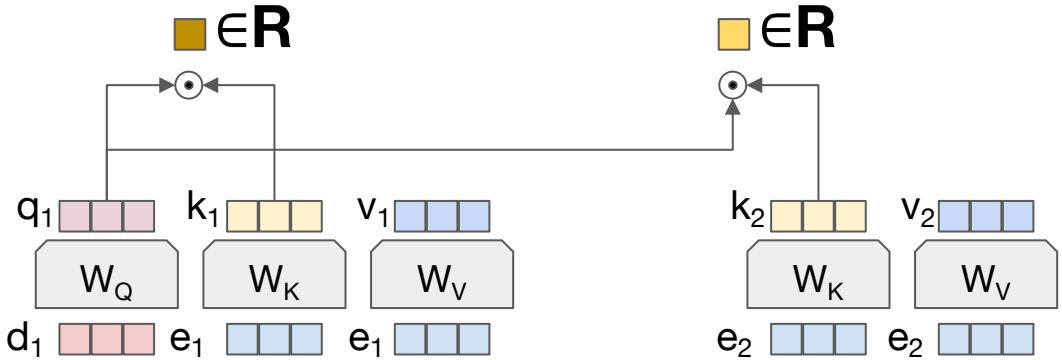


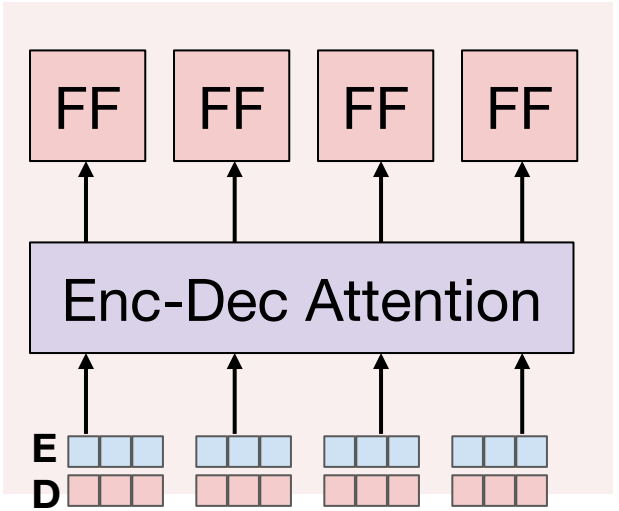


Encoder-Decoder Attention

Attention(**D**, **E**, **E**)

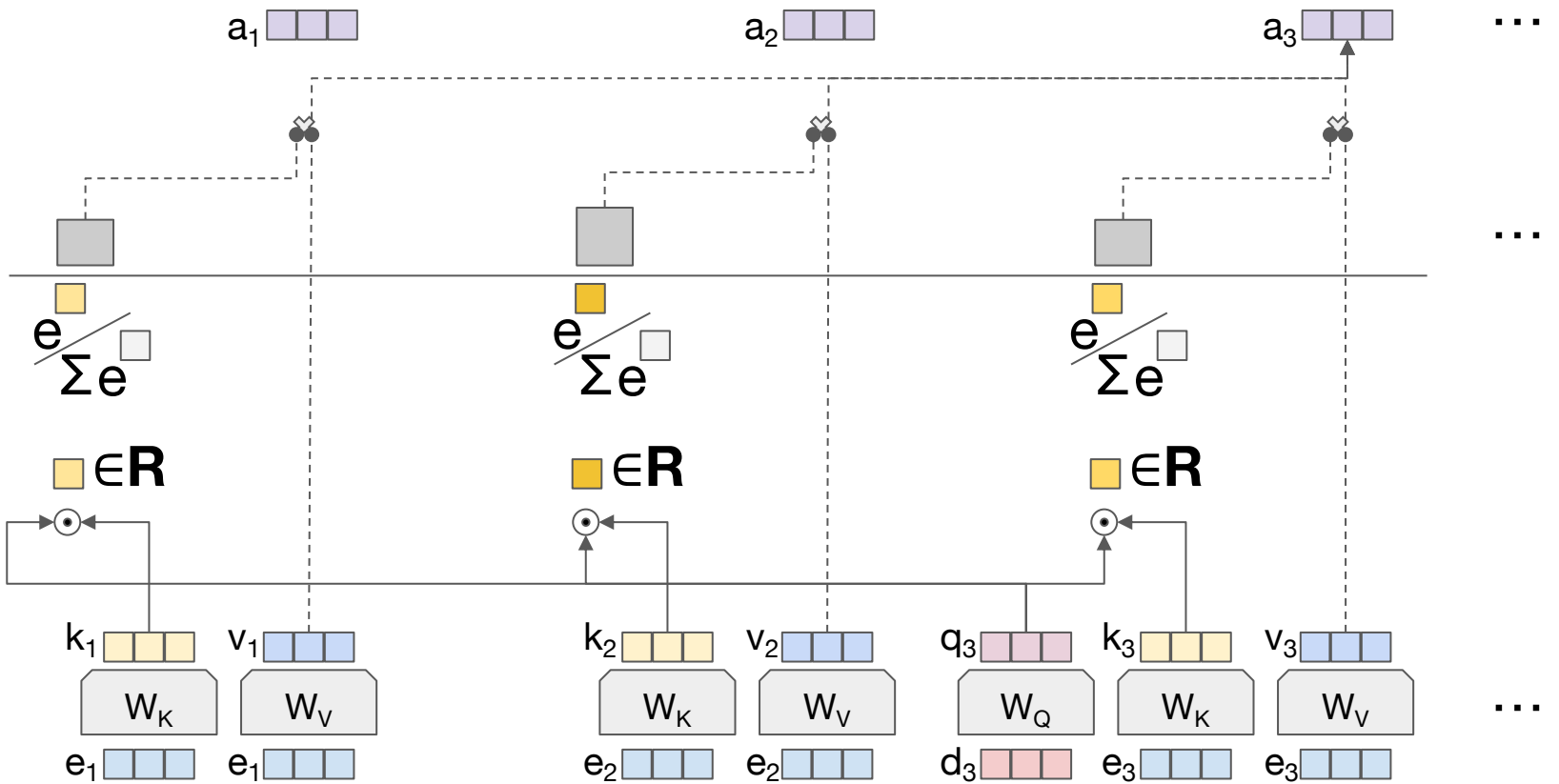
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$



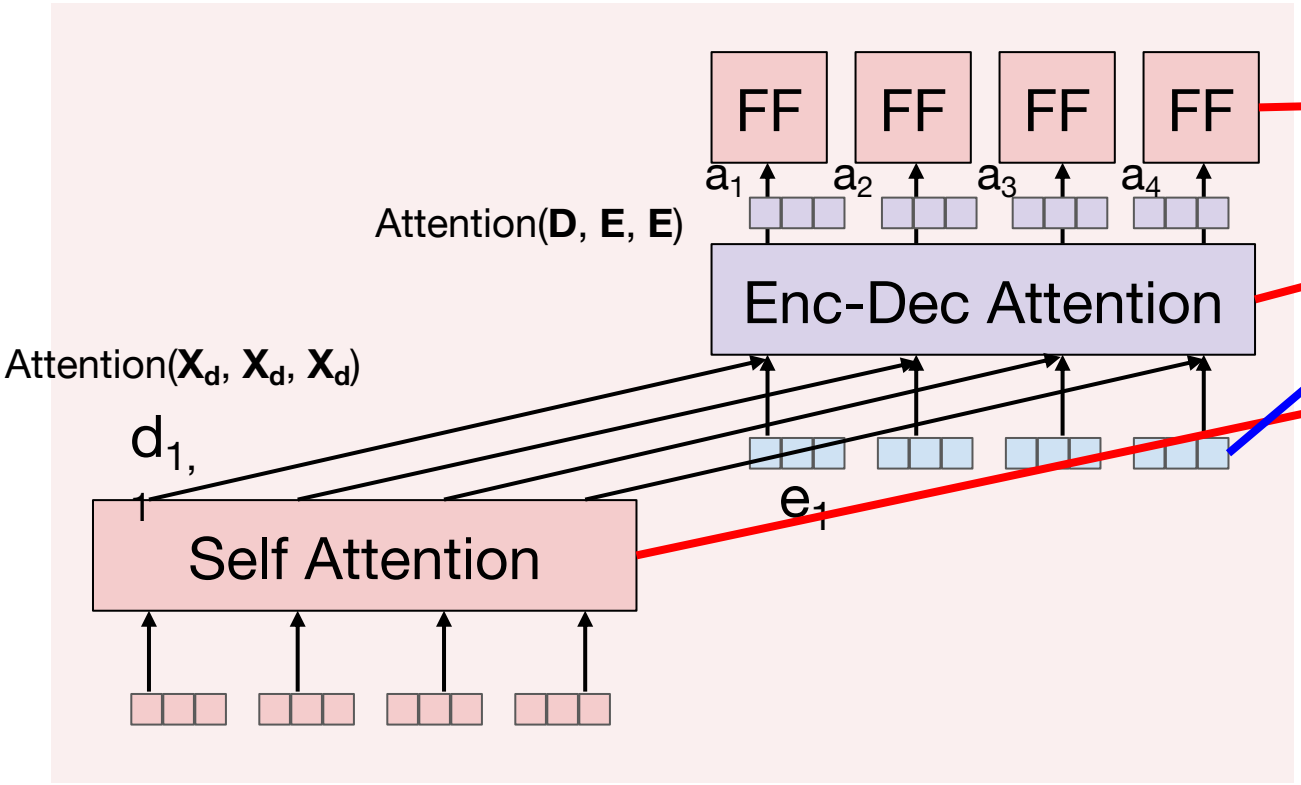


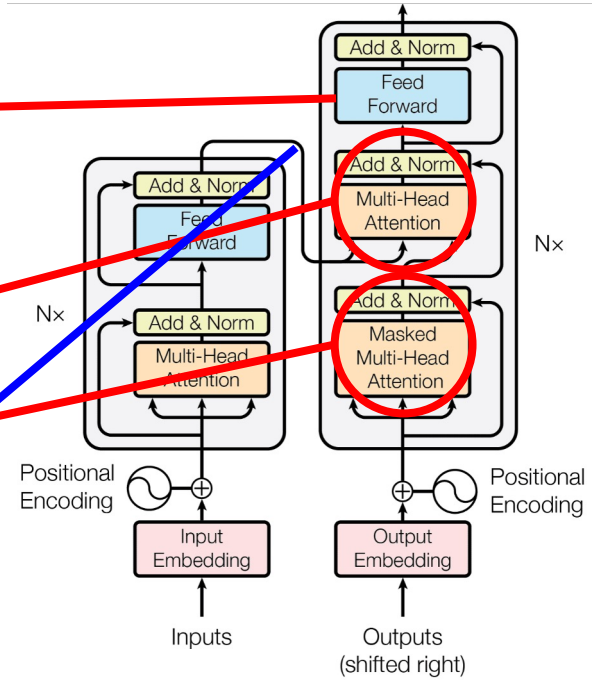
Encoder-Decoder Attention

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



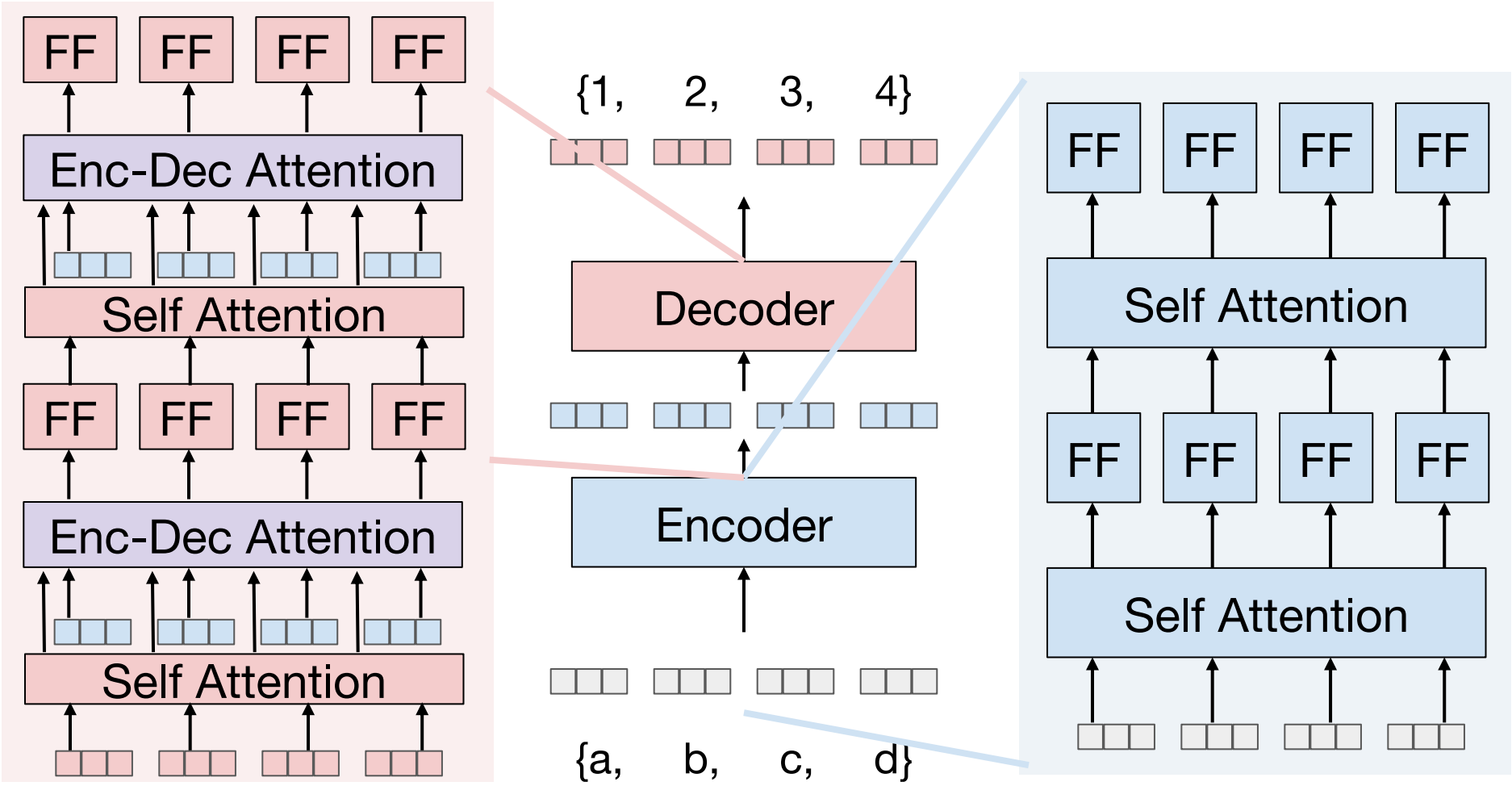
Encoder-Decoder Attention





[Attention Is All You Need]

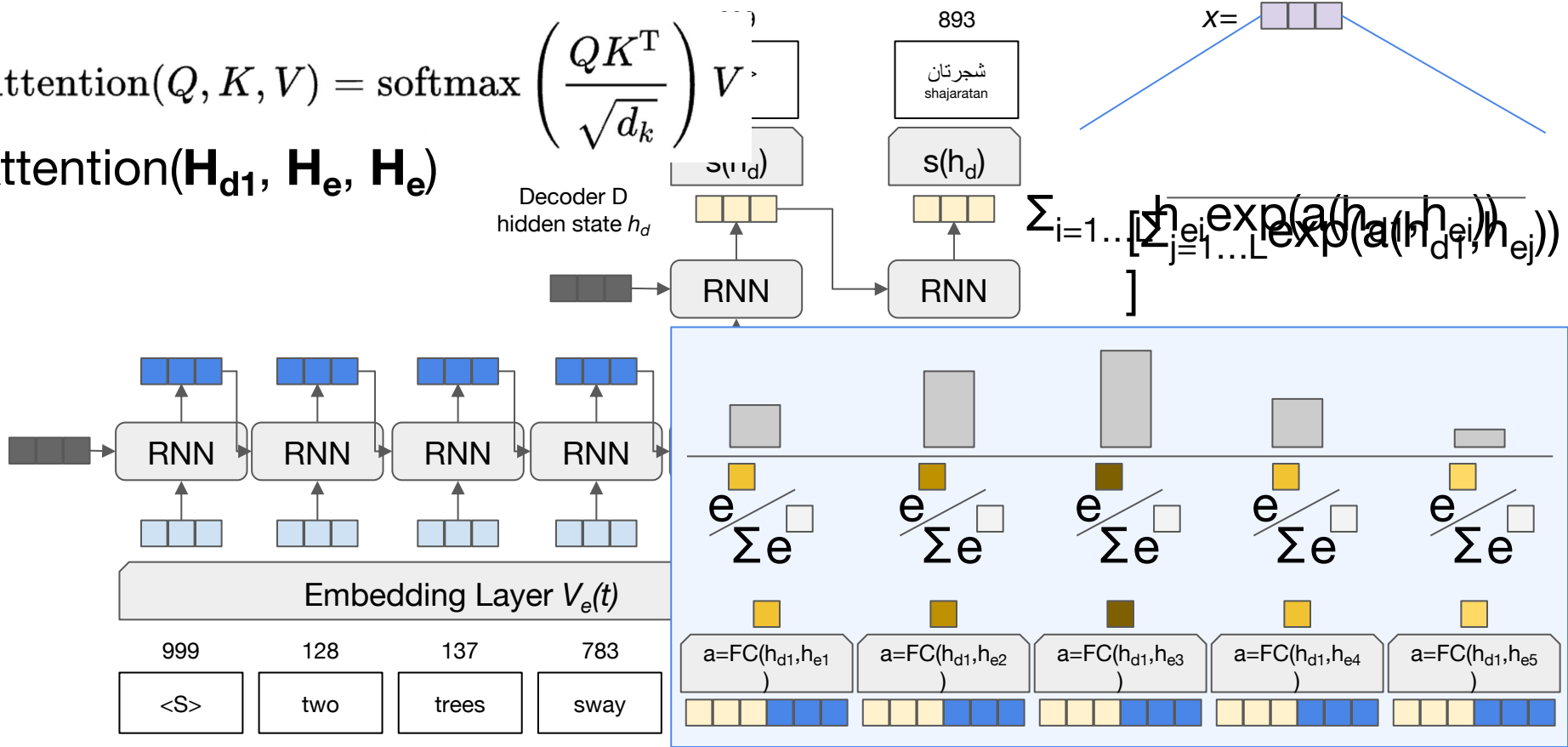
Deeper Encoders / Decoders



Relationship to *Global Soft Attention*

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

Attention($\mathbf{H_{d1}}$, $\mathbf{H_e}$, $\mathbf{H_e}$)



Relationship to *Global Soft Attention*

- Global Soft Attention: *Attention*($\mathbf{H_{d1}}$, $\mathbf{H_e}$, $\mathbf{H_e}$)
  - Stack last decoder hidden state into a *Query* tensor because we use it to query each *Key*, the encoder hidden states at each timestep, to get a weighted *Value*, the encoder hidden states at each timestep
- Transformer Encoder-Decoder Attention: *Attention*(**D**, **E**, **E**)
  - Decoder self-attention *Query* to encoder self-attention *Keys* to get weighted encoder self-attention *Values*
- Key difference: in the Transformer decoder, we have access to multiple timesteps because Transformers operate on *sets*!

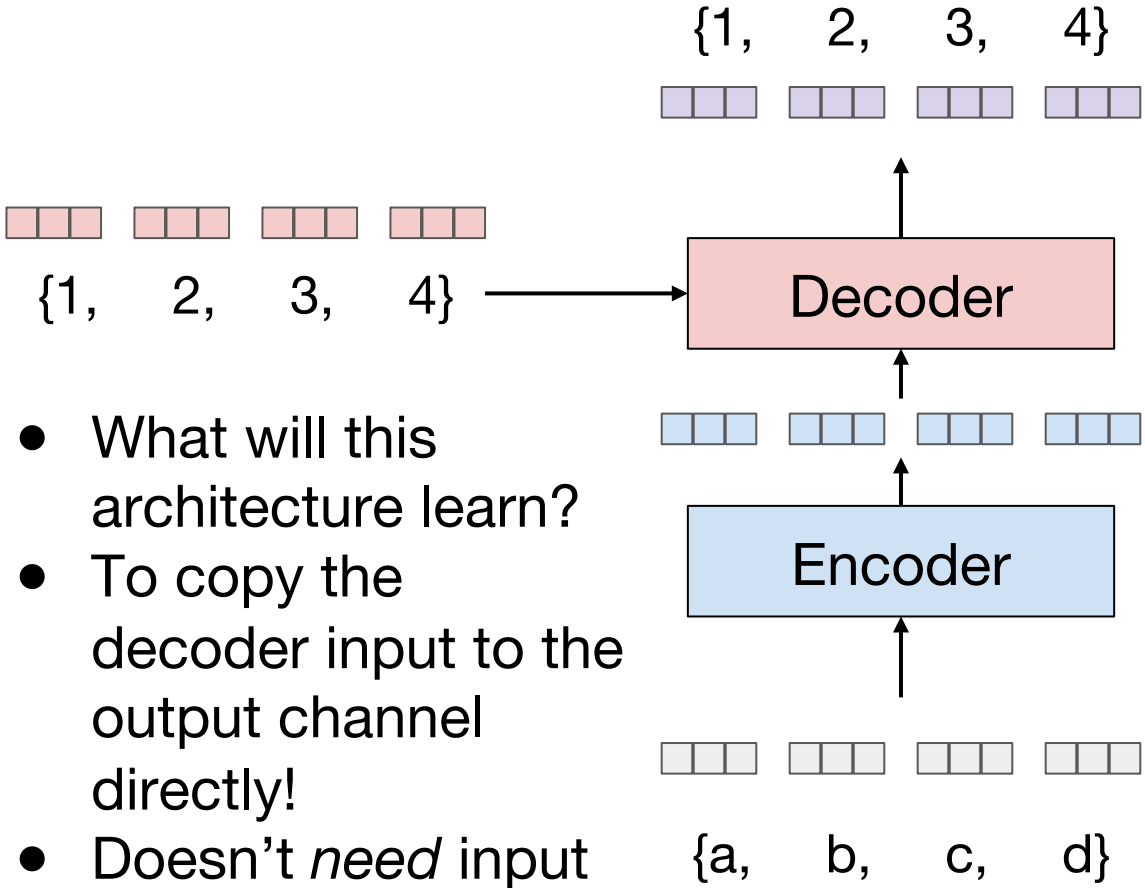
Self-Attention Intuitions

- Transformer Encoder Self Attention: *Attention*($\mathbf{X_i}$, $\mathbf{X_i}$, $\mathbf{X_i}$)
  - How the words in the input sequence relate to one another
  - *Query* how each word relates to the *Keys* of other words in the set (and itself), to get weighted *Value* of each words' representation with respect to others in the sequence
- Key takeaway: in self-attention, we encode the entire set of tokens with respect to one another, ending with a size L set of self-conditioned embeddings

Self-Attention Intuitions

- Transformer Decoder Self Attention: *Attention*($\mathbf{X_o}$, $\mathbf{X_o}$, $\mathbf{X_o}$)
  - How the words in the output sequence relate to one another
  - *Query* how each word relates to the *Keys* of other words in the set (and itself), to get weighted *Value* of each words' representation with respect to others in the sequence
- Key takeaway: In a Transformer, we have access to the input and output sets at encoding time, so the decoder self-attention can create a latent embedding too

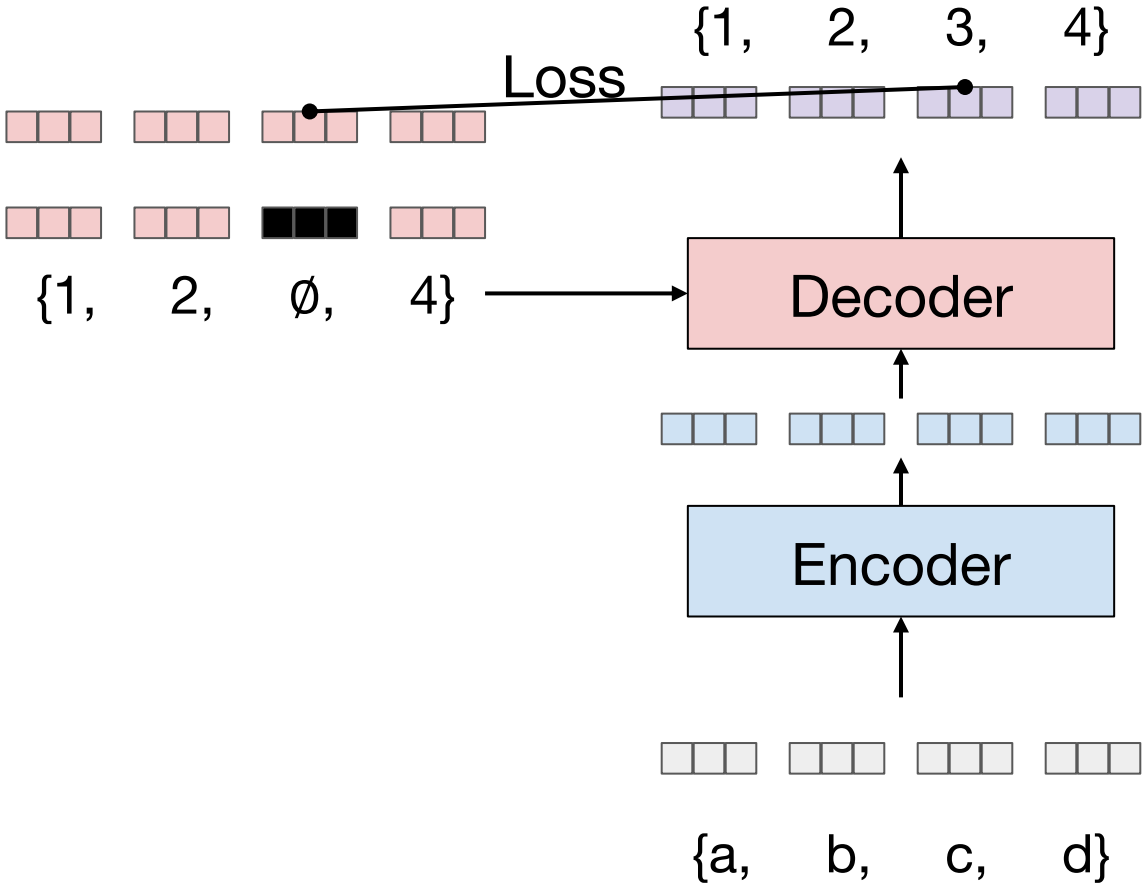
How Would We Train a Transformer?



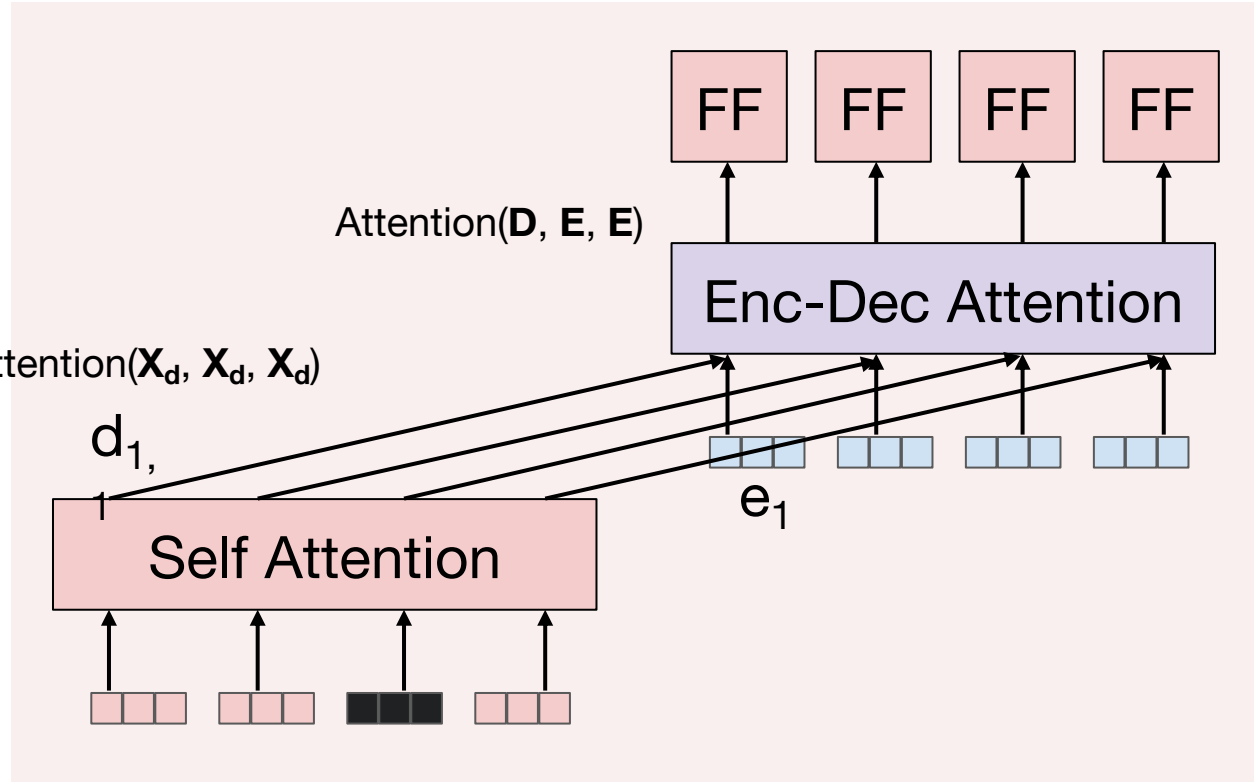
- What will this architecture learn?
- To copy the decoder input to the output channel directly!
- Doesn't *need* input

- Say we want to learn something really easy
- a->1, b->2, …
- Inputs: sets of letters
- Outputs: sets of numbers

Supervision: Masking

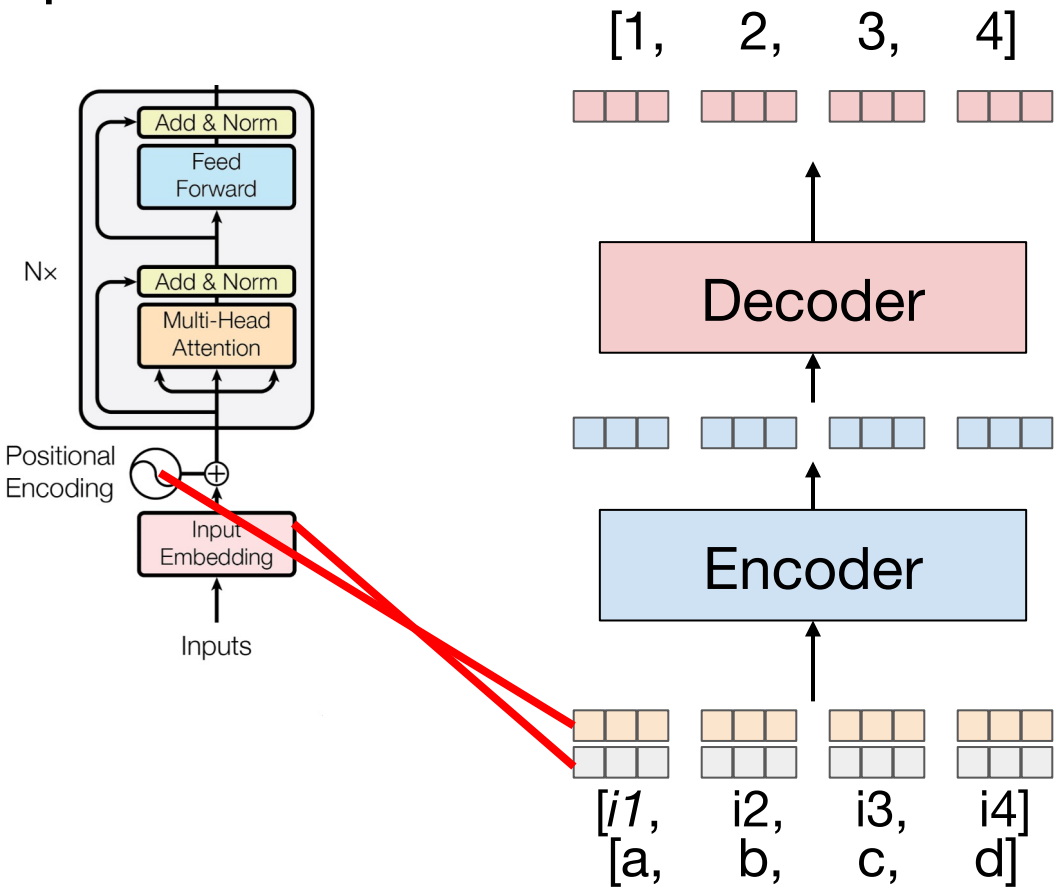


Supervision: Masking



- Learn to *recover* missing elements of the target output sequence conditioned on information from the input.

Sequences



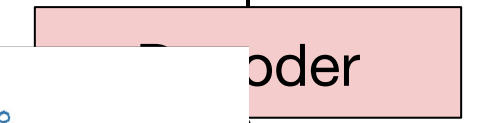
- What if we want our input/output to be *sequences*?
- E.g., sentences!
- Prev:
  - {a, c, d} -> {4, 3, 1},
- Want:
  - [a, c, d] -> [1, 3, 4]

Sequences: What *is* a Positional Encoding?

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

[1, 2, 3, 4]



Sequence	Index of token, k	Positional Encoding Matrix with d=4, n=100			
		i=0	i=0	i=1	i=1
I	0	P_{00}=sin(0) = 0	P_{01}=cos(0) = 1	P_{02}=sin(0) = 0	P_{03}=cos(0) = 1
am	1	P_{10}=sin(1/1) = 0.84	P_{11}=cos(1/1) = 0.54	P_{12}=sin(1/10) = 0.10	P_{13}=cos(1/10) = 1.0
a	2	P_{20}=sin(2/1) = 0.91	P_{21}=cos(2/1) = -0.42	P_{22}=sin(2/10) = 0.20	P_{23}=cos(2/10) = 0.98
Robot	3	P_{30}=sin(3/1) = 0.14	P_{31}=cos(3/1) = -0.99	P_{32}=sin(3/10) = 0.30	P_{33}=cos(3/10) = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

cos(p(0), p(1)) = 0.77		
cos(p(0), p(2)) = 0.28	cos(p(1), p(2)) = 0.77	
cos(p(0), p(3)) = -0.01	cos(p(1), p(3)) = 0.28	cos(p(2), p(3)) = 0.77

i3, i4]

- Creating a consistent "indicator vector" for each position
- "Nearness" between adjacent positions preserved by encoding functions.

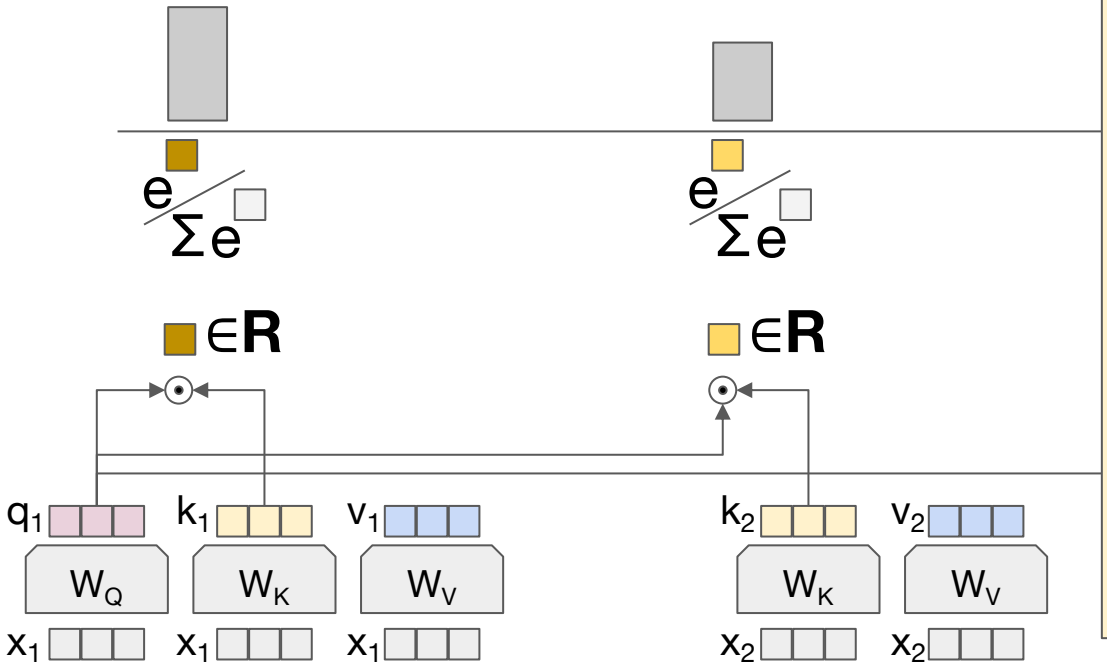
https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Positional Encodings

cos(p(1), p(2)) = 0.77

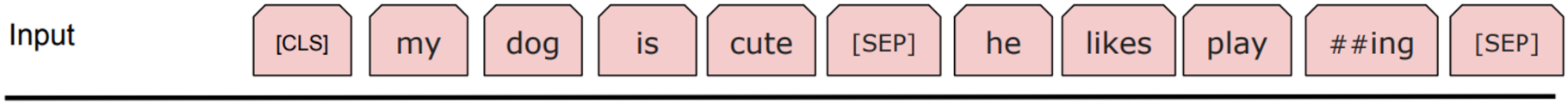
cos(p(1), p(3)) = 0.28

cos(p(1), p(4)) = -0.01



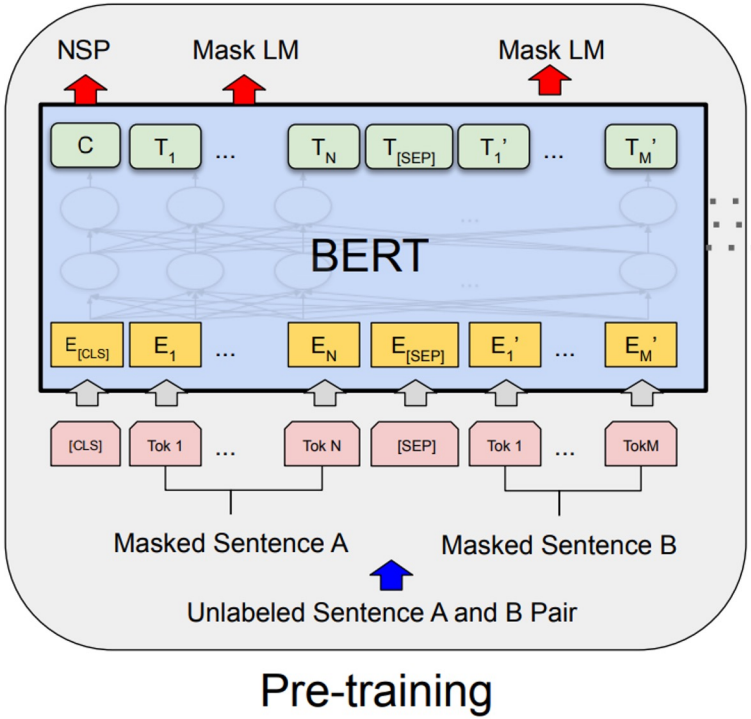
- **Assume:** W_Q and W_K matrix transformations preserve distance.
- Then in general:
  - $q_1 * k_2 > q_1 * k_3$
- Positional encodings *bias* the attention mechanism to give more weight to adjacent words without respect to the words themselves

Bidirectional Encoder Representations from Transformers



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

Bidirectional Encoder Representations from Transformers

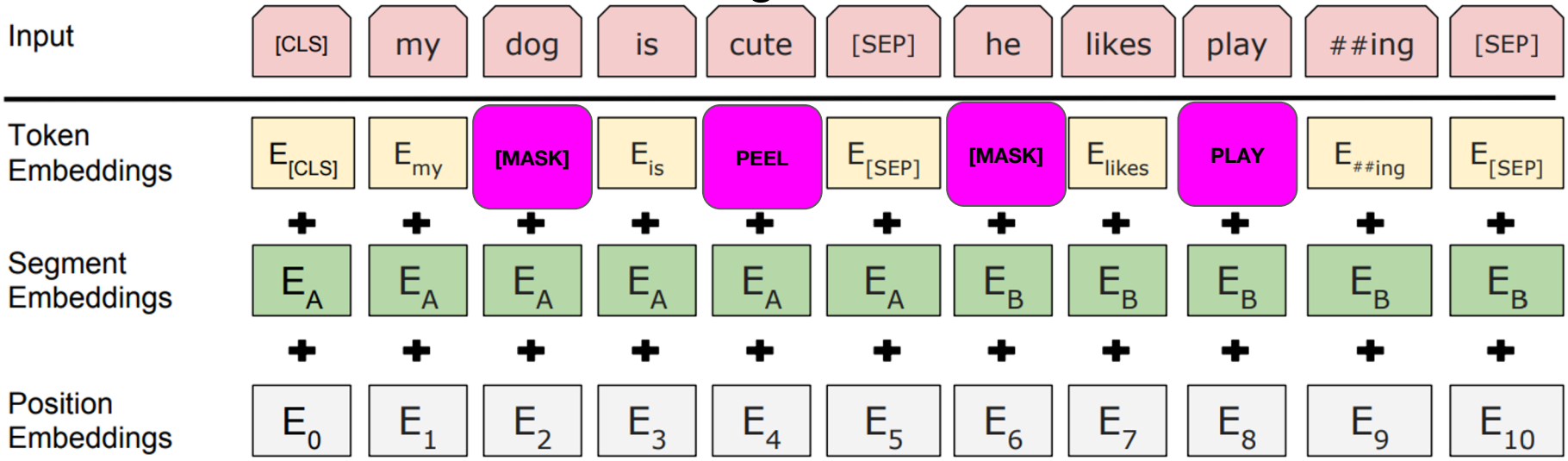


Pre-training

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

Masked Language Modeling Pretraining

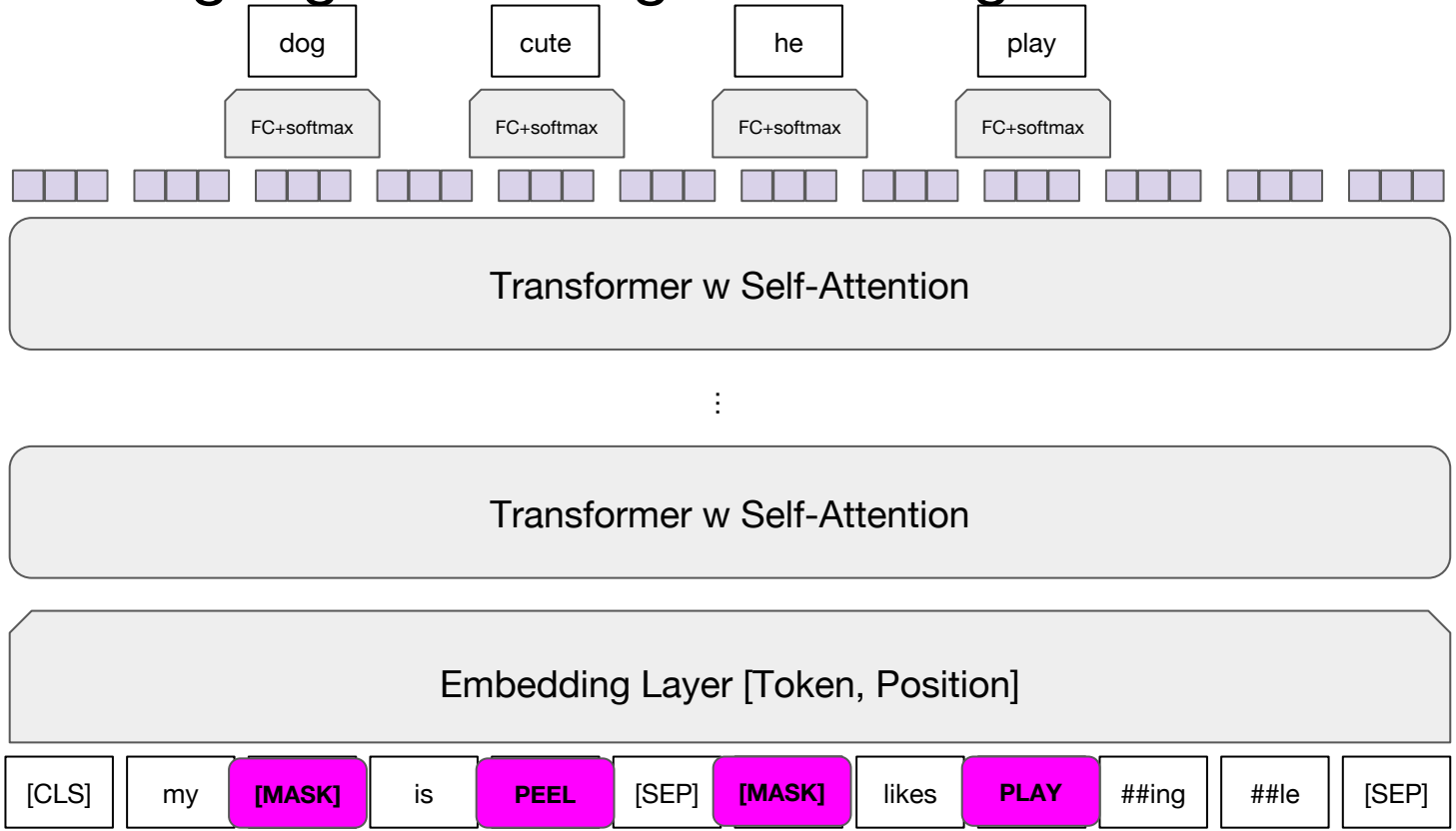
15% of words selected for training



80% of those [MASK] 10% random token 10% original token

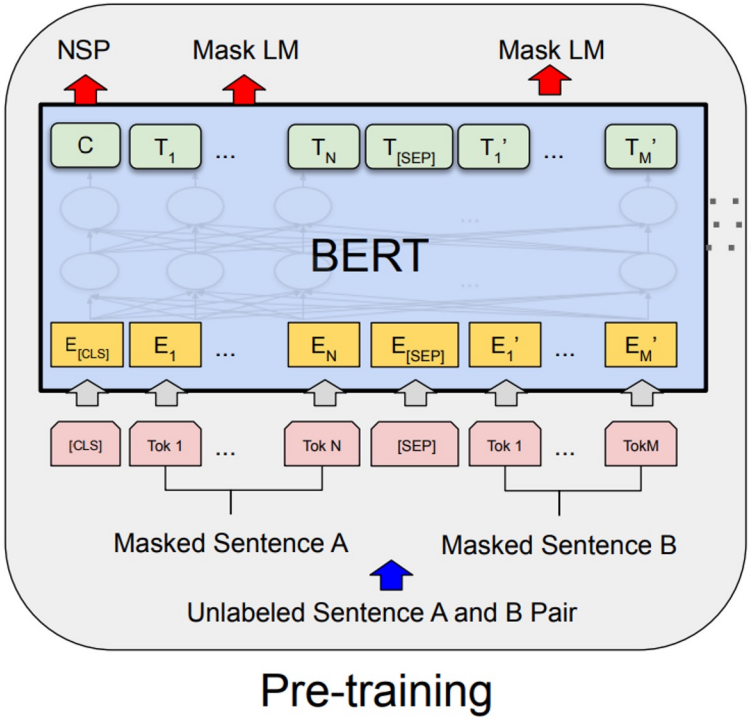
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Masked Language Modeling Pretraining



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

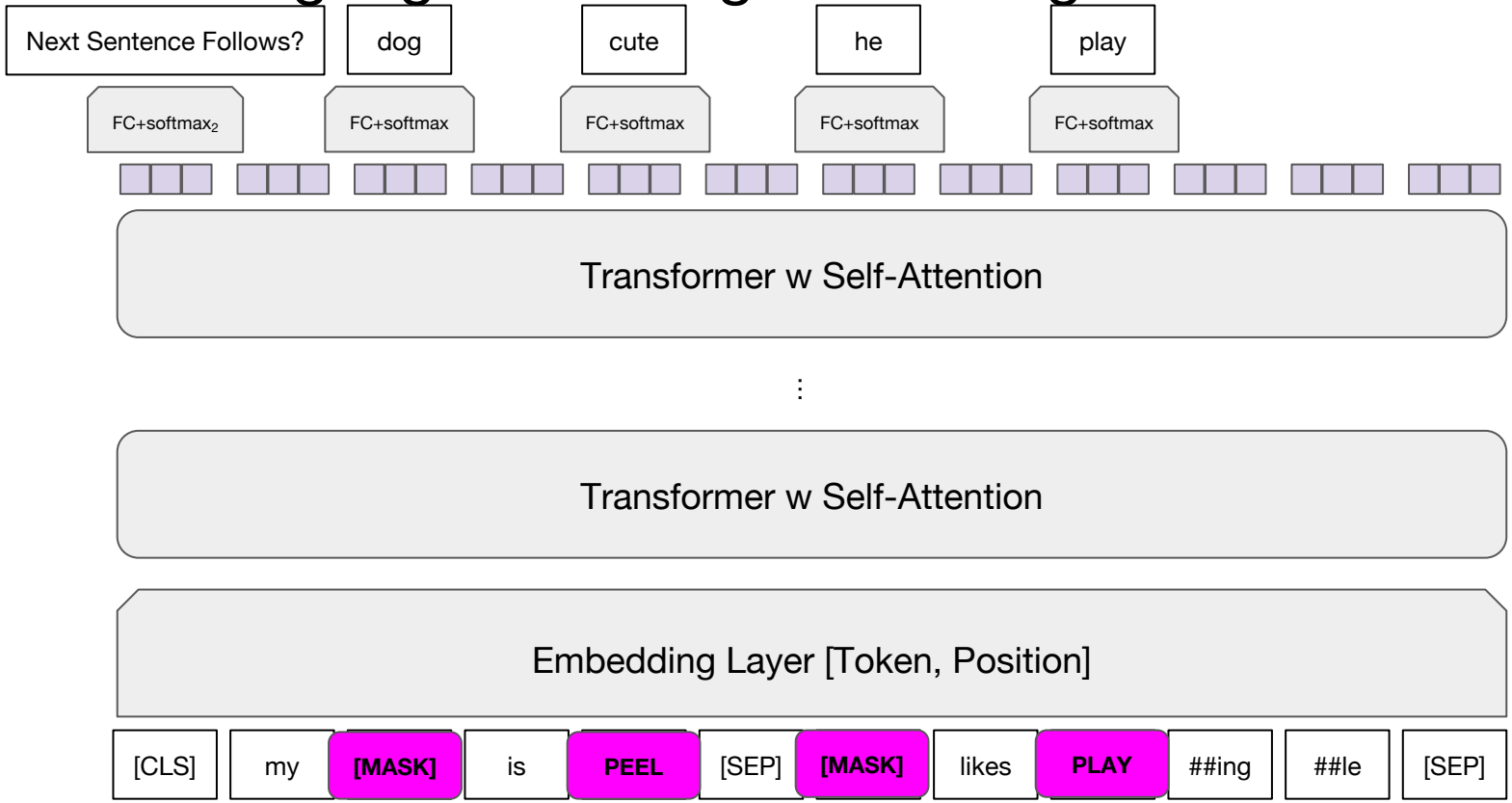
Bidirectional Encoder Representations from Transformers



Pre-training

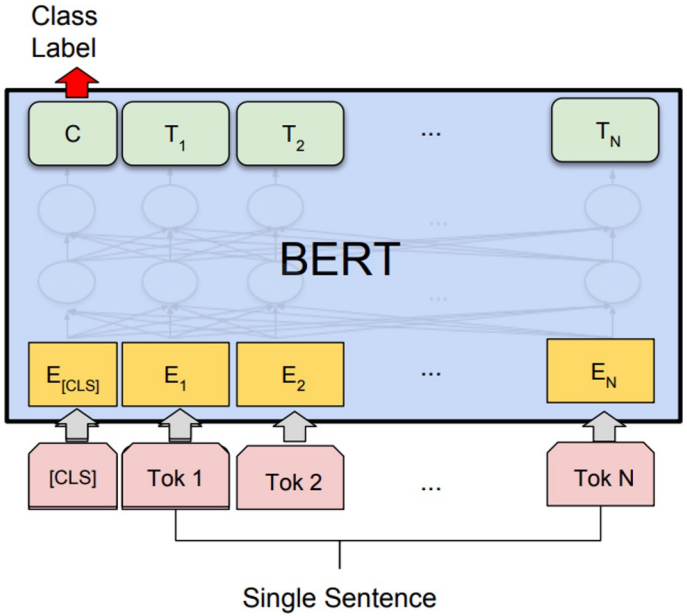
[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding]

Masked Language Modeling Pretraining



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

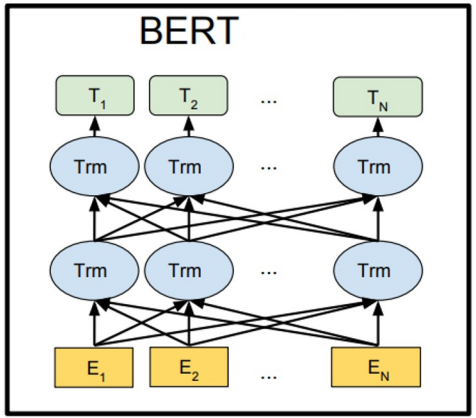
Bidirectional Encoder Representations from Transformers



(b) Single Sentence Classification Tasks:
SST-2, CoLA

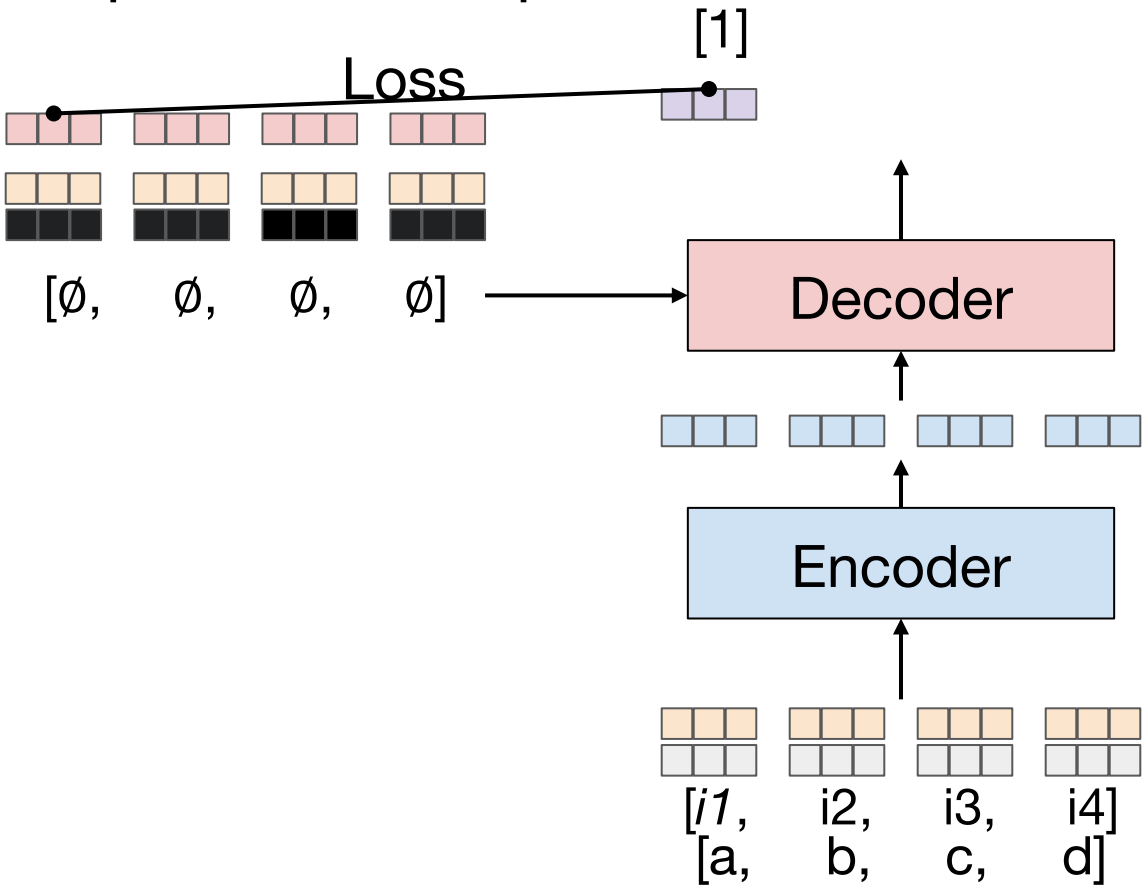
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Bidirectional versus Autoregressive



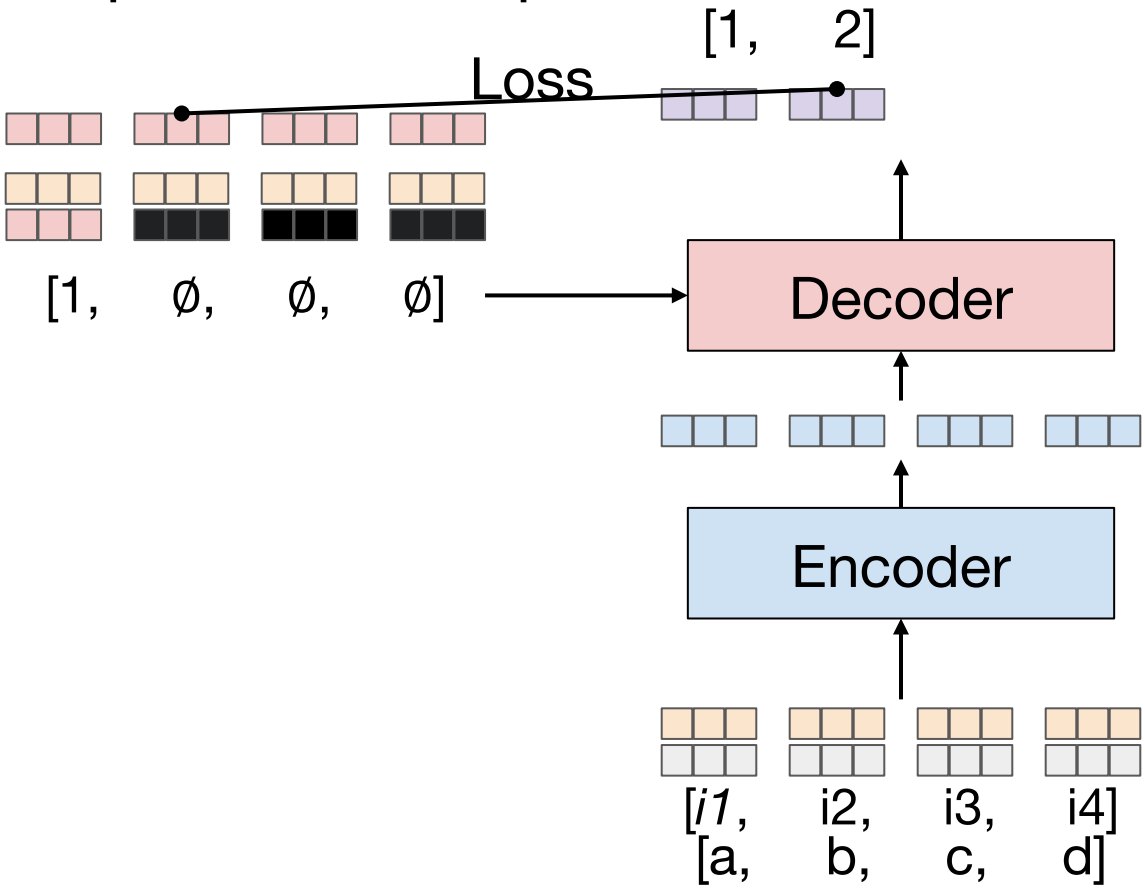
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Supervision: Sequences

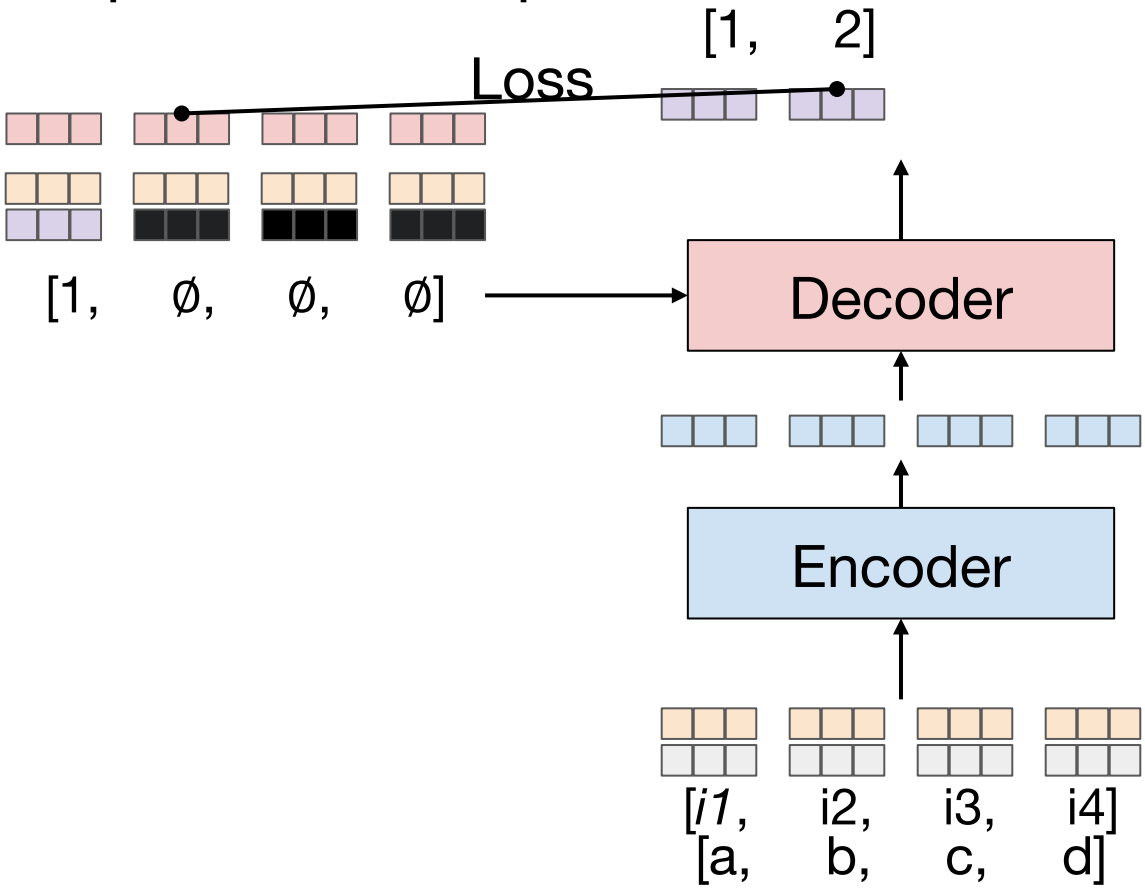


- Left-to-right masking of decoder input
- On each run, predict the next token.
- Need position information for decoder inputs as well.

Supervision: Sequences

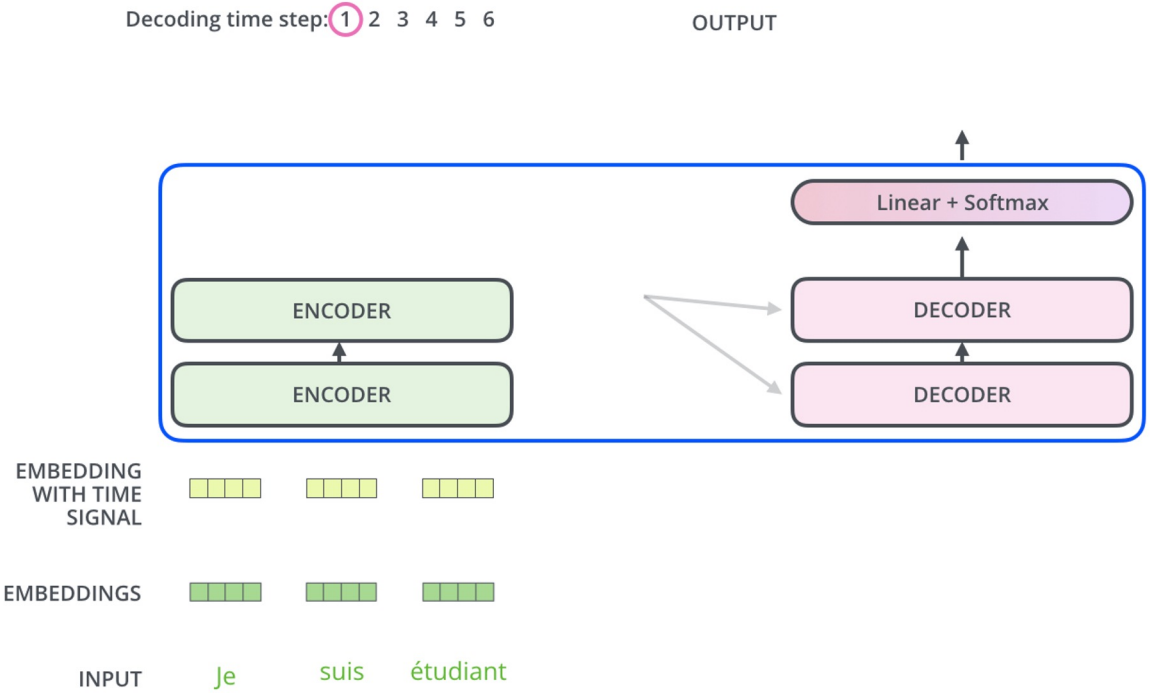


Supervision: Sequences



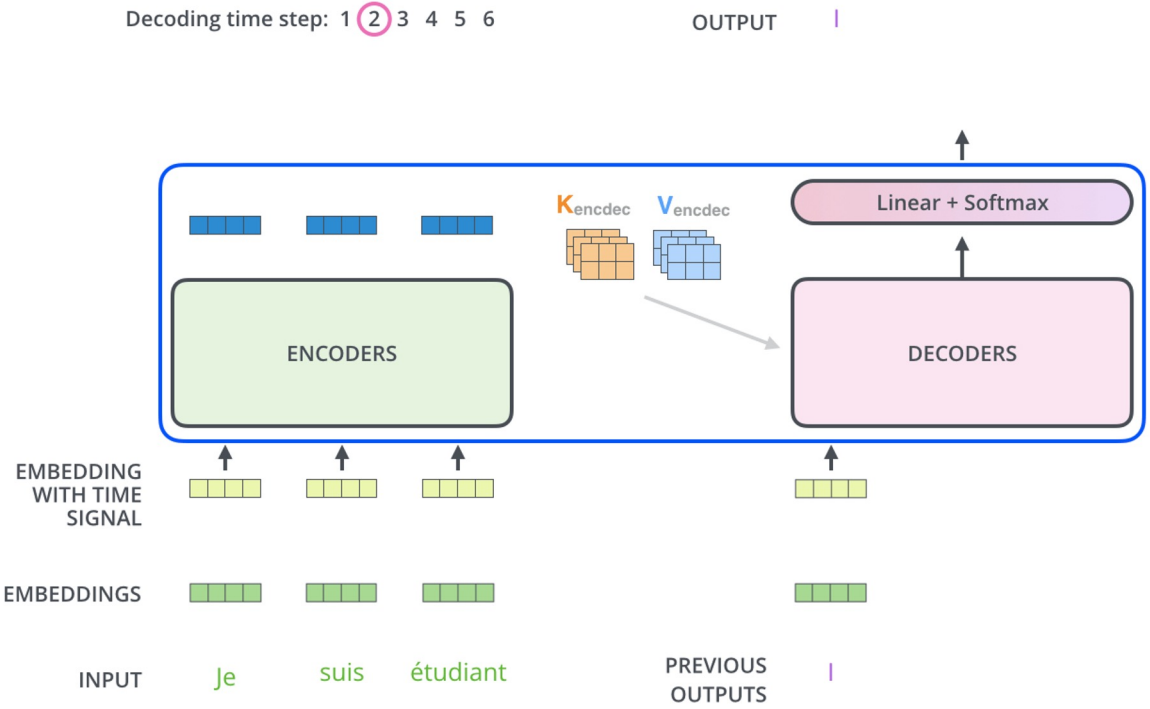
- In practice, decoder inputs will be previous timestep outputs.
- Related to general language modeling questions for decoding: sampling, beam search, etc.

Supervision: Sequences



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

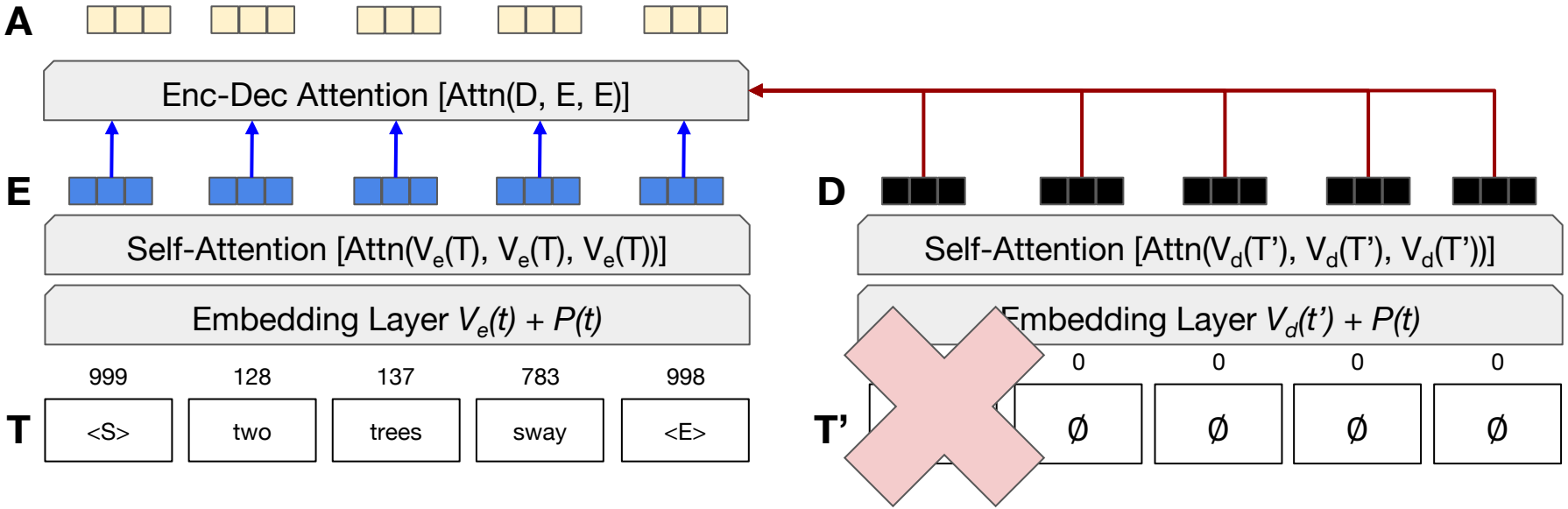
Supervision: Sequences



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

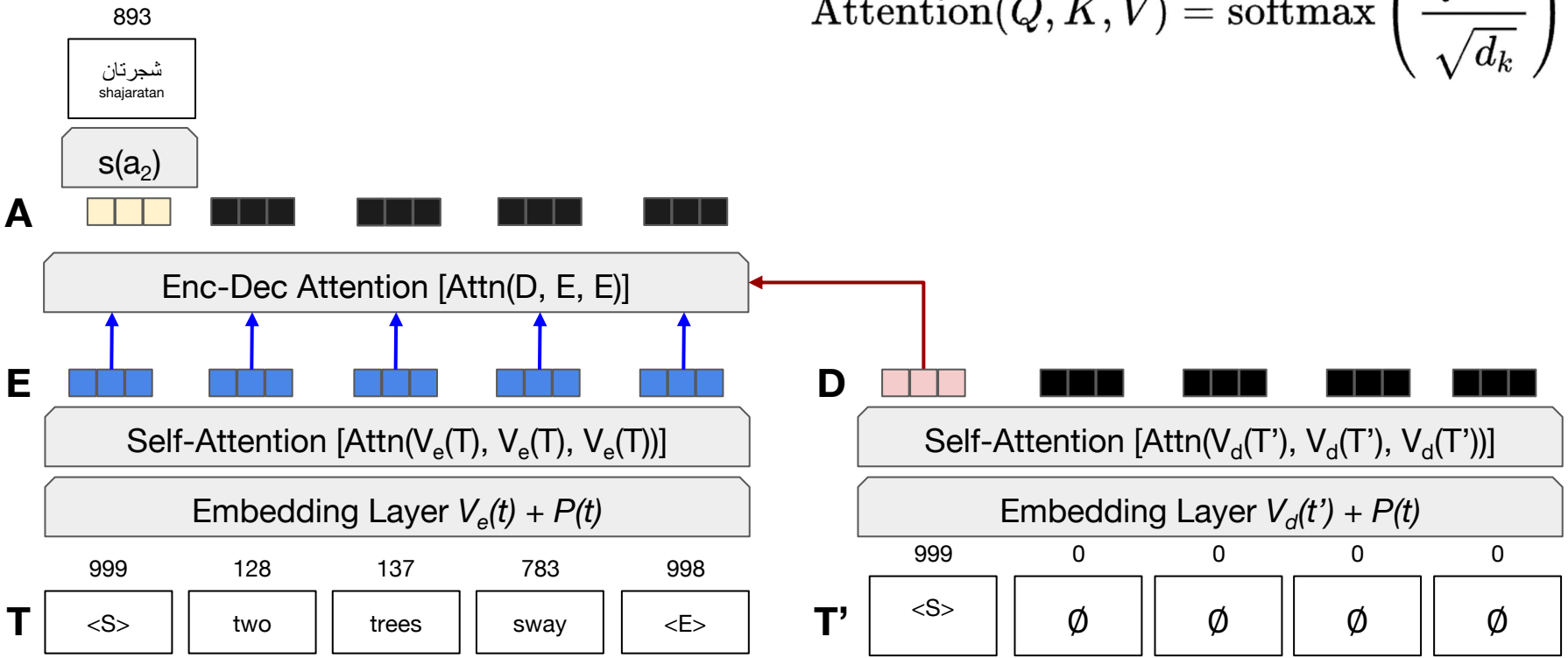
Sequence-to-Sequence Transformer

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

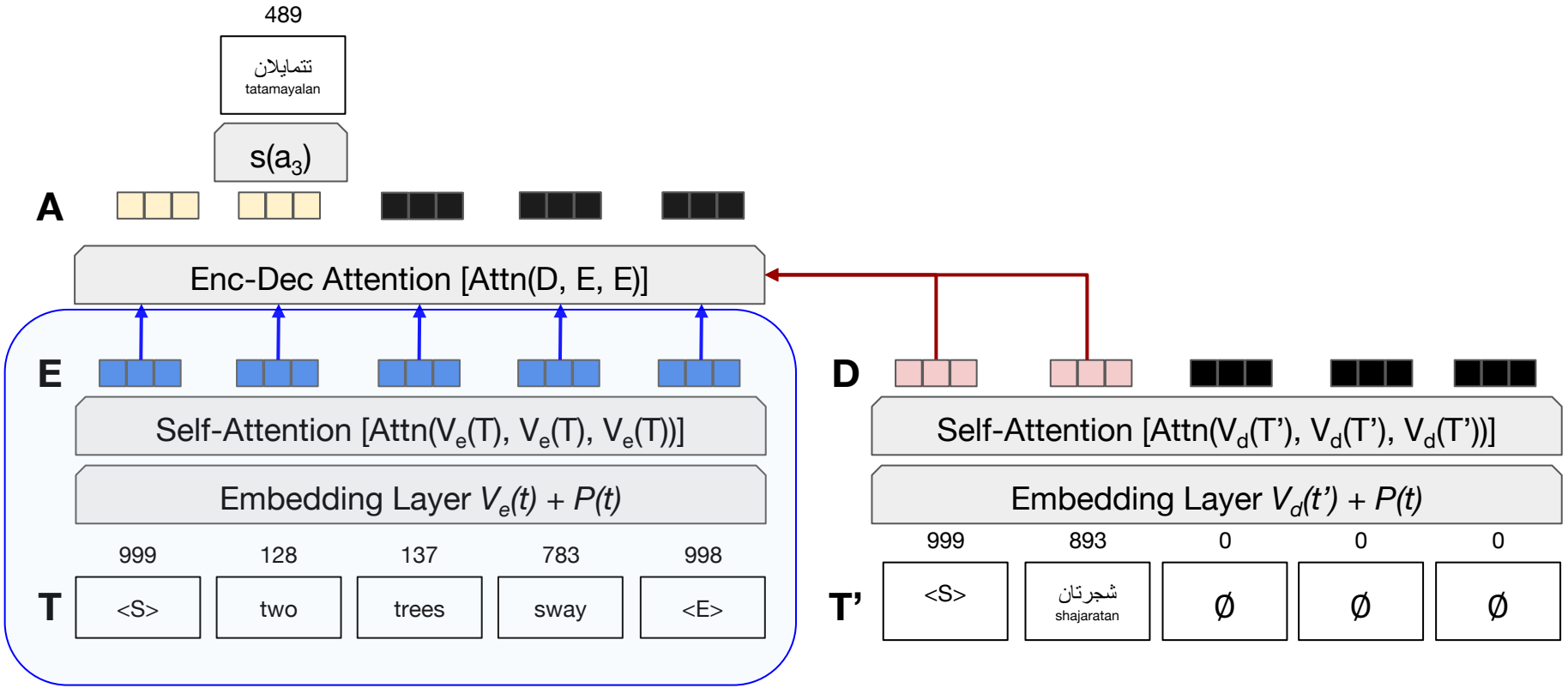


Sequence-to-Sequence Transformer

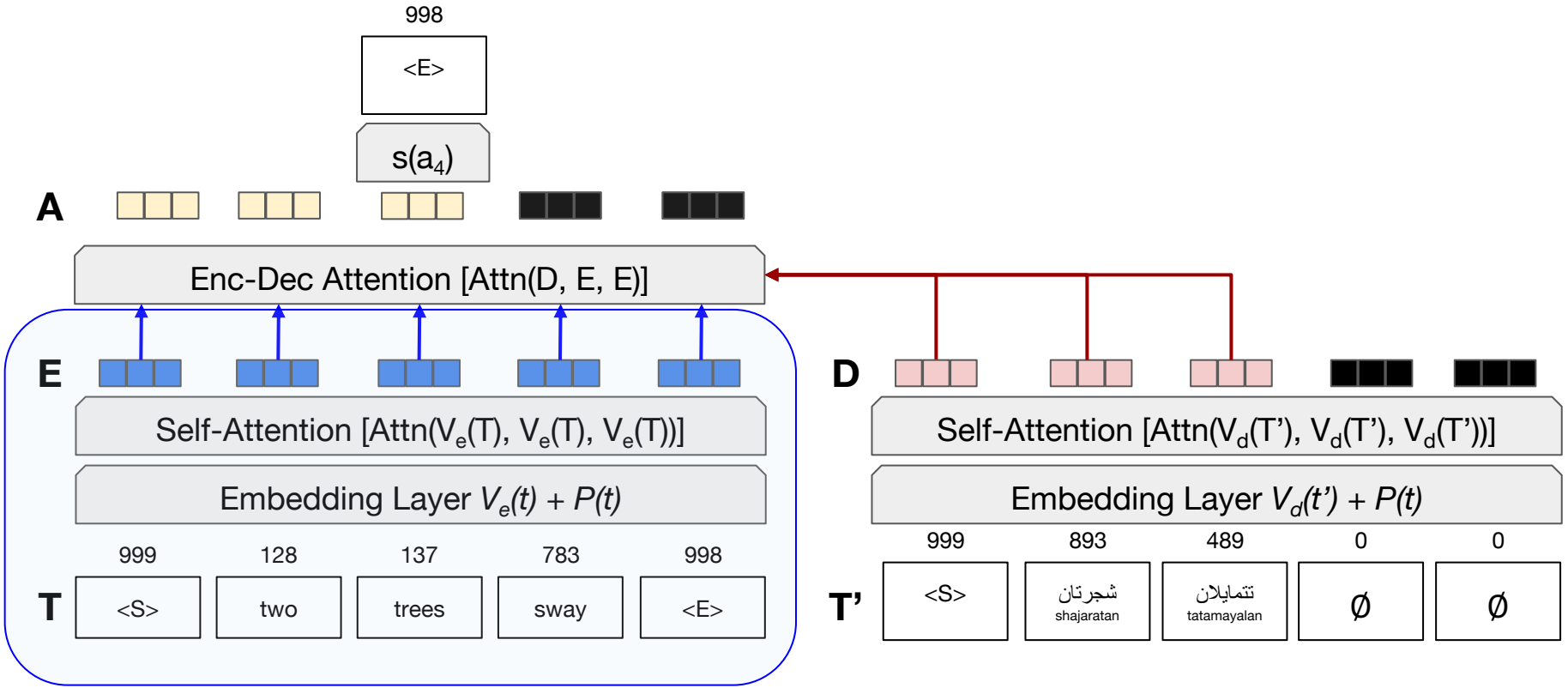
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{T}}}{\sqrt{d_k}}\right)V$$



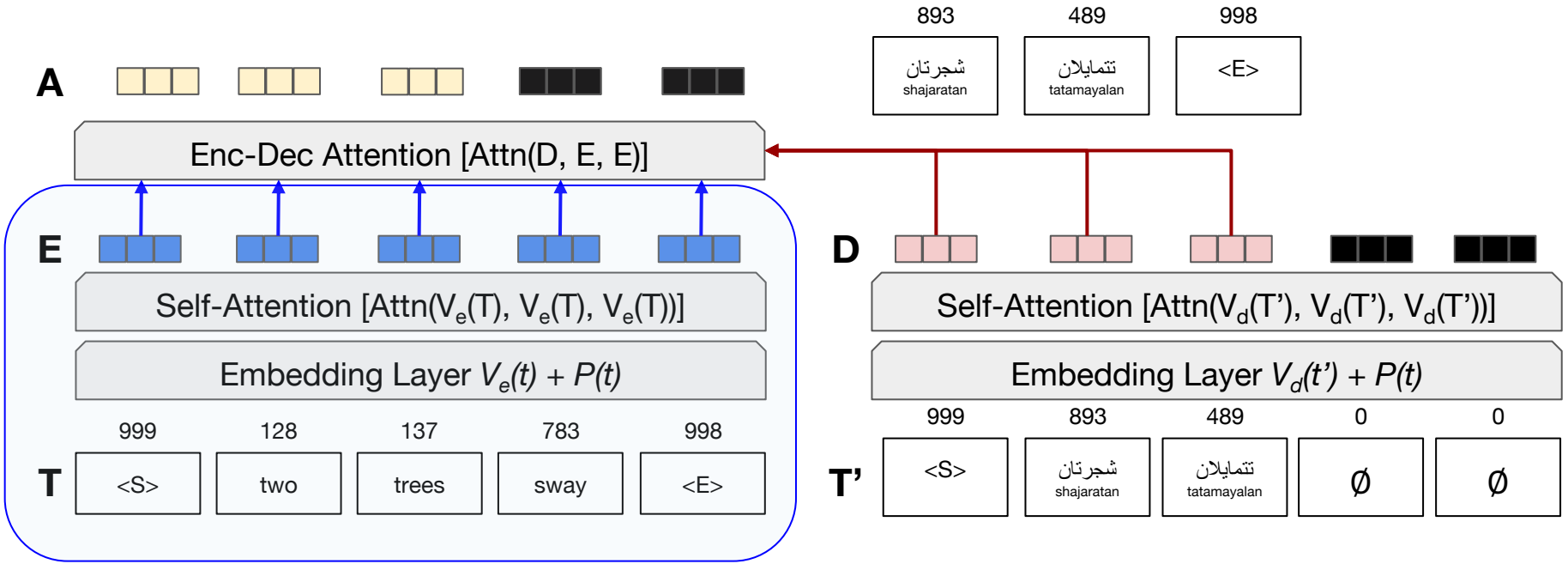
Sequence-to-Sequence Transformer



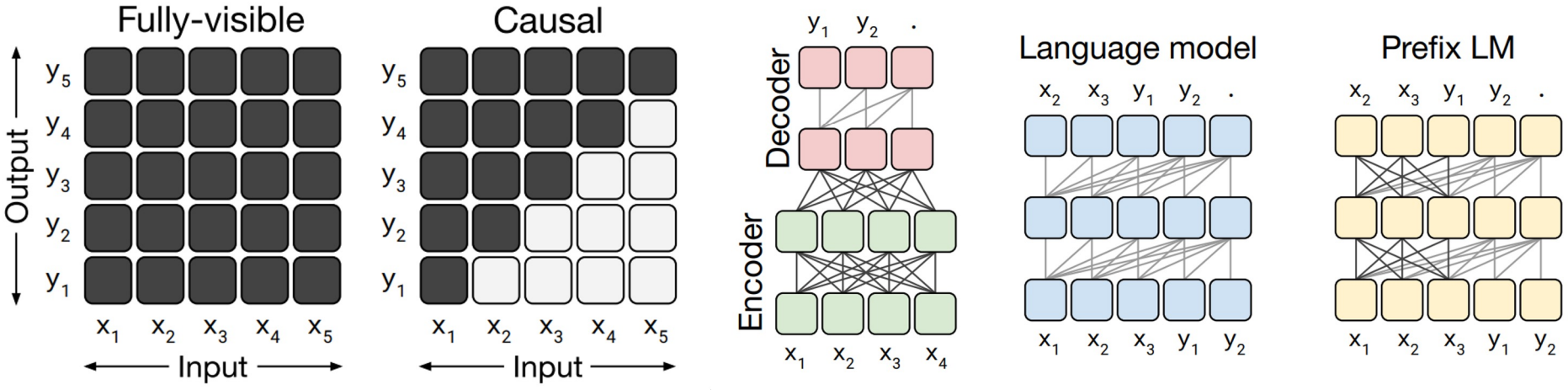
Sequence-to-Sequence Transformer



Sequence-to-Sequence Transformer



Redundant Terminology



- Autoregressive -> left-to-right; causal (e.g., GPT)
- Bidirectional -> fully-visible (e.g., BERT)
- What differences do you see here for Enc/Dec vs LM vs Pref LM?

INDUSTRY PRACTICE: Layer Normalization

- Like BatchNorm, except compute the estimated mean and variance of the activations of a *neurons* in a *single layer*

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- Learnable ɣ, β vectors; more stability and faster convergence
- Same operation at training and inference time
- Has become *industry standard* to include LayerNorm between stacked Transformer layers

Layer Normalization (original paper and method); On Layer Normalization in the Transformer Architecture

Transformers: The Future of the Present

- Transformers *seem* like they might be the universal encoders we've been waiting for in multimodal space
- Language tokens, image regions, image patches, audio snippets, joint positions, transformers don't care!
- We'll keep talking about Transformers next time with Deep Learning for Multimodality and Agents

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Transformers~~
  - Any questions before we move on?
- DL for Computer Vision
- Midterm Debrief

Deep Learning and Machine Vision

- The resurgence in popularity of Deep Learning started in vision
  - AlexNet!
- In our lectures on CNNs, we covered some “classic” CV problems like image classification
- Object detection is similar, but typically involves identifying “regions of interest” where an object might live
- For today’s focus, I want to move towards the “flashier” half of input/output relationships in computer vision

ML FUNDAMENTALS: Discriminative v. Generative

Bayes' Rule

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

Posterior of Y

Likelihood of Y

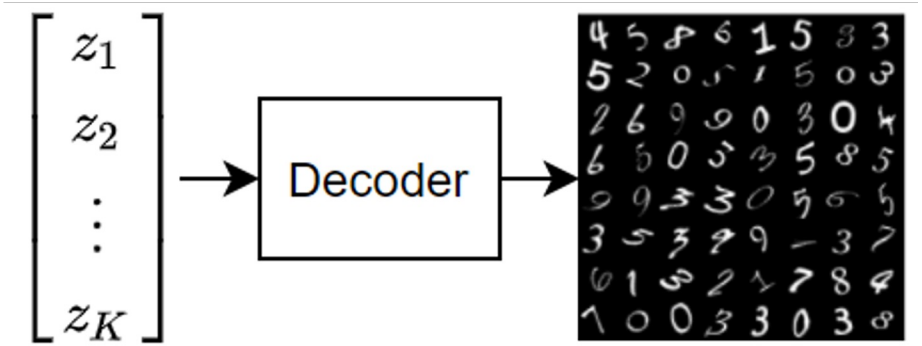
Prior of Y

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Discriminative
Estimate P(Y|X) from data.
E.g., {f(x)=y} ↔ {p(y|x)}

Generative
Estimate P(X|Y) and P(Y) from data.
E.g., {f(x)=y} ↔ {p(y) occurs and p(x|y) that y *explains* observed x}

Deep Generative Modeling for Computer Vision



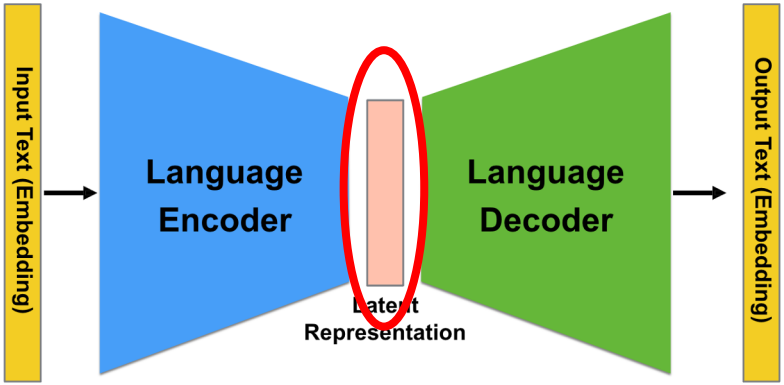
Random Vectors

Generated Images

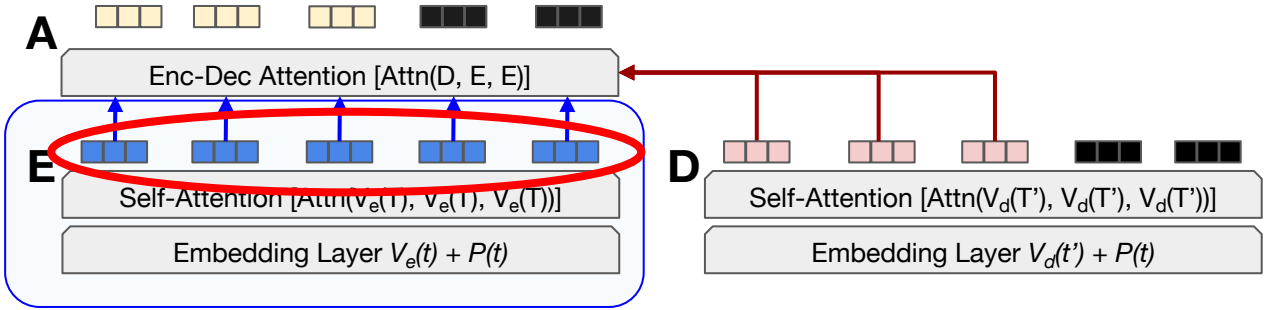
Autoencoder

- That word embeddings are a learned compression of co-occurrence matrices for tokens is part of a larger trend
- One can formulate deep learning models to be *learned compression algorithms* for a given input space
- Autoencoding is *unsupervised*:
  - Given input of size X, shrink to size x<X with learned function (encoder) and unshrink it back to X with a another learned function (decoder)
- Autoencoders learn a *lower dimensional manifold* for input

Autoencoder

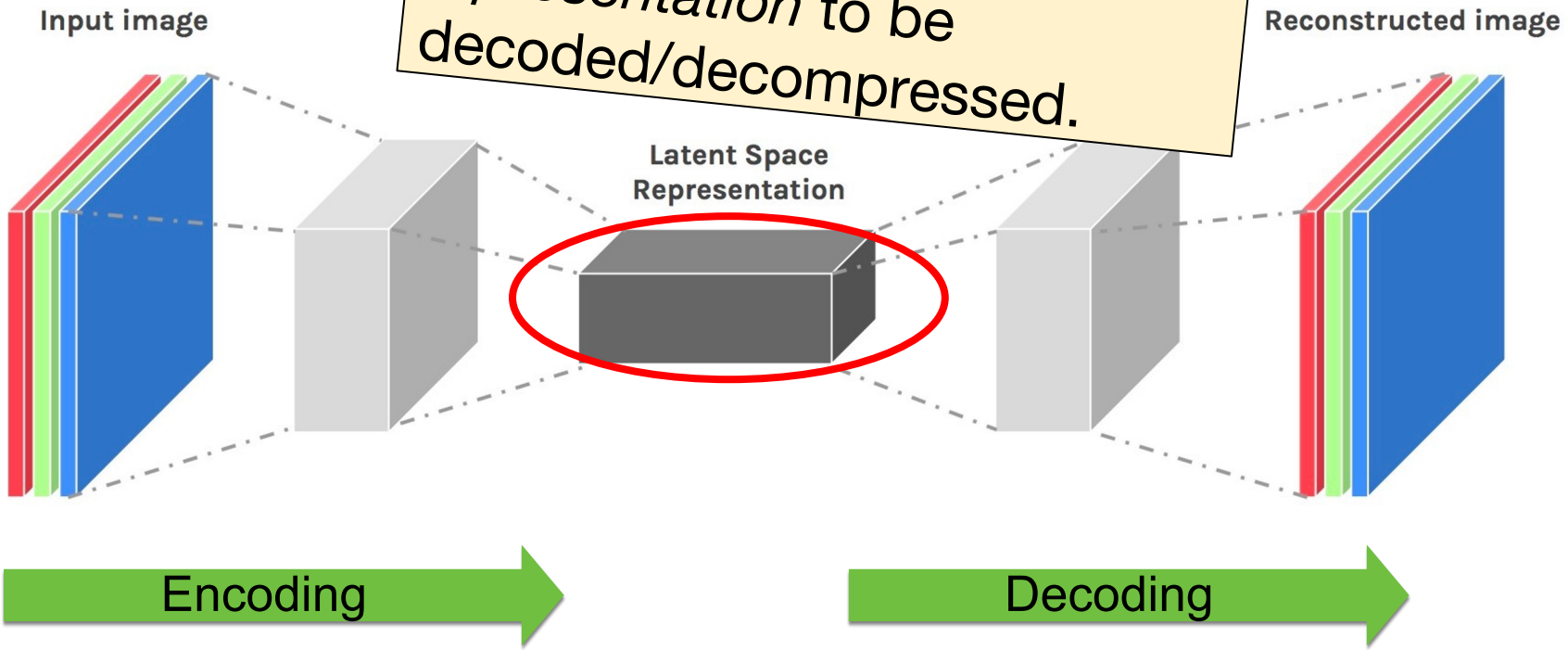


The latent tensor from the encoder is a *compressed representation* to be decoded/decompressed.

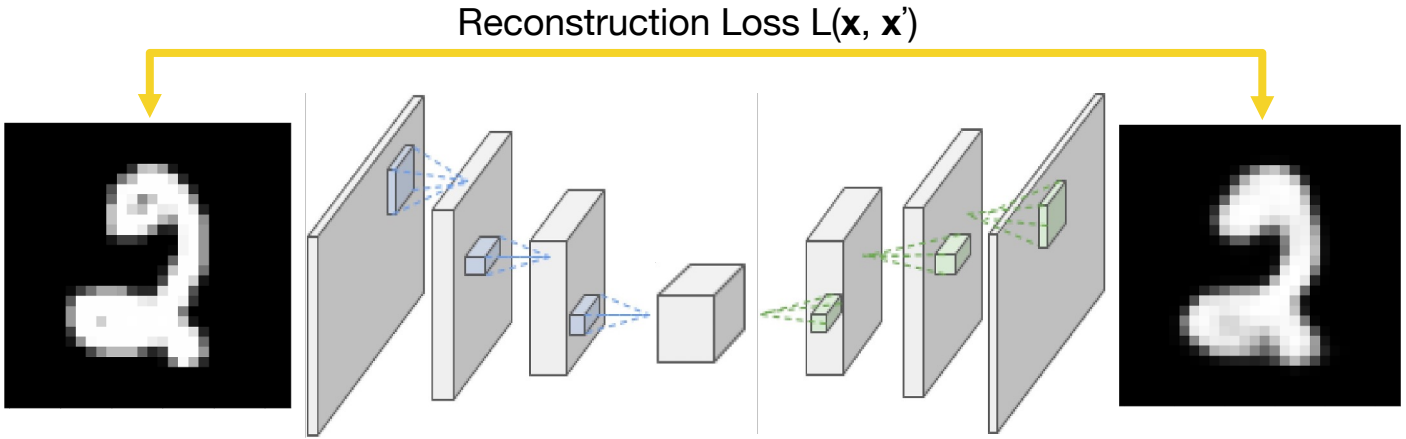


Autoencoder

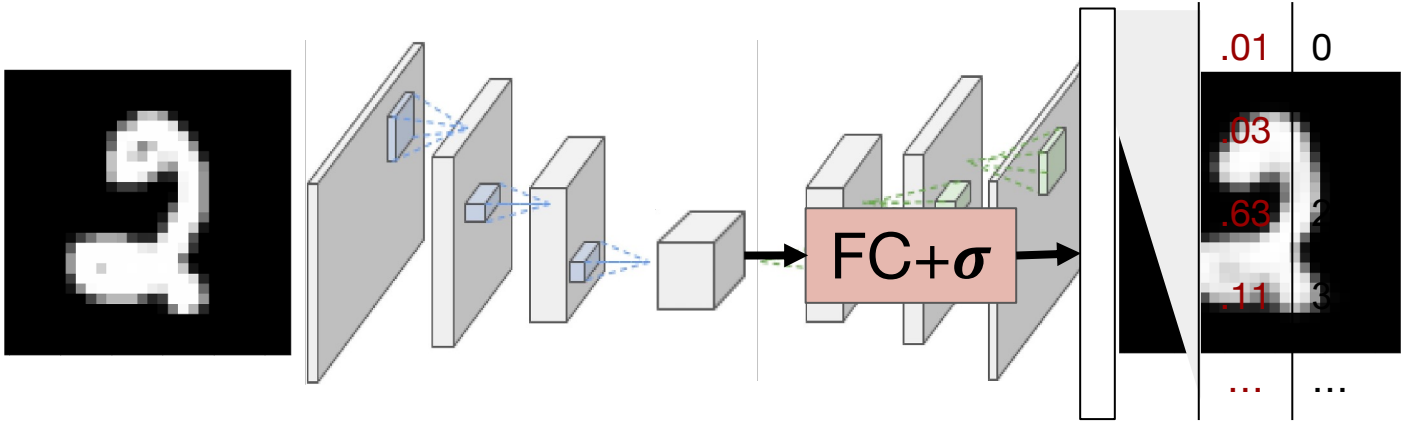
The latent tensor from the encoder is a *compressed representation* to be decoded/decompressed.



CNN Autoencoder

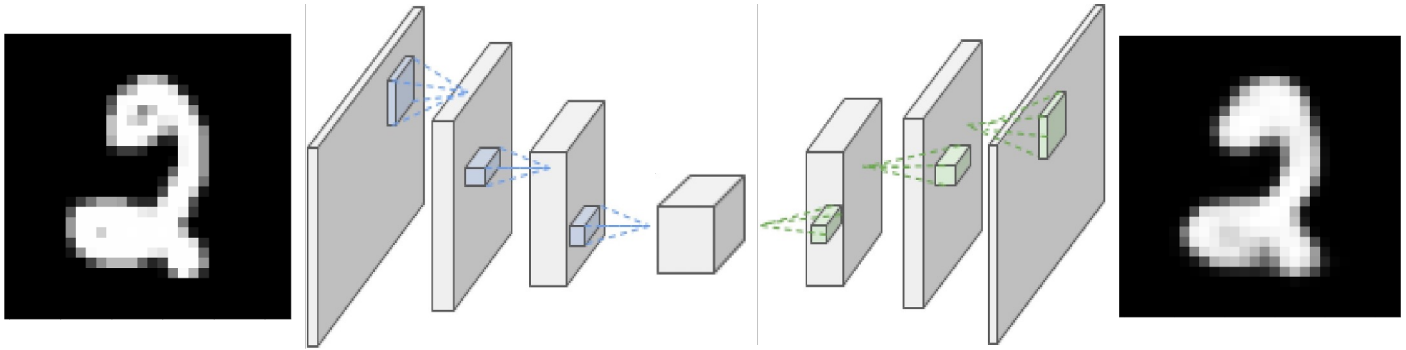


Autoencoder Encoder As Feature Representation



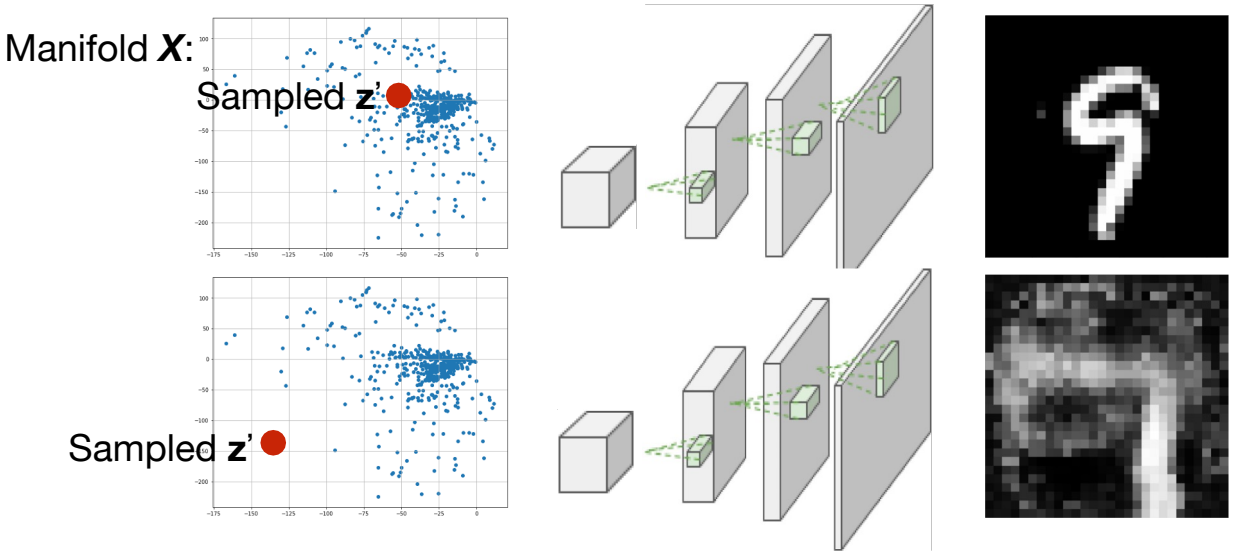
- Like word embeddings, autoencoder representations are compact manifolds for the input that capture salient information (in this case, what's needed to reconstruct)

Autoencoder Decoder As Data Generator



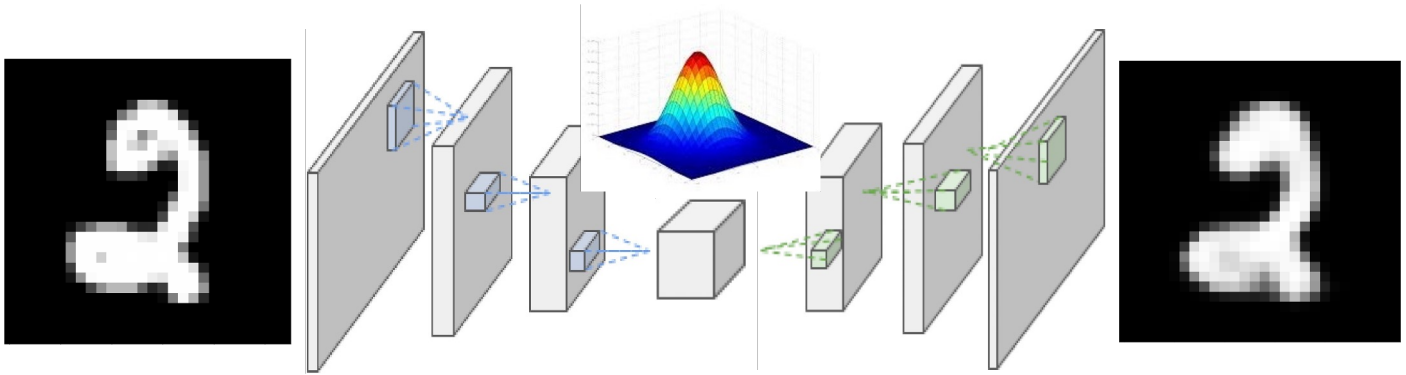
- What if we sample vector ***z'*** as draws from a Gaussian?
- Will we achieve reconstructions that look like our input data *X*?
  - No! Our data *X* encodes to a particular manifold in $R^{|z|}$

Autoencoder Decoder As Data Generator



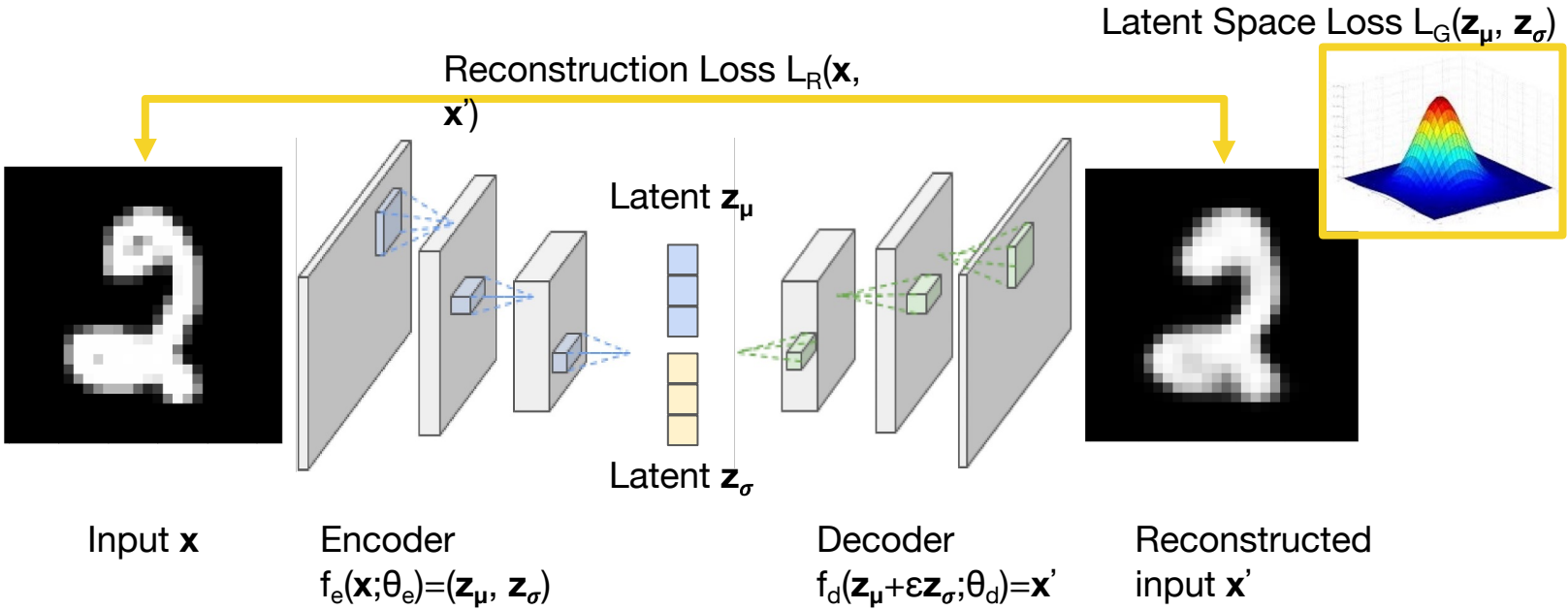
- What if we sample vector ***z'*** as draws from a Gaussian?
- Will we achieve reconstructions that look like our input data *X*?
  - No! Our data *X* encodes to a particular manifold in $R^{|z|}$

Autoencoder Decoder As Data Generator



- What if we *make* the latent manifold Gaussian?
- By reformulating the nature of the latent space and adding some additional regularizing loss, we can.

Variational Autoencoder



- We learn a mean and variance for the latent manifold for every input, then constrain those to be Gaussian

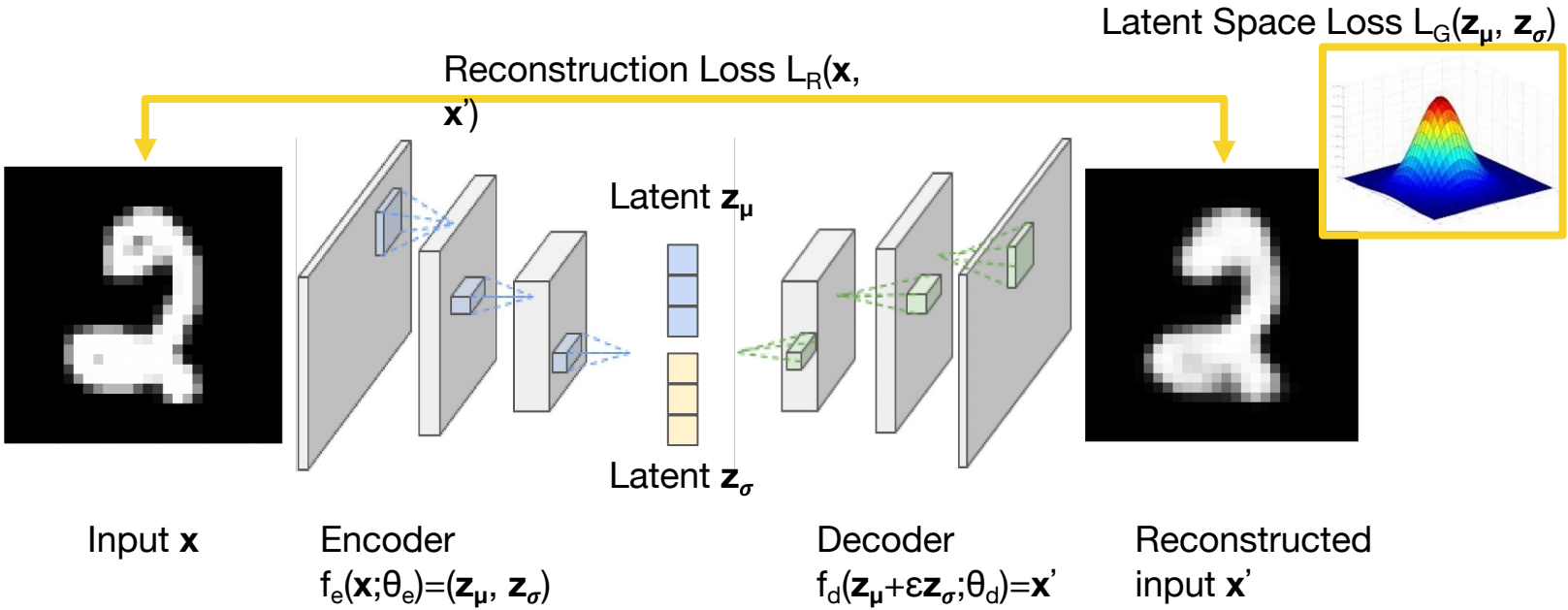
Variational Autoencoder

- Reconstruction Loss $L_R(\boldsymbol{x}, \boldsymbol{x}')$
  - Typically mean-squared error loss (for images)
  - For language, could be cross entropy over decoded tokens
  - Enforces that learned decoder function $f_d(\boldsymbol{z}_{\boldsymbol{\mu}}+\varepsilon\boldsymbol{z}_{\boldsymbol{\sigma}};\theta_d)$ is a good approximation of $p(\boldsymbol{x}|\boldsymbol{z})$ for $\boldsymbol{x}$ the original input and $\boldsymbol{z}$ the learned latent manifold vector obtained from $f_e(\boldsymbol{x})$

Variational Autoencoder

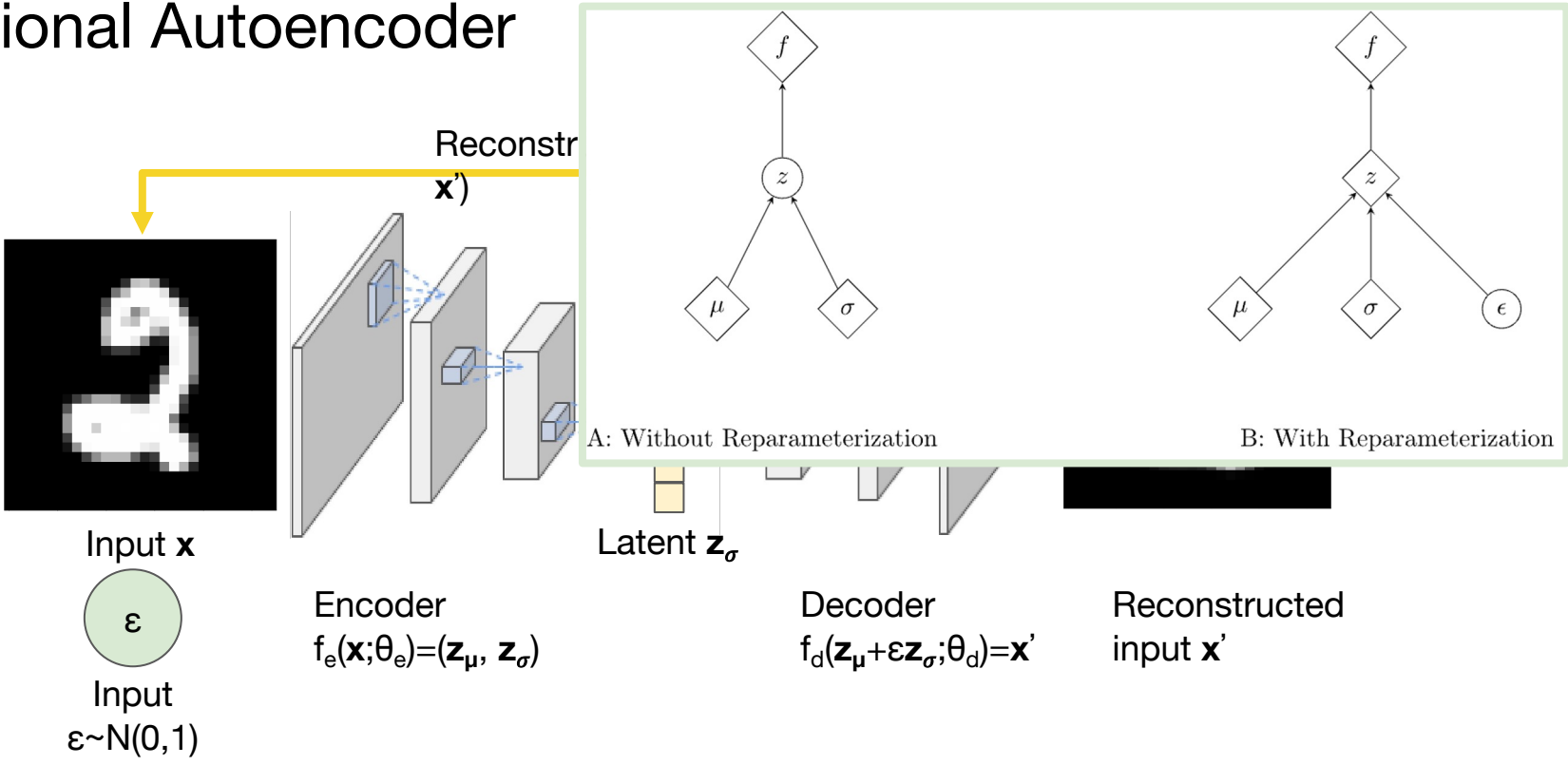
- Latent Space Loss $L_G(\boldsymbol{z_\mu}, \boldsymbol{z_\sigma})$
  - Typically KL divergence between encoder outputs $f_e(\boldsymbol{x};\theta_e)$ and the ideal Gaussian distribution we want latent space in
  - Note that $f_e(\boldsymbol{x};\theta_e)$ approximates $p(z|x)$, and the goal of this loss is to drive $p(z)$ towards a Gaussian, e.g., $N(0, 1)$
  - Without going into deep detail, the KL divergence of P from Q is the expected excess surprise from using Q as a model when the actual distribution is P.
    - $L_G(\mathbf{z_\mu}, \mathbf{z_\sigma}) = KL(N(\mathbf{z_\mu}, \mathbf{z_\sigma}), N(0, 1))$

Variational Autoencoder



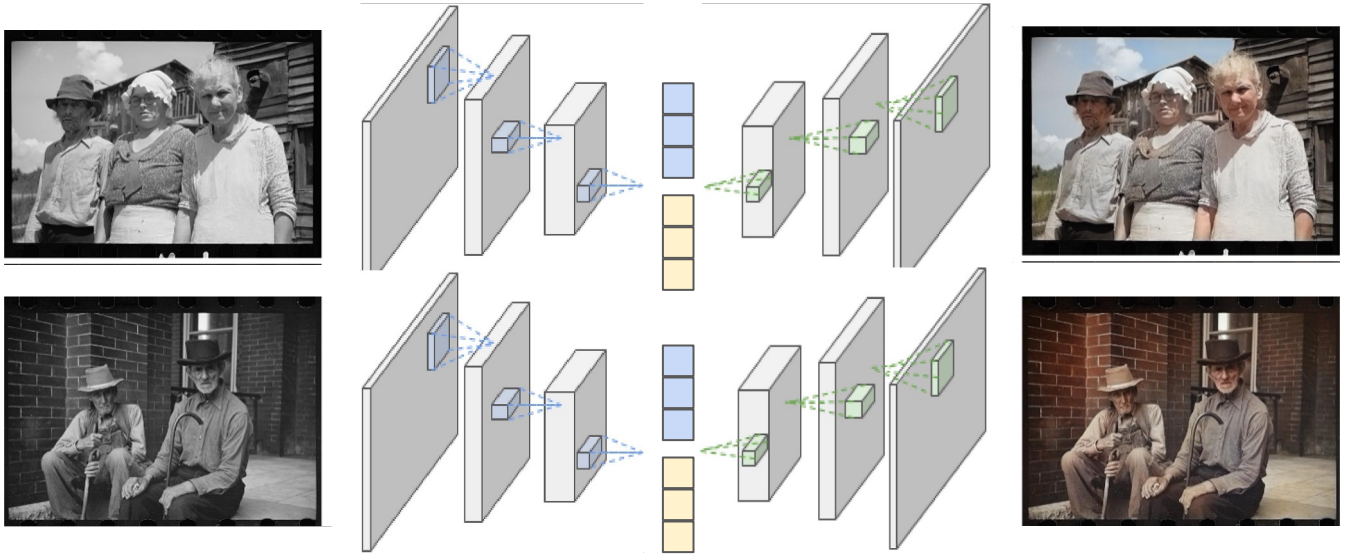
- We have one small hiccup yet.
- Does the above network define a function?

Variational Autoencoder



[The Reparameterization Trick in Variational Autoencoders]

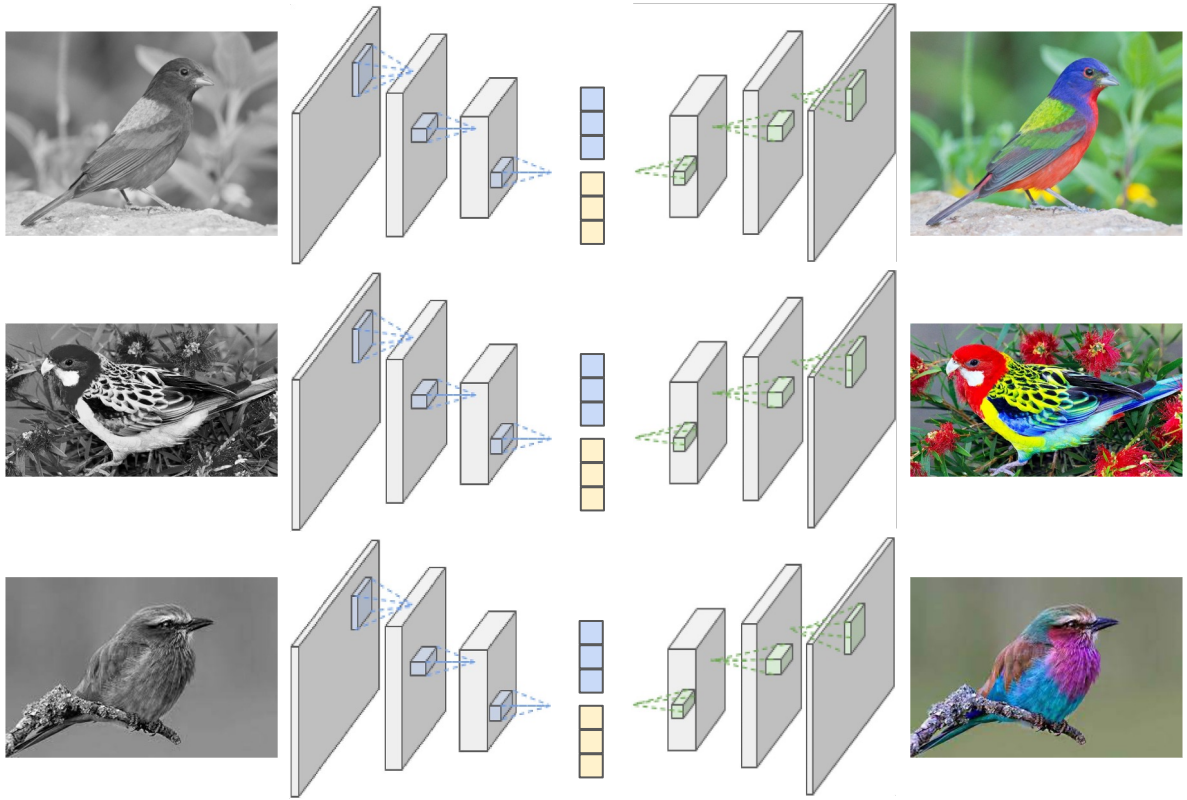
Autoencoder Decoder For Downstream Tasks



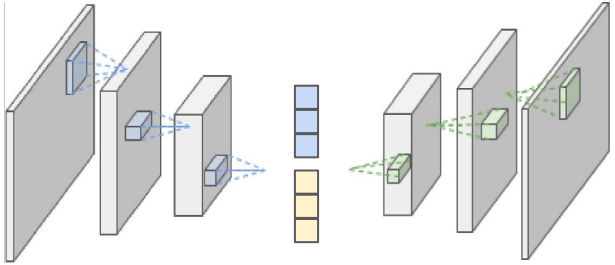
- Initialize decoder from autoencoder process
- Fine tune to recover *missing* information, such as grayscale (single channel) to RGB (3 channel)

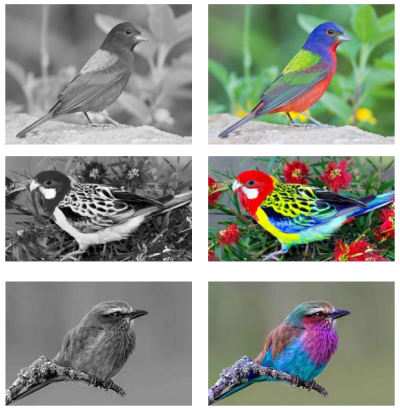
https://developer.nvidia.com/blog/automatic-image-colorization-of-grayscale-images/

Autoencoder Decoder For Downstream Tasks



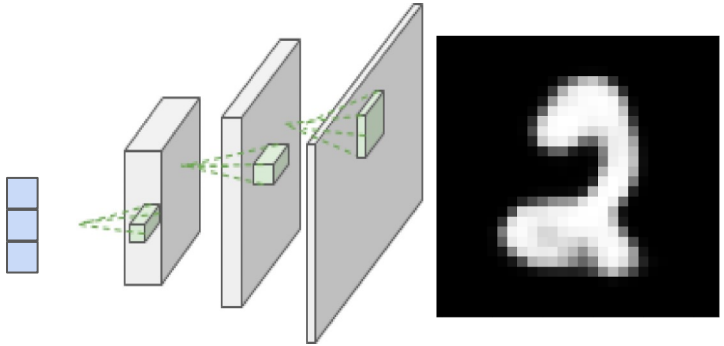
Variational Autoencoder Mode Collapse





- Minimizing reconstruction loss means defaulting to *averages* for high variance tasks like “coloring a bird”
- Average minimizes the loss but never gets the right color!
- How can we penalize taking these “safe bet” local minima?

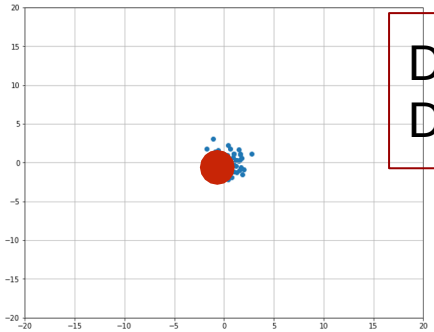
Generative Adversarial Networks



Rand **z**

Decoder
$f_d(\mathbf{z};\theta_d)=\mathbf{x}'$

Upsampled **x**'



Discriminator D(**x'**) -> 0
Discriminator D(**x**) -> 1

Rand *N(0, 1)*

Generator f(**z**) -> **x'**

Generator Network

Gen Images

Train Images

Discriminator Network D

Discriminator Network D

Real/Fake Decision

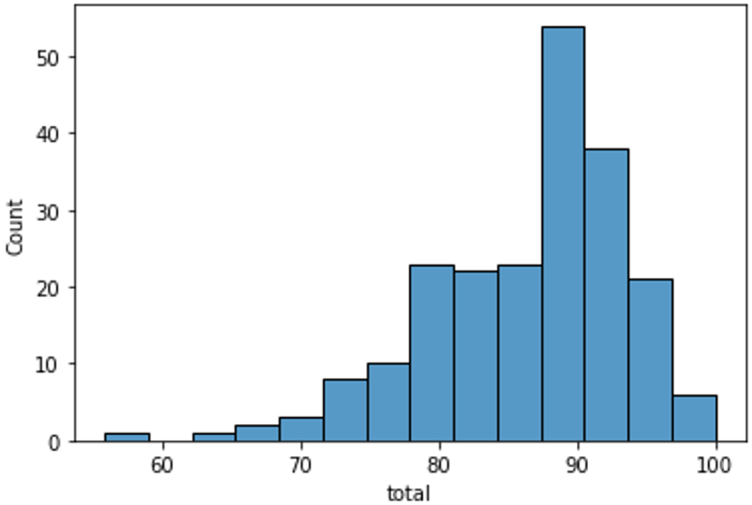
Real/Fake Decision

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Transformers~~
- ~~DL for Computer Vision~~
  - Any questions before we move on?
- Midterm Debrief

Midterm

- (μ=86.5; σ=7.2); median=88.8



- Today, we'll go over the 10 most popular incorrect answers.

Midterm: Most Missed Questions

25. (4 points) Which of the following are inductive biases used in unidirectional, forward RNNs? (**Multiple Answers Possible**)

A. To encode an input at time t, we need only to know the input at time $t-1$; that is, RNNs can be thought of as only "looking back" one step.

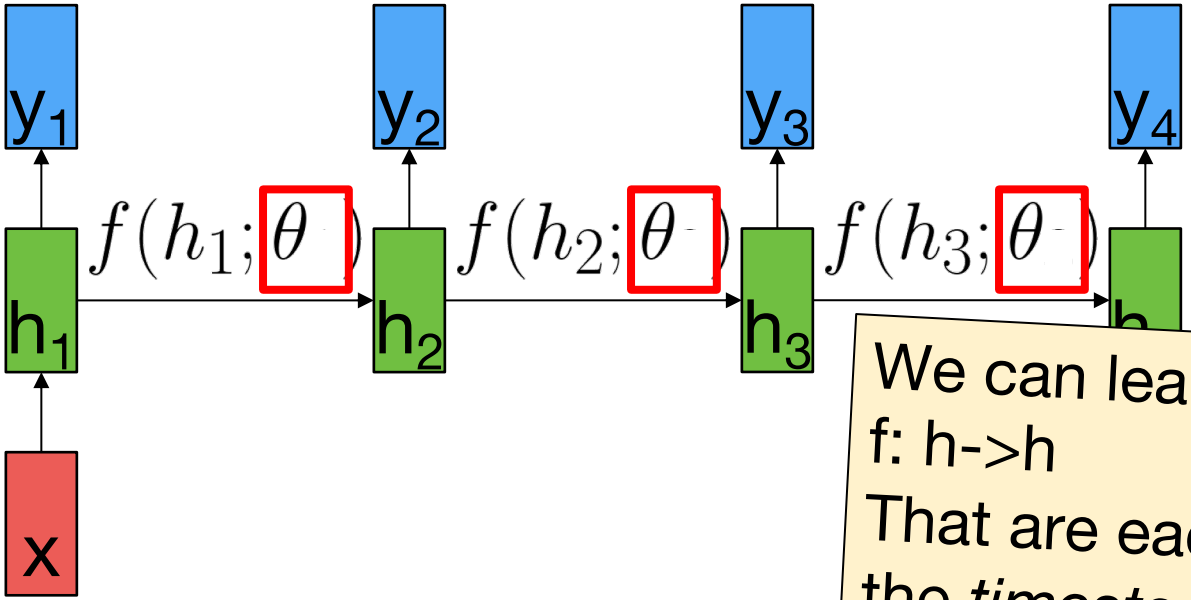
B. Tokens in a sequence are influenced by all tokens occurring earlier in the sequence.

C. Tokens should be encoded independent of their position in the sequence as input to the RNN.

D. Tokens in a sequence can be treated as input to the RNN without respect to their original order.

E. The underlying state of an encoded sequence can be updated one token input at time.

Sequence Modeling with Learned Neural Functions [Lecture 5]



We can learn functions:
f: h->h **f: x->h** f: h->y
That are each independent of the *timestep* dimension.

Midterm: Most Missed Questions

25. (4 points) Which of the following are inductive biases used in unidirectional, forward RNNs? (**Multiple Answers Possible**)

A. To encode an input at time t, we need only to know the input at time $t-1$; that is, RNNs can be thought of as only "looking back" one step.

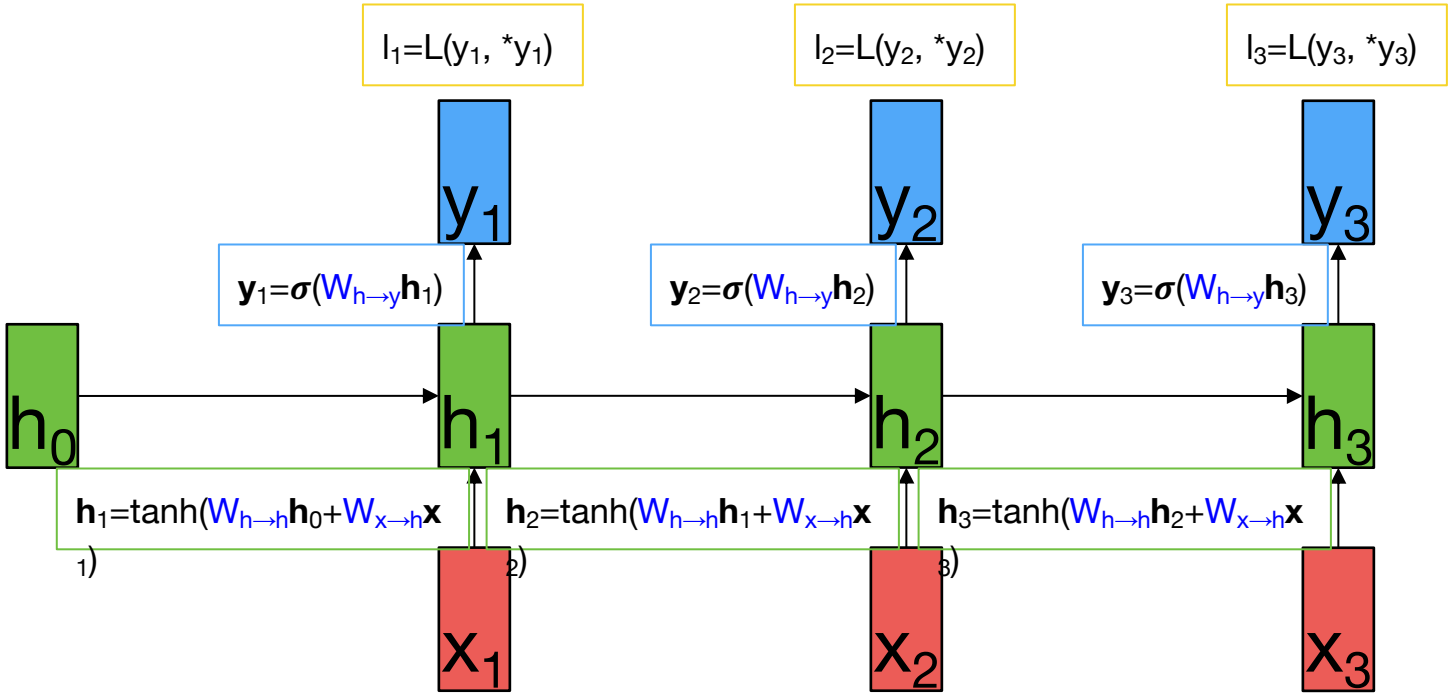
B. Tokens in a sequence are influenced by all tokens occurring earlier in the sequence.

C. Tokens should be encoded independent of their position in the sequence as input to the RNN.

D. Tokens in a sequence can be treated as input to the RNN without respect to their original order.

E. The underlying state of an encoded sequence can be updated one token input at time.

High Level Computation Graph for RNN [Lecture 5]



Midterm: Most Missed Questions

22. (4 points) **Select all** of the following statements that are true about Recurrent Neural Networks (RNNs). (**Multiple Answers Possible**)

A. RNNs can take variable-length input sequences.

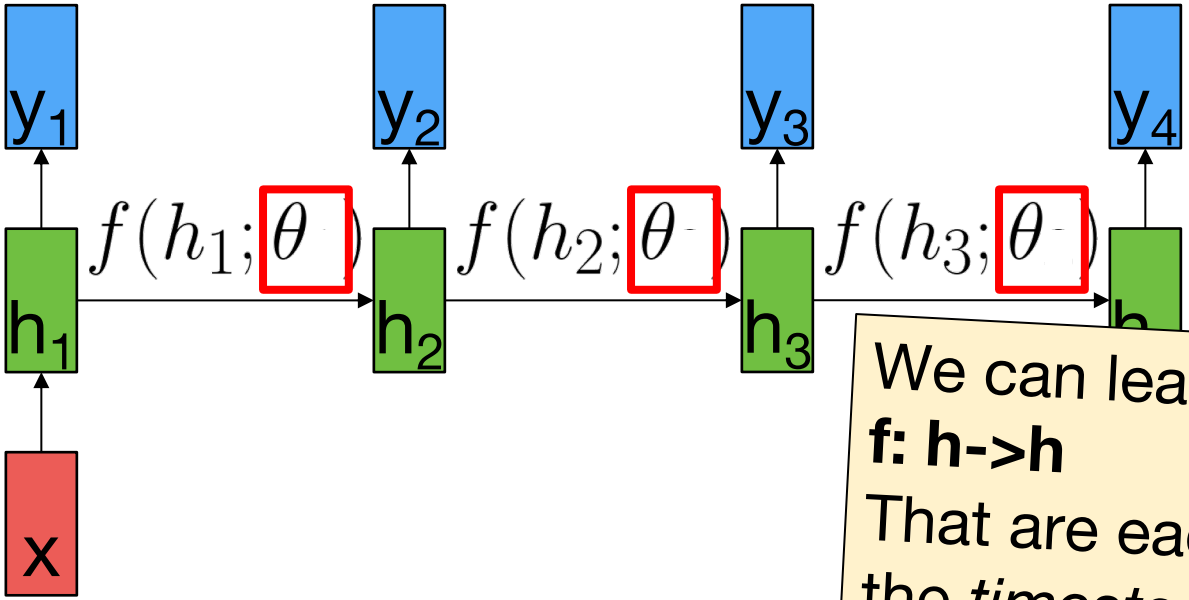
B. Typically, RNNs learn transition functions that are positionally dependent, achieving better accuracy at the cost of more parameters compared to processing a sequence with a CNN filter.

C. RNNs cannot be used with CNN output since the flattening operation on the CNN output tensor would be non-differentiable.

D. Due to the vanishing gradient problem, RNNs can only be trained with truncated backpropagation.

E. RNNs can be applied to text, speech, vision, and time series data.

Sequence Modeling with Learned Neural Functions [Lecture 5]



We can learn functions:
f: h->h f: x->h f: h->y
That are each independent of the *timestep* dimension.

Midterm: Most Missed Questions

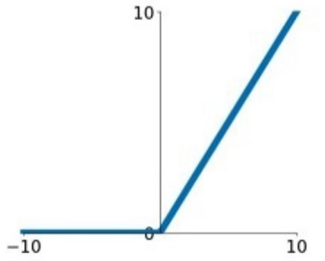
7. (4 points) Which of the following is true? (**Multiple Answers Possible**)
   A. The loss function will always decrease after a parameter update when performing Stochastic Gradient Descent.
   B. During backpropagation, as the gradient flows backwards through any of sigmoid/tanh/ReLU non-linearities, the gradient in each dimension cannot change sign.
   C. If the training loss rapidly increases with each gradient step in SGD, one possibility is that the learning rate is too high.
   D. Dropout, L2 weight penalties, and Xavier initialization are all examples of parameter regularization techniques that encourage generalization under distribution shift.

Activation Functions [Lectu

These activation functions are monotonically non-decreasing; their gradient is never negative.

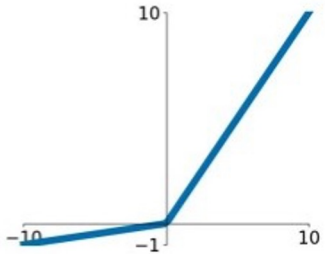
ReLU

$\max(0, x)$



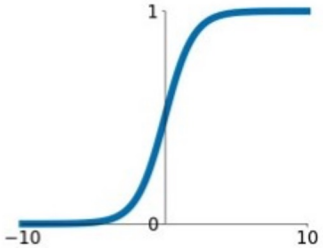
Leaky ReLU

$\max(0.1x, x)$



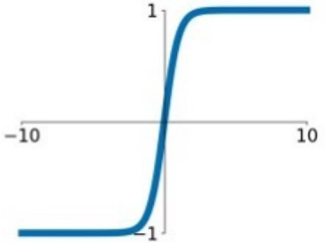
Sigmoid

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



tanh

$\tanh(x)$



Midterm: Most Missed Questions

7. (4 points) Which of the following is true? (**Multiple Answers Possible**)
   A. The loss function will always decrease after a parameter update when performing Stochastic Gradient Descent.
   B. During backpropagation, as the gradient flows backwards through any of sigmoid/tanh/ReLU non-linearities, the gradient in each dimension cannot change sign.
   C. If the training loss rapidly increases with each gradient step in SGD, one possibility is that the learning rate is too high.
   D. Dropout, L2 weight penalties, and Xavier initialization are all examples of parameter regularization techniques that encourage generalization under distribution shift.

Xavier (Glorot) initialization is not a parameter regularization technique; the motivation of Xavier and He initializations are related to training convergence, not to generalization performance [Lecture 4]

Midterm: Most Missed Questions

10. (4 points) Which of the following statements about various neural network layers is **true**? (**Multiple Answers Possible**)

A. A perceptron, or fully-connected layer, lacks the ability to capture class imbalance information in training data.

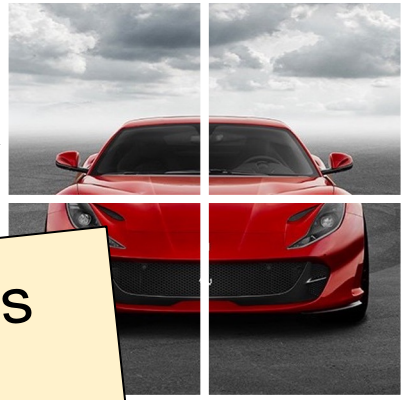
B. A multi-layer perceptron can consider the interactions between multiple input features and an output class.

C. A convolutional layer is best applied when the input to that layer contains local structure.

D. A pooling layer can only be applied to 2-dimensional or higher input data.

E. A vanilla RNN cell can be formulated as a fully-connected layer operation, half of whose inputs at each timestep from previous output and half from an input sequence.

Linear Image Classification [Lecture 2]



10	150

2x2 numbers

(4 numbers)

W: how much each input feature indicates each class (e.g., p(y|x))

each class is alone (e.g., p(y))

0.2	-0.3	0	0.6

Great, so “Car” is the highest number. How do we get **W** and **b** though?

150		0.2		96.4	Cat
7	+	-1	=	212.8	Car
232		-0.9		181.1	Airplane

W x b f(x; W,b)

Midterm: Most Missed Questions

18. (4 points) Which of these options is an appropriate expression for the multiplicative mask on input values created by dropout during training and testing? Denote the probability of *dropping units* as p and the function that performs random dropping of units on input x as $d(x, p)$. (**Multiple Answers Possible**)

A. Training: $\frac{d(x,p)}{1-p}$, Testing: $\frac{d(x,p)}{1-p}$

B. Training: $\frac{d(x,p)}{p}$, Testing: $\frac{d(x,p)}{p}$

C. Training: p, Testing: $\frac{d(x,p)}{p}$

D. Training: $\frac{d(x,p)}{1-p}$, Testing: 1

E. Training: $d(x, p)$, Testing: $1 - p$

Need E[x] at train/test time to be the same. [Lecture 4]

D. “inverted dropout” scales train activations up.

E. “scaling activation” scales inference activations down.

Same effect! This one required some thought, rather than lookup directly.

Midterm: Most Missed Questions

27. (4 points) Which of the following are **true** regarding the use of attention in RNNs? (**Multiple Answers Possible**)

A. Attention can be used when the final hidden state from an encoder RNN may not contain all the information needed to decode the target sequence.

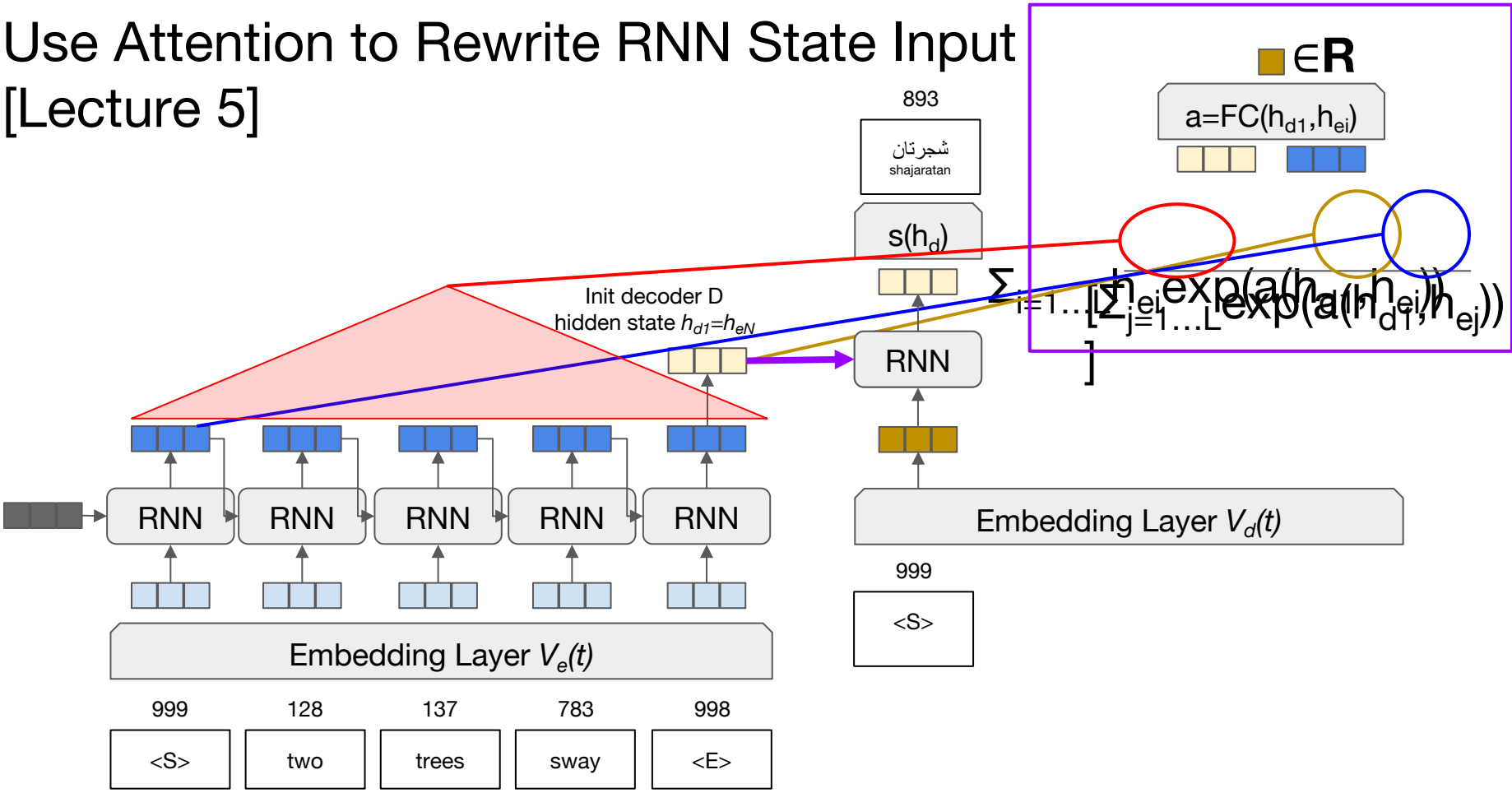
B. Attention can help with gradient flow in RNN training since it gives a more direct connection from loss function to encoder network units.

C. Global soft attention selects the most relevant hidden state from among the globally available options in the encoder and conditions the next decoder hidden state exclusively on that most relevant encoder state.

D. In multi-headed attention, k decoder RNNs are learned in parallel, each served by one of k attention heads.

E. Local attention restricts the considered encoder states at each decoder timestep t to a neighborhood in the encoder sequence corresponding to t.

Use Attention to Rewrite RNN State Input [Lecture 5]



Midterm: Most Missed Questions

23. (4 points) Which of the following statements about RNNs are false? (**Multiple Answers Possible**)

A. Bidirectional RNNs are most appropriate when the output sequence needs to be produced one element at a time as the input sequence is observed.

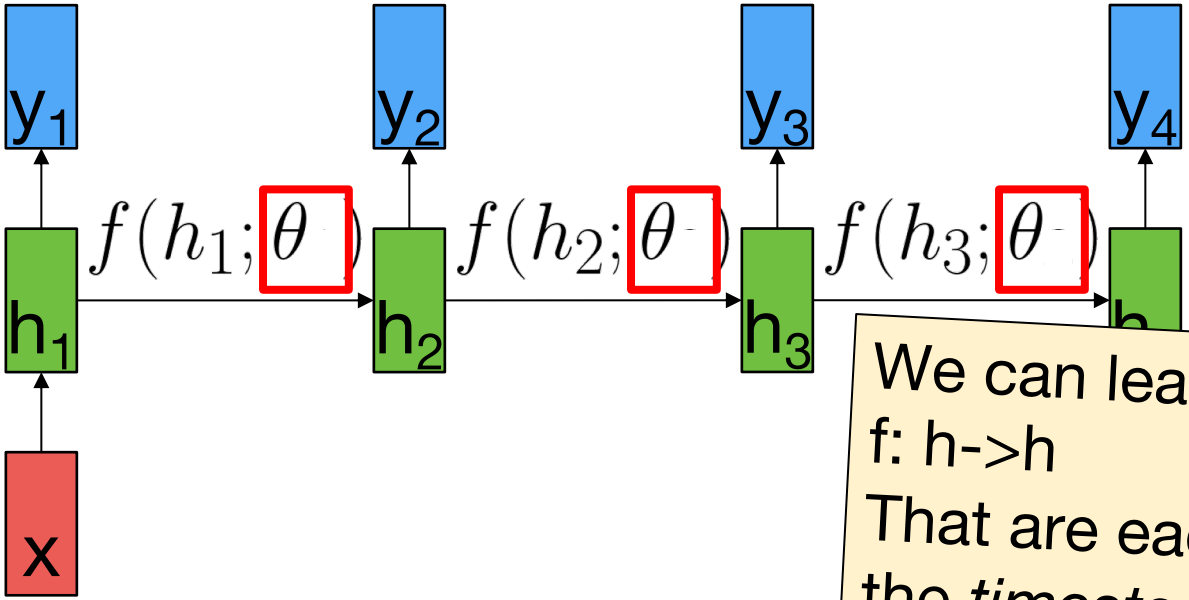
B. The number of parameters in a vanilla RNN has no relationship to the number of input elements in the sequences over which it operates.

C. The output of each RNN unit can be used as input to a feedforward network.

D. Generally, RNNs are harder to train when tasks involve long-range dependencies in the input sequence.

E. The number of input tokens in an RNN input sequence cannot be greater than the number of parameters in the RNN hidden state.

Sequence Modeling with Learned Neural Functions [Lecture 5]

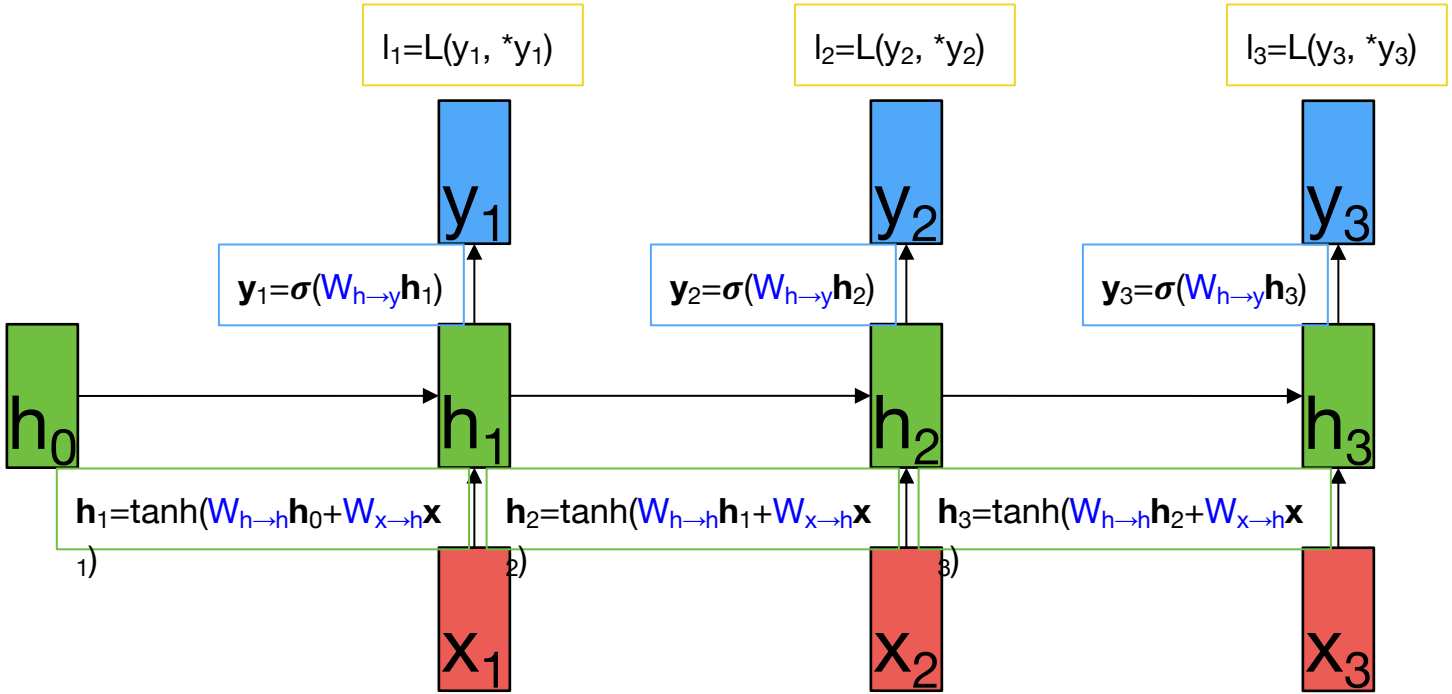


We can learn functions:
f: h->h f: x->h f: h->y
That are each independent of the *timestep* dimension.

Midterm: Most Missed Questions

13. (4 points) **Select all** of the following statements that are **true** about loss functions for neural networks. (**Multiple Answers Possible**)
    A. Loss functions can contribute to model parameter regularization.
    B. Lower training loss strictly implies lower validation loss.
    C. Loss functions are not part of the computation graph for backpropagation, and need not be differentiable.
    D. Cross entropy loss is calculated over the vector output of a softmax activation.
    E. In order to preserve gradient flow, all loss functions used during a single step of SGD must be applied on the same output vector.

High Level Computation Graph for RNN [Lecture 5]



Action Items for You

- Your project surveys are due **tonight** at 11:59pm
- We have released Coding Assignment 2; details on Piazza
  - CA2 is due in *three weeks* on March 31, 11:59pm
- There is no class next week; enjoy spring break!

CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 7: Transformers and DL for Vision