# CS 189/289

Today:

1. Recurrent Neural Networks
2. Attention and Transformers

Assigned reading:

- 5.3 intro (softmax reminder)
- 12.1 (attention & transformers), i.e., 12.1-12.9

*Slides adapted from Prof. Levine's CS 182, slides and lectures, here.*

# CS 189/289

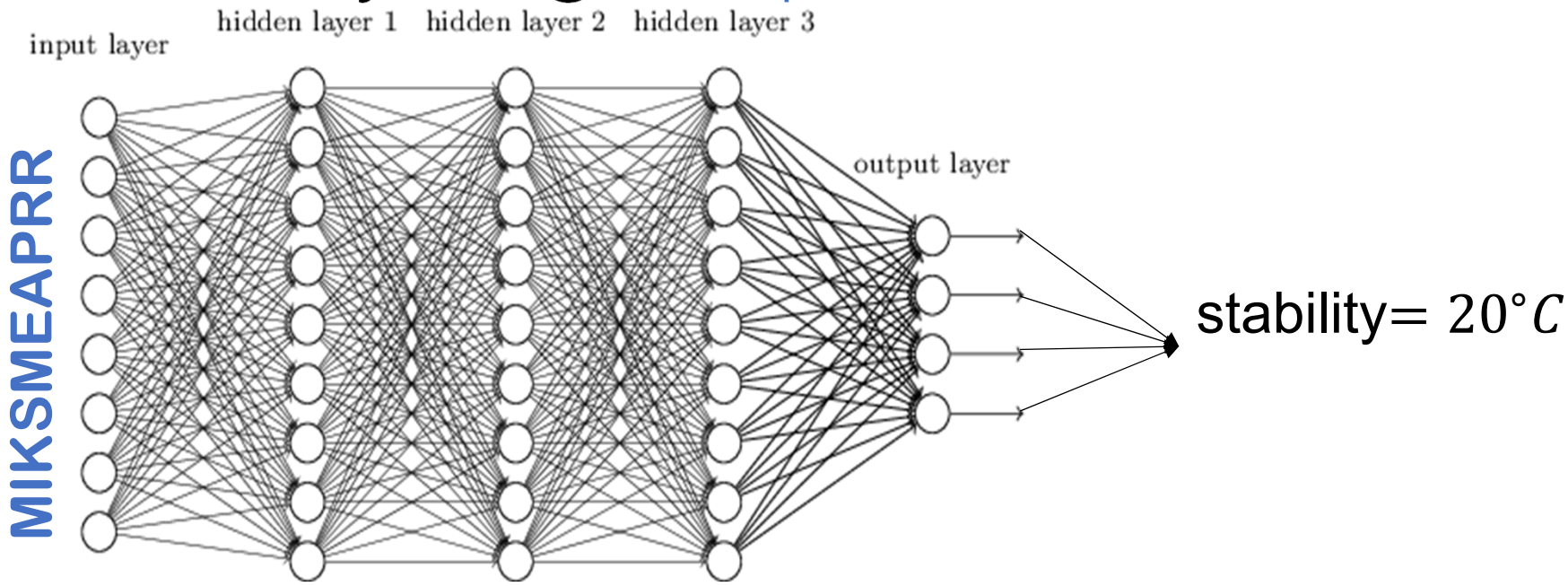
Today:

1. Recurrent Neural Networks
2. Attention and Transformers

*Slides adapted from Prof. Levine's CS 182, slides and lectures, here.*

# How to handle arbitrary length inputs?

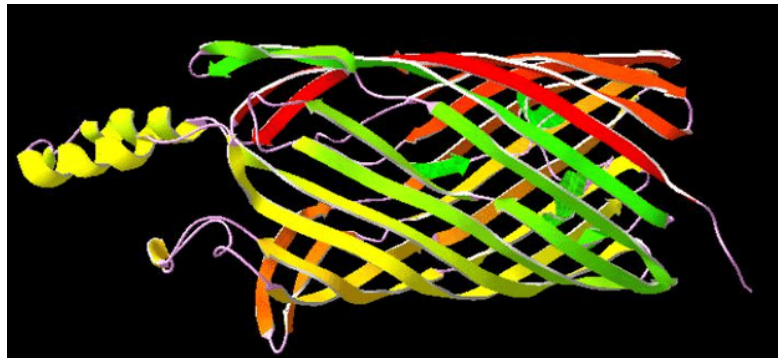
*e.g.* predict scalar property from protein sequence




stability = $20°C$

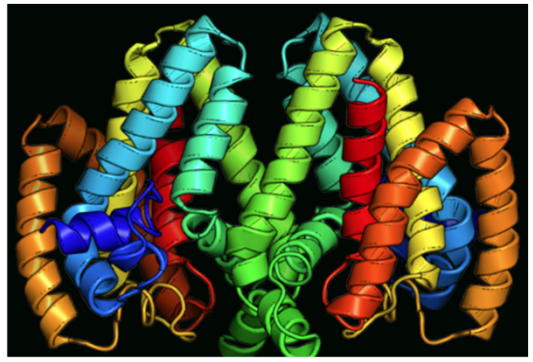
MIKSMEAPRR

stability = $42°C$



ALKELIKSANVIALIDMME

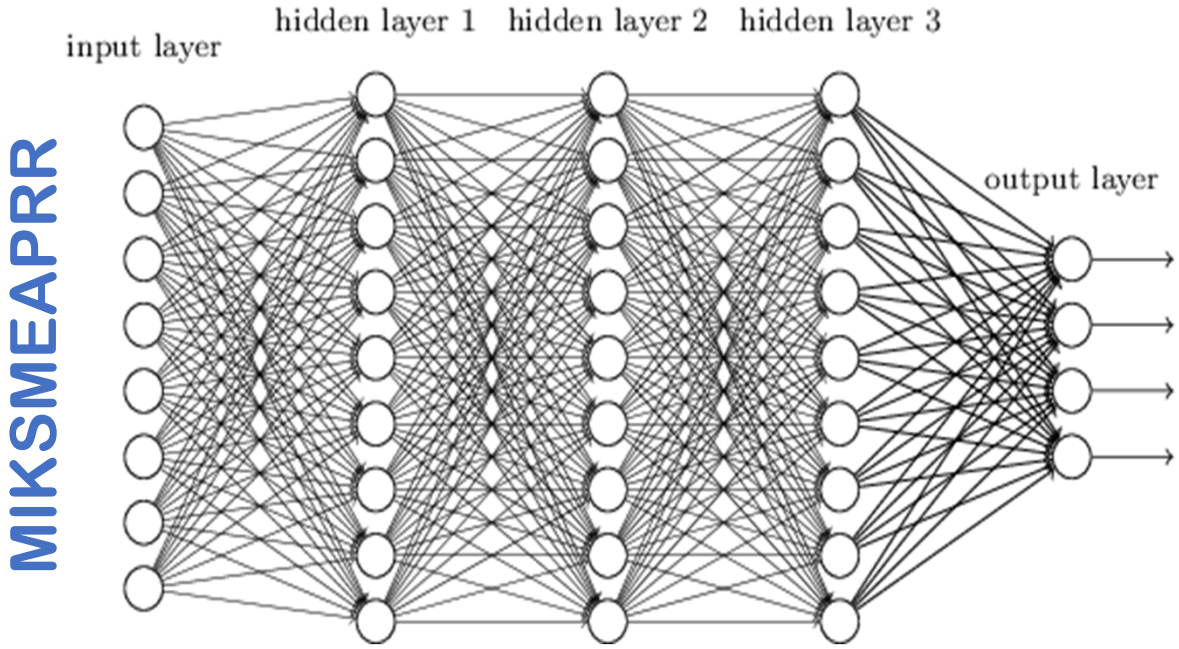
stability = $36°C$



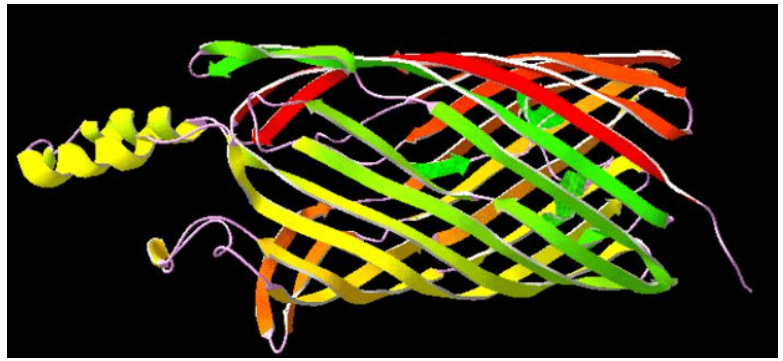
TCAGVLWYFHD

# How to handle arbitrary length inputs-outputs?

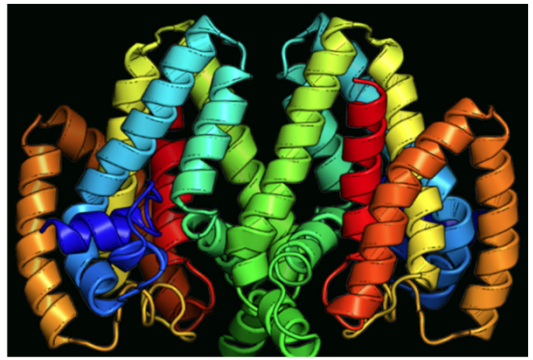
*e.g.* protein structure prediction




MIKSMEAPRR



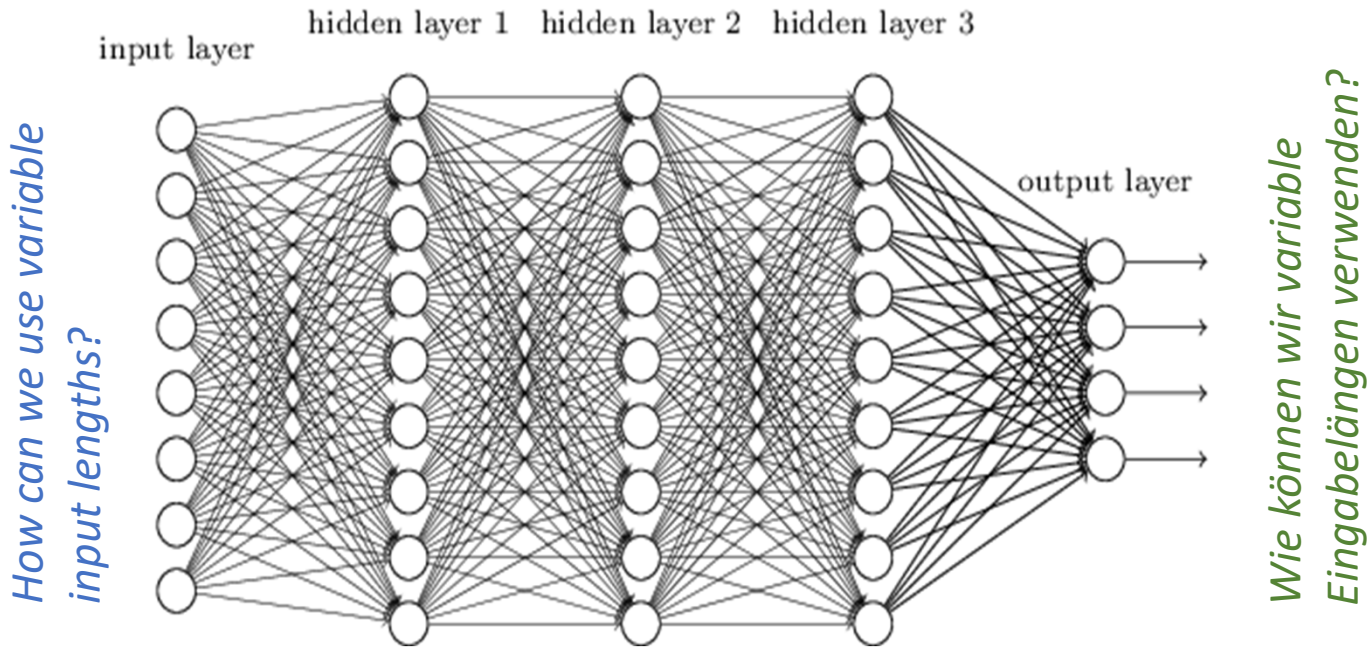
ALKELIKSANVIALIDMME



TCAGVLWYFHD

# How to handle arbitrary length inputs and outputs?

*e.g.* language translation




*Wie können wir variable Eingabelängen verwenden?*

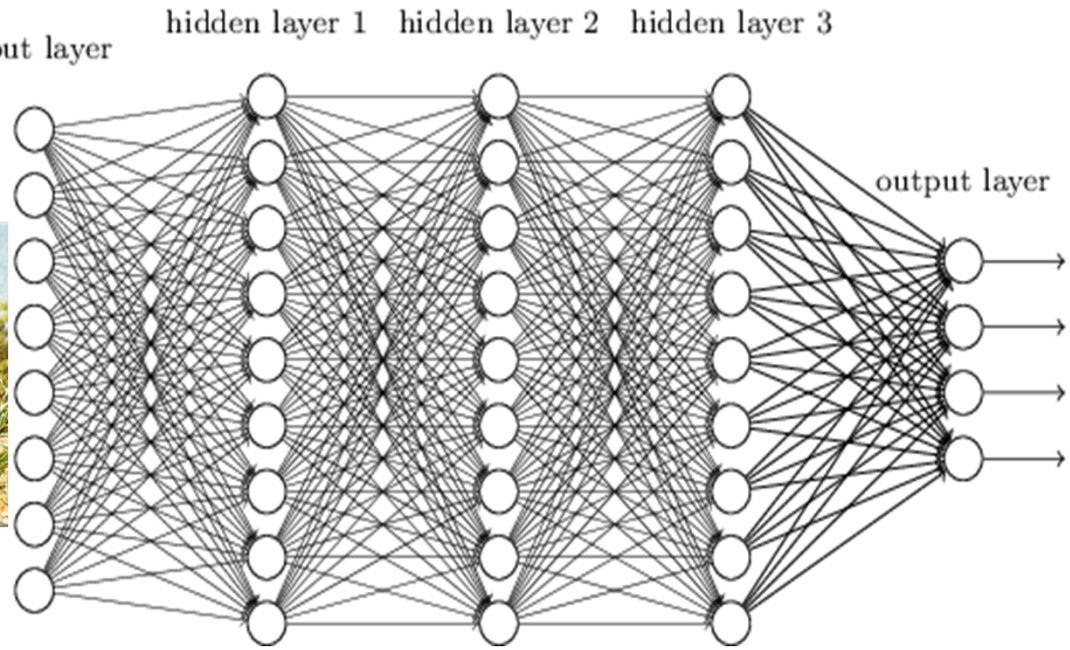
*How can we use variable input lengths?*

*Neuronale Netze verwenden nur Eingaben fester Länge*

*Neural networks only uses fixed length inputs*

# How to handle arbitrary length inputs and outputs?

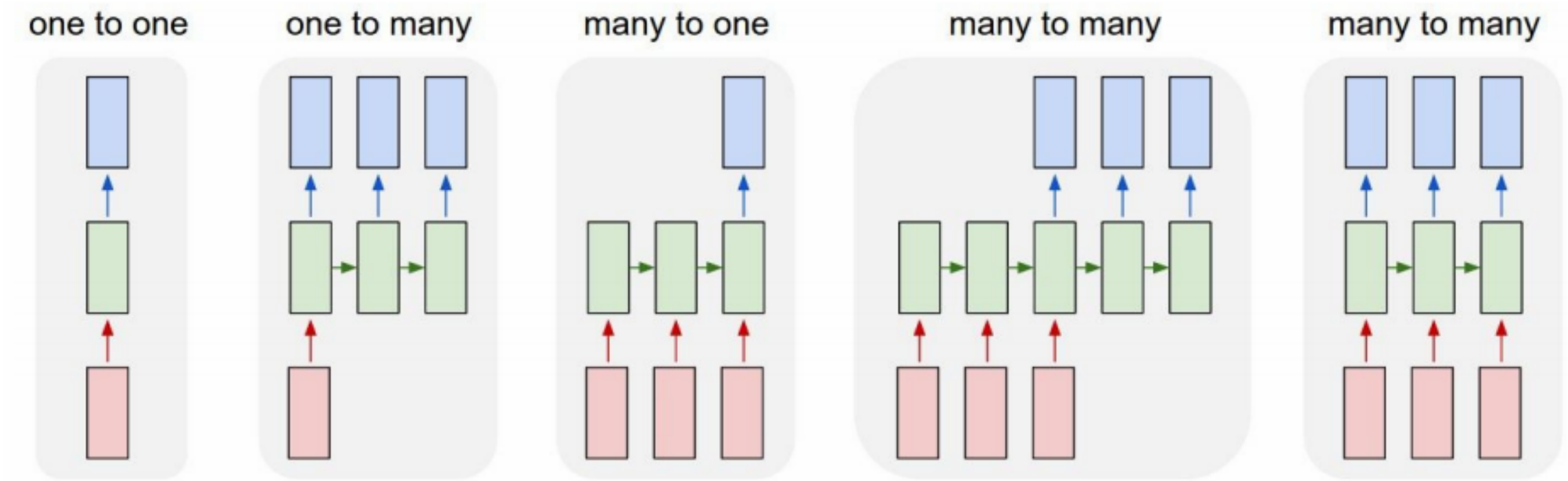
*e.g.* image captioning




A dog is sitting on the beach next to a dog.



# Generally called sequence-to-sequence models.




e.g., image captioning

e.g., activity recognition

e.g., machine translation

e.g., frame-level video annotation

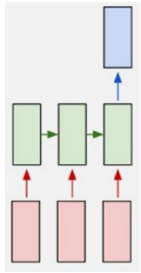
Image: Andrej Karpathy

# CS 189/289

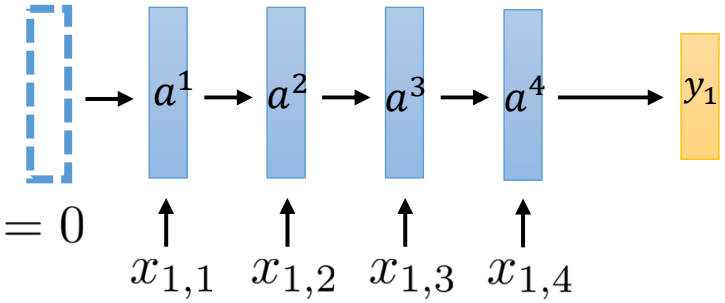
Today:

1. Recurrent Neural Networks
2. Attention and Transformers

# First, consider only variable-length inputs



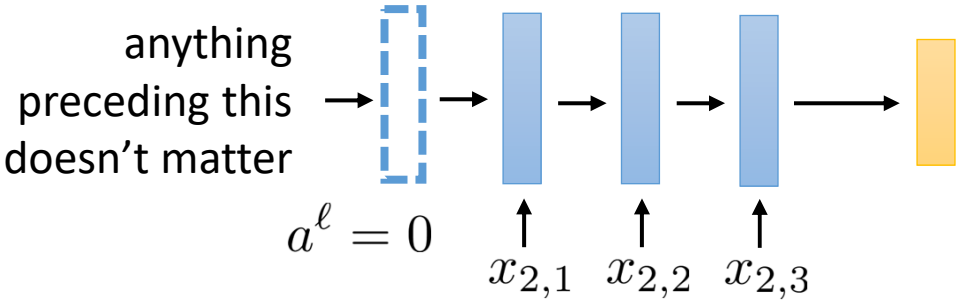
$$x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4})$$
$$x_2 = (x_{2,1}, x_{2,2}, x_{2,3})$$
$$x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5})$$



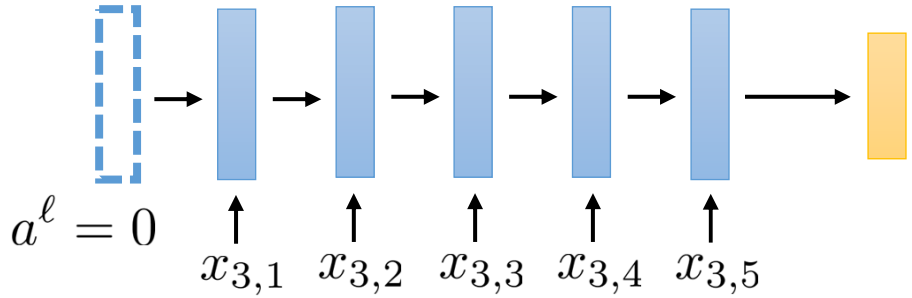

each layer:

$$\bar{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix}$$
$$z^\ell = W^\ell \bar{a}^{\ell-1} + b^\ell$$
$$a^\ell = \sigma(z^\ell)$$

Can we use one input variable per layer?




Obvious question: what happens to the missing layers?




Problem:

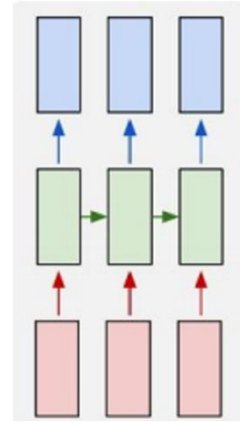
- #of $W_l$ increases with max sequence length!
- for small $l$ few samples to train with.

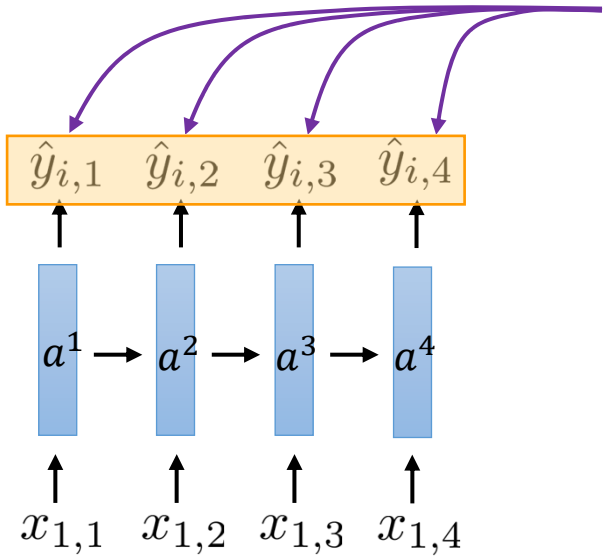
Fix: tie layer parameters:

- $W^l = W$ (and $b^l = b$)
- *Recurrent Neural Network*

$a^l$ is the running "memory" of the system

# Variable # inputs and outputs






each input gets its own output

$$\bar{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix}$$

$$z^\ell = W\ \bar{a}^{\ell-1} + b$$

$$a^\ell = \sigma(z^\ell)$$

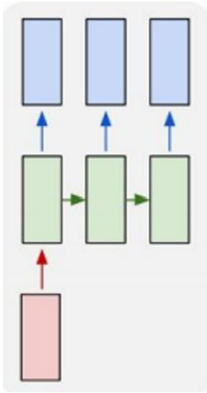
just like before

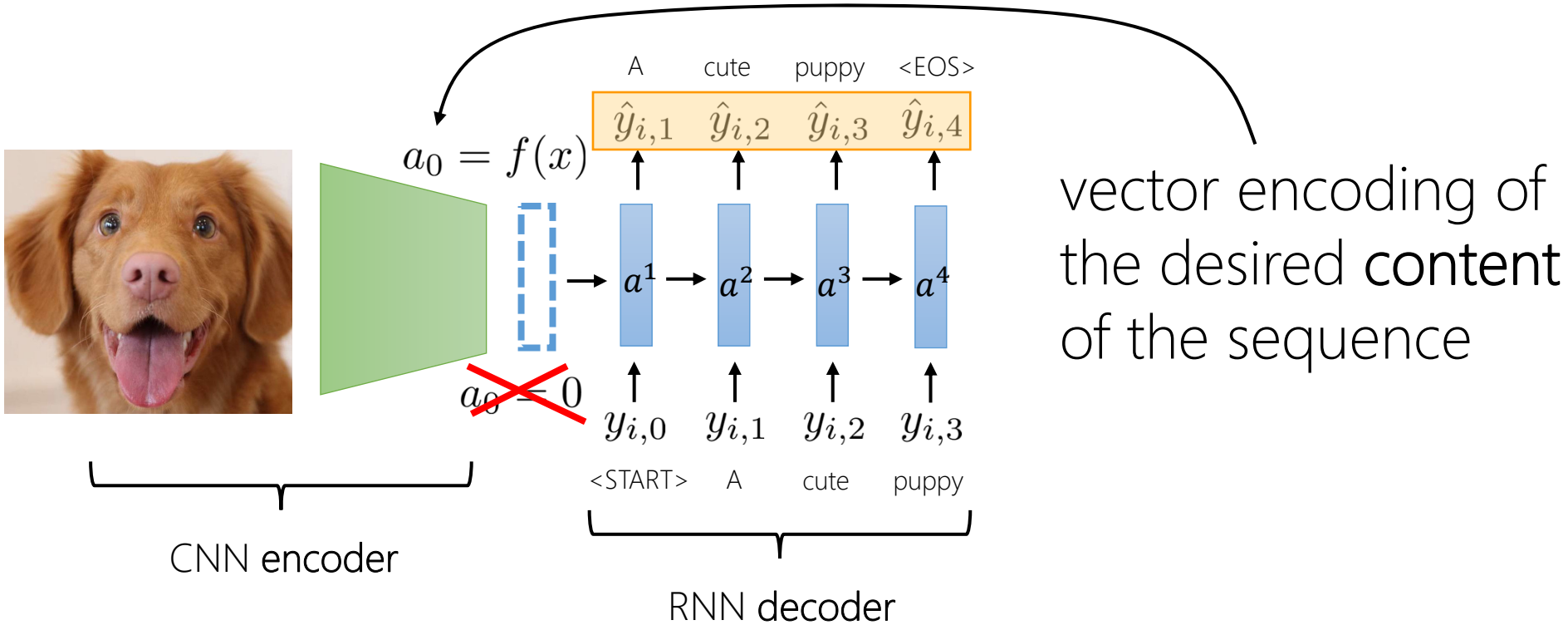
$$\hat{y}_\ell = f(a^\ell)$$

some kind of readout function, a "decoder"

A more general *Recurrent Neural Network*

# An image-conditional model

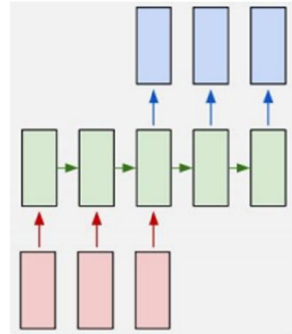


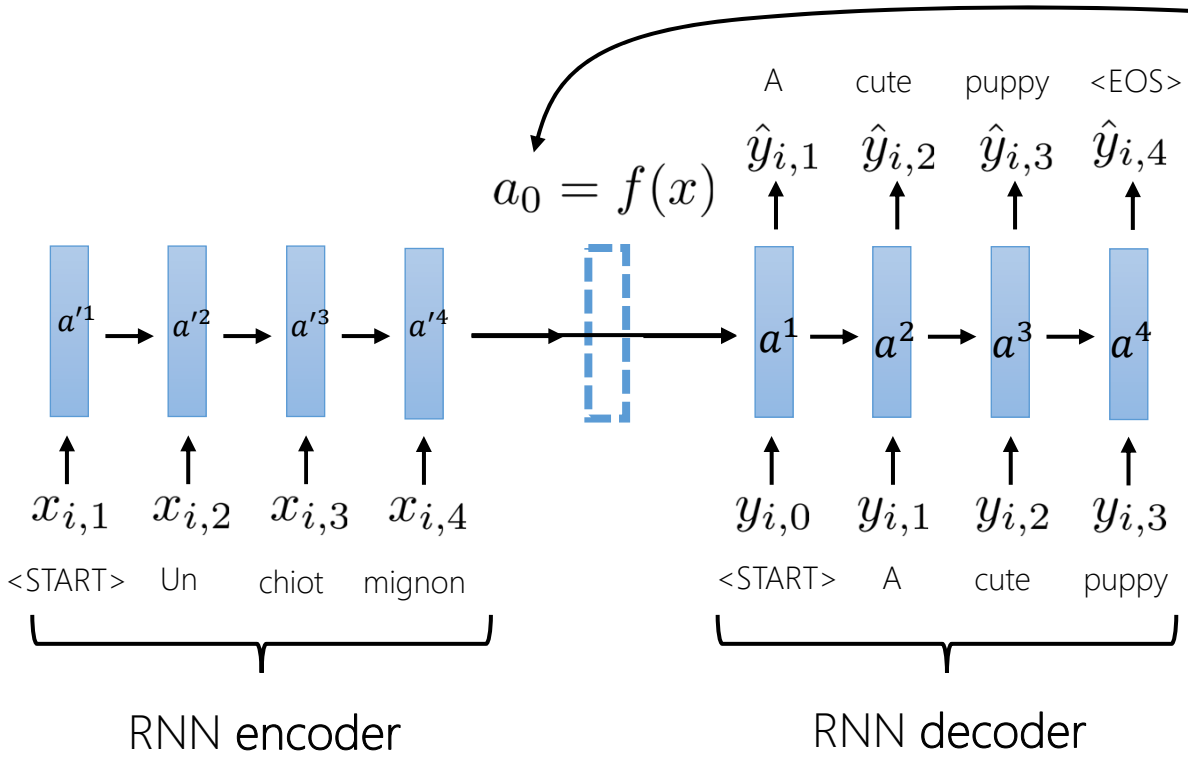



vector encoding of the desired **content** of the sequence

This is an *autoregressive* generative model: we generate each new word, $\hat{y}_{i,j}$, one at a time, having fed in the previous ones, $y_{i,0:j-1}$

# What if we condition on *another* sequence?

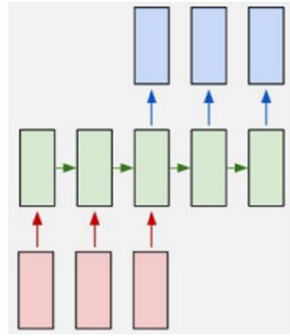


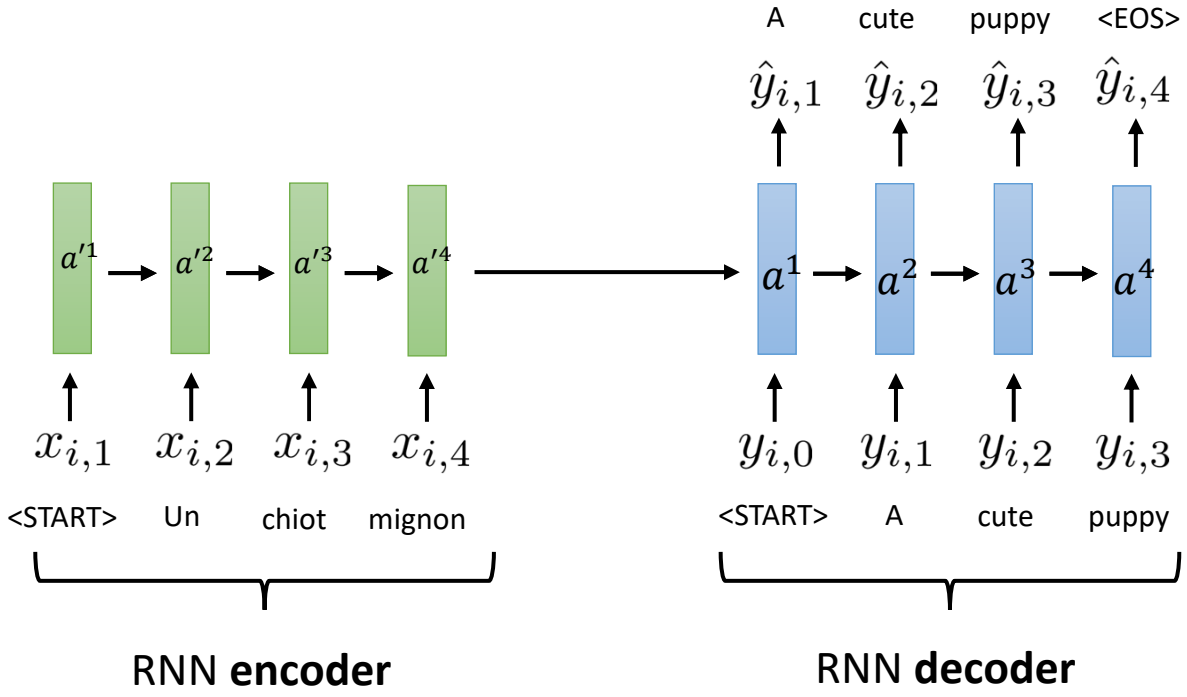



vector encoding of the desired **content** of the sequence

This is an *autoregressive* generative model: we generate each new word, $\hat{y}_{i,j}$, one at a time, having fed in the previous ones, $y_{i,0:j-1}$

# Sequence to sequence models

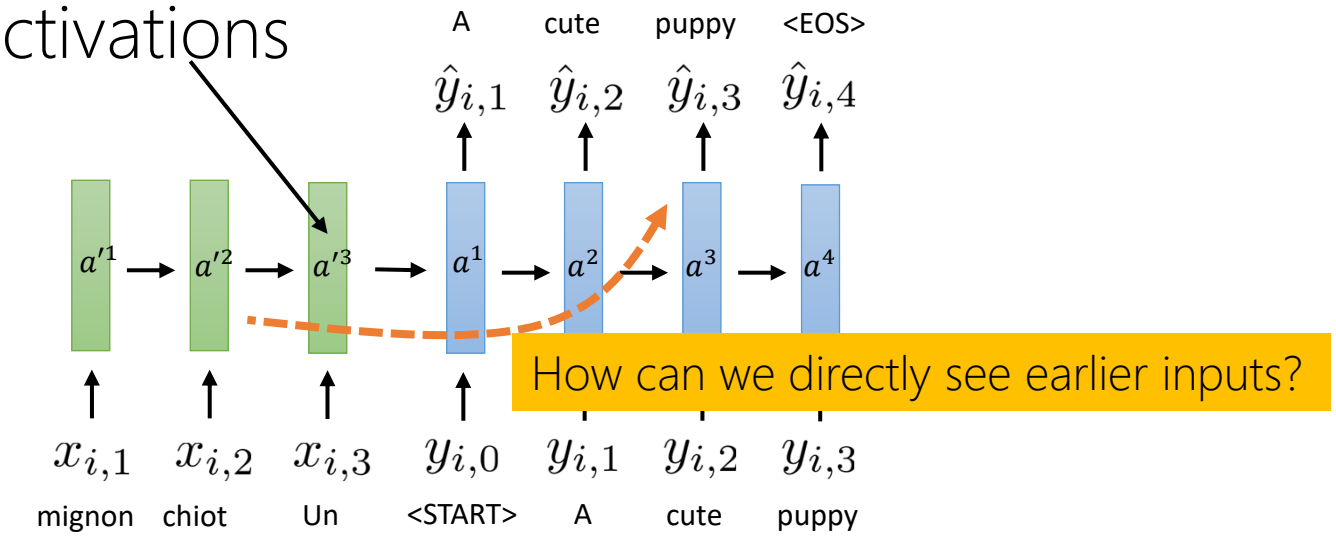





- Two separate RNNs: encoder & decoder
- Trained **end-to-end** on paired data (*e.g.*, pairs of French & English sentences)
- Likelihood/cross-entropy loss, summing over each decoded word, in each sentence.

# RNN bottleneck problem

all information about the conditioned sequence is contained in these activations




Attention to the rescue!

# CS 189/289

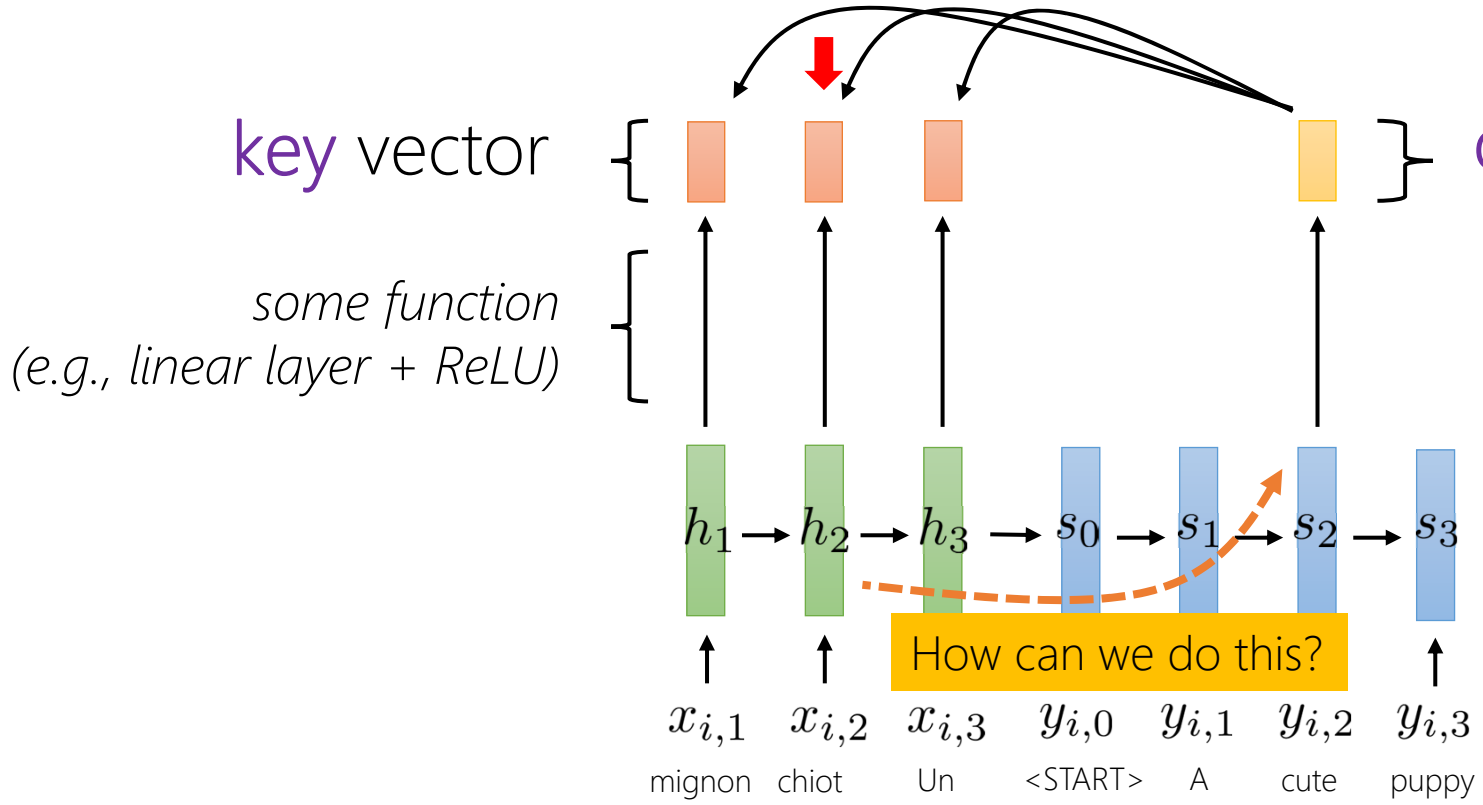
Today:

1. Recurrent Neural Networks
2. Attention and Transformers

# Attention overview

compare query to each key to find the closest one to get the right value

Keys and queries are **learned.**




**Intuition:** key might encode "the subject of the sentence," and query might ask for "the subject of the sentence".

# Attention details

attention score for encoder input $t$ to decoder step $l$

RNN encoder activations at step $t$

$e_{t,l} = k_t \cdot q_l$

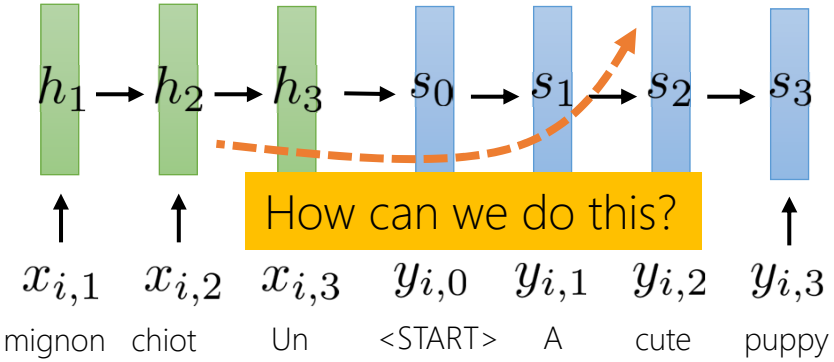
key: $k_t = k(h_t)$

query: $q_l = q(s_l)$

learned function

e.g., $k_t = \sigma(W_k h_t + b_k)$

not differentiable!




intuitively: send $h_t$ for $\arg\max_t e_{t,l}$ to step $l$

let $\alpha_{\cdot,l} = \text{softmax}(e_{\cdot,l})$

$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

send $a_l = \sum_t \alpha_{t,l} h_t$ ⟵ approximates $h_t$ for $\arg\max_t e_{t,l}$

each layer:

$$\bar{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix}$$

$$z^\ell = W^\ell \bar{a}^{\ell-1} + b^\ell$$

$$a^\ell = \sigma(z^\ell)$$

what does "send" mean?

who receives it?

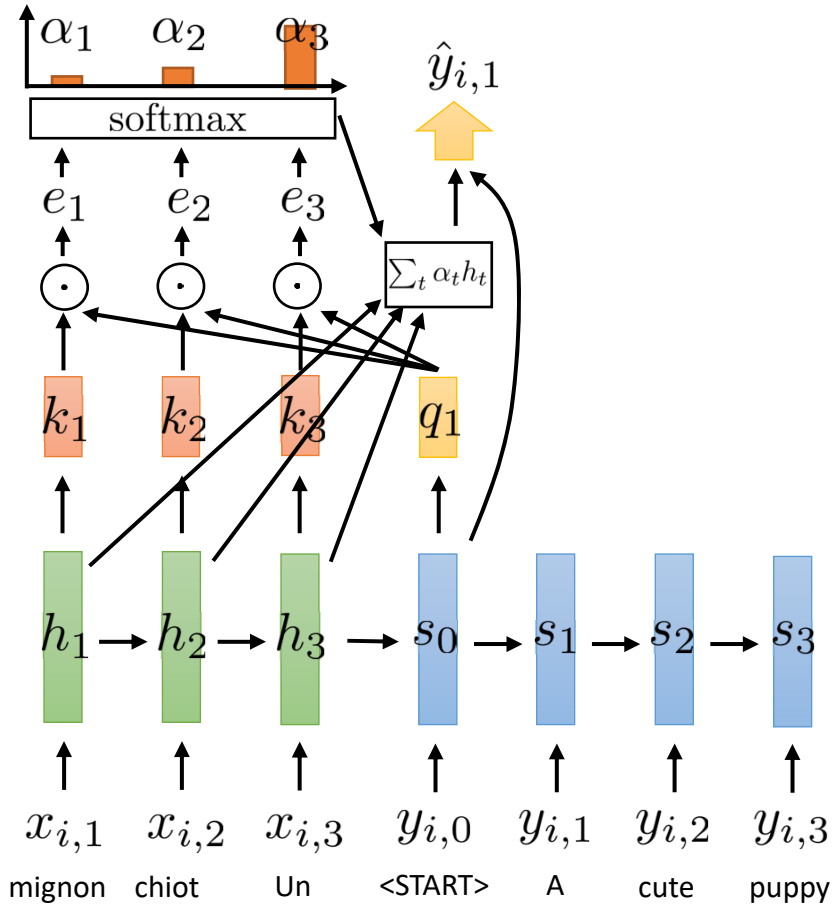
output: $\hat{y}_l = f(s_l, a_l)$

next decoder step $\bar{s}_l = \begin{bmatrix} s_{l-1} \\ a_{l-1} \\ y_l \end{bmatrix}$

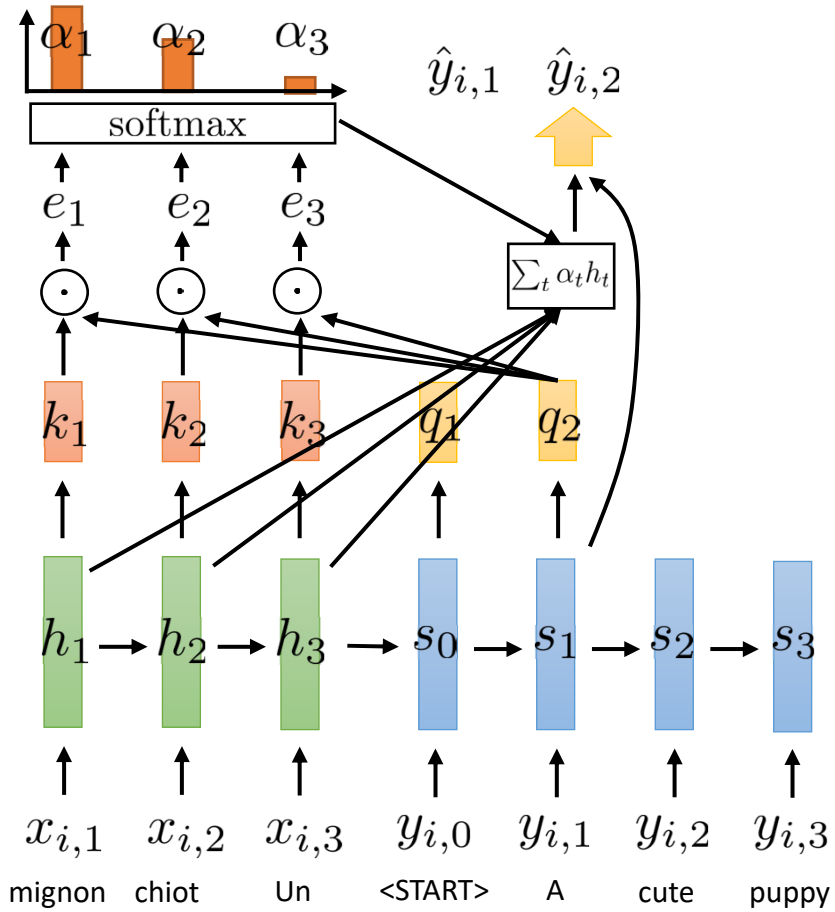
(*i.e.*, append a to the input)

# Attention Walkthrough (Example)

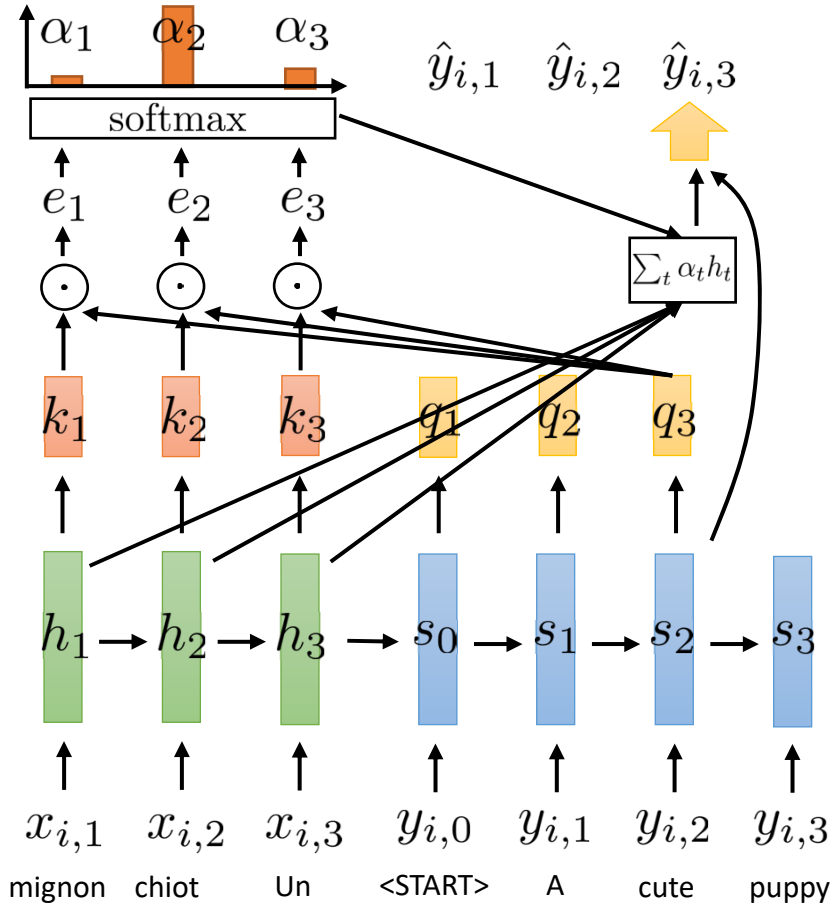
$$e_{t,l} = k_t \cdot q_l$$



# Attention Walkthrough (Example)



# Attention Walkthrough (Example)



# Attention Variants

Simple key-query choice: $k$ and $q$ are identity functions

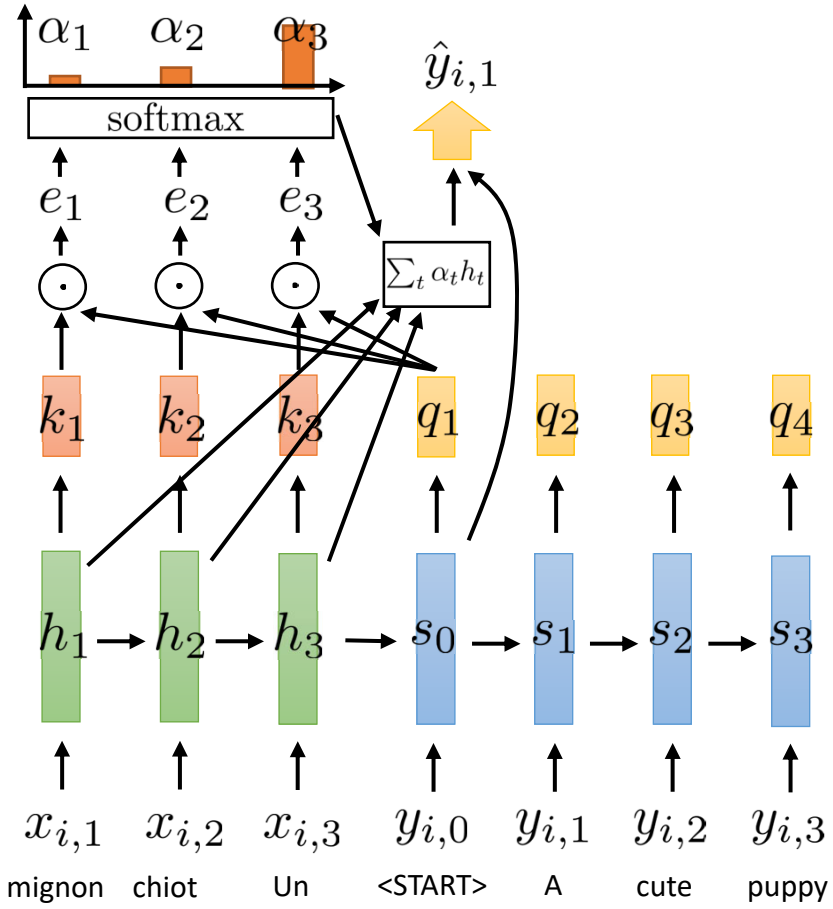
$$k_t = h_t \qquad q_l = s_l$$

Decoder-side:

$$e_{t,l} = h_t \cdot s_l$$

$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

$$a_l = \sum_t \alpha_t h_t$$



# Attention Variants

Linear multiplicative attention:

$$k_t = W_k h_t \qquad q_l = W_q s_l$$

Decoder-side:

just learn this matrix

$$e_{t,l} = h_t^T W_k^T W_q s_l = h_t^T W_e s_l$$

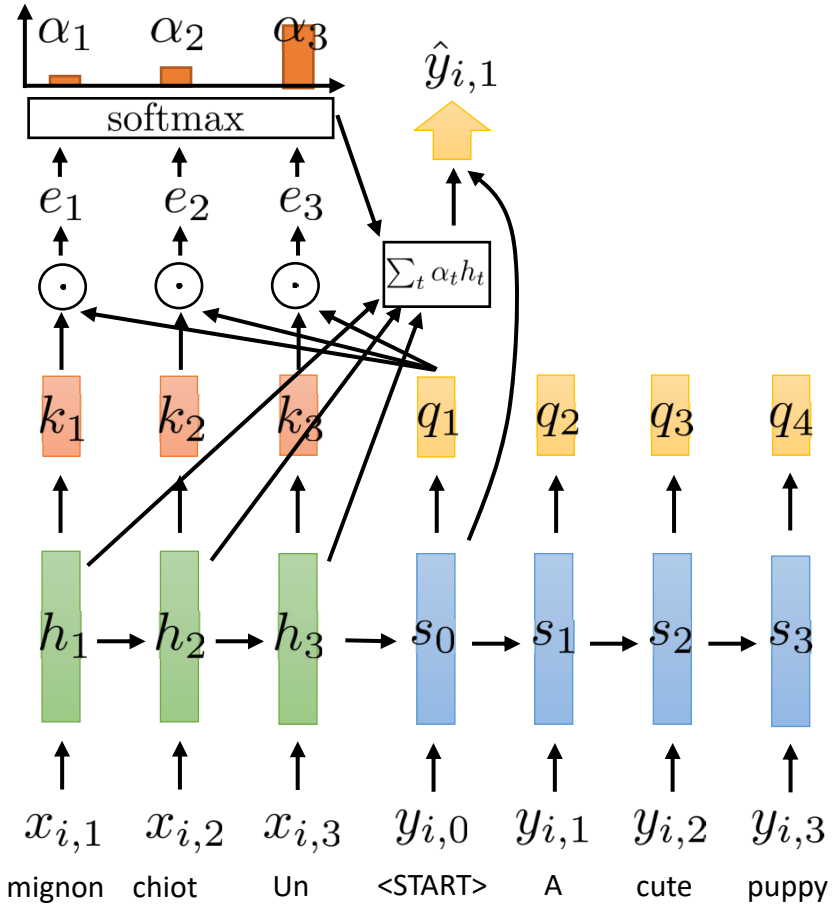
$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

$$a_l = \sum_t \alpha_t h_t$$

Learned value encoding:

$$a_l = \sum_t \alpha_t v(h_t)$$

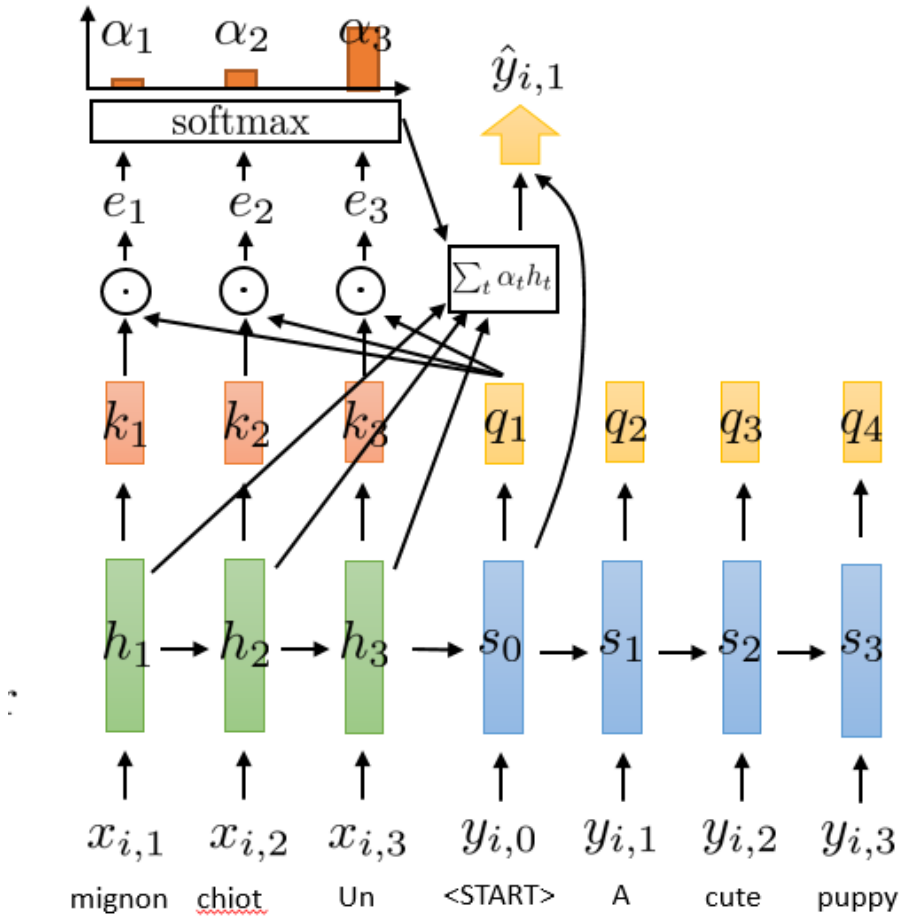
some learned function



# Attention is **very** powerful

- All decoder steps are connected to **all** encoder steps!
- Connections can *skip* directly ahead to where needed.
- Thus gradients can be much better behaved than RNN without attention.

Is attention all we need?



# CS 189/289

Today:

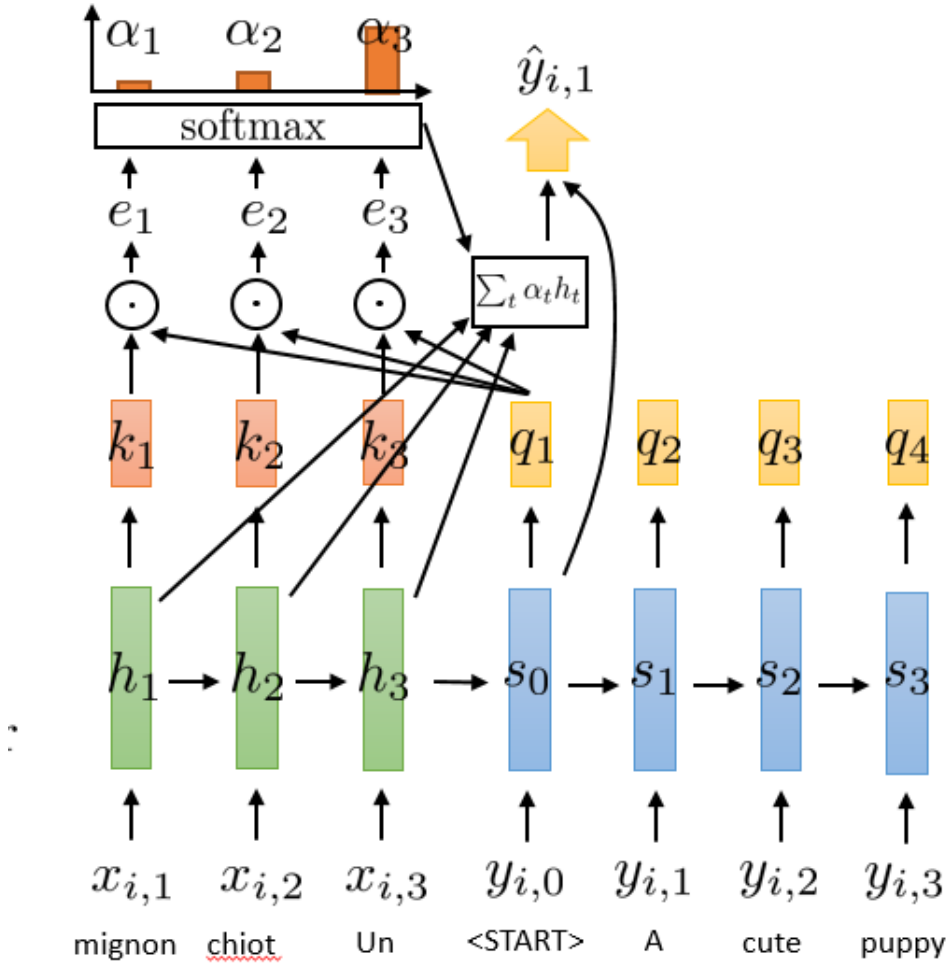
1. Recurrent Neural Networks
2. Attention and Transformers

# Is Attention All We Need?

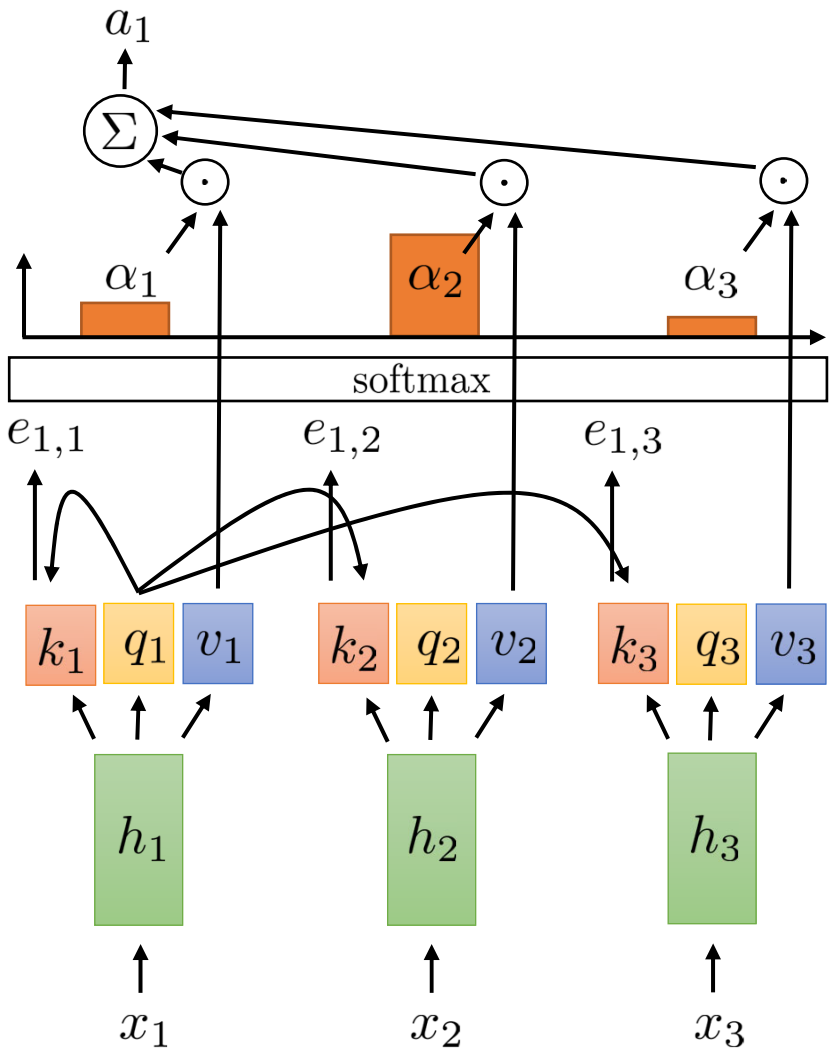
- If we have **attention**, do we even need recurrent connections?
- Can we **transform** our RNN into a **purely attention-based** model?

This has a few issues we must overcome:

- Decoding position 3 can't access $s_1$ or $s_0$.
- Solution: self-attention.



# Self-Attention (one layer)




$$a_l = \sum_t \alpha_{l,t} v_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

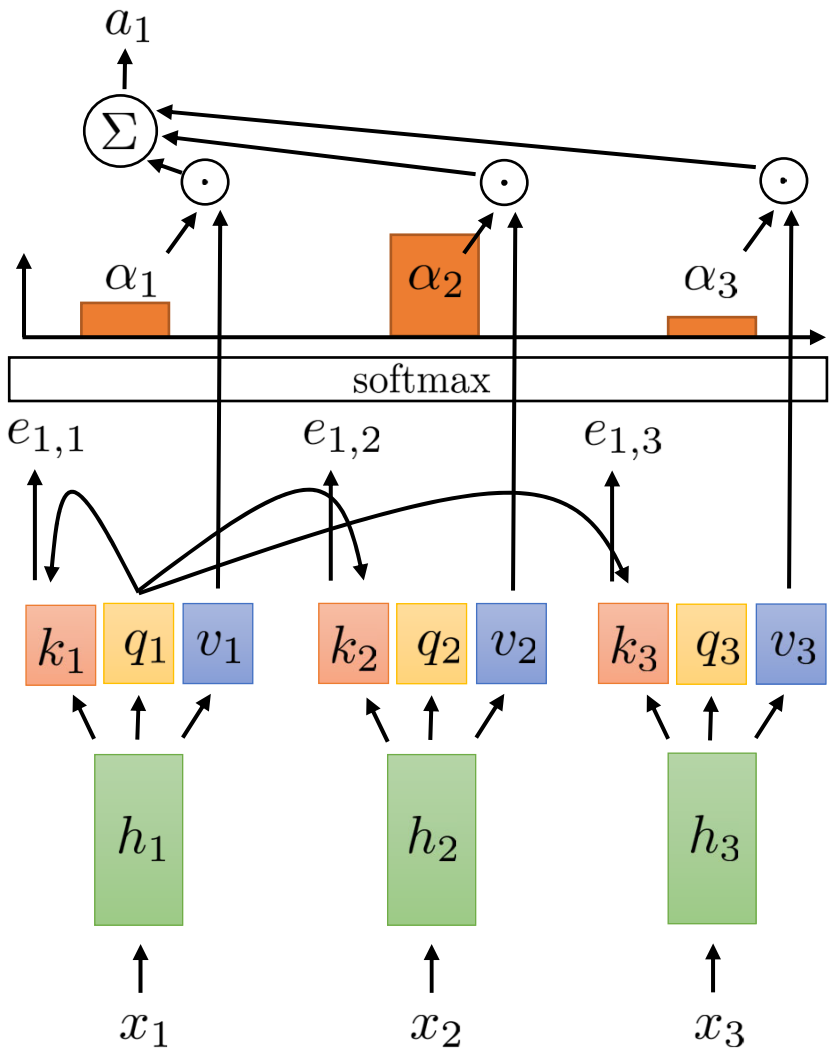
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

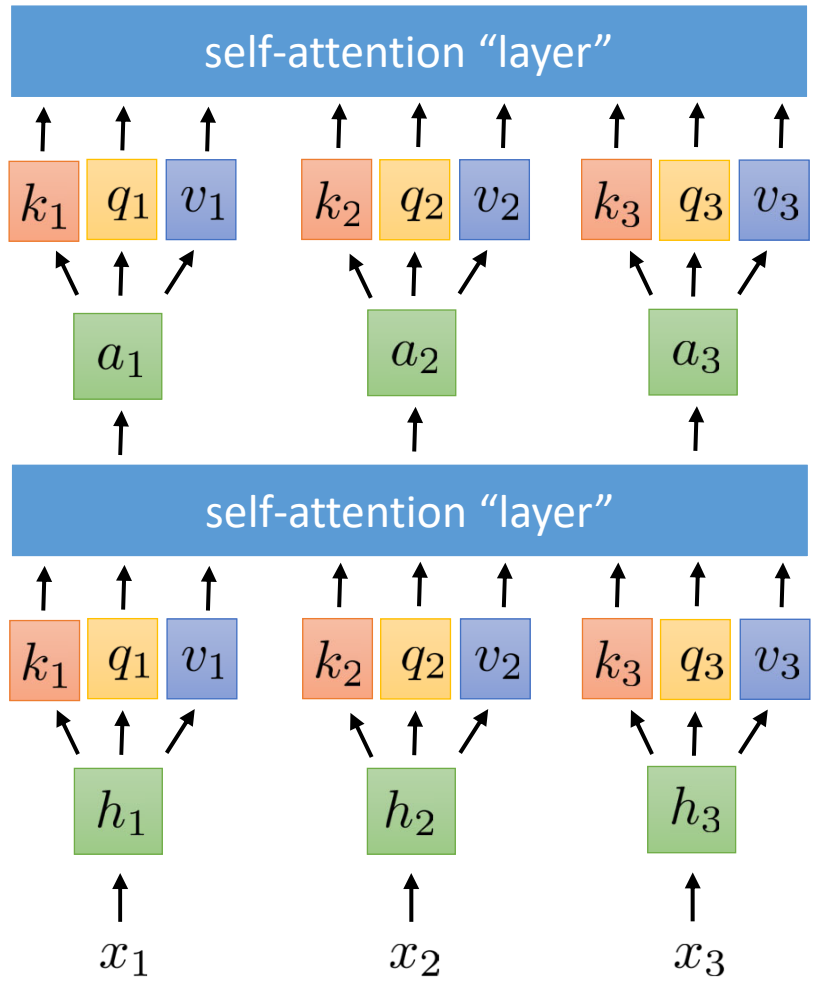
(or any other nonlinear function)

# Self-Attention




keep repeating until we've processed this enough

then hand off to next part of overall model

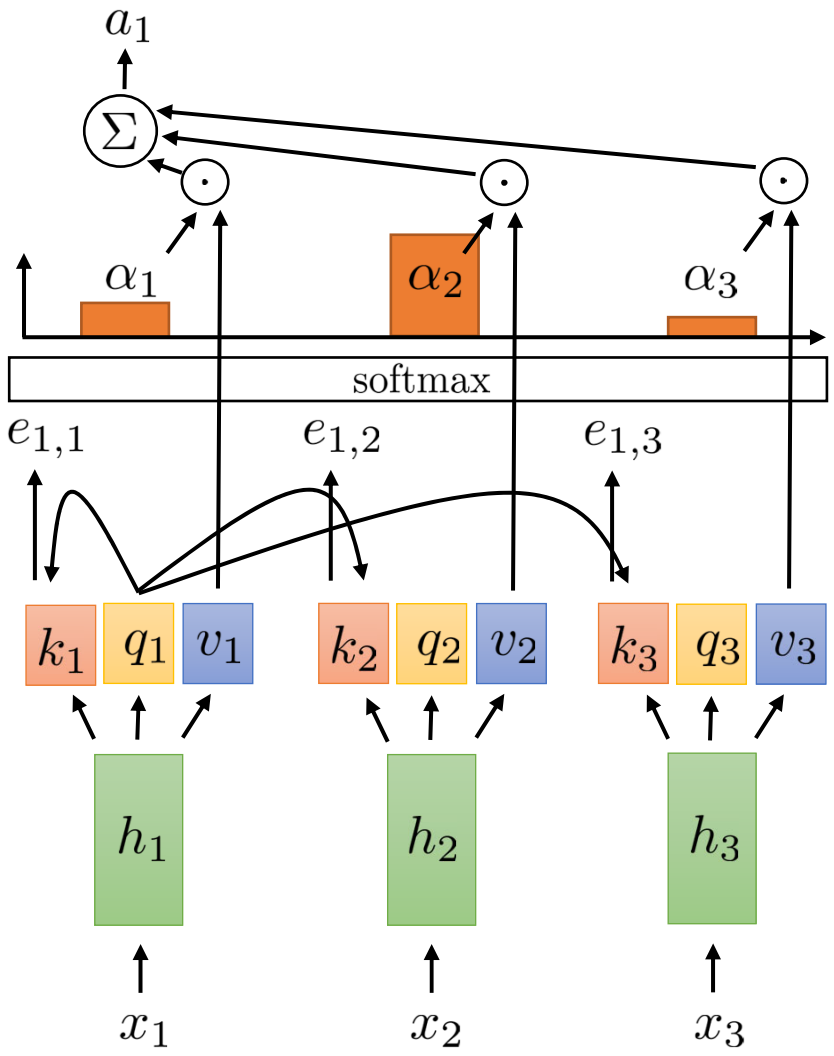


# From Self-Attention to Transformers

- Self-attention lets us remove recurrence entirely, yielding the now pervasively used Transformer model for sequences.
- But we need a few additional components to fix some problems:

| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Positional encoding: what is the order?




**what we see:**

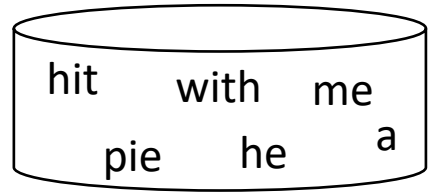
he hit me with a pie

**what naïve self-attention sees:**

a pie hit me with he

a hit with me he pie

he pie me with a hit




Permutation Equivariant!

**most** alternative orderings are nonsense, but some change the meaning

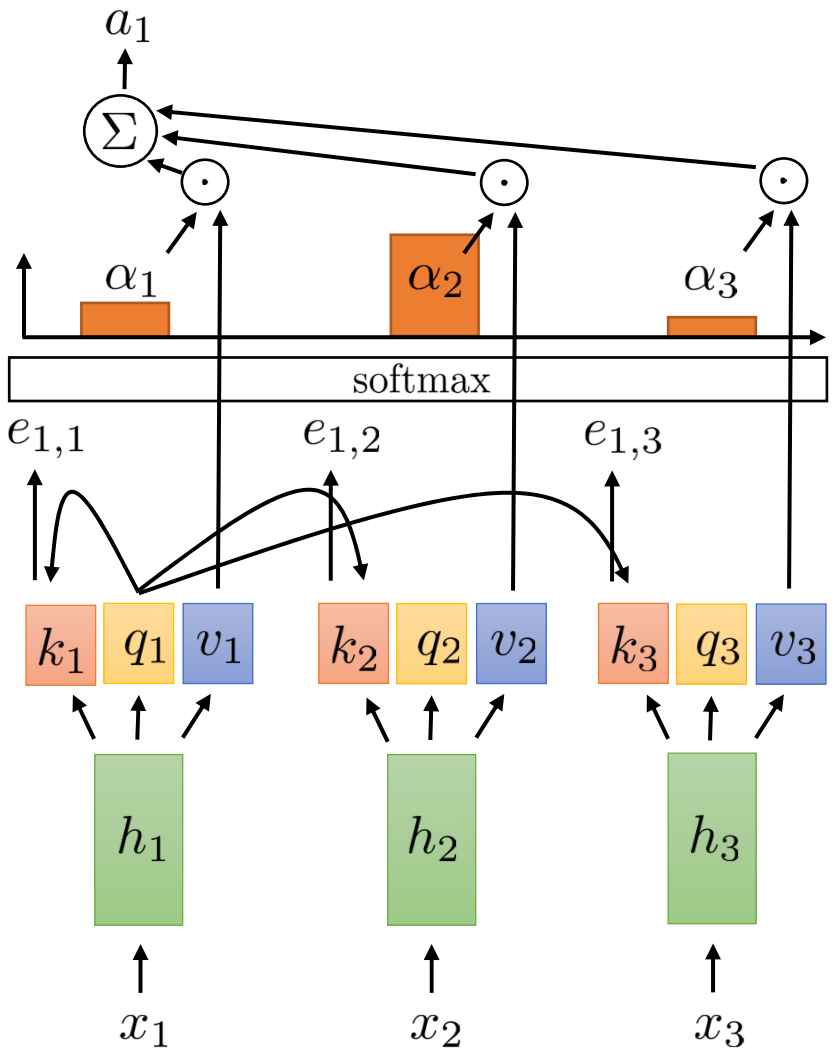
**in general** the position of words in a sentence carries information!

**Idea:** add some information to the representation at the beginning that indicates where it is in the sequence!

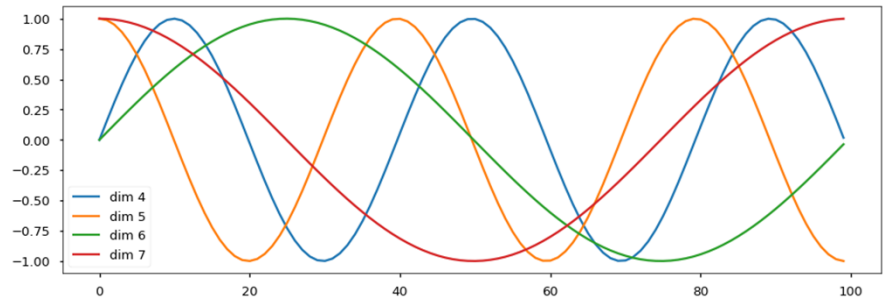
$$h_t = f(x_t, t)$$

some function

# Positional encoding: what is the order?




$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$




$d$, is the dimensionality of positional encoding

$$h_t = f(x_t, t)$$

some function

(N.B. textbook discusses adding, $x_t + p_t$)

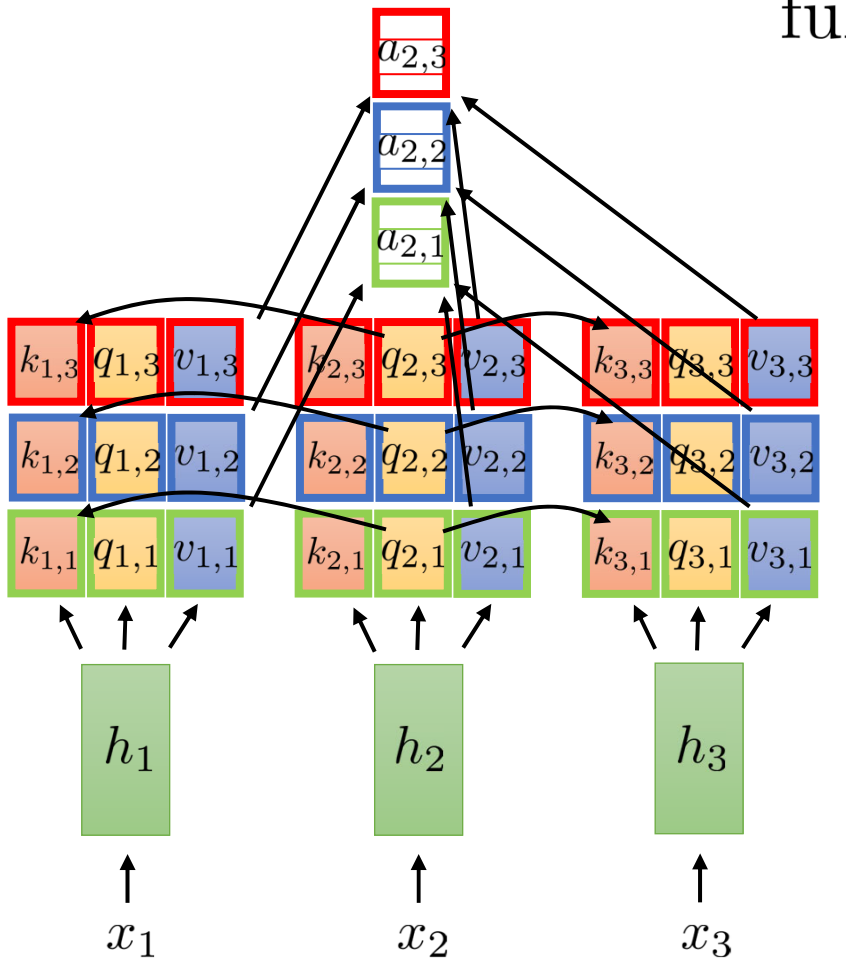
# From Self-Attention to Transformers

- The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**
- But to make this actually work, we need to develop a few additional components to address some fundamental limitations

| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Multi-head attention

Idea: have multiple keys, queries, and values for every time step!




full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

compute weights **independently** for each head

$$e_{l,t,i} = q_{l,i} \cdot k_{t,i}$$

$$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$$

$$a_{l,i} = \sum_{t} \alpha_{l,t,i} v_{t,i}$$

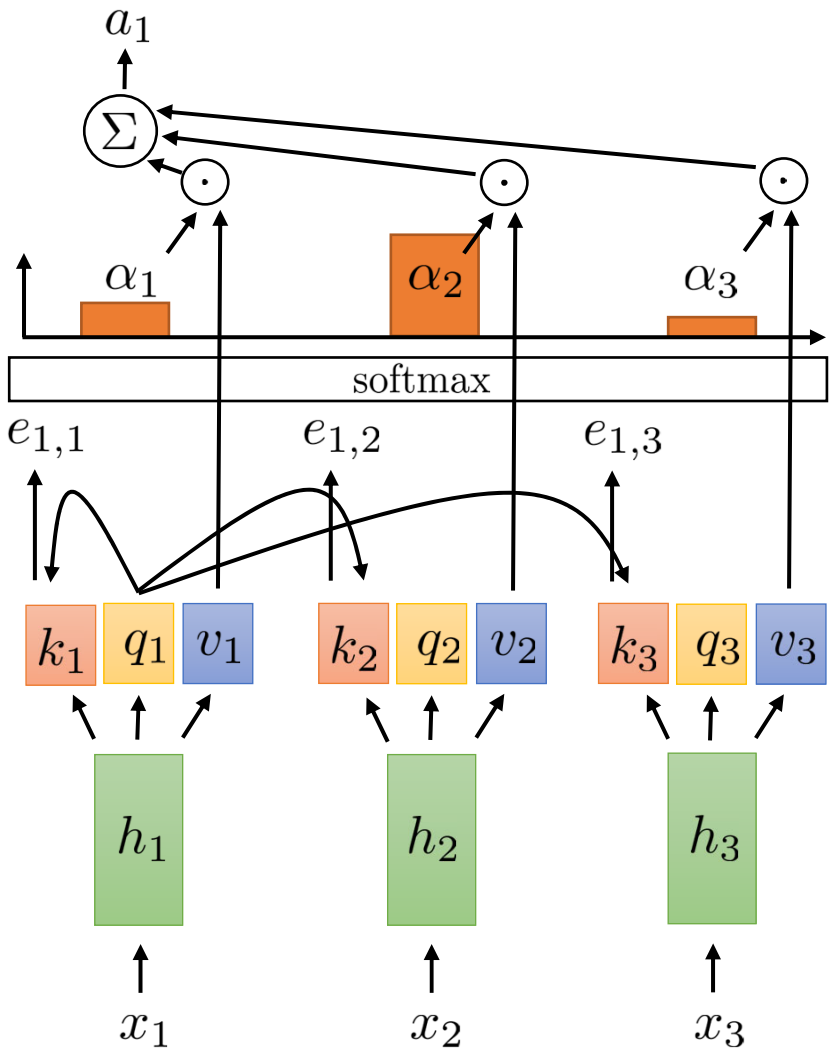
*around 8 heads seems to work pretty well for big models*

# From Self-Attention to Transformers

- The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**
- But to make this actually work, we need to develop a few additional components to address some fundamental limitations

| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Self-Attention is **Linear**




$$k_t = W_k h_t \qquad q_t = W_q h_t \qquad v_t = W_v h_t$$

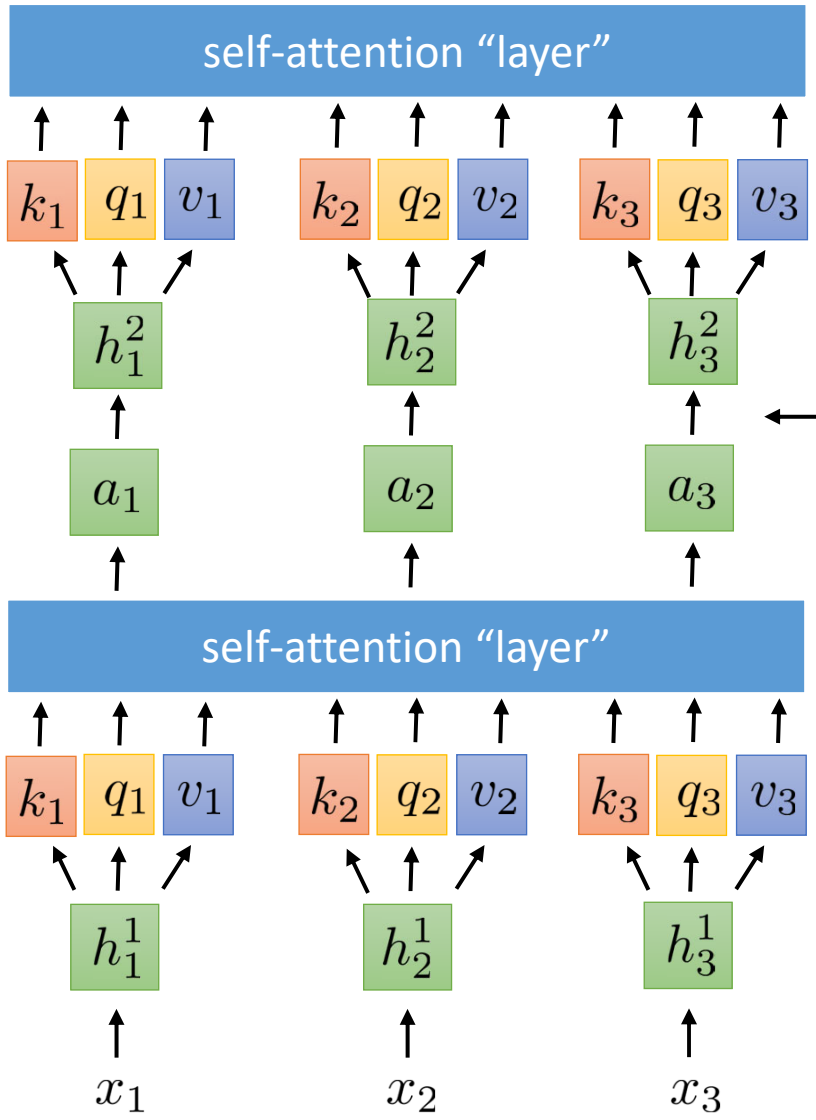
$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$$a_l = \sum_t \alpha_{l,t} v_t = \sum_t \alpha_{l,t} W_v h_t = W_v \sum_t \alpha_{l,t} h_t$$

Every self-attention “layer” is a linear transformation of the previous layer

# Alternating self-attention & non-linearity




some non-linear (learned) function

$$\text{e.g., } h_t^\ell = \sigma(W^\ell a_t^\ell + b^\ell)$$

just a neural net applied at every position after every self-attention layer

# From Self-Attention to Transformers

- The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**
- But to make this actually work, we need to develop a few additional components to address some fundamental limitations

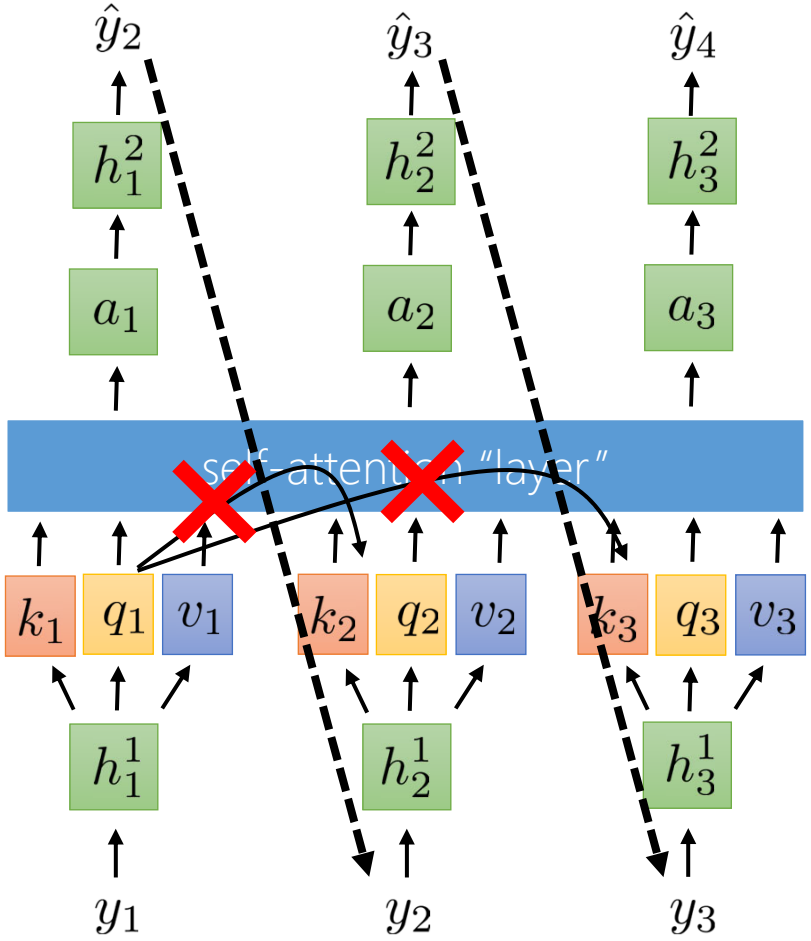
| | |
|---|---|
| 1. Positional encoding | addresses lack of sequence information |
| 2. Multi-headed attention | allows querying multiple positions at each layer |
| 3. Adding nonlinearities | so far, each successive layer is *linear* in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

# Self-attention can see the future!

*e.g.* self-attention "language model":

$$a_l = \sum_t \alpha_{l,t} v_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$




Problem:

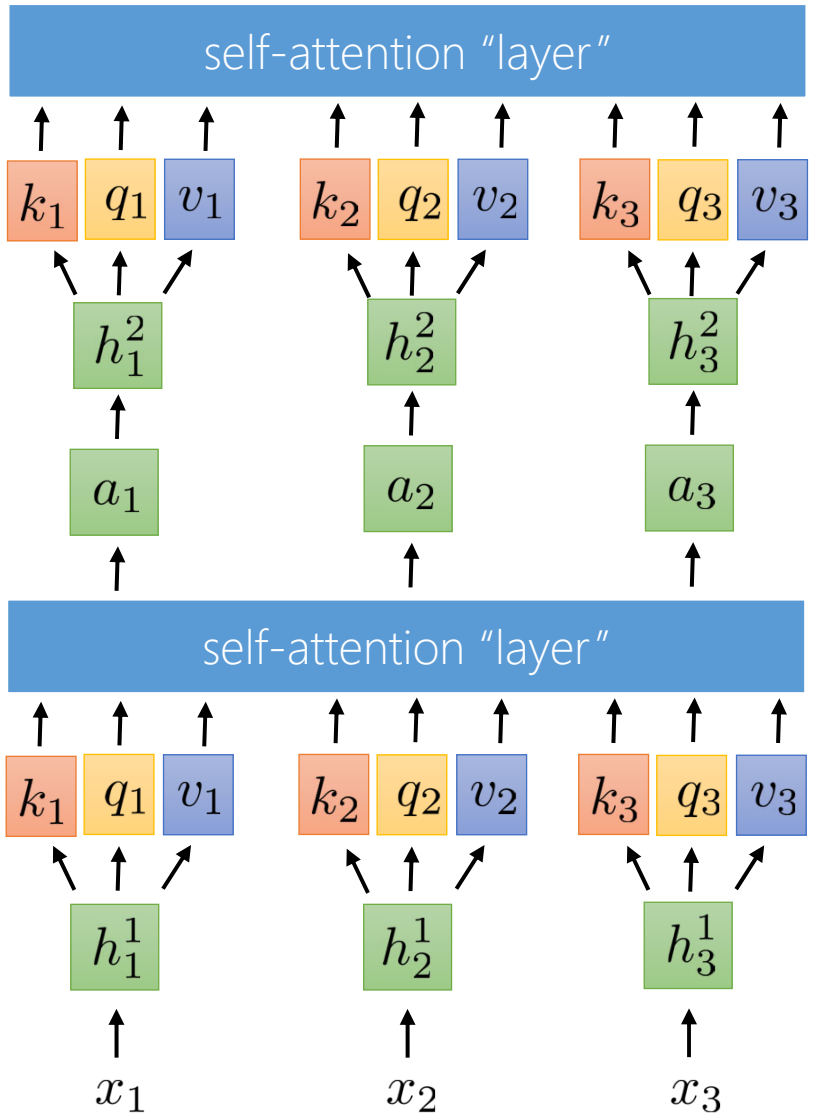
- Step 1 looks at future values.
- The output at step 1 sees the input at step 2 …
- Also cyclic: output 1 depends on input 2 which depends on output 1.

Solution:

$$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } t \leq l \\ -\infty & \text{otherwise} \end{cases}$$

Now we are read for
The Transformer!

# Sequence-to-sequence with self-attention




"*Transformer*" architecture:

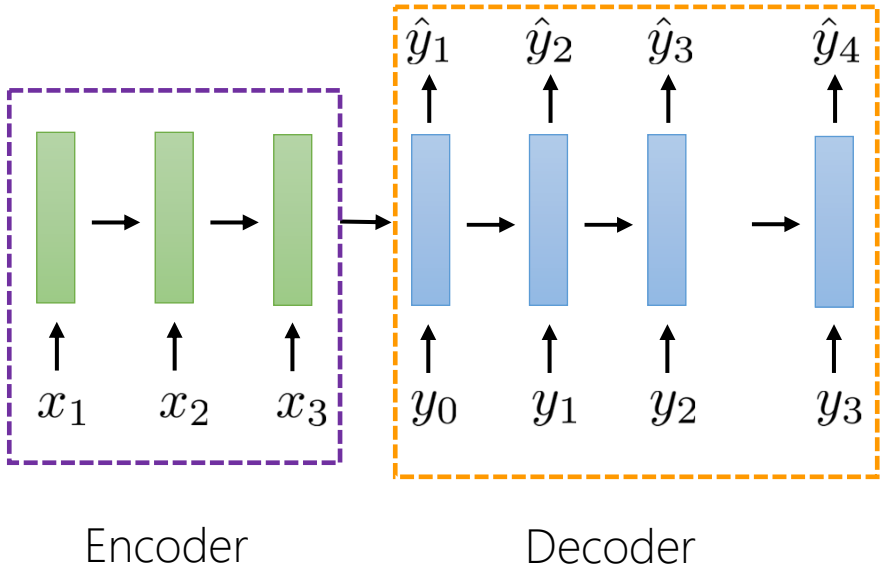
- Stacked self-attention layers with position-wise nonlinearities.
- *Transform* one sequence into another at **each** layer.
- For sequence data.

[Vaswani et al. **Attention Is All You Need.** 2017]

# Encoder-Decoder Transformer

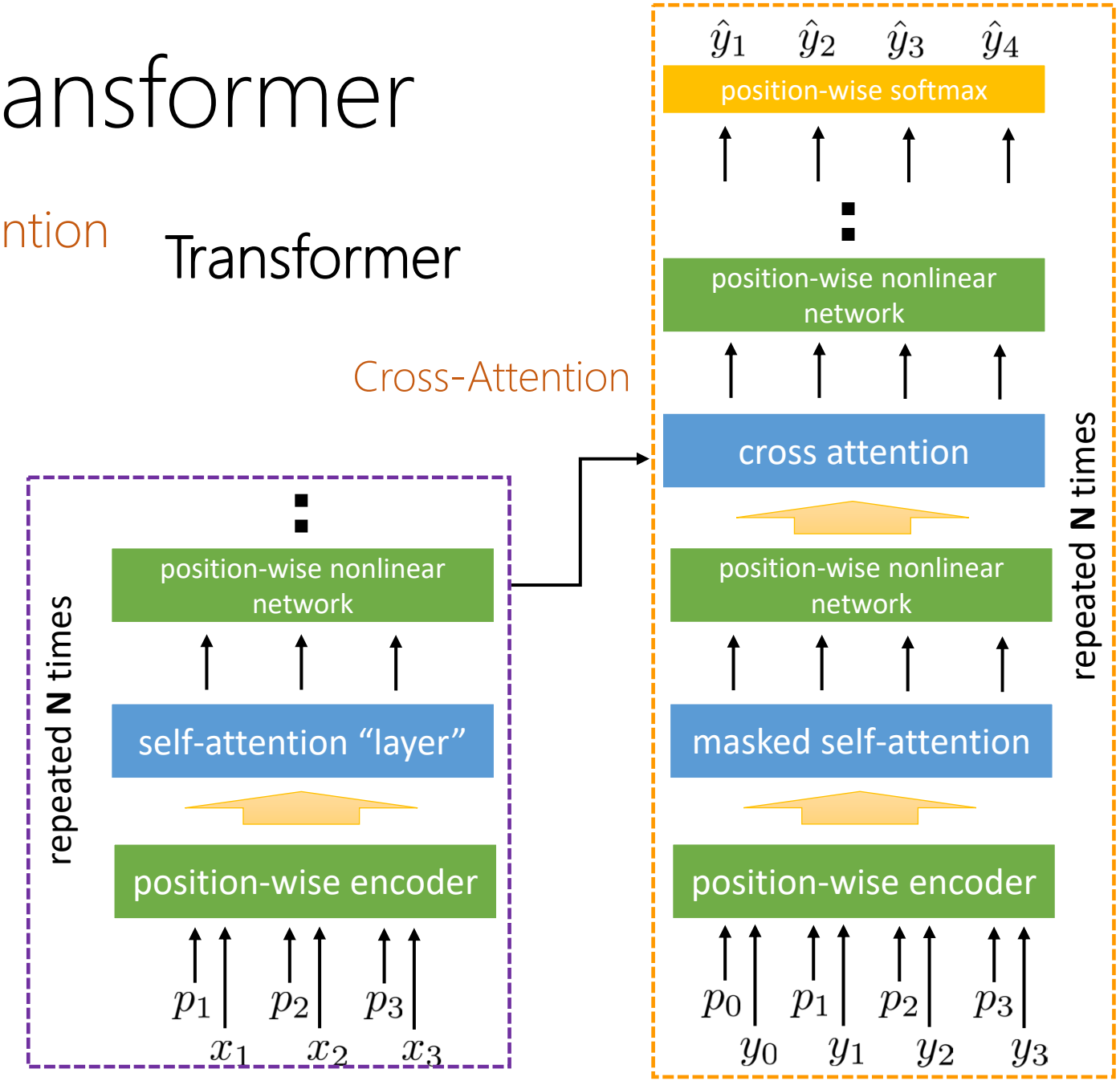
Similar to the **standard (non-self)** attention from the previous lecture

RNN




Transformer

Cross-Attention



# Transformers pros and cons

Downsides:

- Attention computations are technically $O(n^2)$
- Somewhat more complex to implement (positional encodings, etc.)

Benefits:

+ Much better long-range connections
+ Much easier to parallelize
+ In practice, can make it much deeper (more layers) than RNN

- Benefits often **vastly** outweigh the downsides.
- Transformers work **much** better than RNNs (and LSTMs) in many cases
- One of the most important sequence modeling improvements of the past decade.