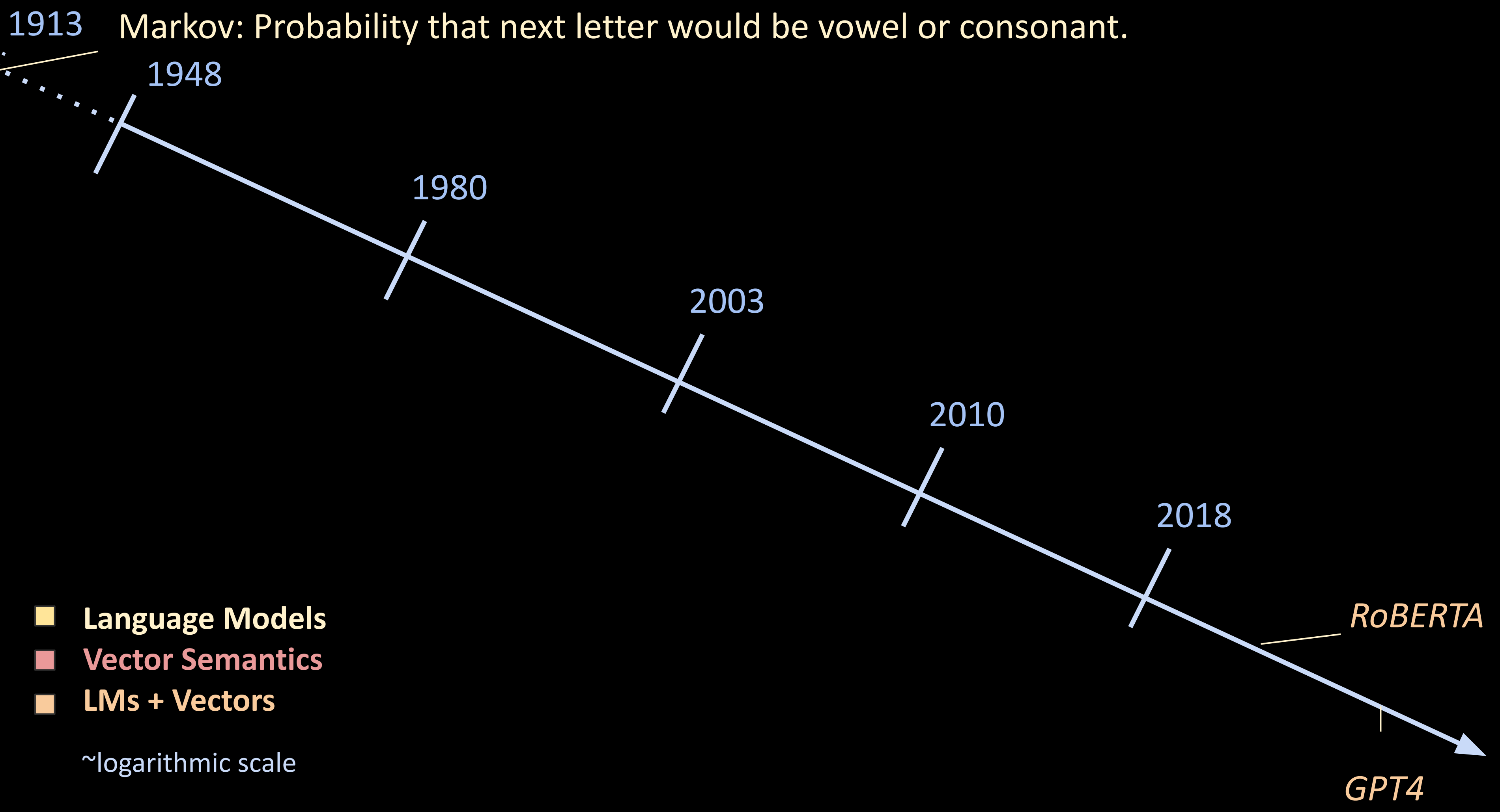
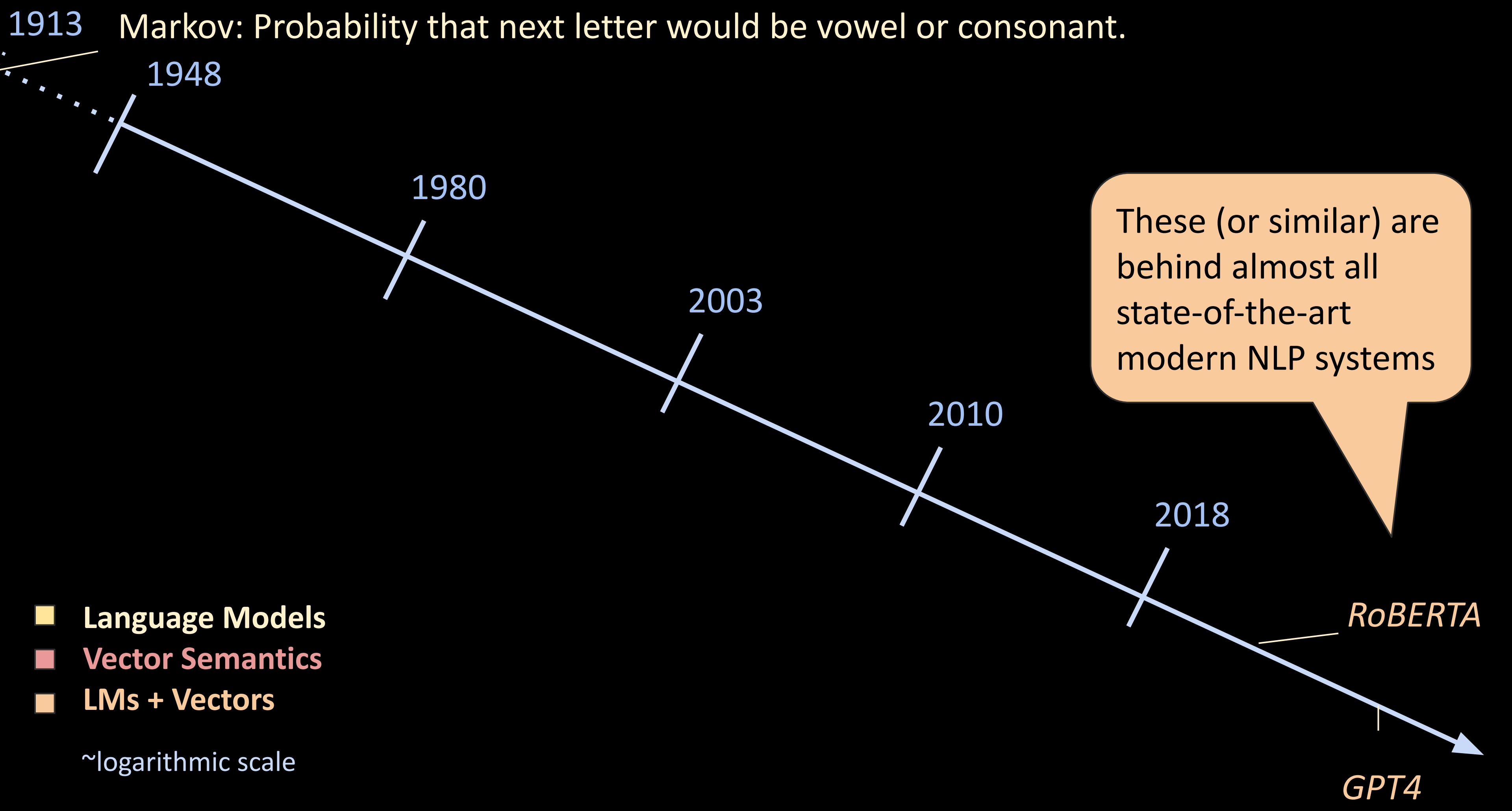
Deep Learning and Transformers

CSE538 - Spring 2024

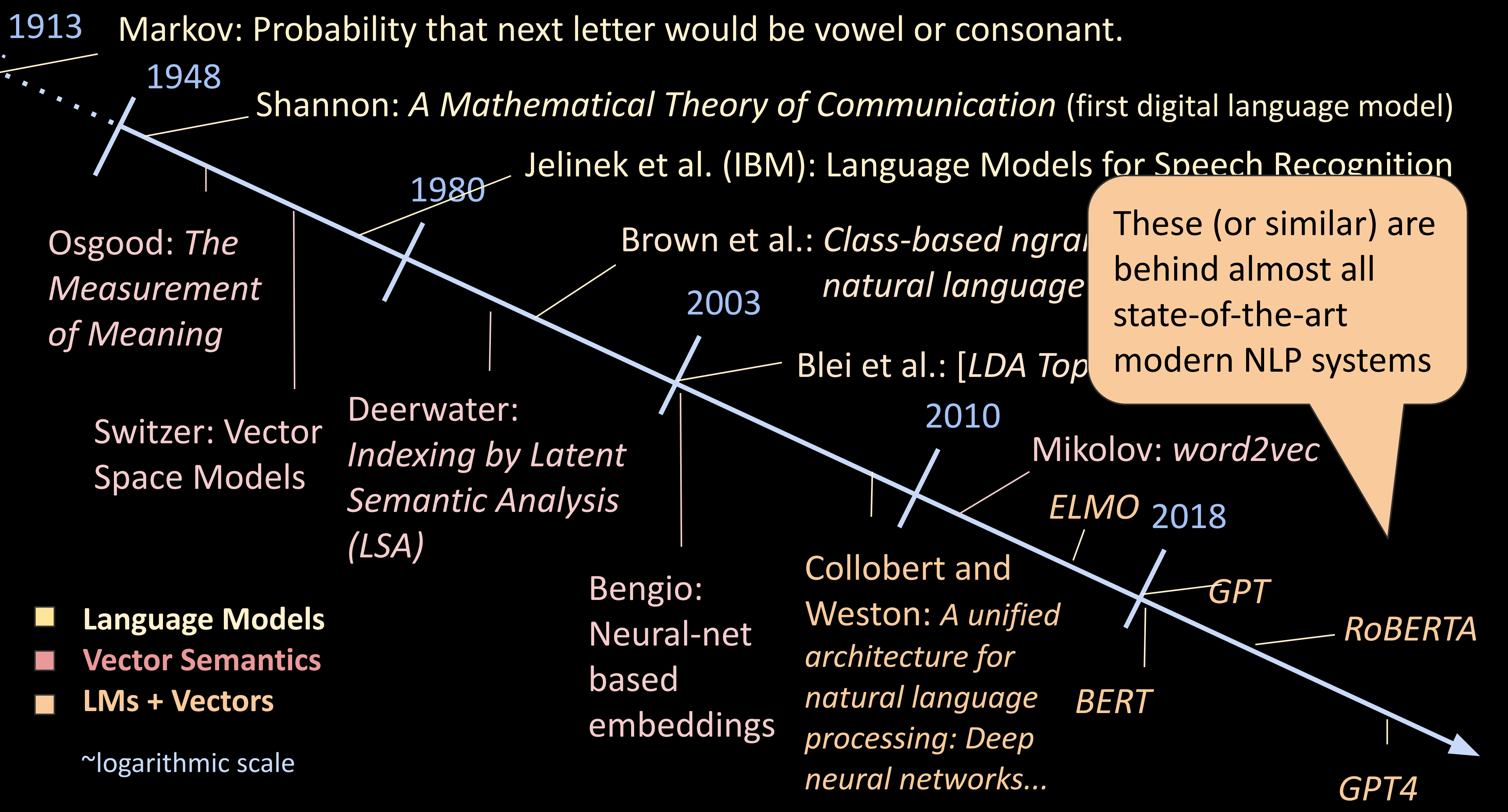
Timeline: *Language Modeling* and *Vector Semantics*



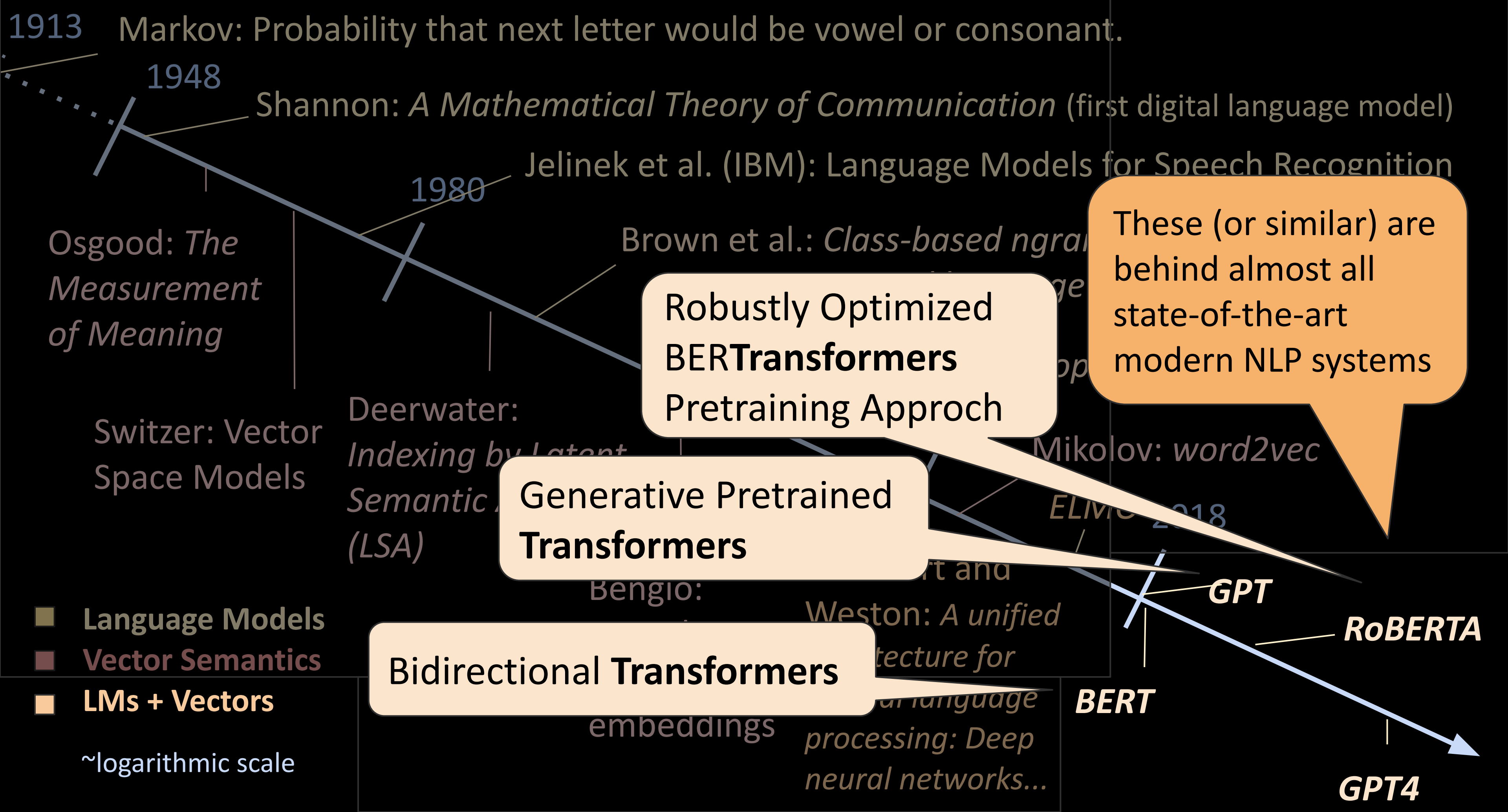
Timeline: *Language Modeling* and *Vector Semantics*



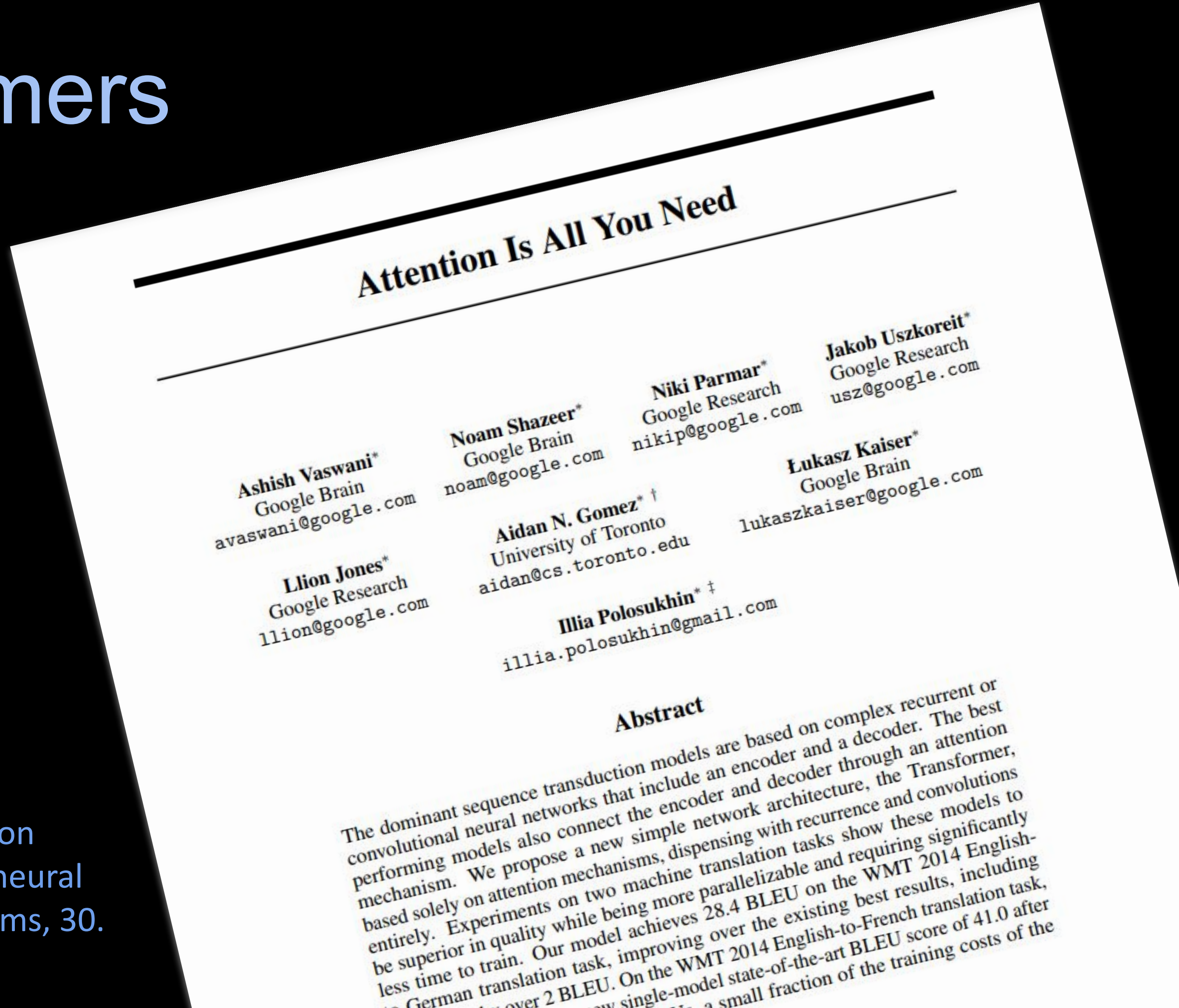
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Transformers


Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ... over 2 BLEU. On the WMT 2014 English-to-French translation task, ... single-model state-of-the-art BLEU score of 41.0 after ... a small fraction of the training costs of the

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Transformers

Self-Attention

Deep Learning

Neural Networks

Transformers

- multi-headed attention
- positional embeddings
- residual links

(to be introduced later)

Self-Attention

Deep Learning

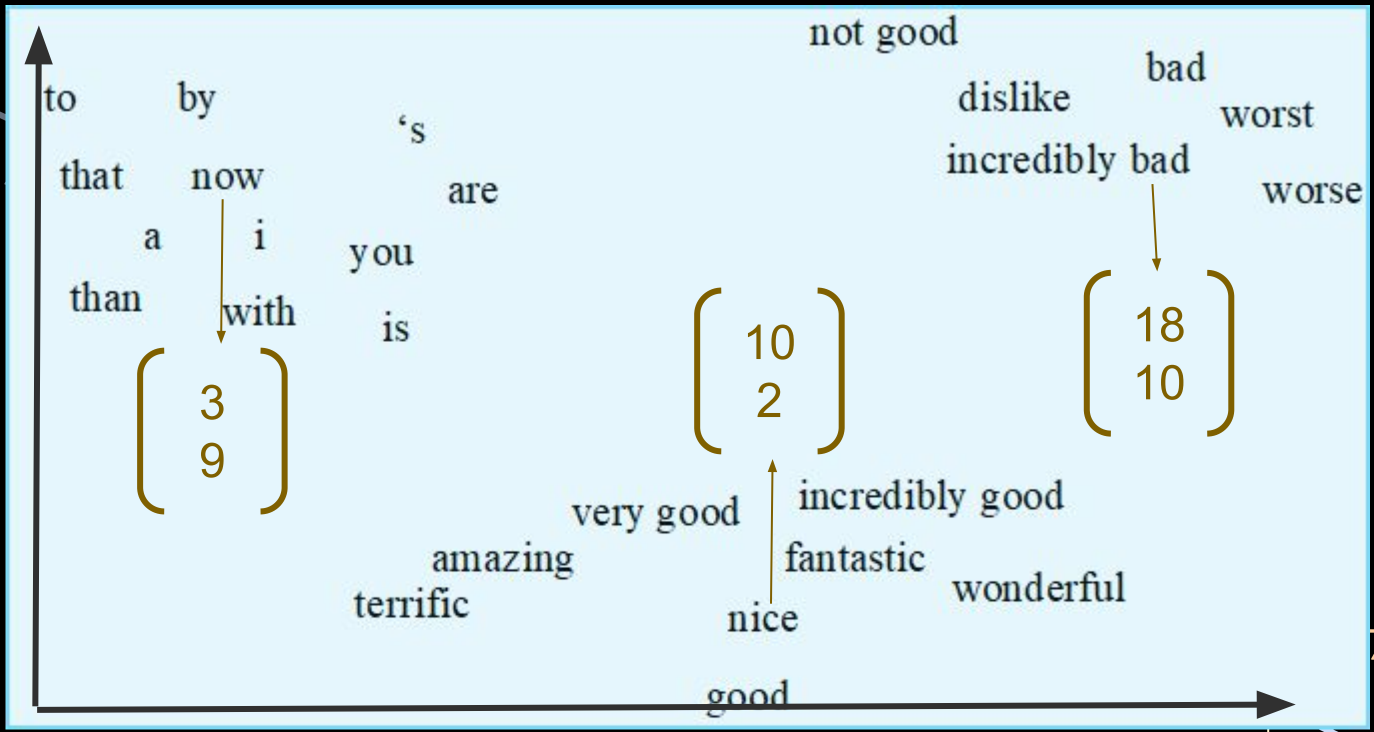
Neural Networks

Timeline: *Language Modeling* and *Vector Semantics*

1913 Markov: Probability that next letter would be vowel or consonant.

1948 Shannon: *A Mathematical Theory of Communication* (first digital language model)

Osgood: *The Measurement of Meaning*



(Li et al. ,2015; Jurafsky et al., 2019)

- Language Models
- Vector Semantics
- LMs + Vectors

~logarithmic scale

GPT4

Word Vectors

To embed: convert a token (or sequence) to a vector that represents **meaning.**

Wittgenstein, 1945: "*The meaning of a word is its use in the language*"

Distributional hypothesis -- A word's meaning is defined by all the different contexts it appears in (i.e. how it is "distributed" in natural language).

Firth, 1957: "*You shall know a word by the company it keeps*"

The nail hit the beam behind the wall.

Word Vectors

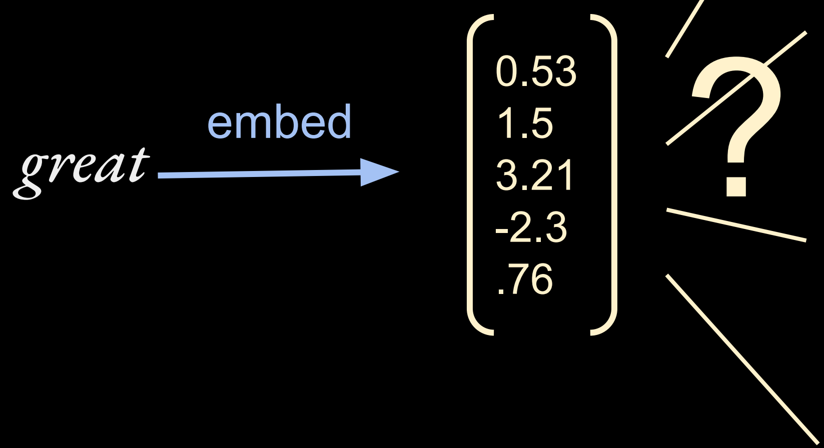
	Person A	**Person B**
How are you?	I feel ***fine*** –even ***great***!	My life is a ***great*** mess! I'm having a very hard time being happy.
What is going on?	Earlier, I ***played*** the ***game*** Yahtzee with my ***partner*****.** I could not get that ***die*** to roll a 1! Now I'm ***lying*** on my bed for a ***rest***.	My business ***partner*** was ***lying*** to me. He was trying to ***game*** the system and ***played*** me. I think I am going to ***die*** –he left and now I have to pay the ***rest*** of his ***fine*****.**

(Kjell, Kjell, and Schwartz, 2023)

Objective

$$\textit{great} \xrightarrow{\text{embed}} \begin{pmatrix} 0.53 \\ 1.5 \\ 3.21 \\ -2.3 \\ .76 \end{pmatrix}$$

Objective



great.a.1 (relatively large in size or number or extent; larger than others of its kind)

great.a.2, outstanding (of major significance or importance)

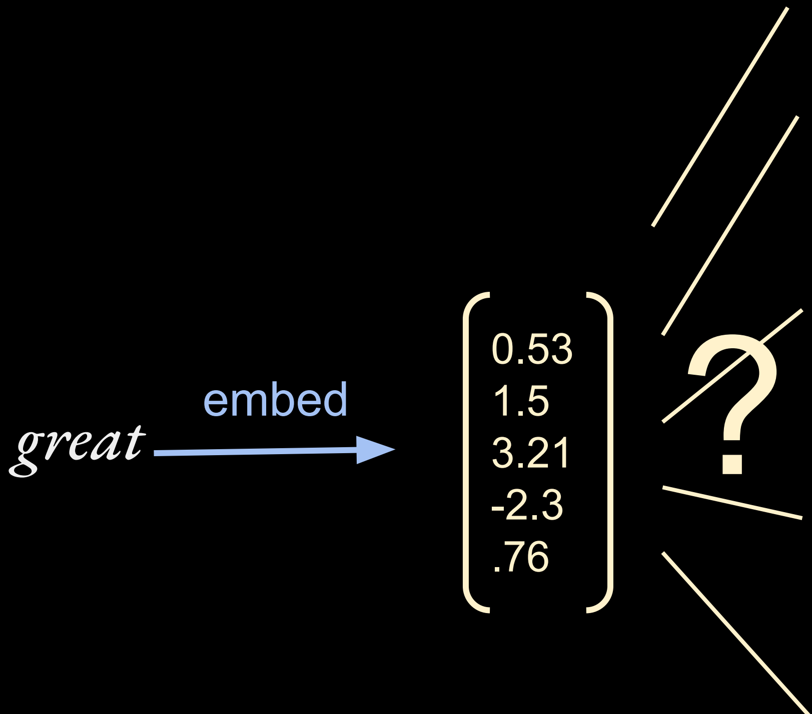
great.a.3 (remarkable or out of the ordinary in degree or magnitude or effect)

bang-up, bully, corking, cracking, dandy, **great.a.4**, groovy, keen, neat, nifty, not bad, peachy, slap-up, swell, smashing, old (very good)

capital, **great.a.5**, majuscule (uppercase)

big, enceinte, expectant, gravid, **great.a.6**, large, heavy, with child (in an advanced stage of pregnancy)

Objective



great.a.1 (relatively large in size or number or extent; larger than others of its kind)

great.a.2, outstanding (of major significance or importance)

great.a.3 (remarkable or out of the ordinary in degree or magnitude or effect)

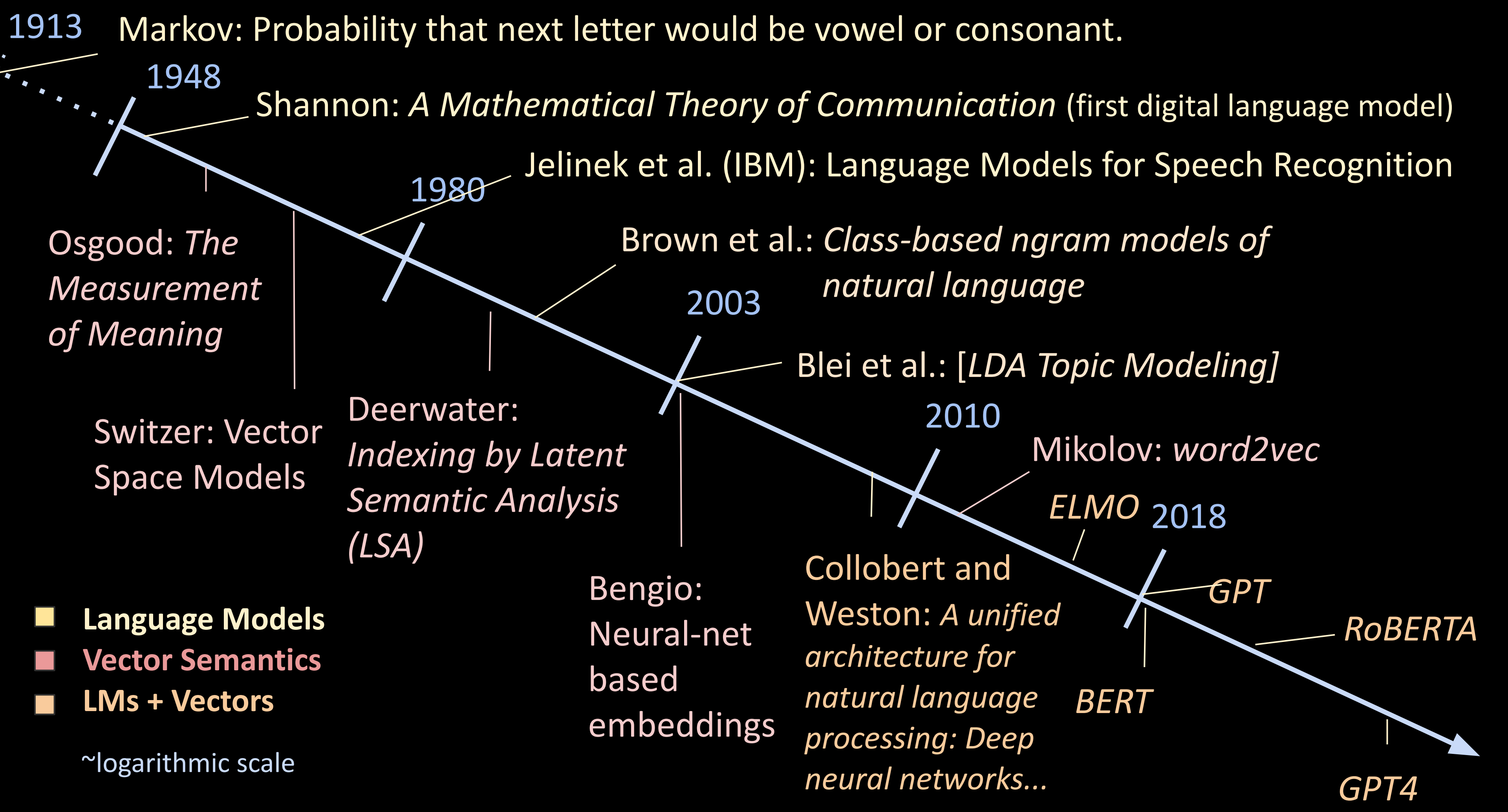
bang-up, bully, corking, cracking, dandy, **great.a.4**, groovy, keen, neat, nifty, not bad, peachy, slap-up, swell, smashing, old (very good)

capital, **great.a.5**, majuscule (uppercase)

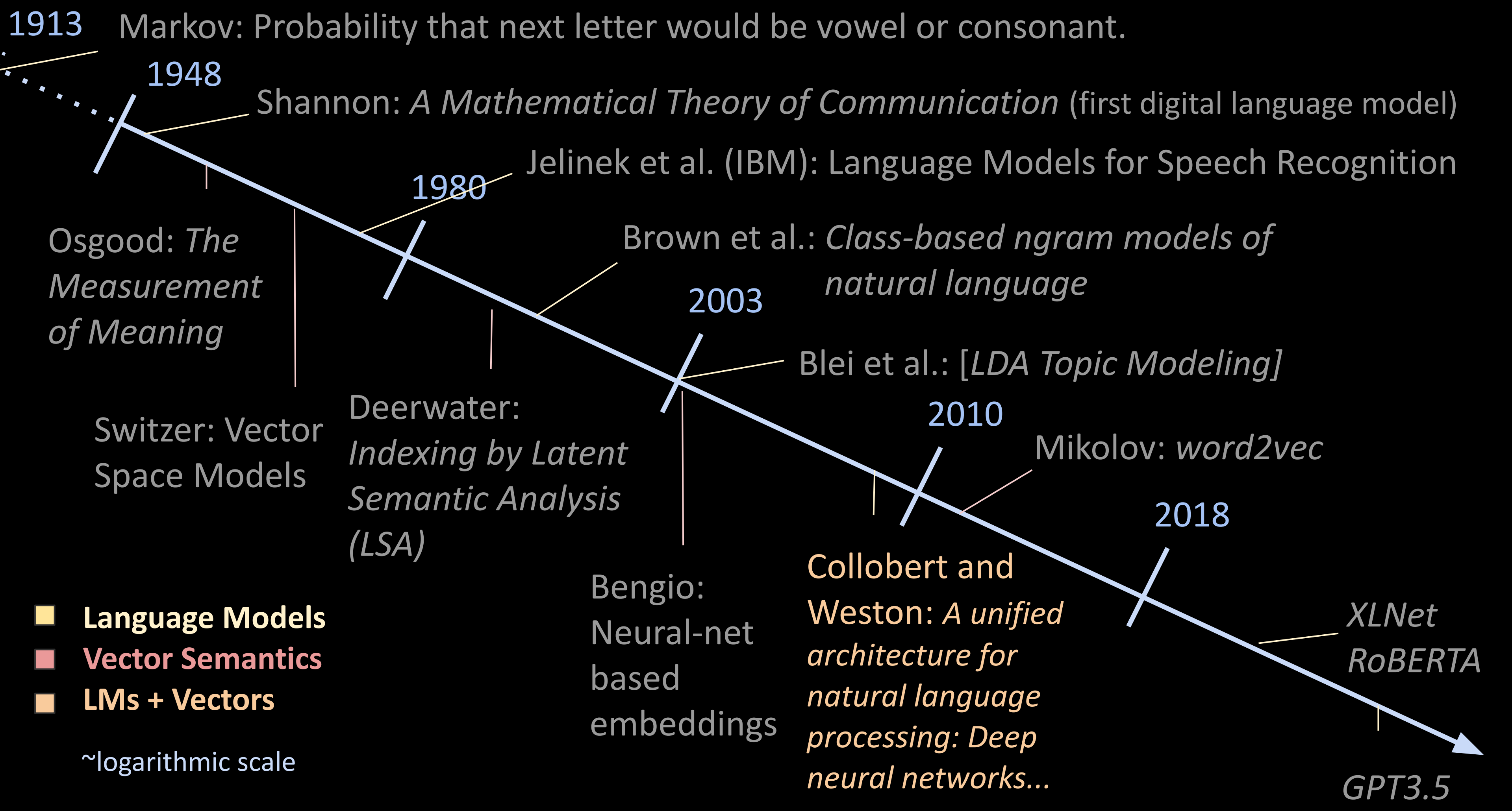
big, enceinte, expectant, gravid, **great.a.6**, large, heavy, with child (in an advanced stage of pregnancy)

great.n.1 (a person who has achieved distinction and honor in some field)

Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



T… odeling and *Vector Semantics*

1913 … r would be vowel or consonant.

… *l Theory of Communication* (first digital language model)

… et al. (IBM): Language Models for Speech Recognition

… rown et al.: *Class-based ngram models of natural language*

Os… *M… of…*

2003

Blei et al.: [*LDA Topic Modeling*]

2010

Mikolov: *word2vec*

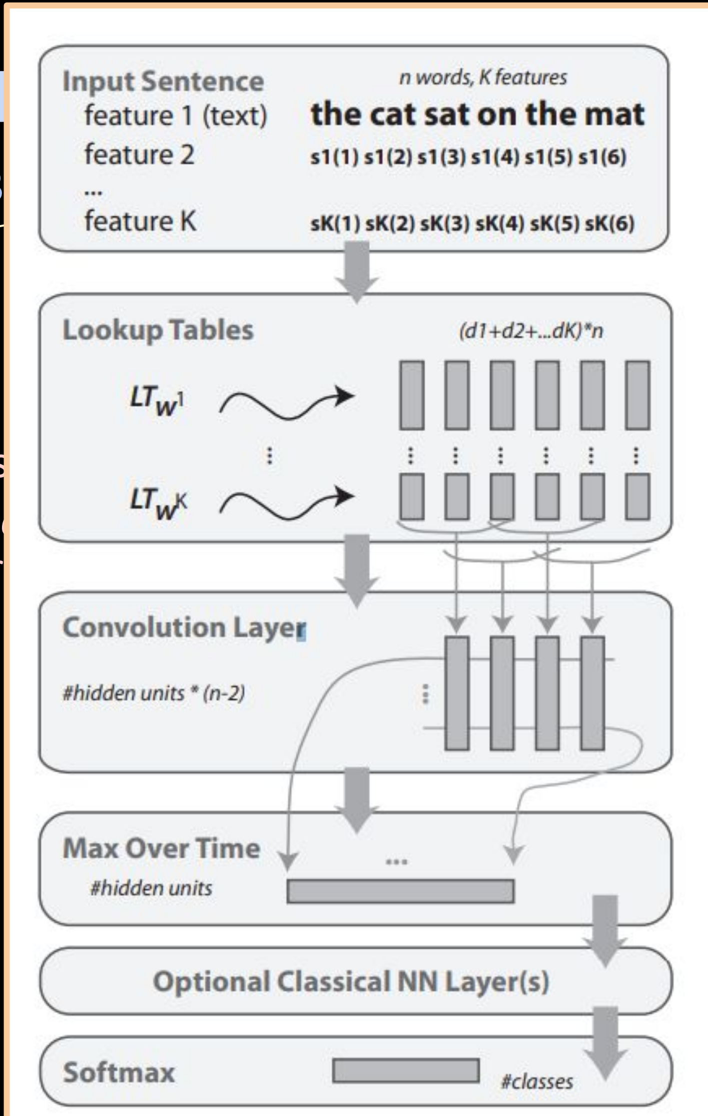
2018

…gio: …ural-net …ed …beddings

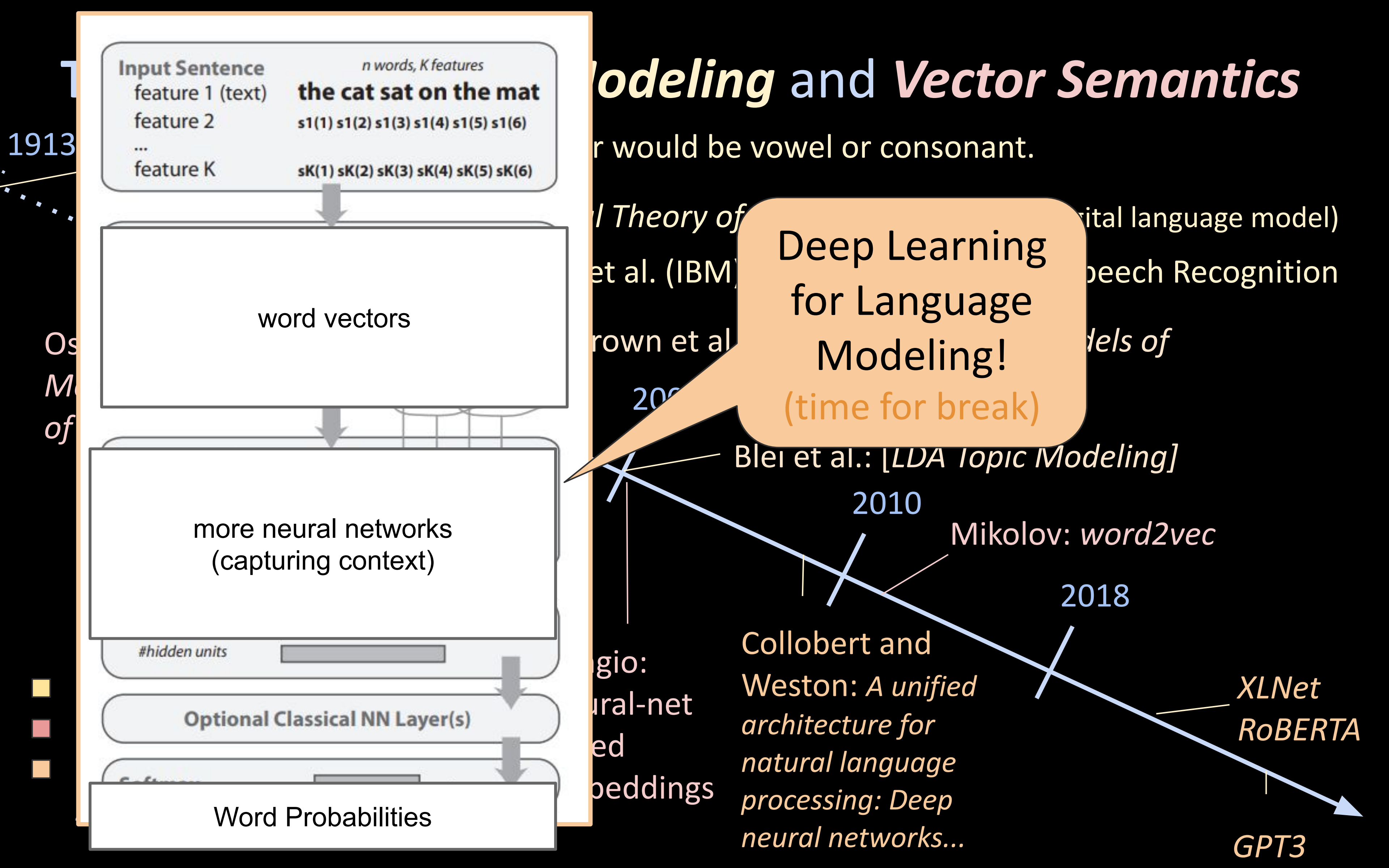
Collobert and Weston: *A unified architecture for natural language processing: Deep neural networks...*

XLNet
RoBERTA

GPT3

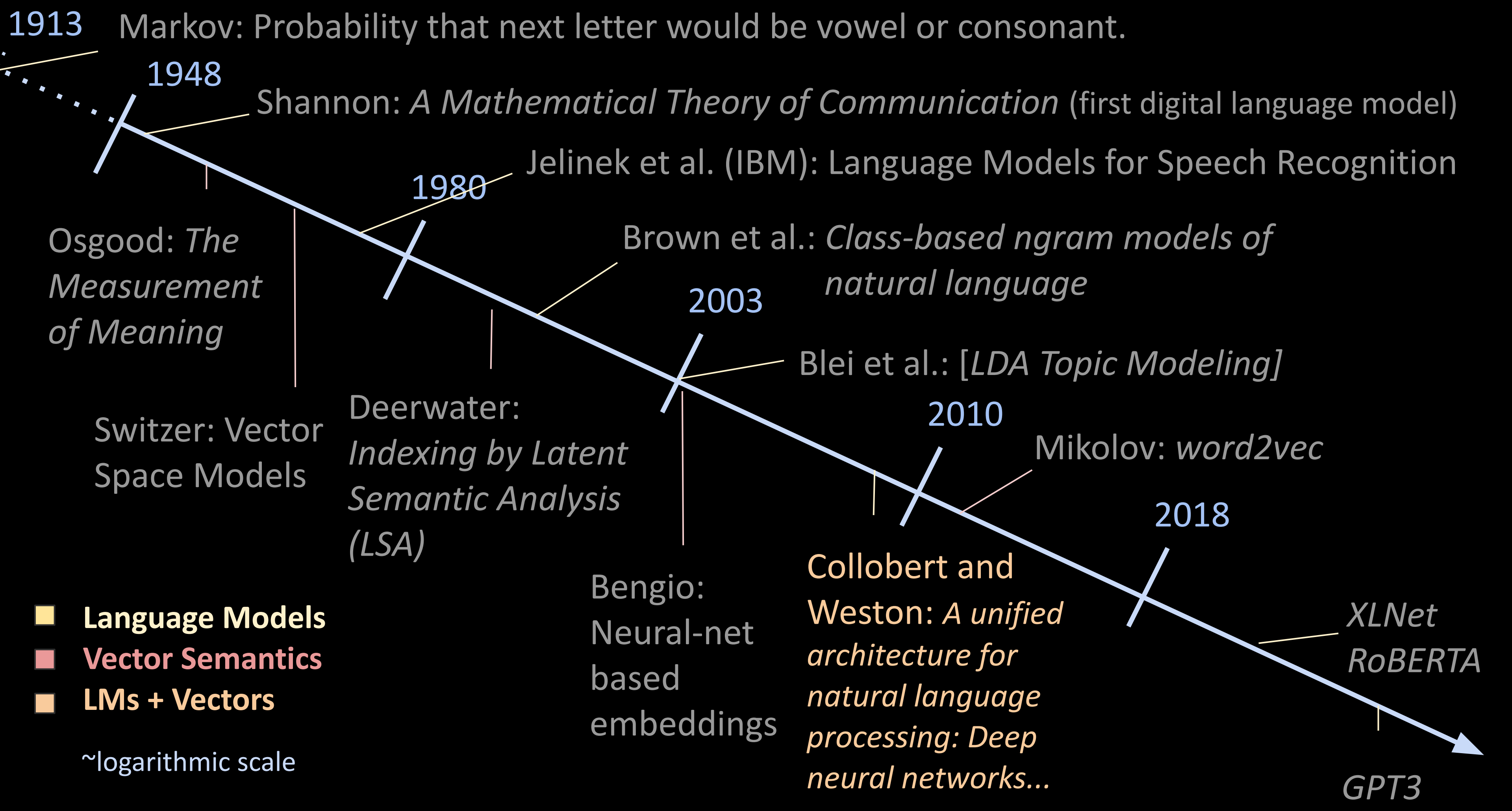


…logarithmic scale

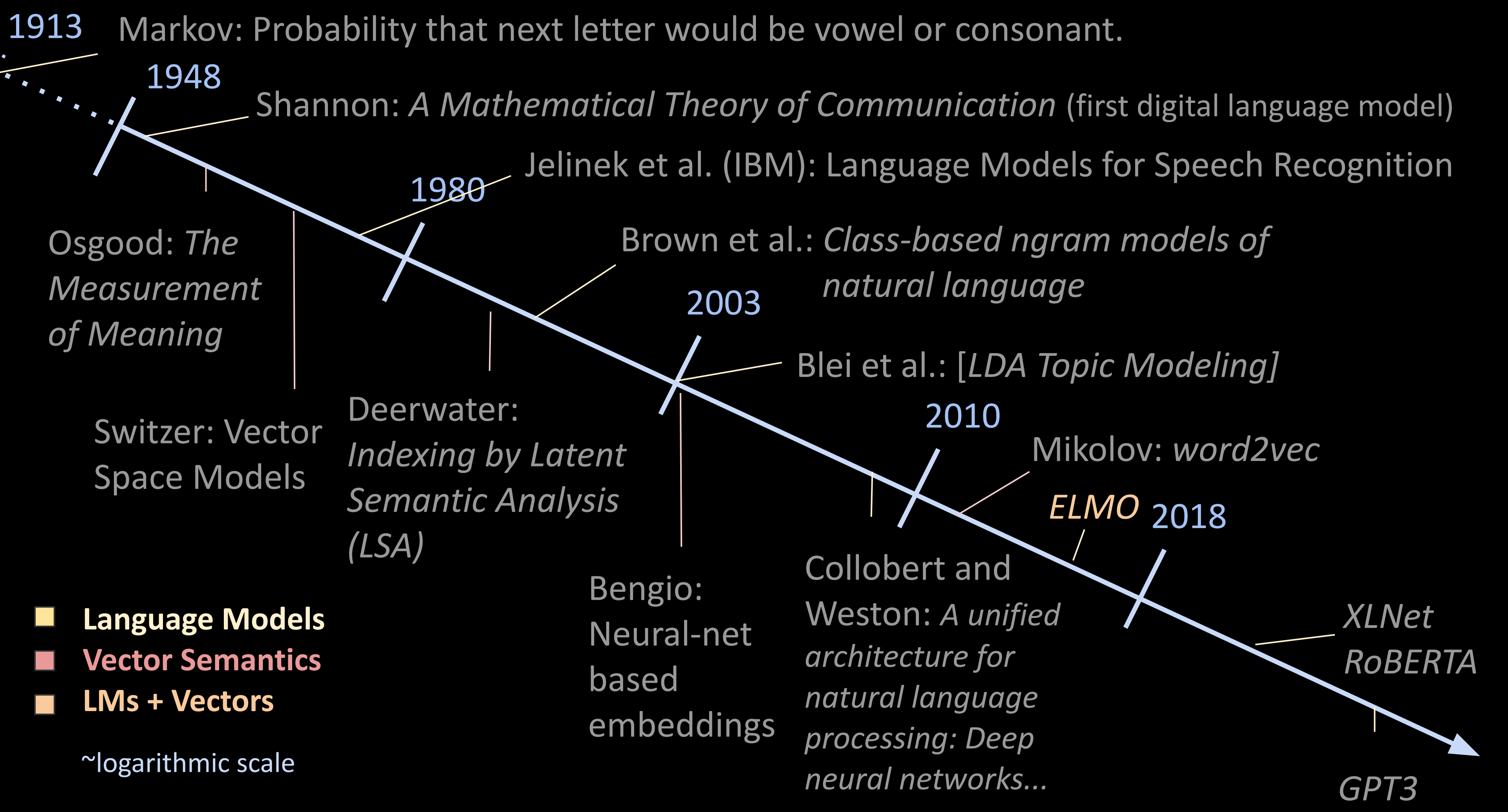


odeling and Vector Semantics
1913
r would be vowel or consonant.
l Theory of
ital language model)
et al. (IBM)
peech Recognition
rown et al
dels of
Os
M
of
20
Blei et al.: [LDA Topic Modeling]
2010
Mikolov: word2vec
2018
gio:
ural-net
ed
beddings
Collobert and Weston: A unified architecture for natural language processing: Deep neural networks...
XLNet
RoBERTA
GPT3
Input Sentence
n words, K features
feature 1 (text)
the cat sat on the mat
feature 2
s1(1) s1(2) s1(3) s1(4) s1(5) s1(6)
...
feature K
sK(1) sK(2) sK(3) sK(4) sK(5) sK(6)
word vectors
more neural networks
(capturing context)
#hidden units
Optional Classical NN Layer(s)
Word Probabilities
Deep Learning for Language Modeling!
(time for break)

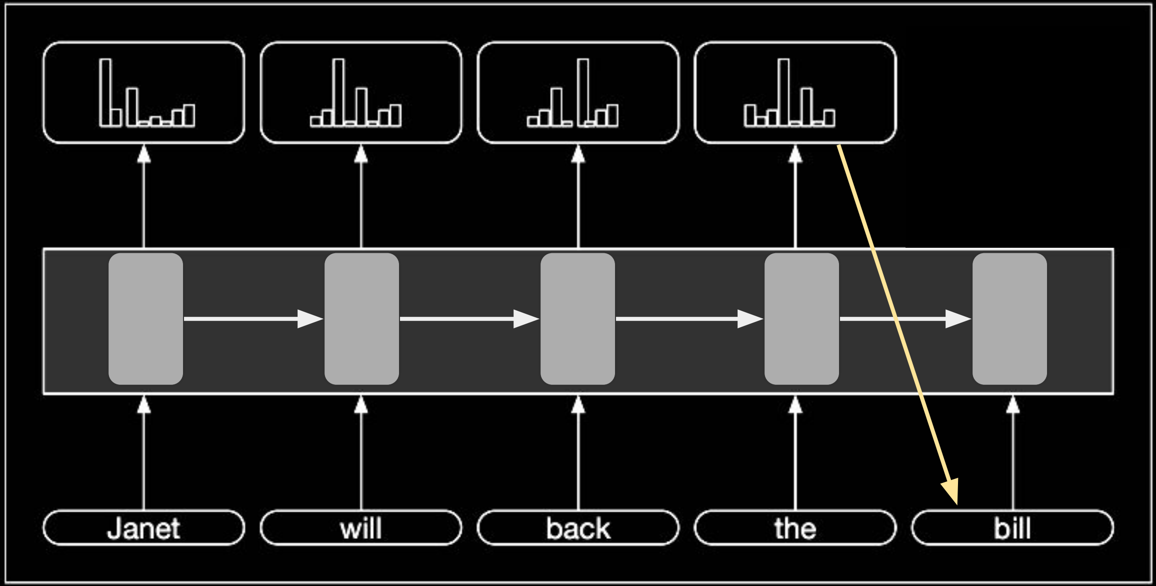
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Recurrent Neural Network

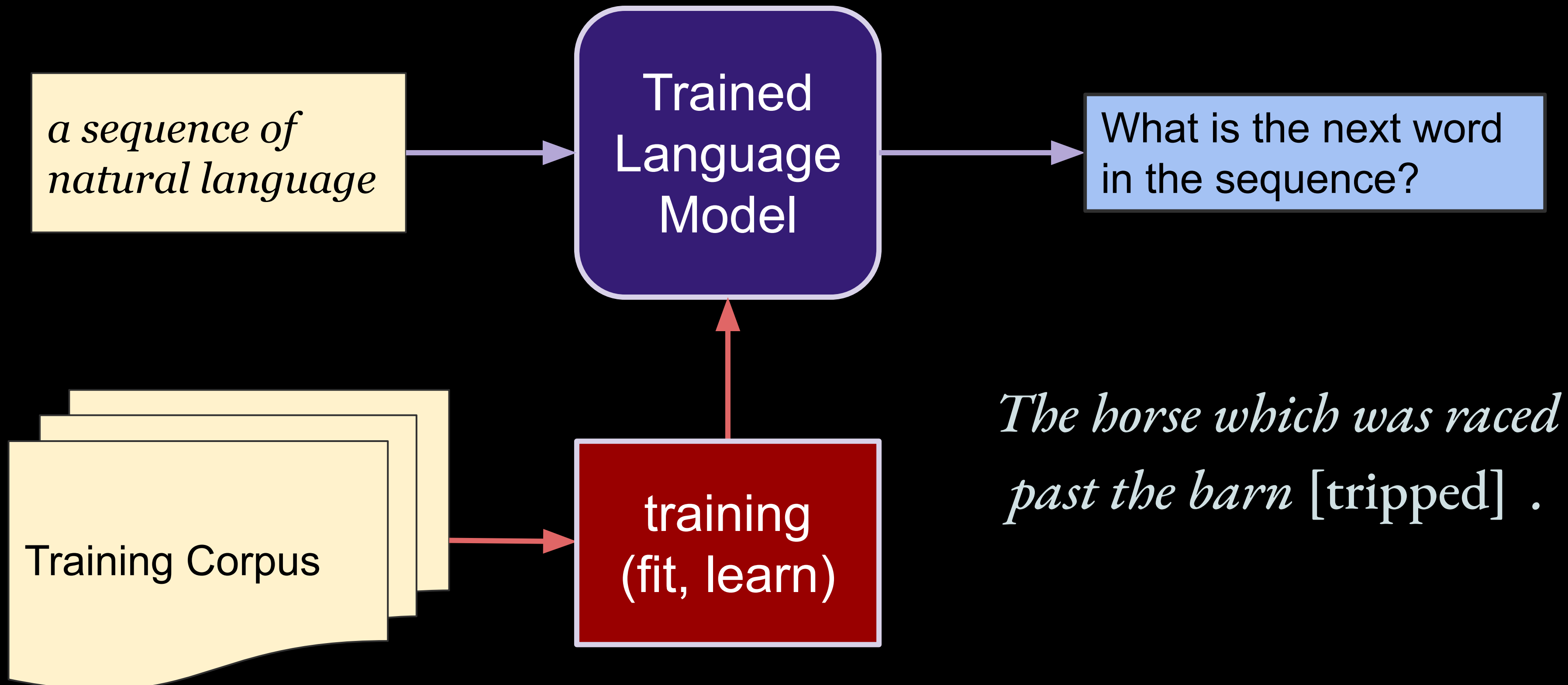


Language modeling with an RNN

Neural Networks

Language Modeling

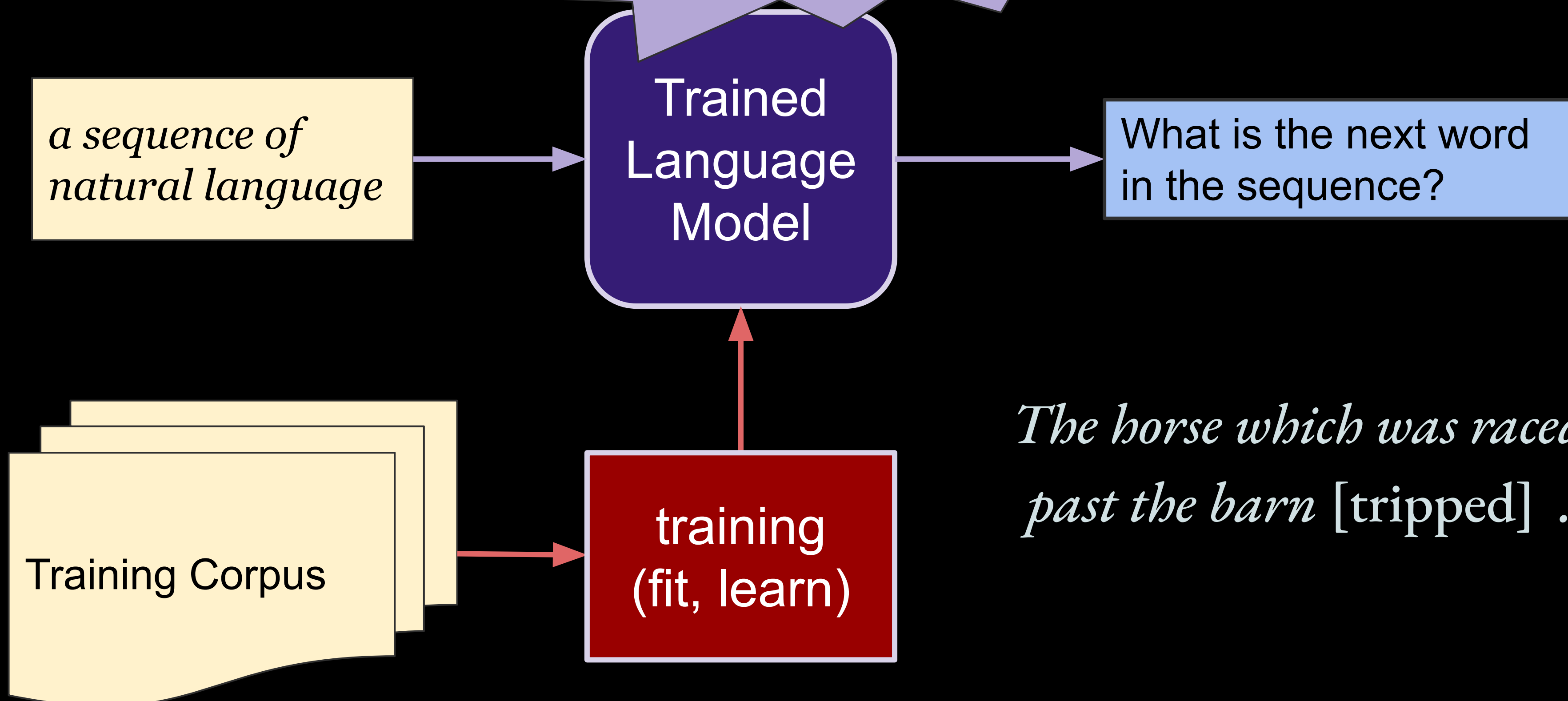
Building a model (or system / API) that can answer the following:



Language Modelin…

To fully capture natural language, models get very complex!

Building a model (or s… …er the following:



Neural Networks: Graphs of Operations

Neural Networks: Graphs of Operations (excluding the optimization nodes)

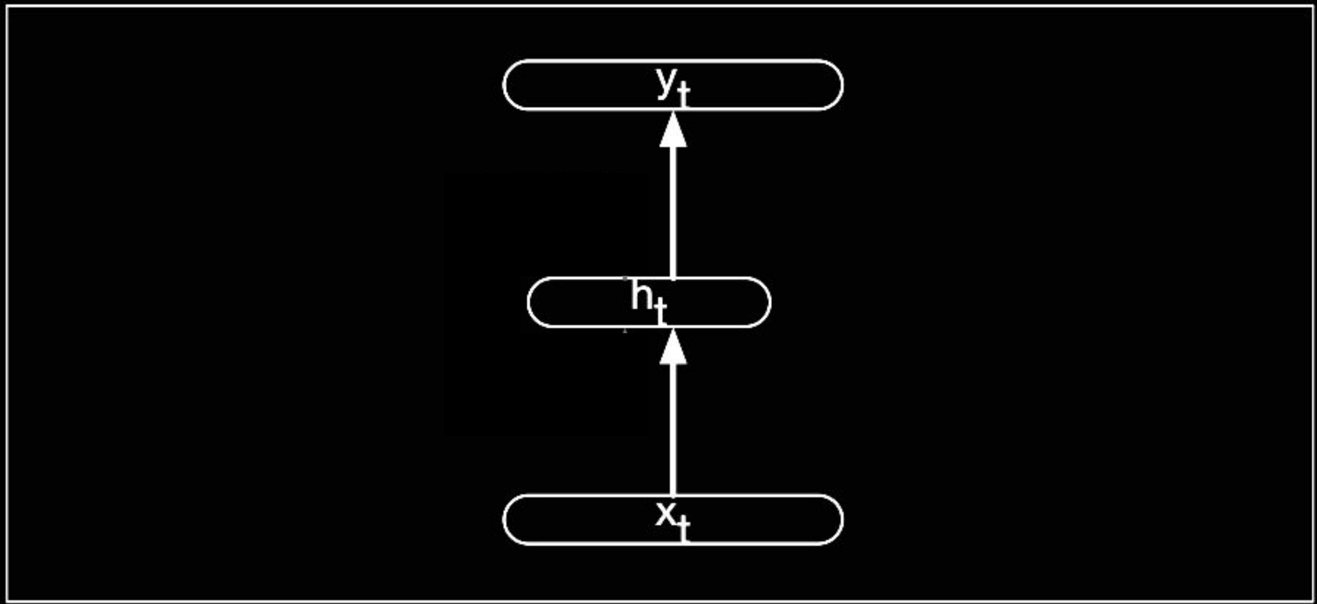


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

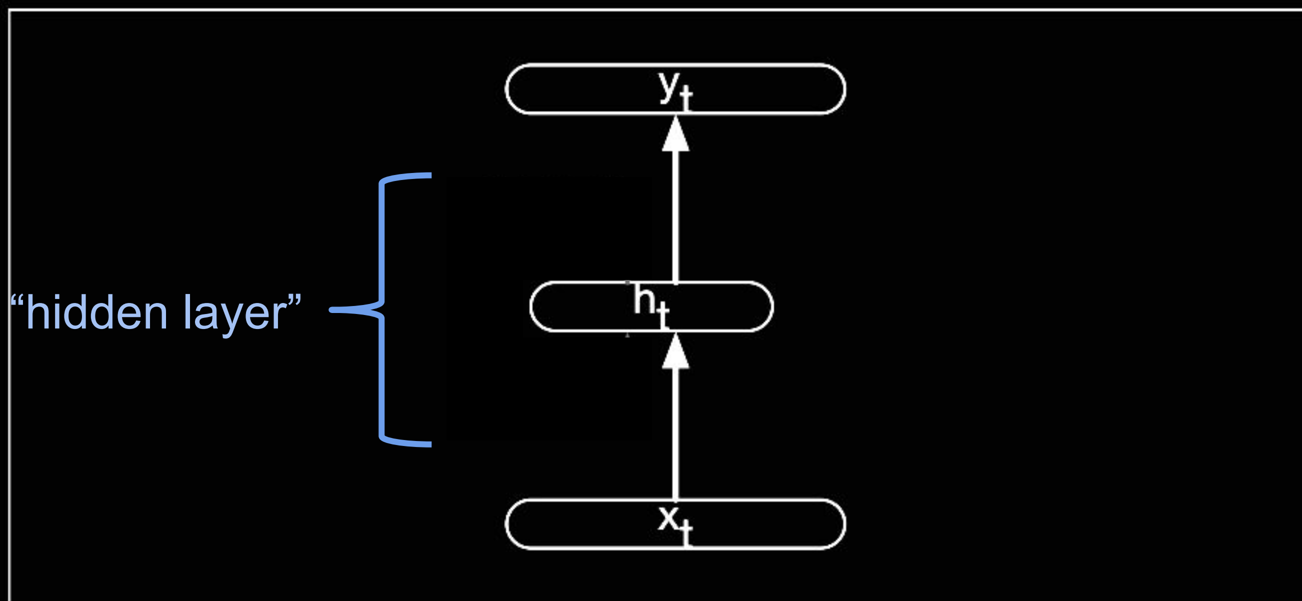


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

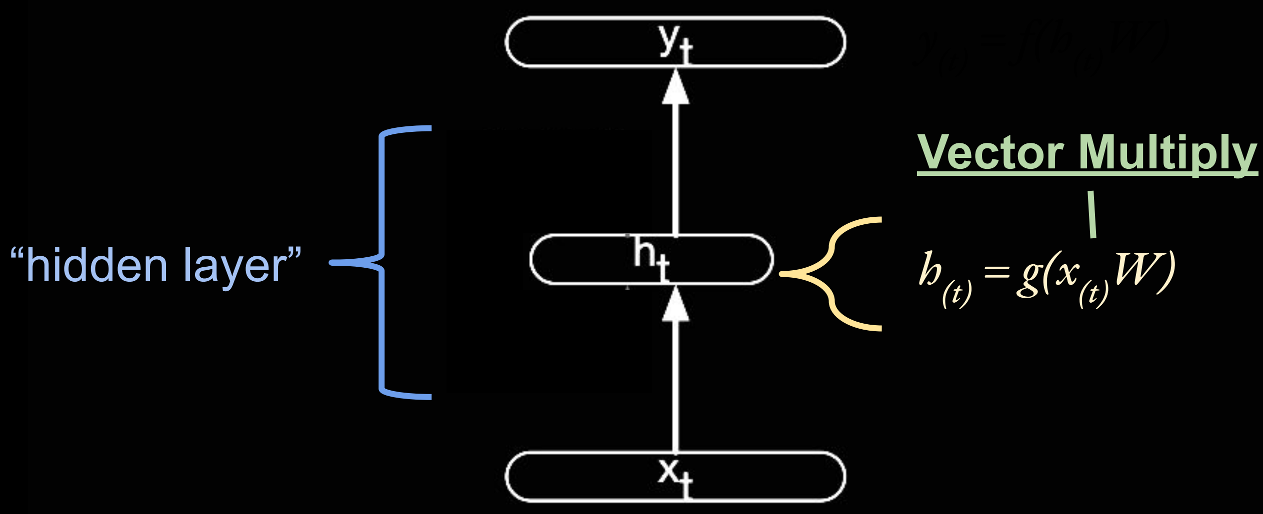
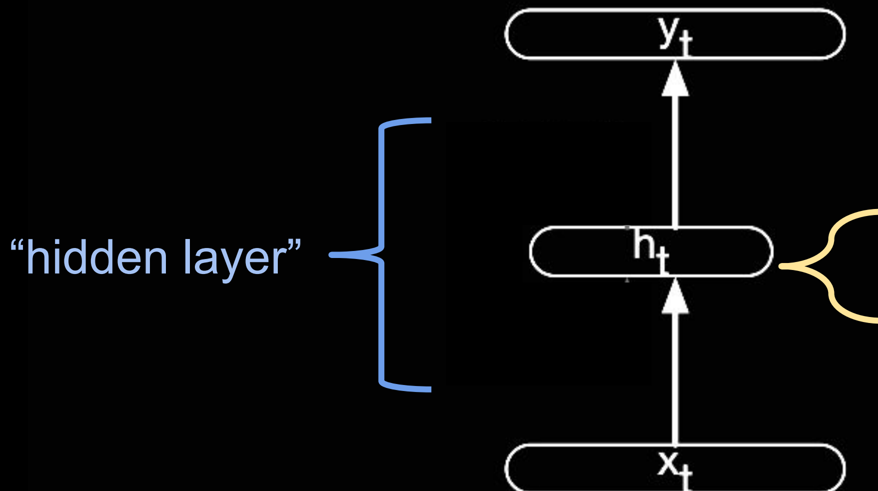


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)



Activation Function

$$h_{(t)} = g(x_{(t)}W)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Common Activation Functions

$z = h_{(t)}W$

Logistic: $\sigma(z) = 1 / (1 + e^{-z})$

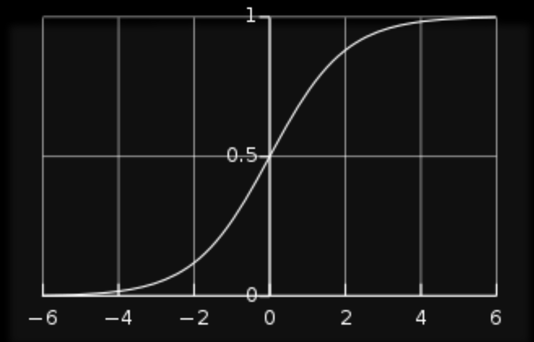
Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$

Common Activation Functions

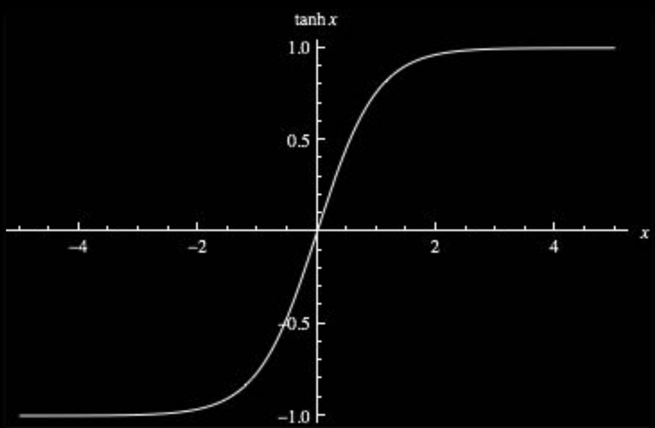
$z = h_{(t)}W$

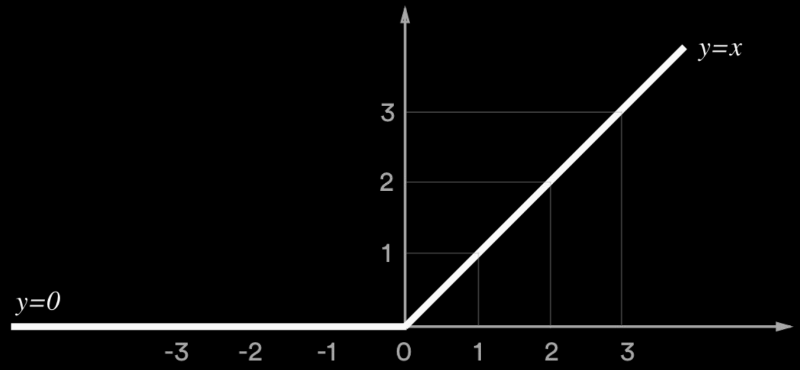
Logistic: $\sigma(z) = 1 / (1 + e^{-z})$



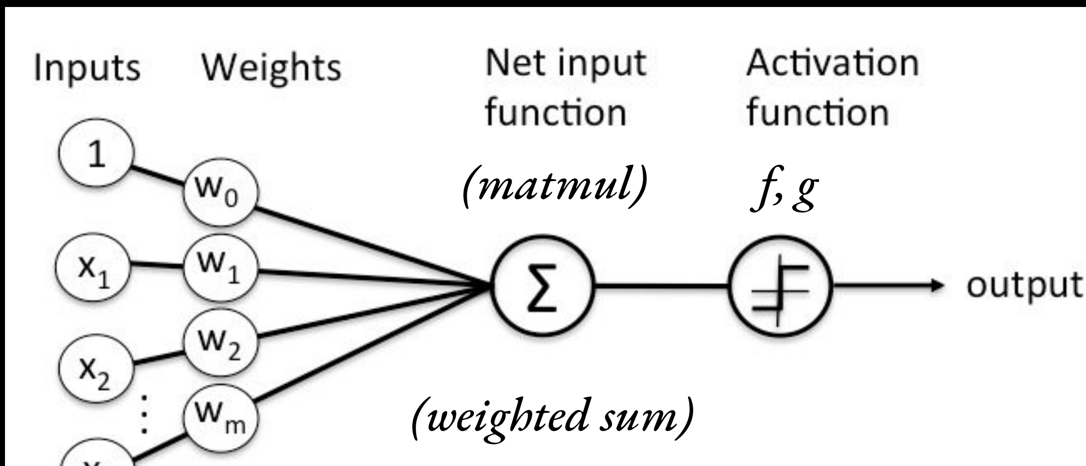
Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$





Neural Networks: Graphs of Operations (excluding the optimization nodes)



(skymind, AI Wiki)

Activation Function

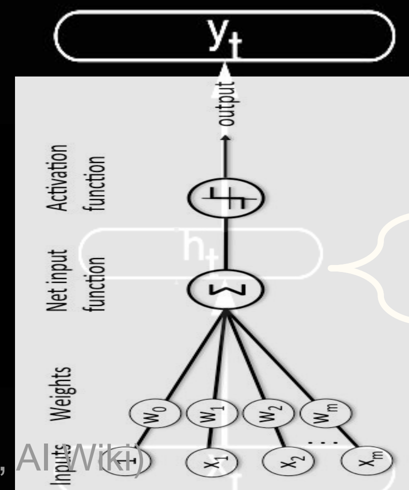
$$h_{(t)} = g(x_{(t)} W)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

“hidden layer”



(skymind, AI Wiki)

Activation Function

$$h_{(t)} = g(x_{(t)}W)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

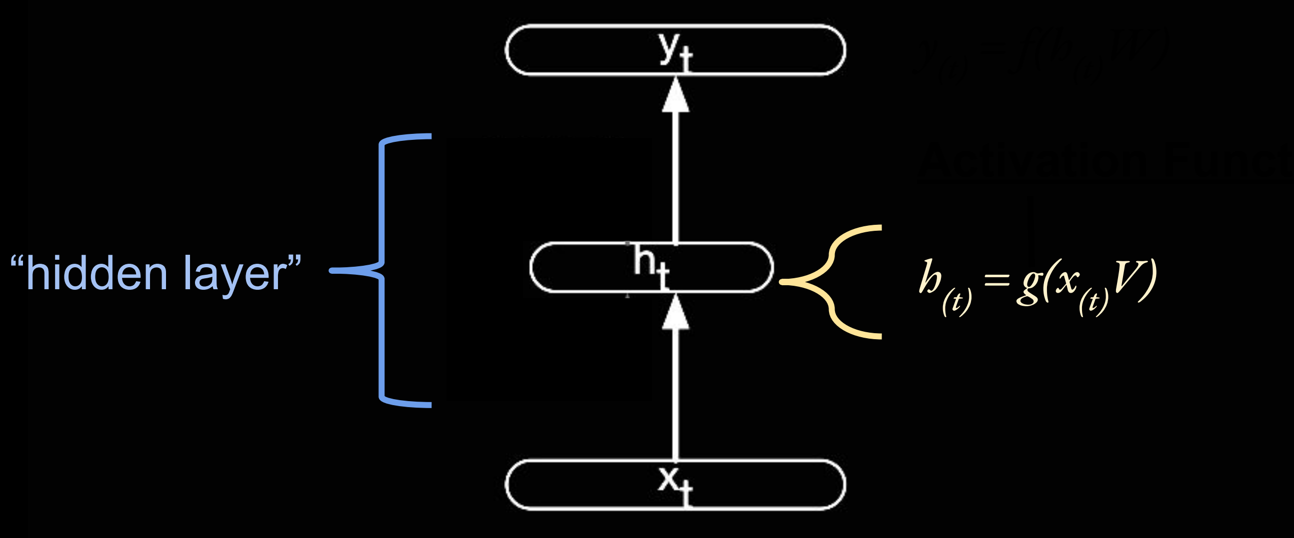
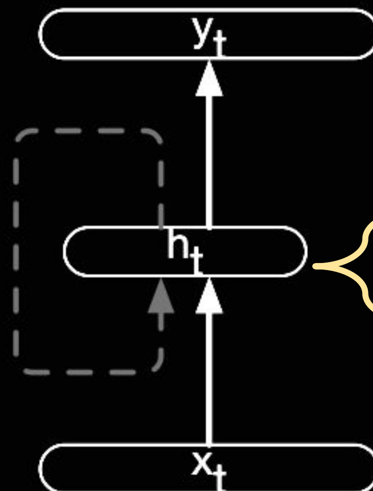


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

“hidden layer”



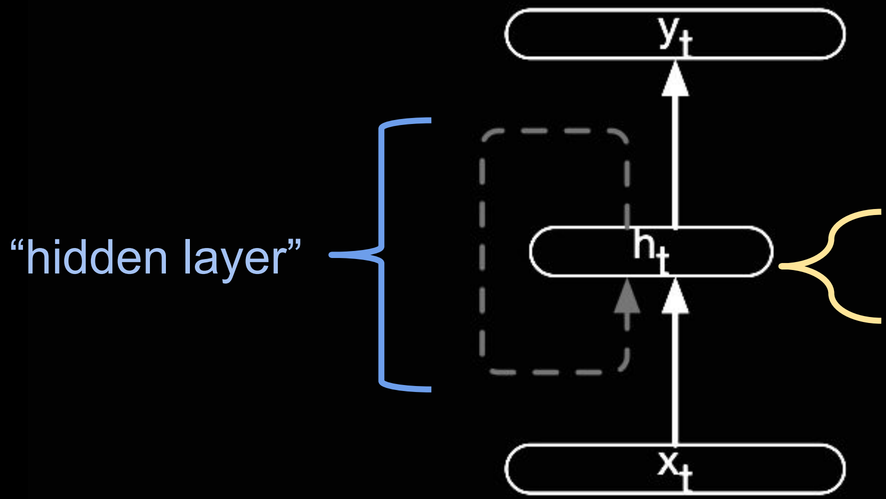
$$h_t = g(h_{t-1}U + x_tV)$$

short hand for vector/ matrix multiply

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)



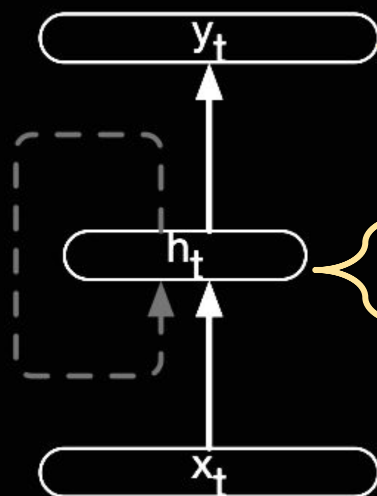
$$h_{(t)} = g(h_{(t-1)}U + x_{(t)}V)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

“hidden layer”



$$y_{(t)} = f(h_{(t)}W)$$

Activation Function

$$h_{(t)} = g(h_{(t-1)}U + x_{(t)}V)$$

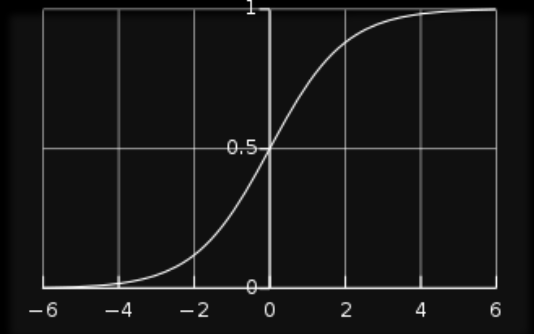
Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Common Activation Functions

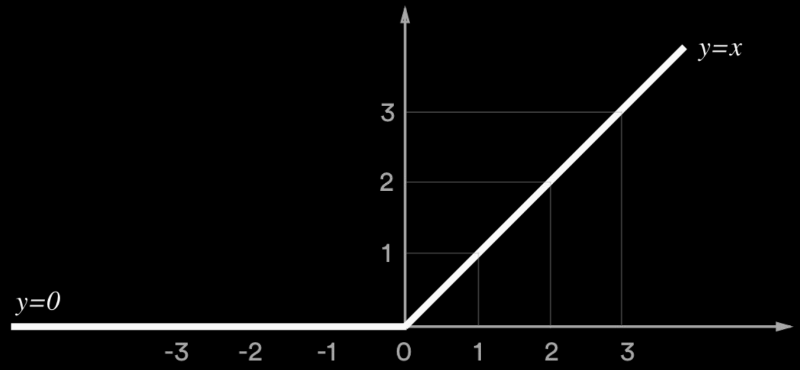
$z = h_{(t)}W$

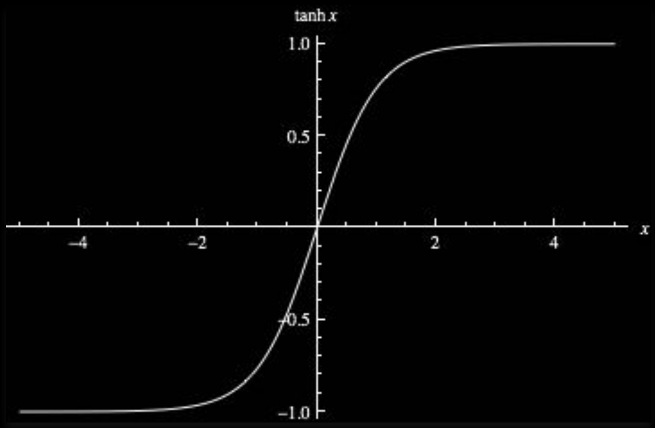
Logistic: $\sigma(z) = 1 / (1 + e^{-z})$



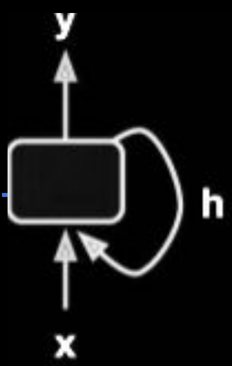
Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$





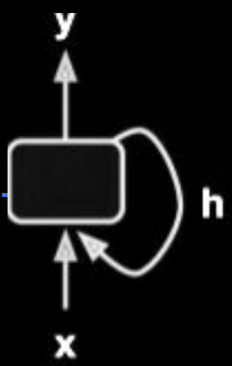
Example: Forward Pass



(Geron, 2017)

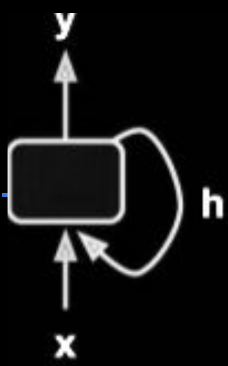
```
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = g(U h(i-1) + W x(i)) #update hidden state
    y(i) = f(V h(i)) #update output
```

Example: Forward Pass




```
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = g(U h(i-1) + W x(i)) #update hidden state
    y(i) = f(V h(i)) #update output
```

Example: Forward Pass



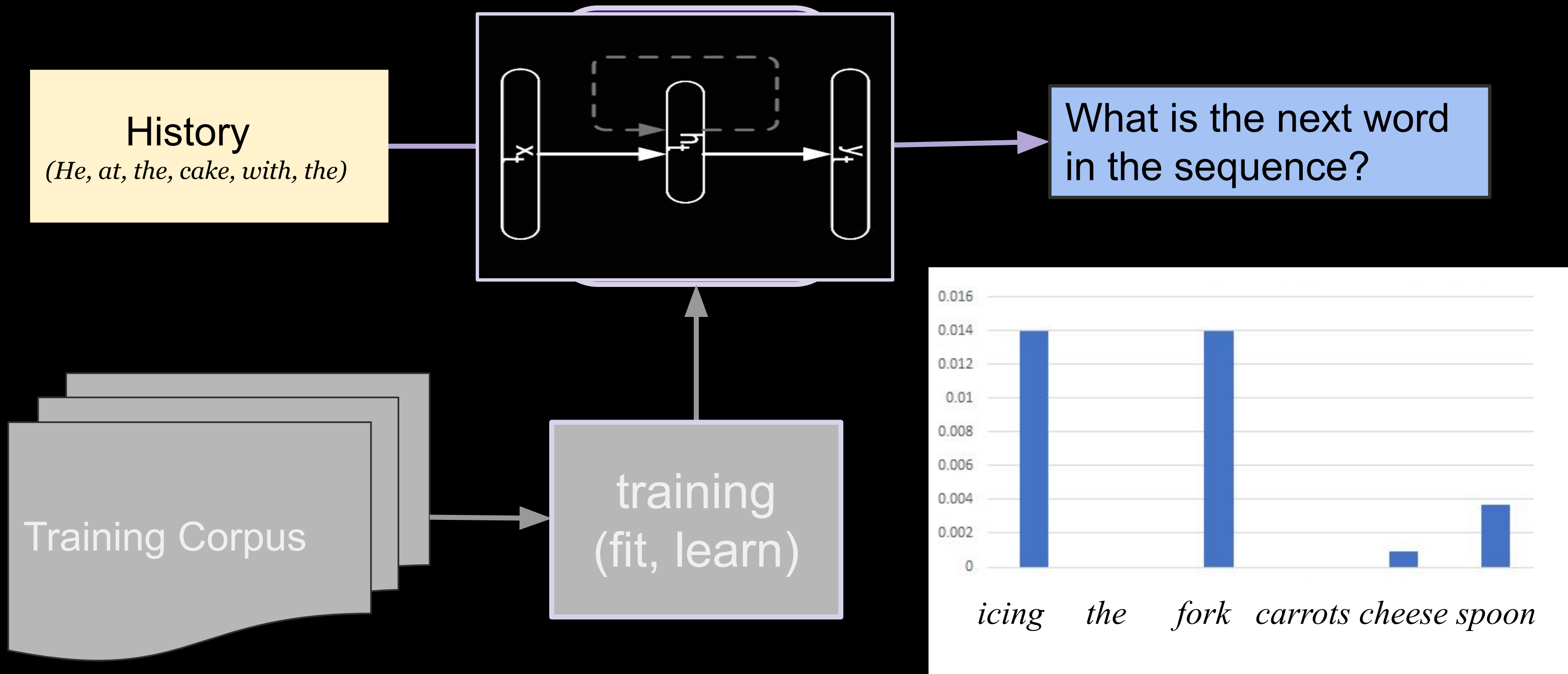

```
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tanh(matmul(U,h(i-1))+ matmul(W,x(i))) #update hidden state
    y(i) = softmax(matmul(V, h(i))) #update output
```

Language Modeling

Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

$P(fork \mid He\ ate\ the\ cake\ with\ the) = ?$

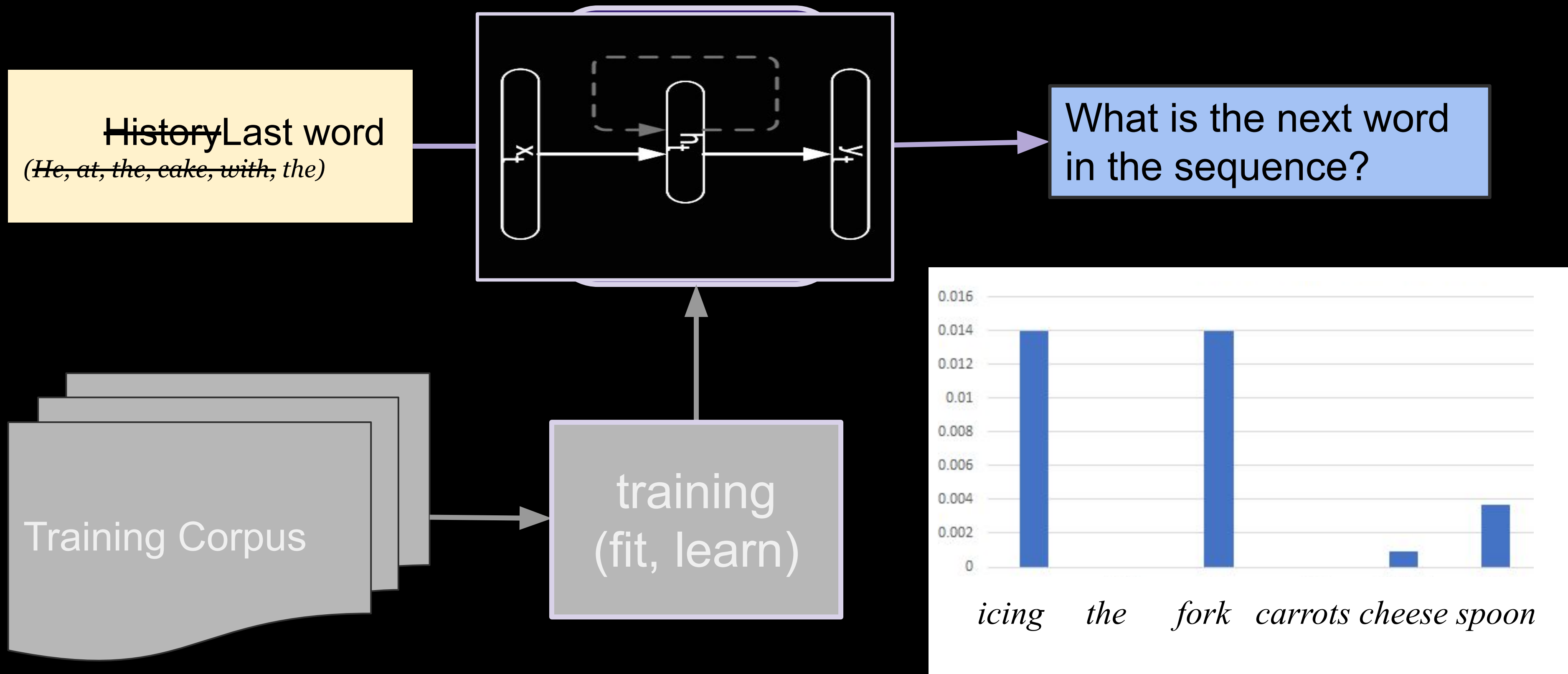


Language Modeling

Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

P(fork | He ate the cake with the) = ?



Language Modeling

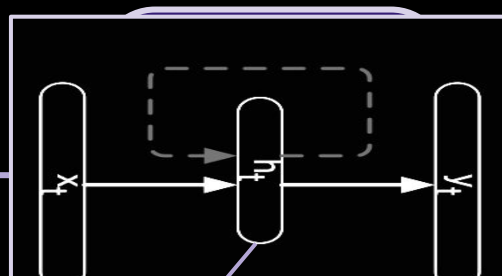
Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

P(fork | He ate the cake with the) = ?

Last word

(the)



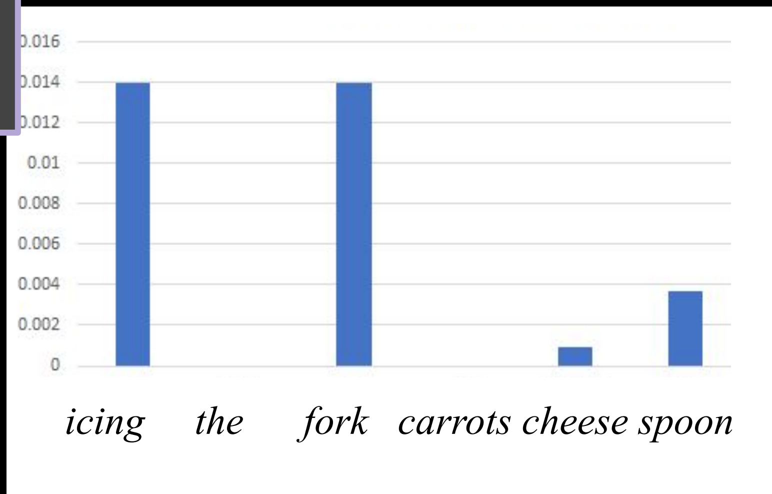
What is the next word in the sequence?

h_t: a vector that we hope "stores" relevant history from previous inputs:

He, at, the, cake, with,

Training Corpus

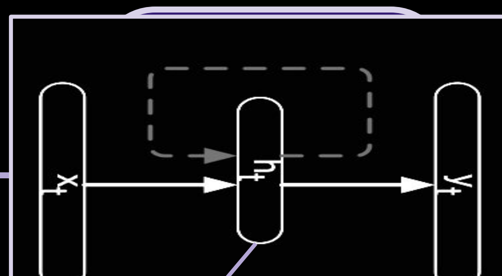
training (fit, learn)



RNN Limitation: Not para

Last word

(the)



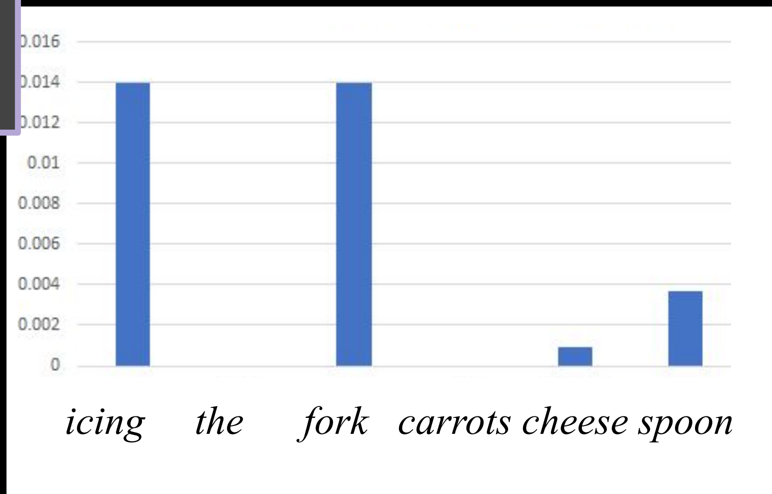
What is the next word in the sequence?

h_t: a vector that we hope “stores” relevant history from previous inputs:

He, at, the, cake, with,

Training Corpus

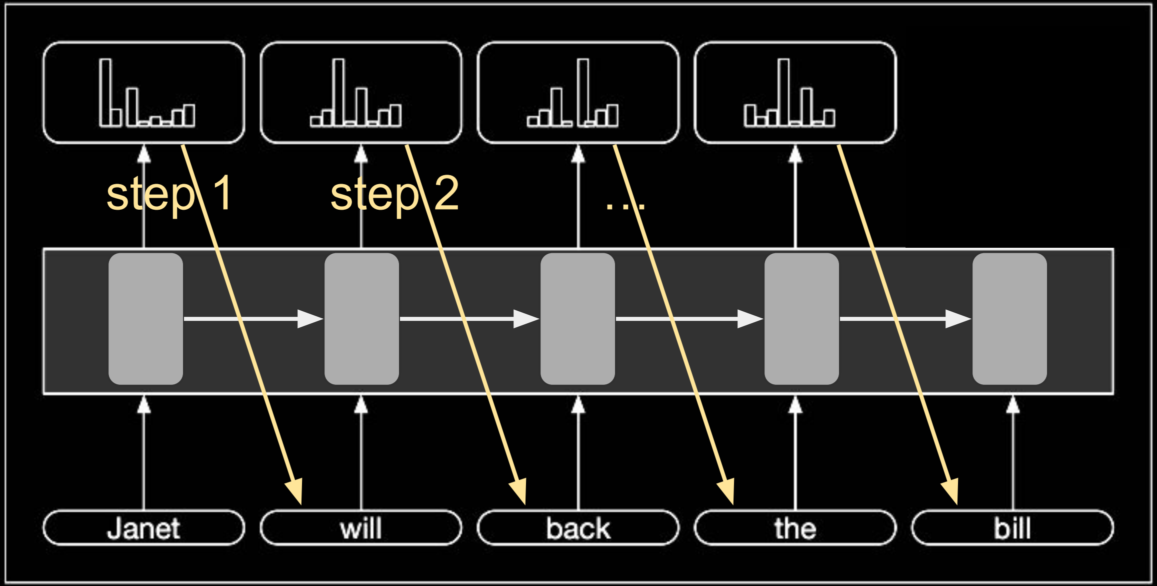
training (fit, learn)



RNN: Limitation:
Losing Track of Long Distance Dependencies

The horse which was raced past the barn tripped .

RNN: Limitation: Not parallelizable



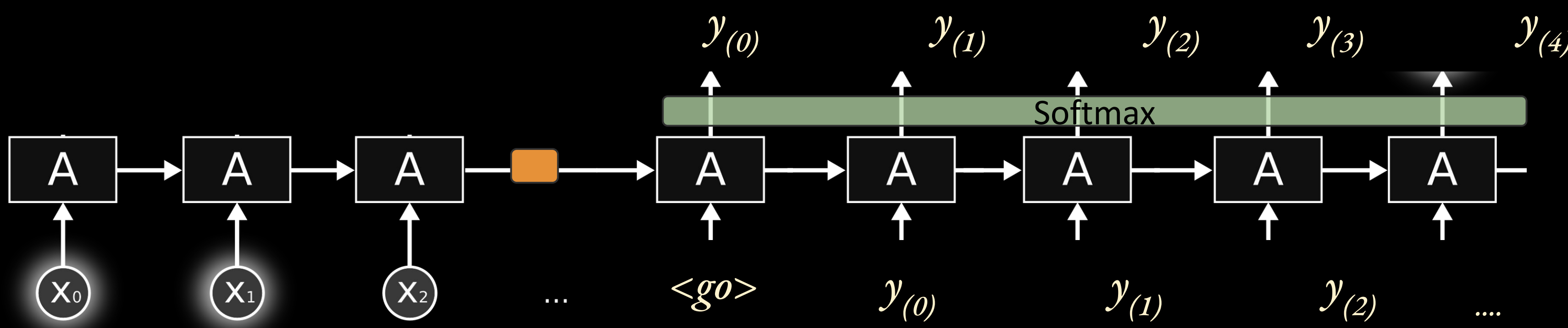
Language modeling with an RNN

The Transformer: Motivation

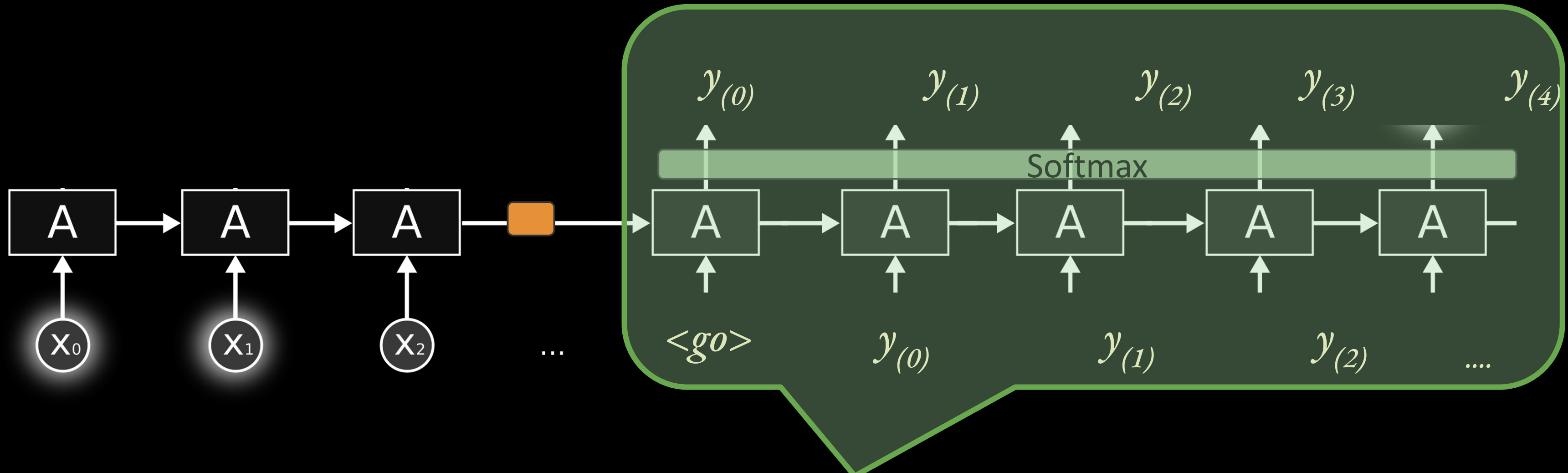
Challenges to sequential representation learning

- Capture long-distance dependencies
- Preserving sequential distances / periodicity
- Capture multiple relationships
- Easy to parallelize -- don’t need sequential processing.

Encoder-Decoder (Simpler Representation)

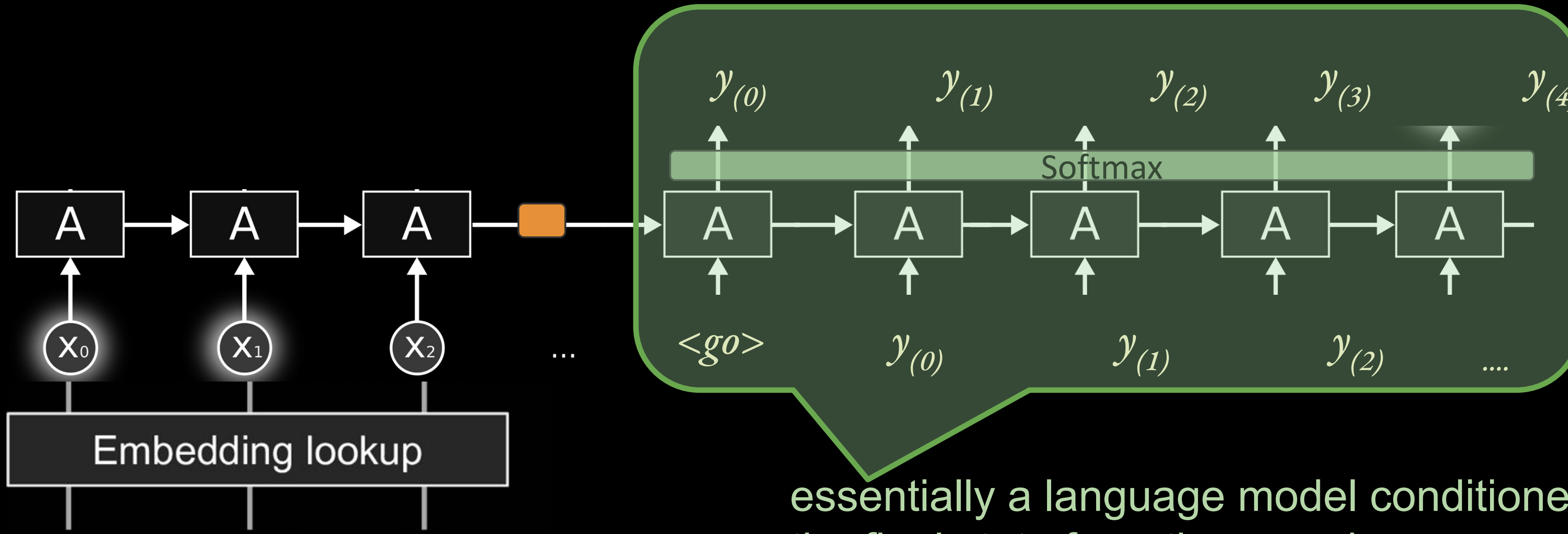


Encoder-Decoder (Simpler Representation)



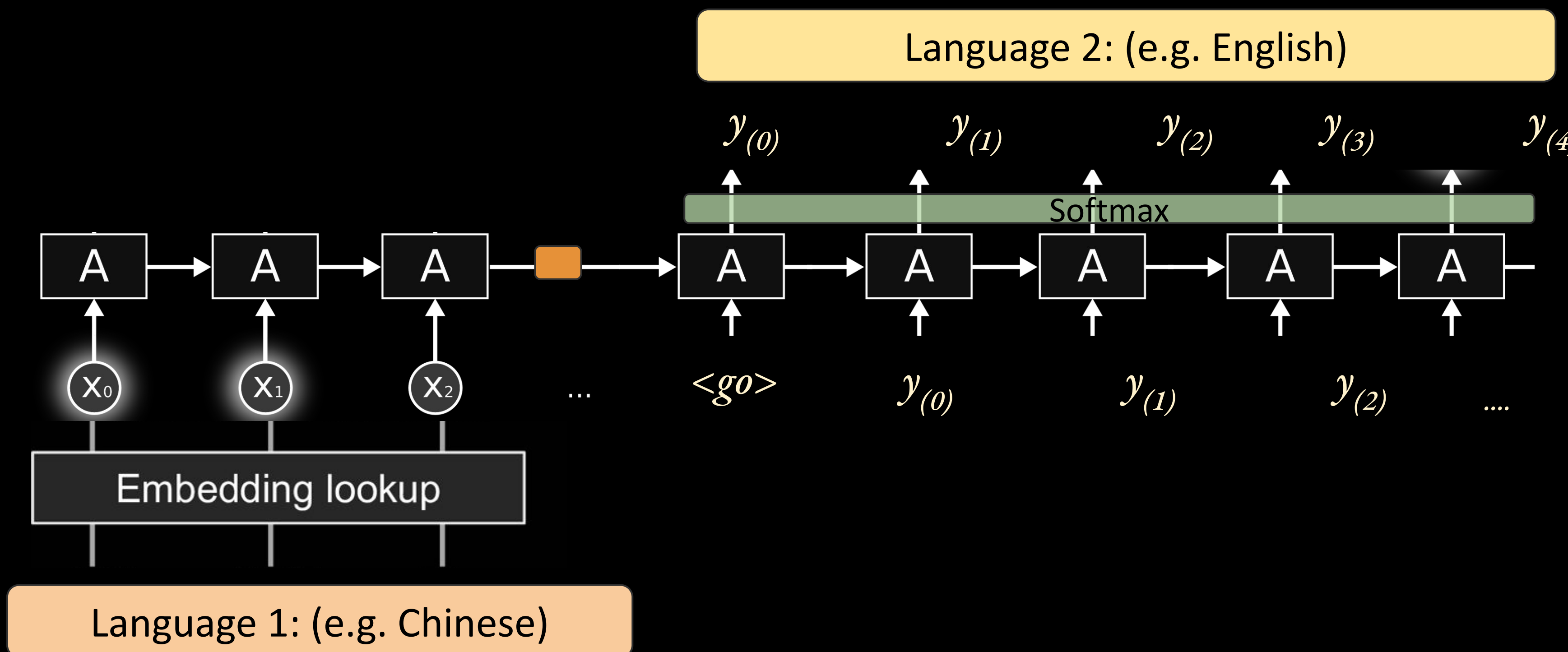
essentially a language model conditioned on the final state from the encoder.

Encoder-Decoder (Simpler Representation)



essentially a language model conditioned on the final state from the encoder.

Encoder-Decoder (Simpler Representation)

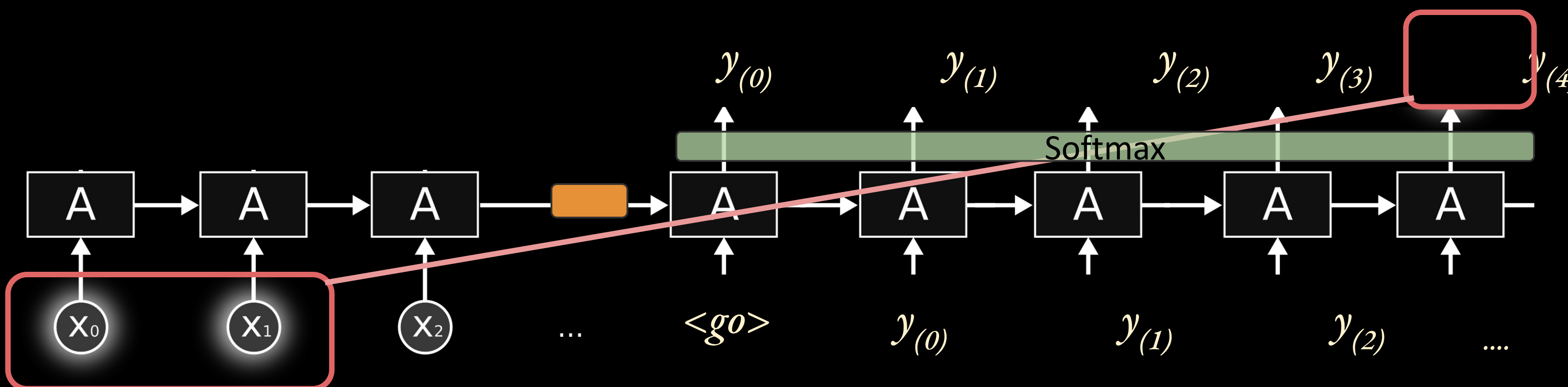


Encoder-Decoder

Challenge:

The ball was kicked by kayla.

- Long distance dependency when translating:



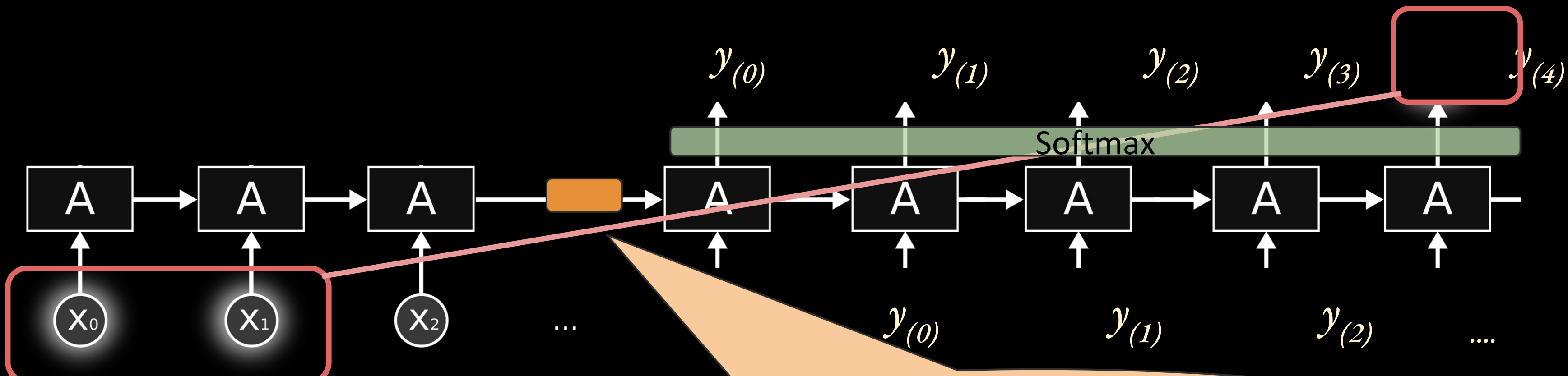
Kayla kicked the ball.

Encoder-Decoder

Challenge:

The ball was kicked by kayla.

- Long distance dependency when translating:



Kayla kicked the ball.

A lot of responsibility put fixed-size hidden state passed from encoder to decoder

Transformer Language Models: Uses multiple layers of a **transformer**

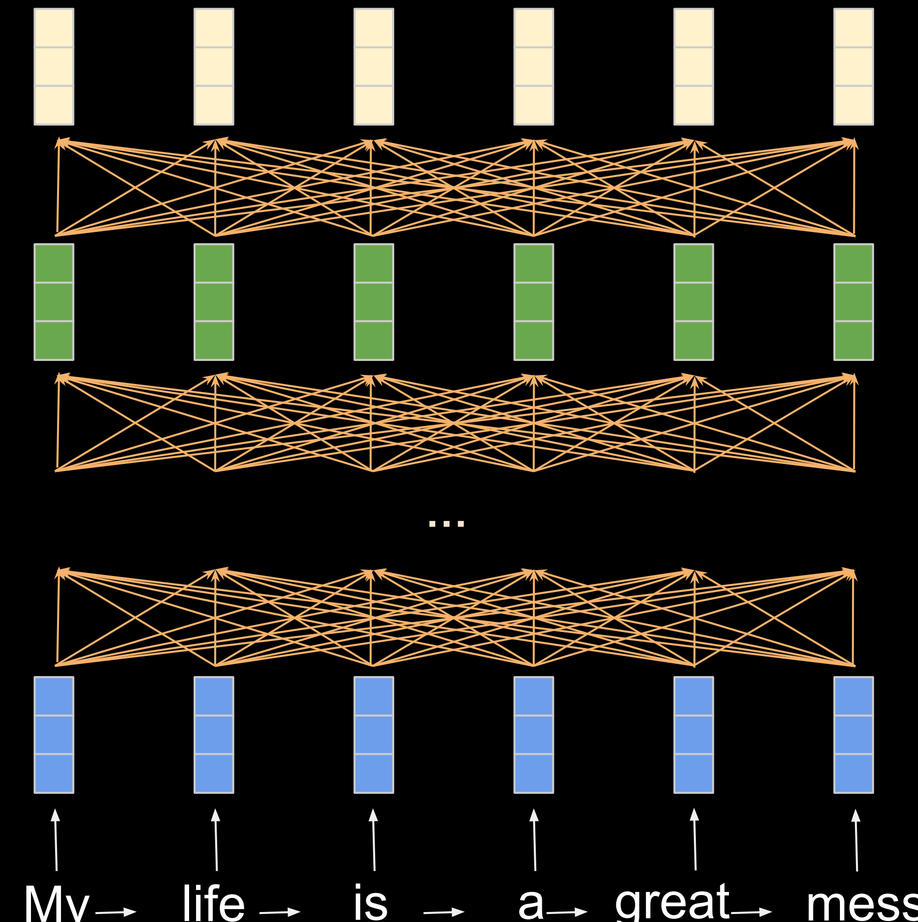
layer k:
(used for language modeling)

layer k-1:
(taken as contextual embedding)

layers 1 to k-2:
(compose embeddings with context)

layer 0:
(input: word-type embeddings)

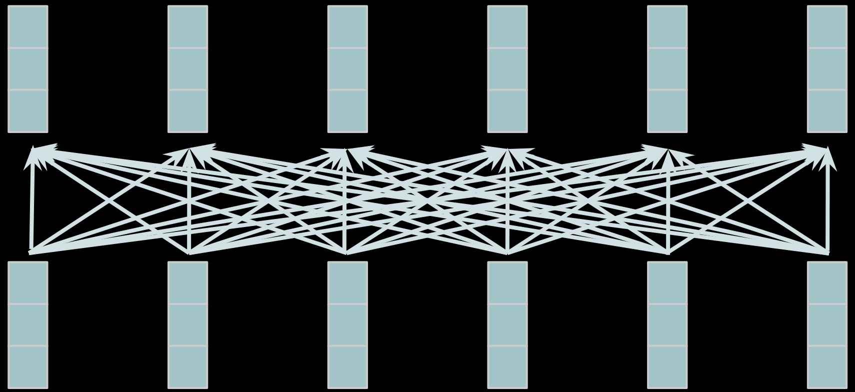
sentence (sequence) input:



(Kjell, Kjell, and Schwartz, 2023)

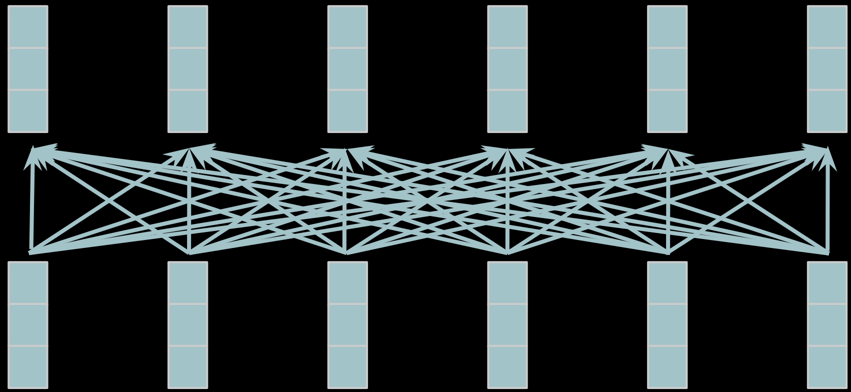
auto-encoder:

- Connections go both directions.
- Task is predict word in middle: p(wi| …, pwi-2, wi-1, wi+1, wi+2…)
- Better for:
  - embeddings
  - fine-tuning (transfer learning)



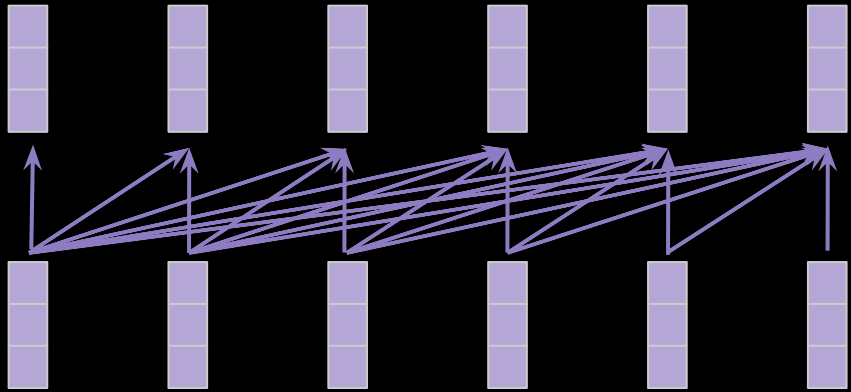
auto-encoder:

- Connections go both directions.
- Task is predict word in middle:
  p(wi| …, pwi-2, wi-1, wi+1, wi+2…)
- Better for:
  - embeddings
  - fine-tuning (transfer learning)



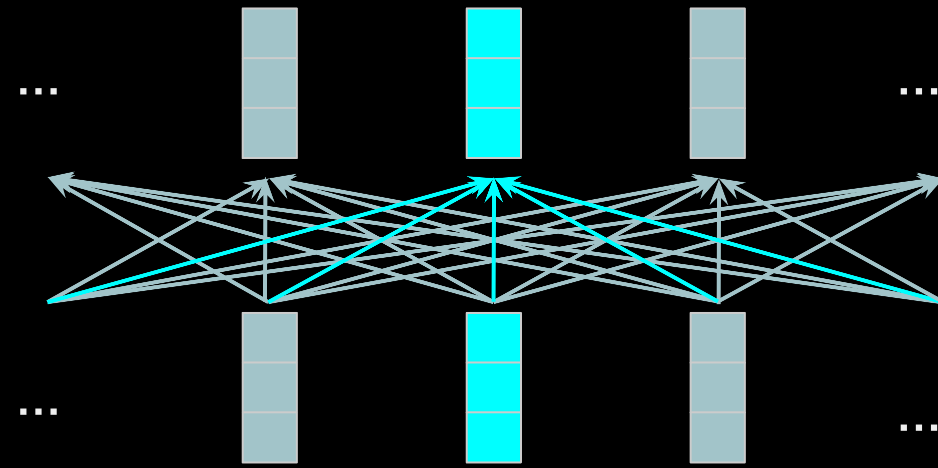
auto-regressor *(generator):*

- Connections go forward only
- Task is predict word next word:
  p(wi| wi-1, wi-2, …)
- Better for:
  - generating text
  - zero-shot learning



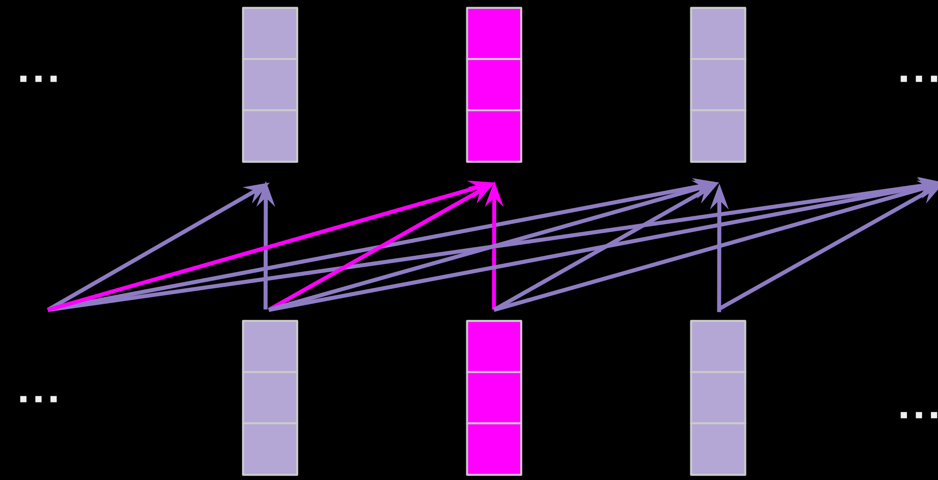
auto-encoder:

- Connections go both directions.
- Task is predict word in middle:
  p(wi| …, pwi-2, wi-1, wi+1, wi+2…)
- Better for:
  - embeddings
  - fine-tuning (transfer learning)

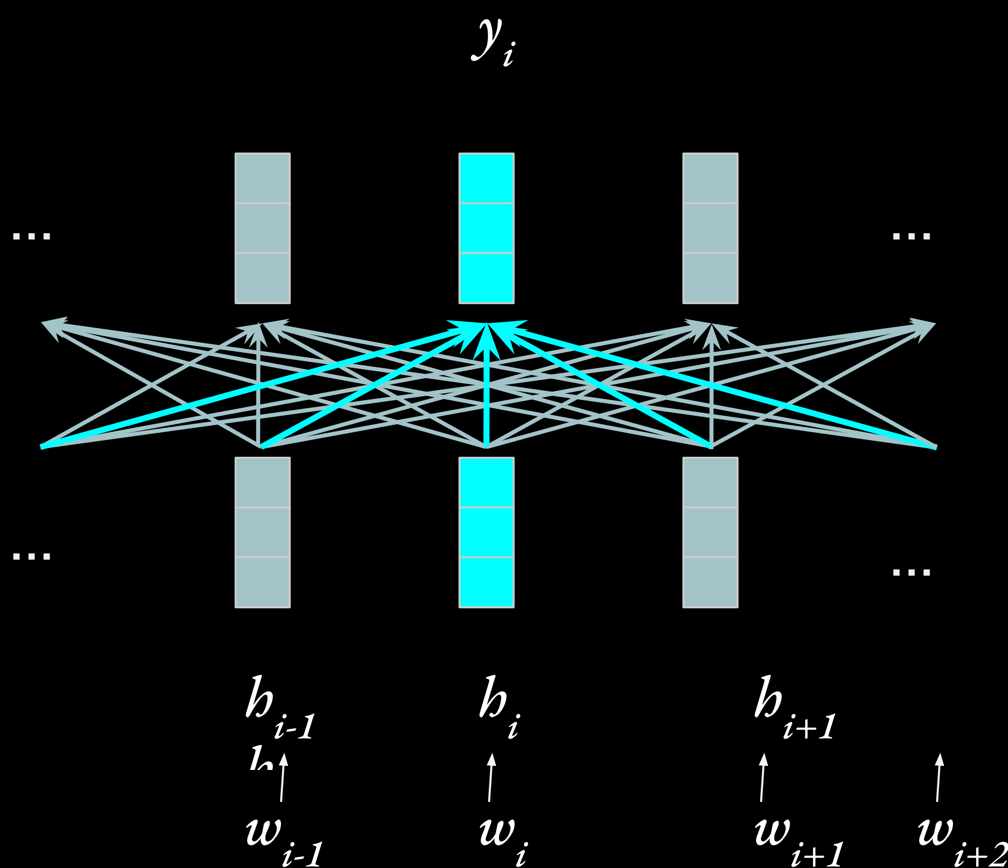


auto-regressor *(generator):*

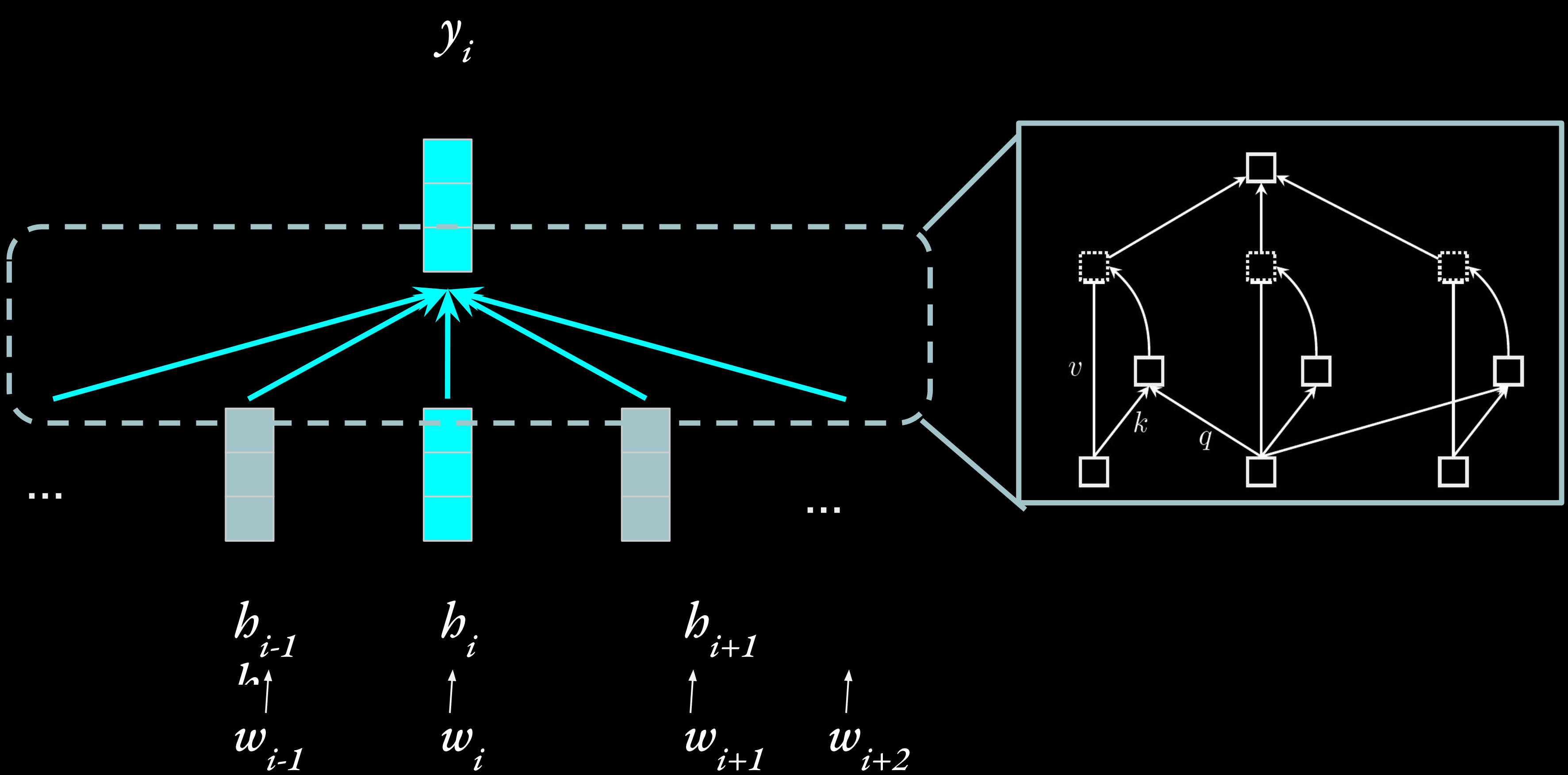
- Connections go forward only
- Task is predict word next word:
  p(wi| wi-1, wi-2, …)
- Better for:
  - generating text
  - zero-shot learning



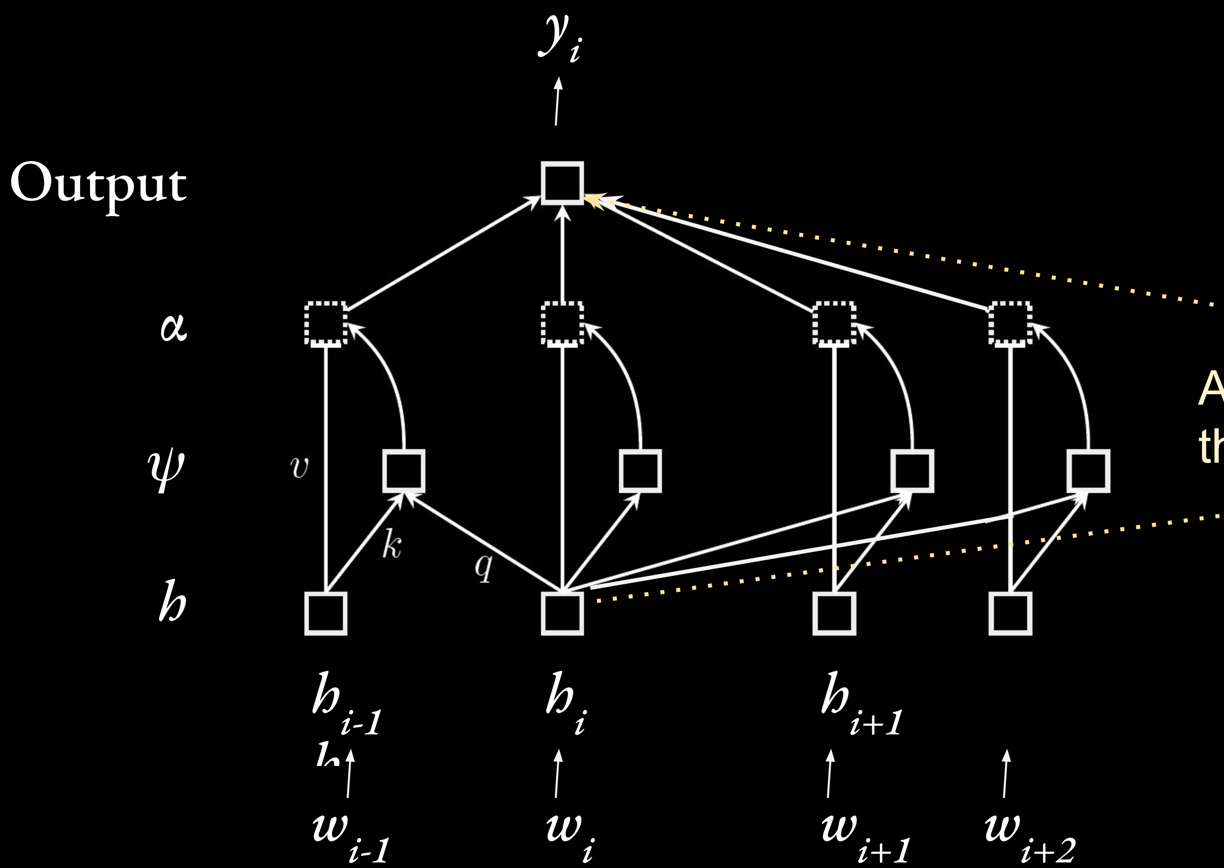
The Transformer's Heart: Self-Attention



The Transformer's Heart: Self-Attention

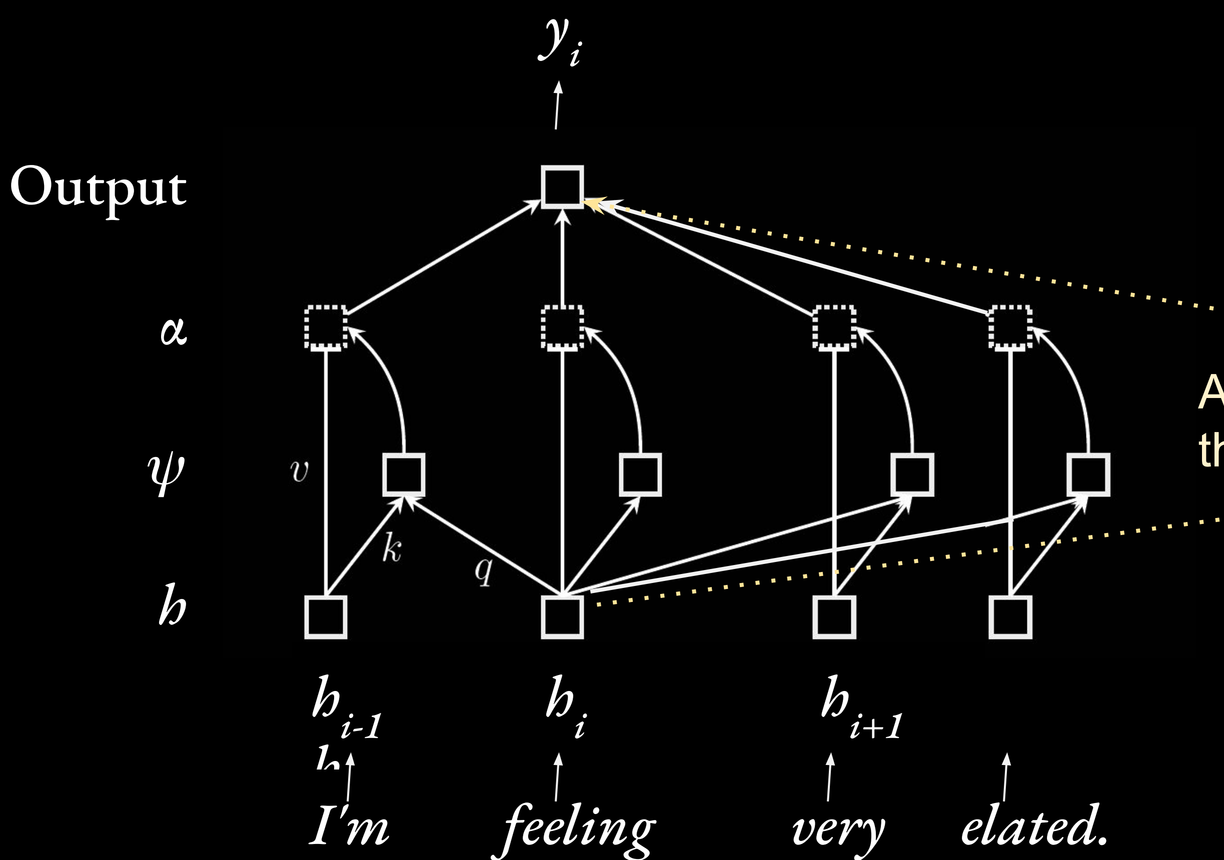


The Transformer's Heart: Self-Attention



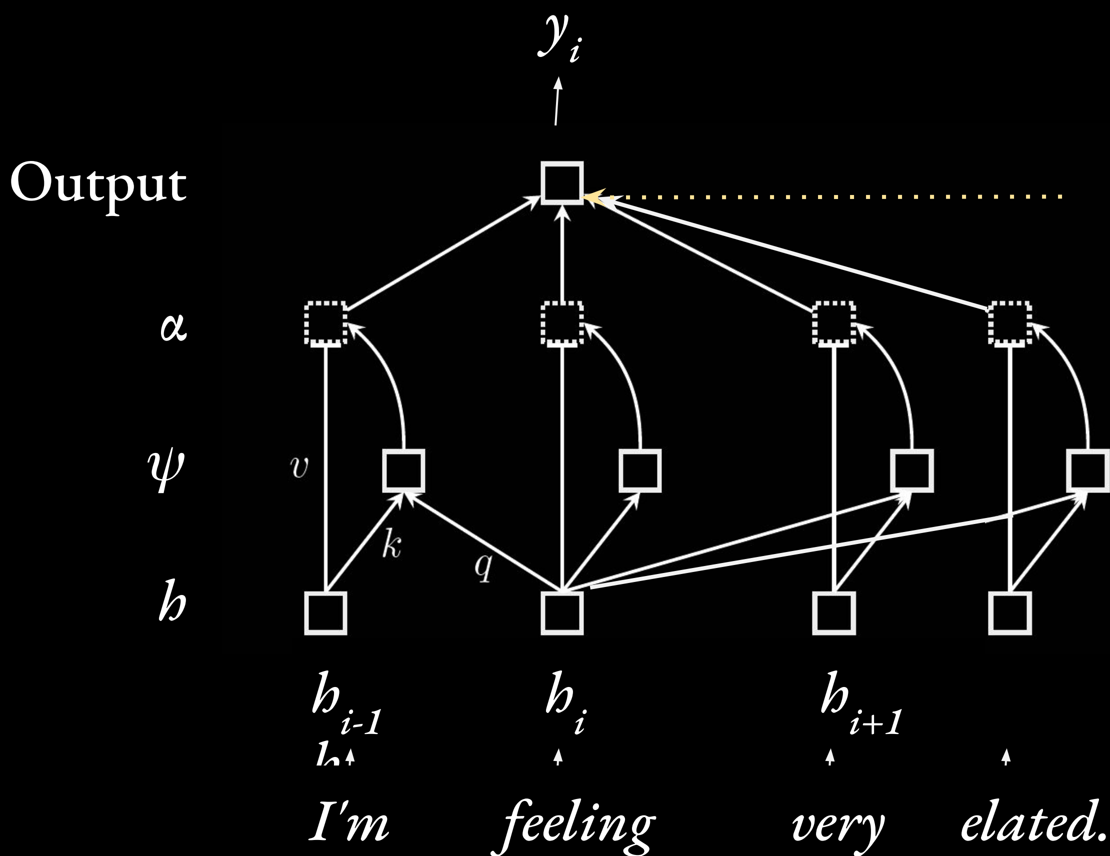
Attend to all other words in the sequence

The Transformer's Heart: Self-Attention



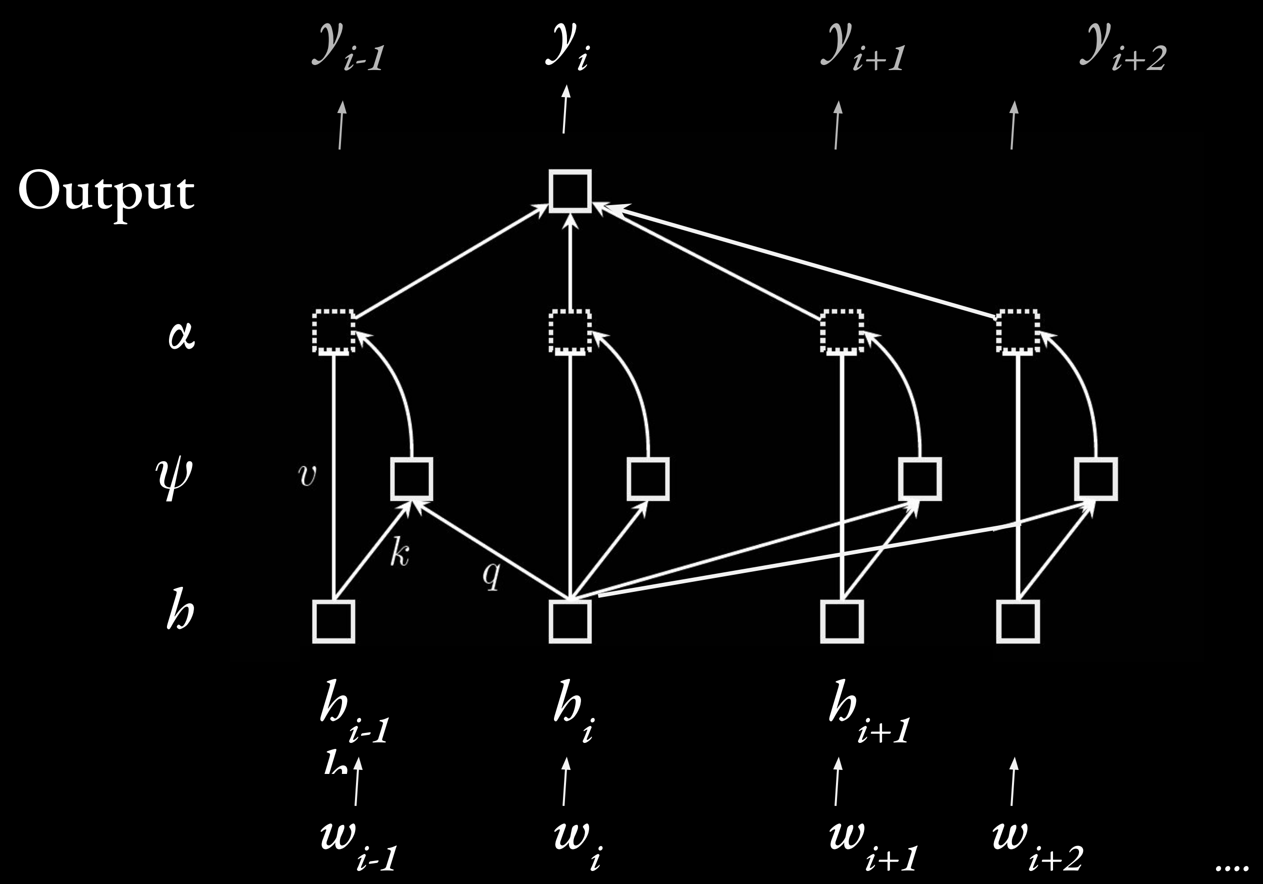
Attend to all other words in the sequence

The Transformer's Heart: Self-Attention

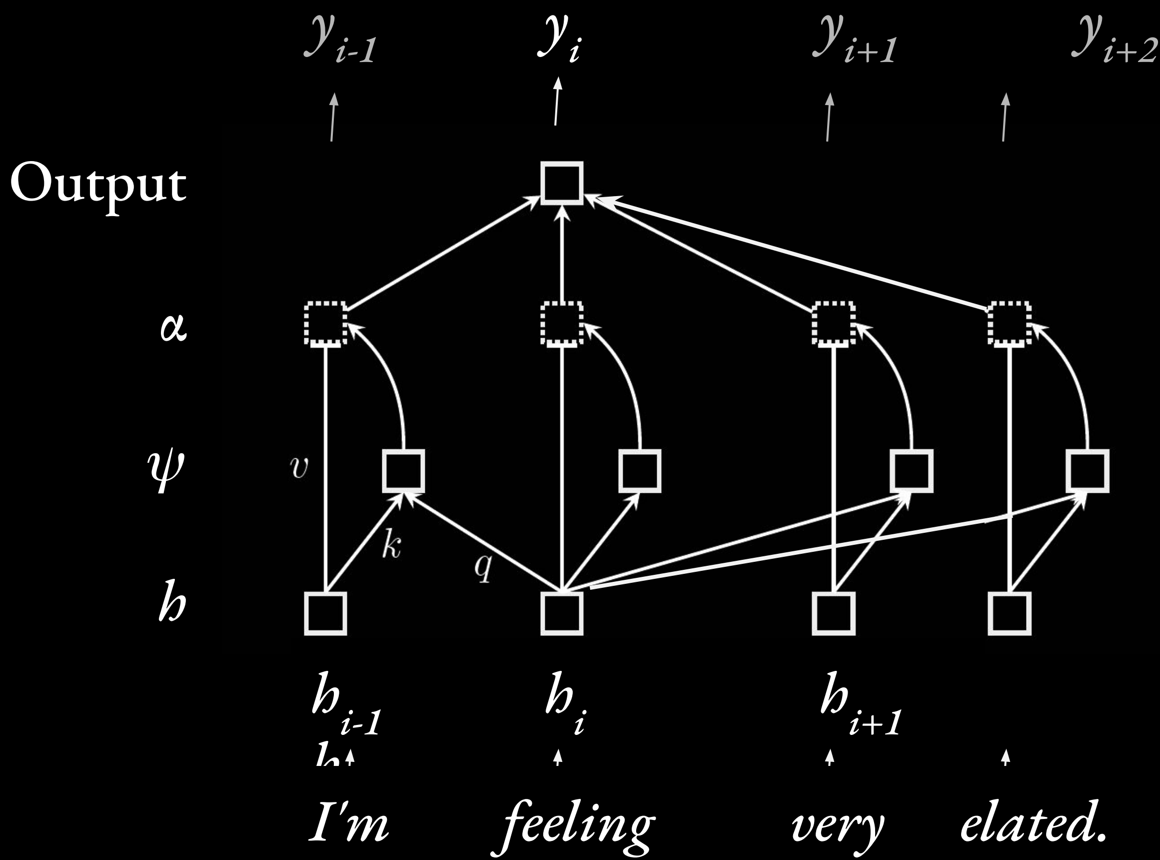


A weighted combination of other words' vectors.

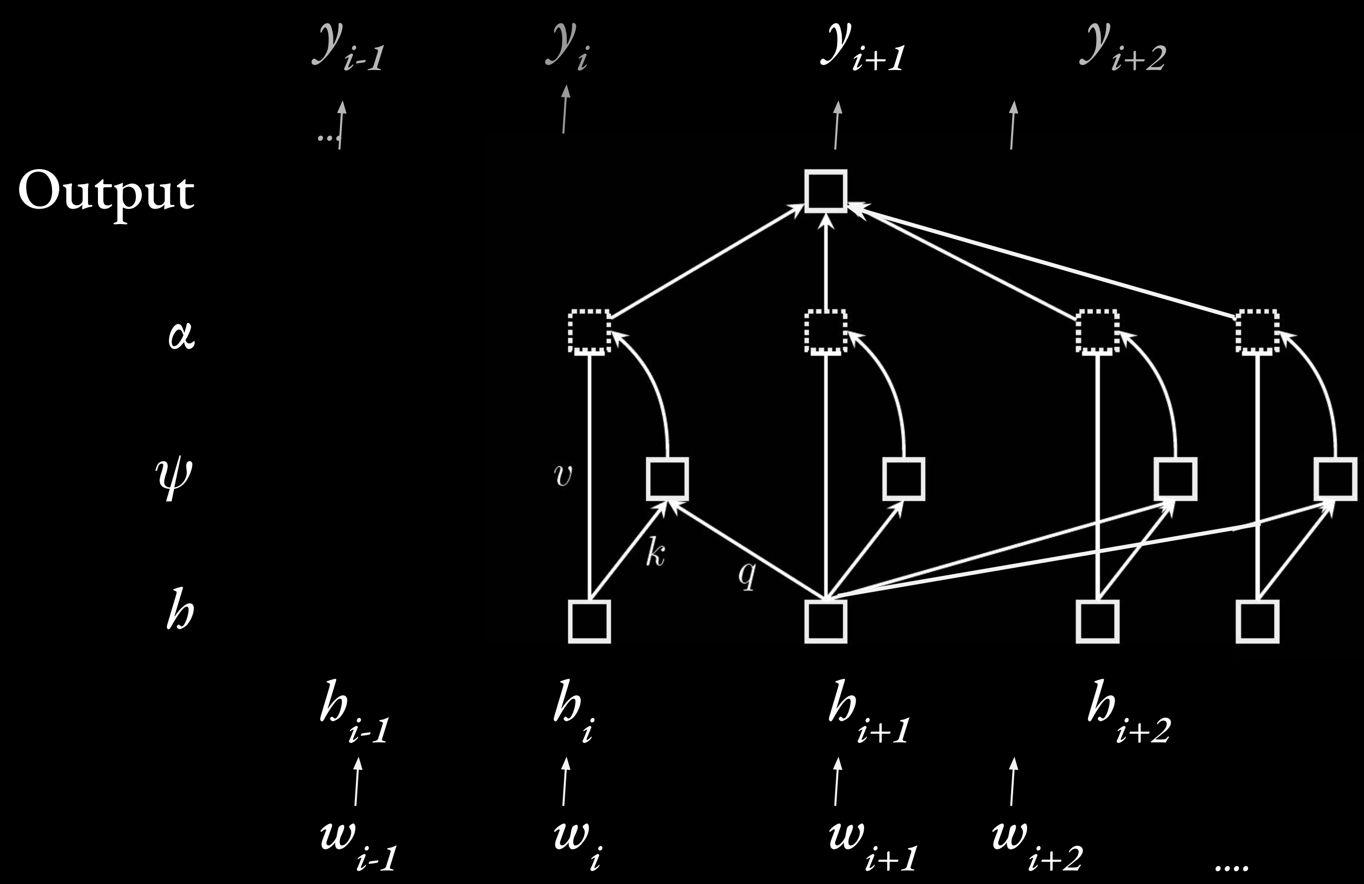
The Transformer's Heart: Self-Attention



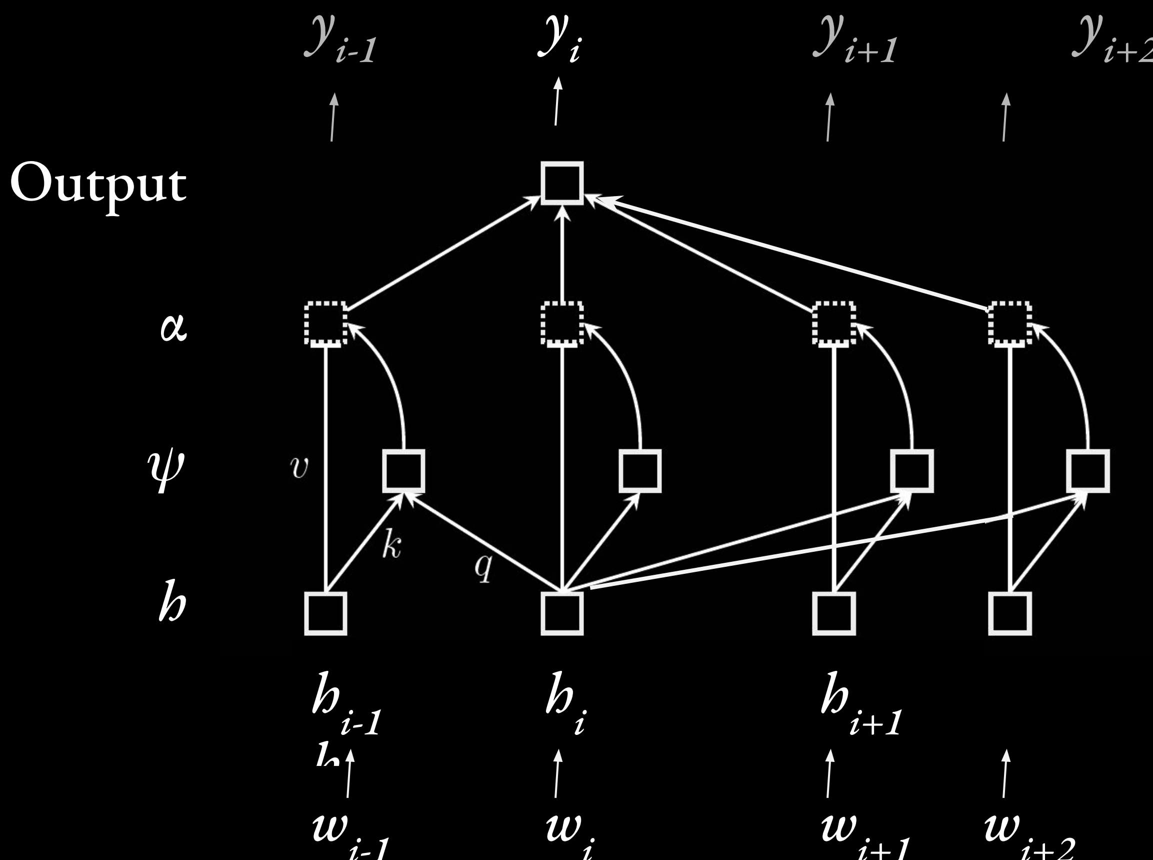
The Transformer's Heart: Self-Attention



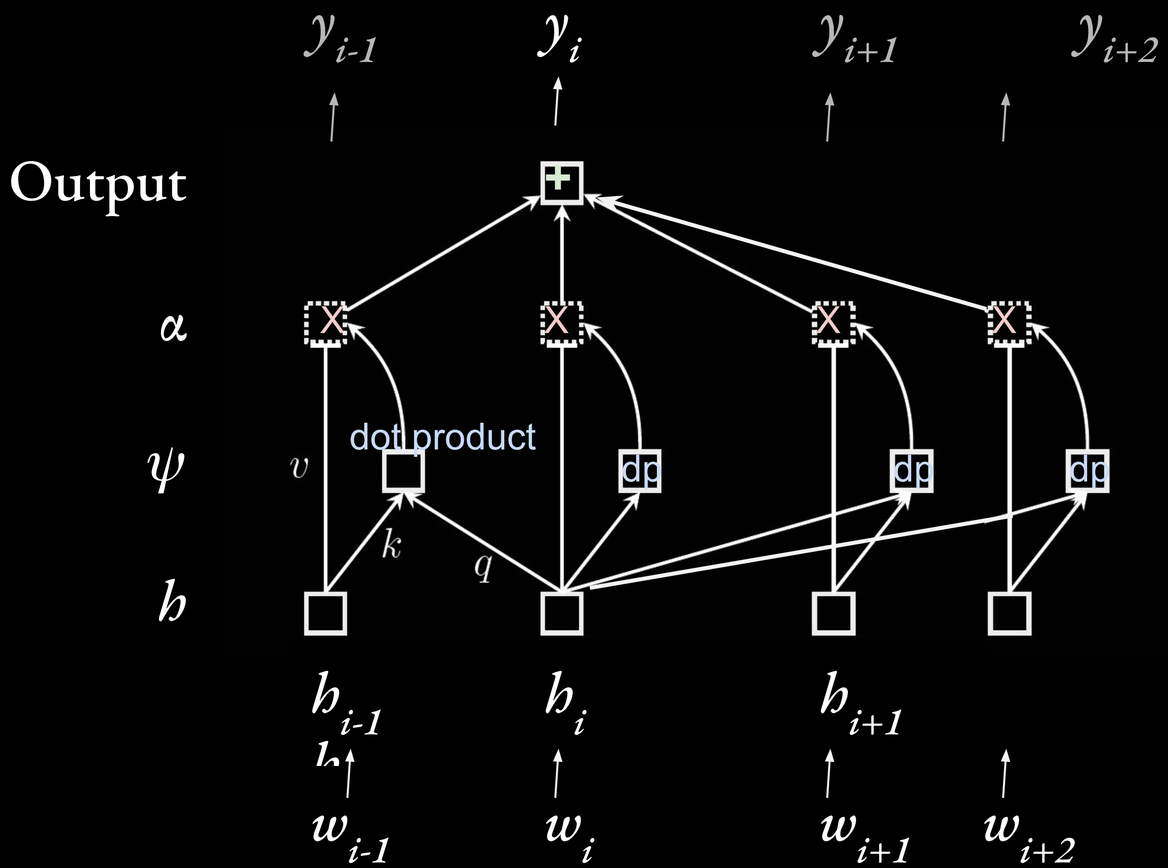
The Transformer's Heart: Self-Attention



The Transformer's Heart: Self-Attention



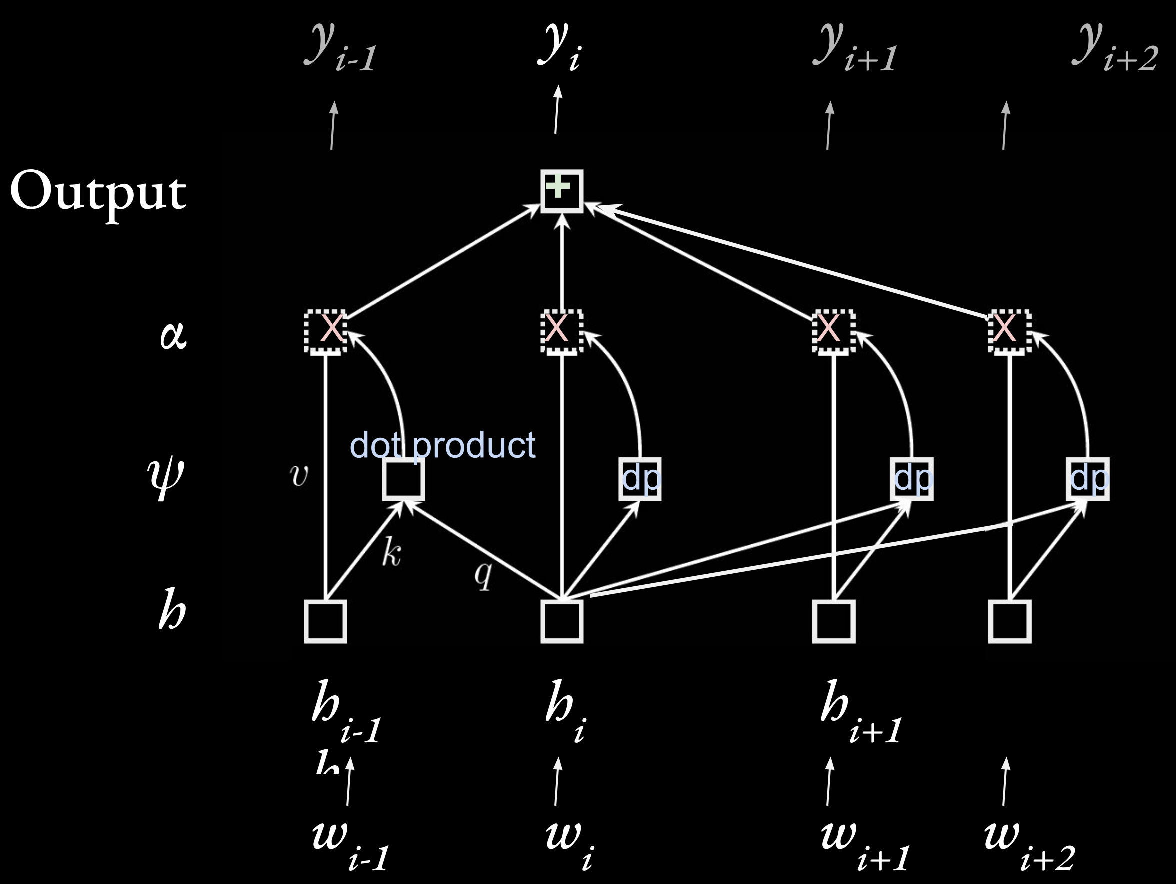
The Transformer's Heart: Self-Attention



$$\psi_{dp}(h_i, s) = s^T h_i$$

$k^t q$

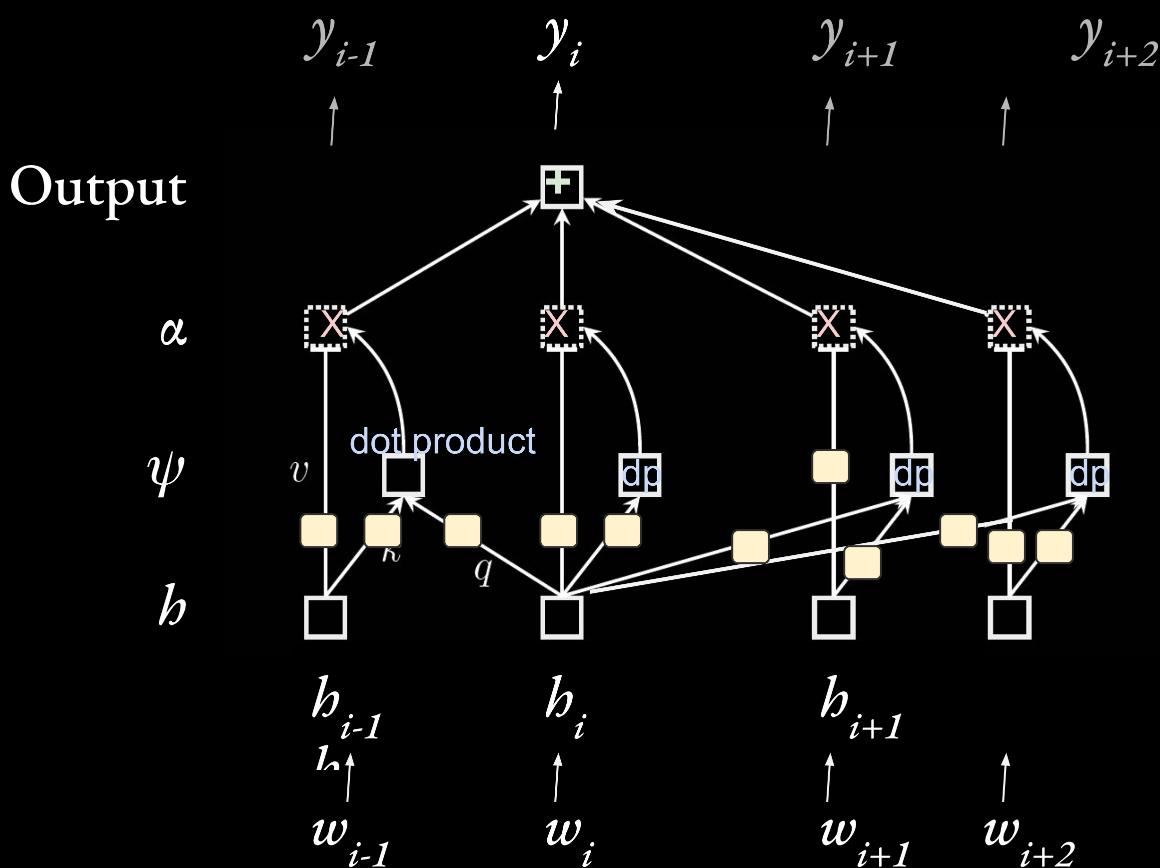
The Transformer's Heart: Self-Attention



scaling parameter

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

The Transformer's Heart: Self-Attention



$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

Linear layer:
$W^T X$

One set of weights for each of for K, Q, and V

Self-Attention in PyTorch

```
import nn.functional as f
class SelfAttention(nn.Module):
    def __init__(self, h_dim:int):
        self.Q = nn.Linear(h_dim, h_dim) #1 head
        self.K = nn.Linear(h_dim, h_dim)
        self.V = nn.Linear(h_dim, h_dim)


    def forward(hidden_states:torch.Tensor):
        v = self.V(hidden_states)
        k = self.K(hidden_states)
        q = self.Q(hidden_states)
        attn_scores = torch.matmul(q, k.T)
        attn_probs = f.Softmax(attn_scores)

        context = torch.matmul(attn_probs, v)
        return context
```

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

Linear layer:
$W^T X$

One set of weights for each of for K, Q, and V

Self-Attention in PyTorch

```
import nn.functional as f
class SelfAttention(nn.Module):
    def __init__(self, h_dim:int):
        self.Q = nn.Linear(h_dim, h_dim) #1 head
        self.K = nn.Linear(h_dim, h_dim)
        self.V = nn.Linear(h_dim, h_dim)
        self.dropout = nn.dropout(p=0.1)

    def forward(hidden_states:torch.Tensor):
        v = self.V(hidden_states)
        k = self.K(hidden_states)
        q = self.Q(hidden_states)
        attn_scores = torch.matmul(q, k.T)
        attn_probs = f.Softmax(attn_scores)
        attn_probs = self.dropout(attn_probs)
        context = torch.matmul(attn_probs, v)
        return context
```

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

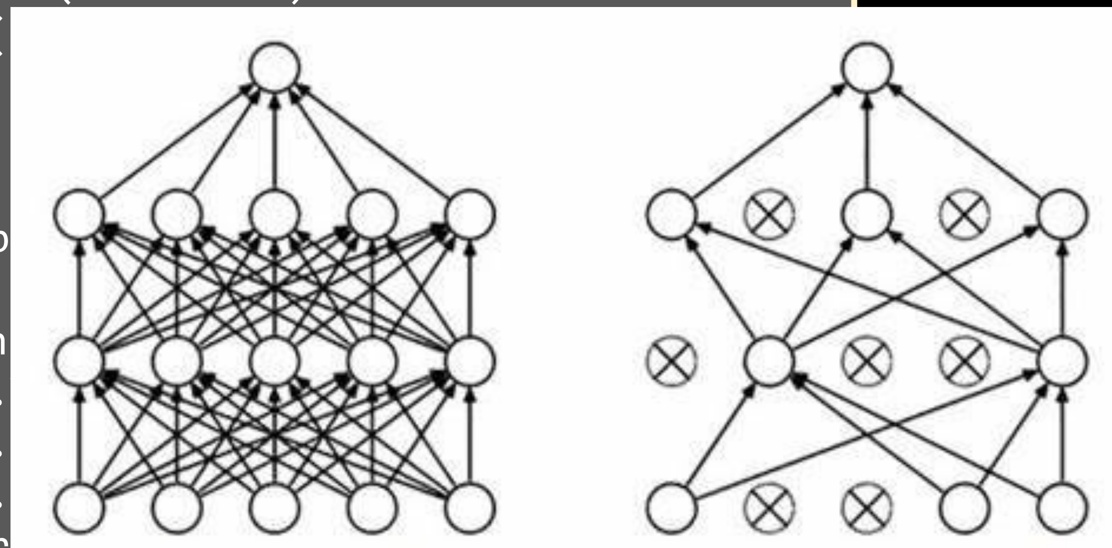
Linear layer:
$W^T X$

One set of weights for each of for K, Q, and V

Self-Attention in PyTorch

```
import nn.functional as f
class SelfAttention(nn.Module):
    def __init__(
        self.Q =
        self.K =
        self.V =
        self.drop

    def forward(h
        v = self.
        k = self.
        q = self.
        attn_scor
        attn_prob
        attn_probs = self.dropout(attn_probs)
        context = torch.matmul(attn_probs, v)
        return context
```



(a) Standard Neural Net (b) After applying dropout.

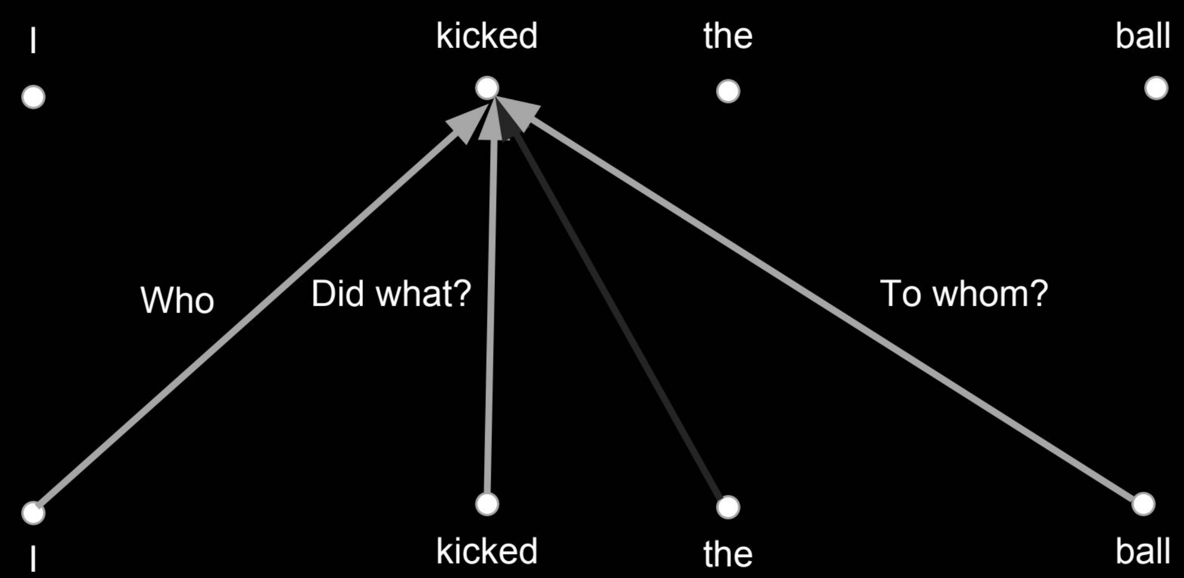
$= (k^t q)\, \sigma$

ayer:

of weights
of for K,
V

The Transformer: Beyond Self-Attention

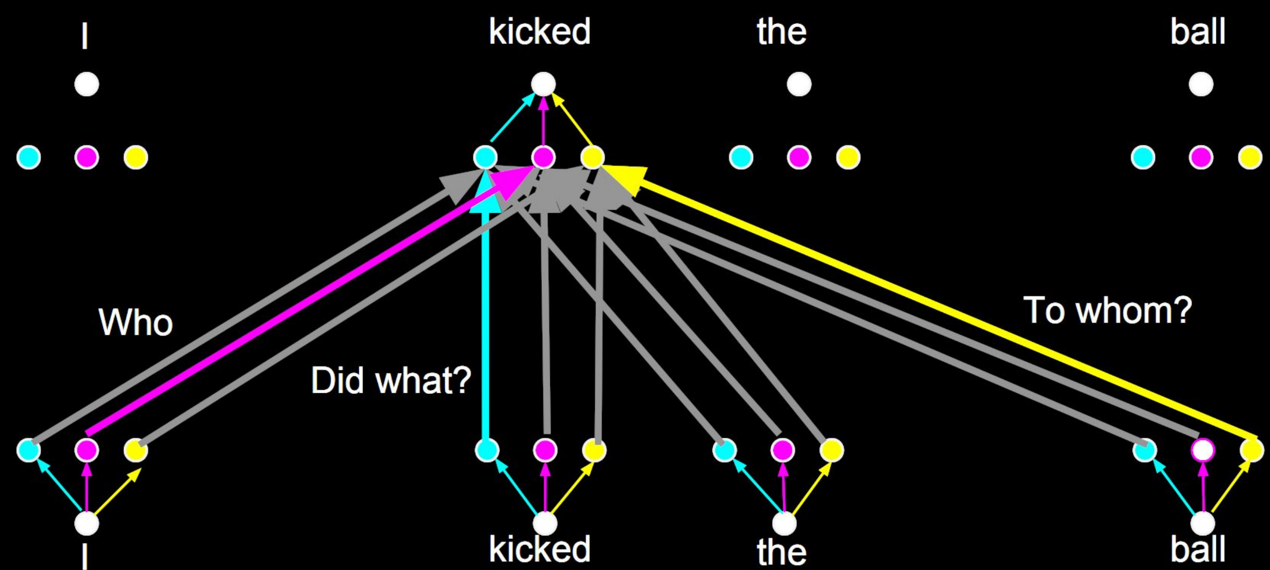
Limitation (thus far): Can't capture multiple types of dependencies between words.



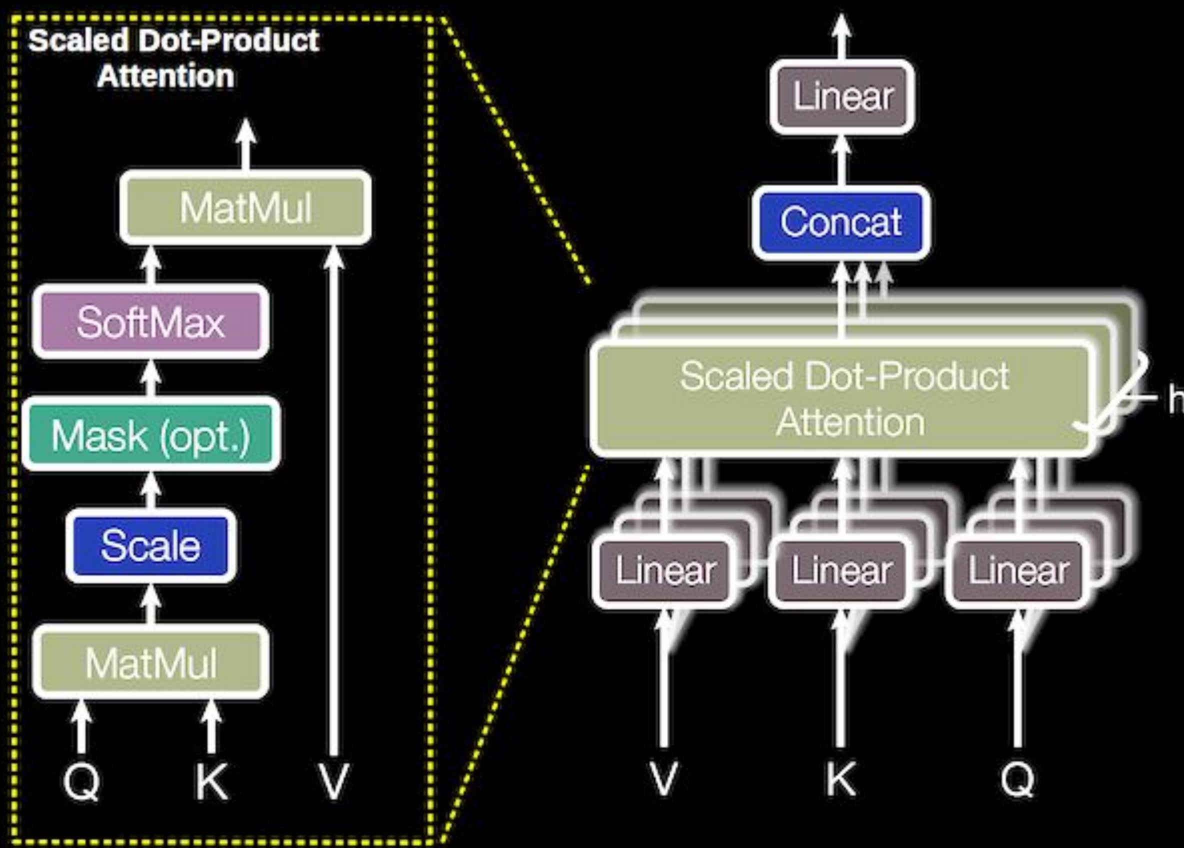
The Transformer: Beyond Self-Attention

Limitation (thus far): Can't capture multiple types of dependencies between words.

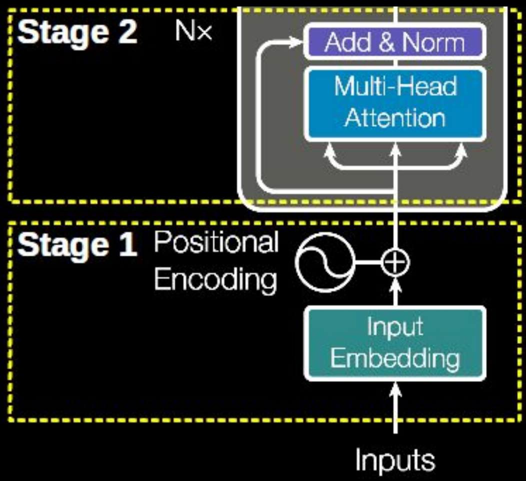
Solution: Multi-head attention



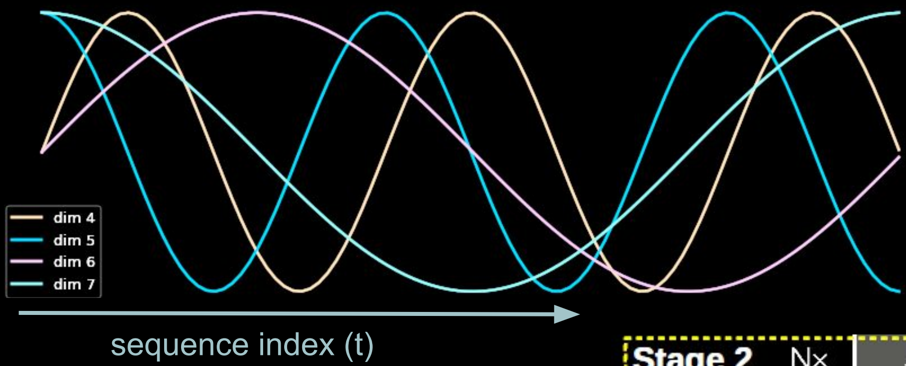
The Transformer: Muli-headed Attention

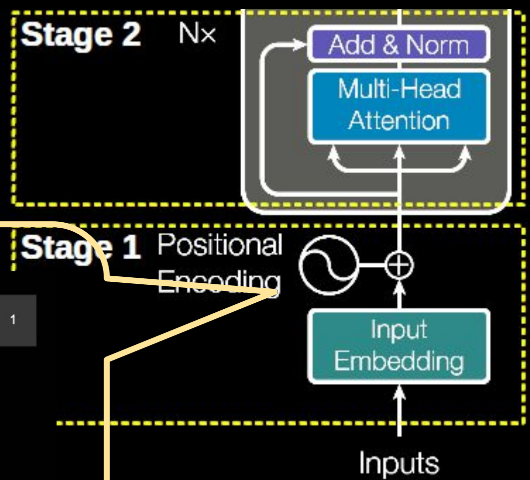


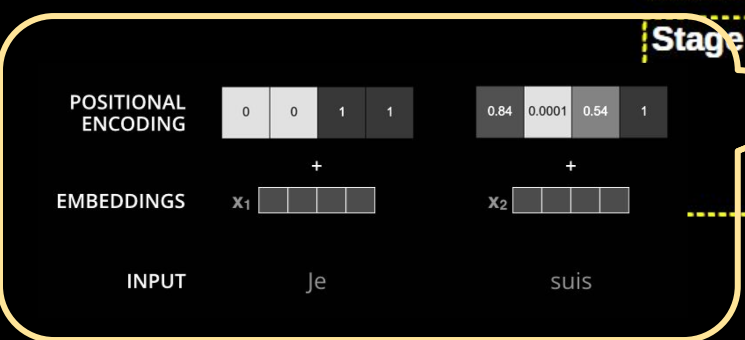
The Transformer



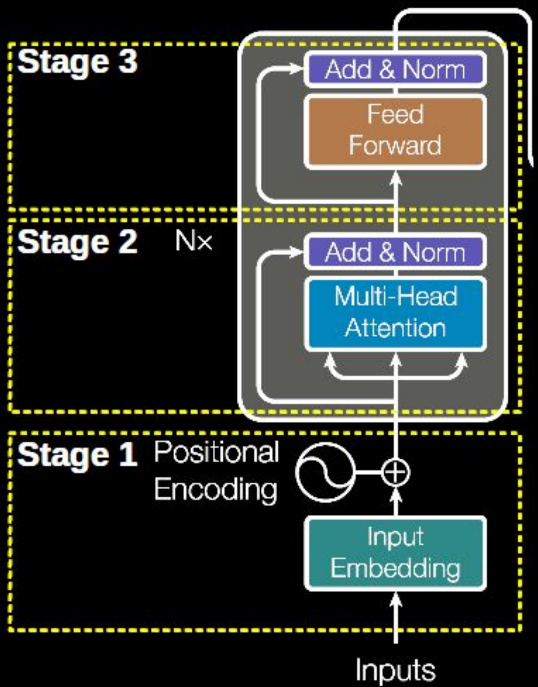
The Transformer





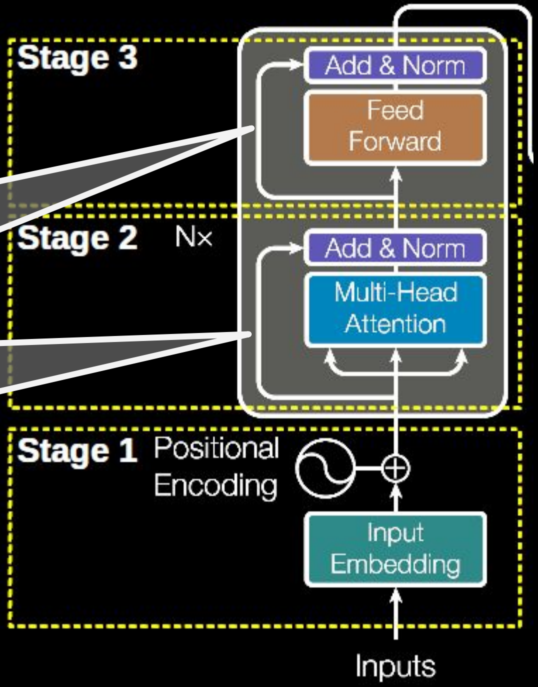


The Transformer

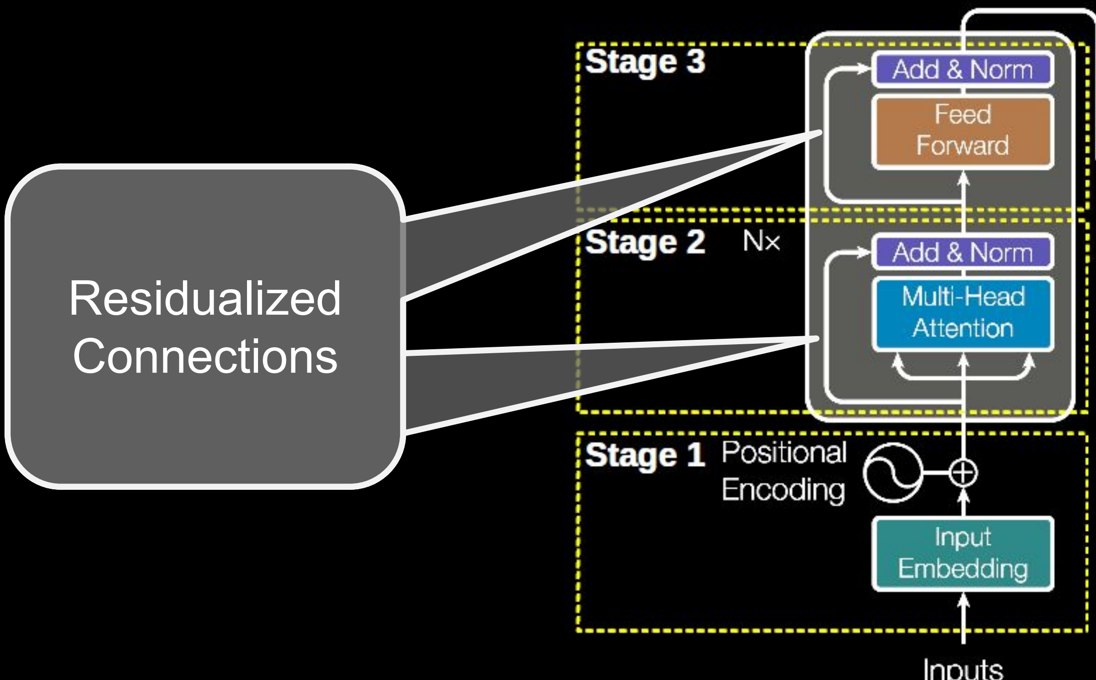


The Transformer

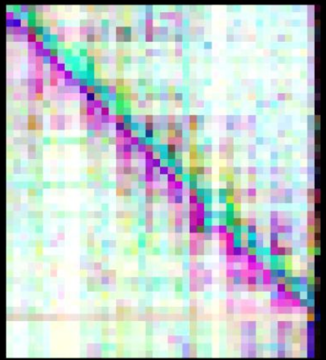
Residualized Connections



The Transformer



residuals enable positional information to be passed along



With residuals

Without residuals

The Transformer: Motivation

Challenges to sequential representation learning

- Capture long-distance dependencies
- Preserving sequential distances / periodicity
- Capture multiple relationships
- Easy to parallelize -- don’t need sequential processing.

The Transformer: Motivation

Challenges to sequential representation learning

- Capture long-distance dependencies
  Self-attention *treats far away words similar to those close.*
- Preserving sequential distances / periodicity
  Positional embeddings *encode distances/periods.*
- Capture multiple relationships
  Multi-headed attention *enables multiple compositions.*
- Easy to parallelize -- don't need sequential processing.
  Entire layer can be computed at once. Is only matrix multiplications + standardizing.

Transformer (as of 2017)

“WMT-2014” Data Set. BLEU scores:

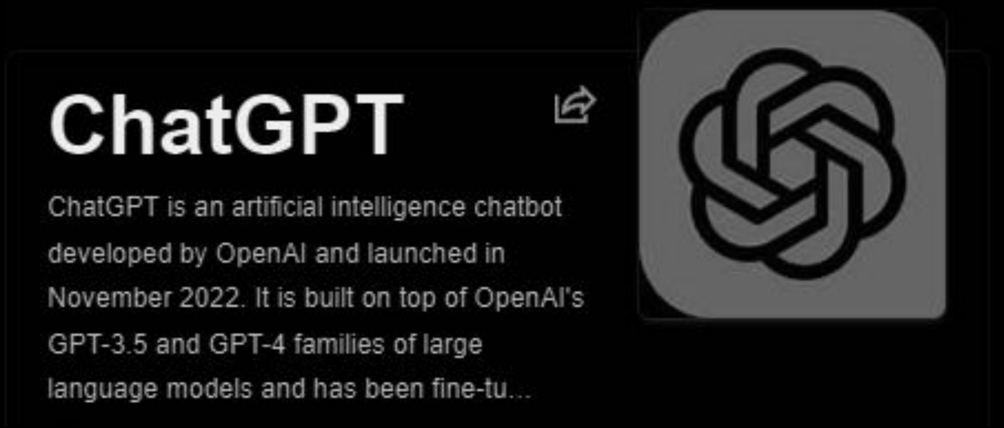
	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

Transformers as of 2023

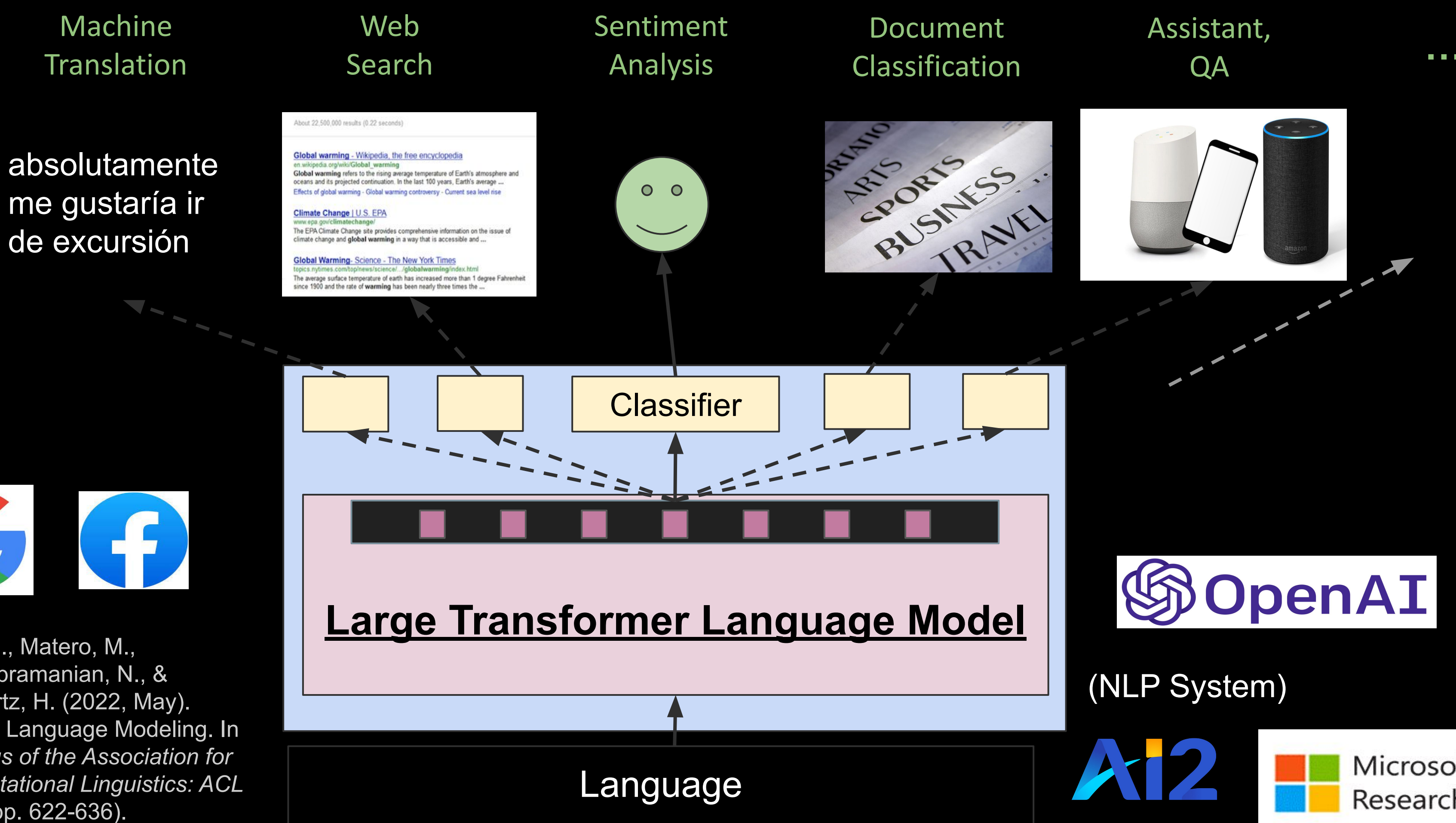
General Language Understanding Evaluations:

https://gluebenchmark.com/leaderboard

https://super.gluebenchmark.com/leaderboard/



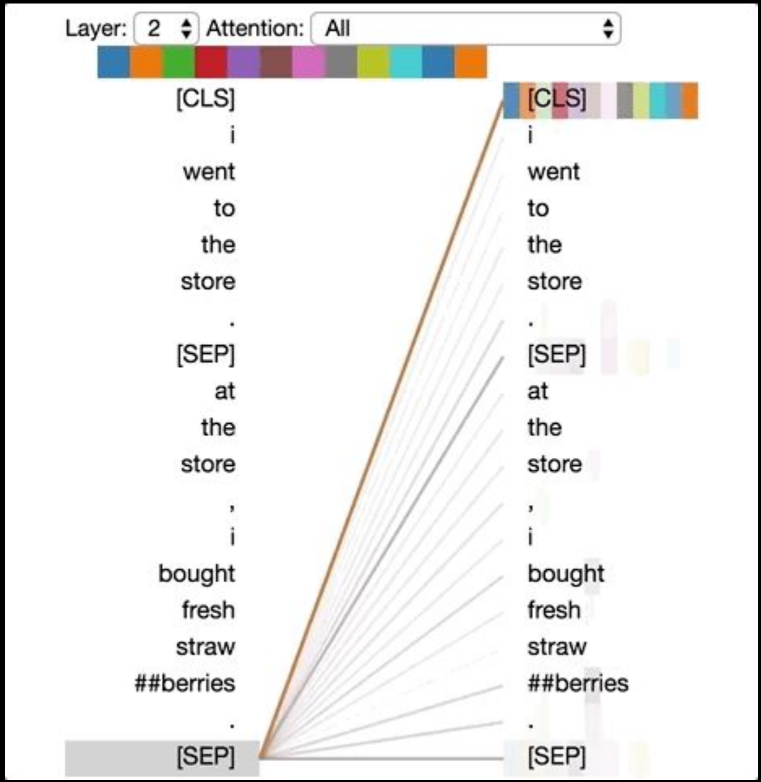
Transformers as of 2023



Soni, N., Matero, M., Balasubramanian, N., & Schwartz, H. (2022, May). Human Language Modeling. In *Findings of the Association for Computational Linguistics: ACL 2022* (pp. 622-636).

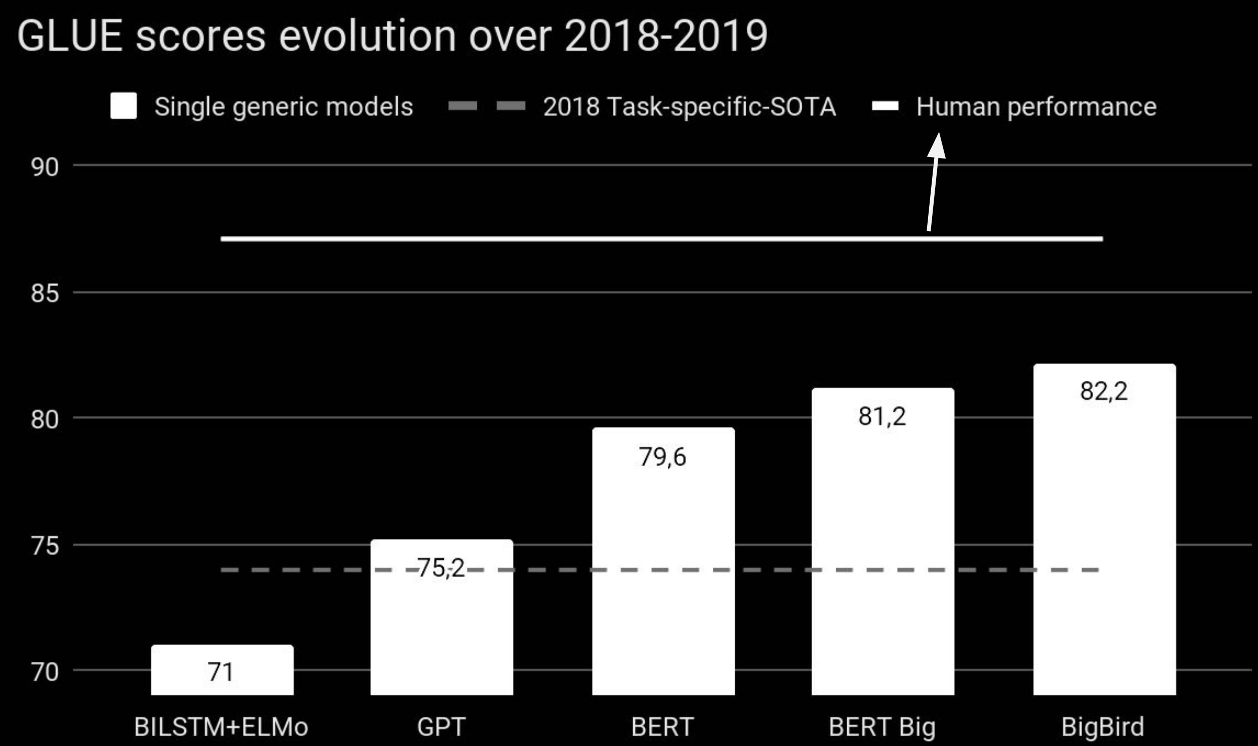
Bert: Attention by Layers

https://colab.research.google.com/drive/1vlOJ1lhdujVjfH857hvYKIdKPTD9Kid8



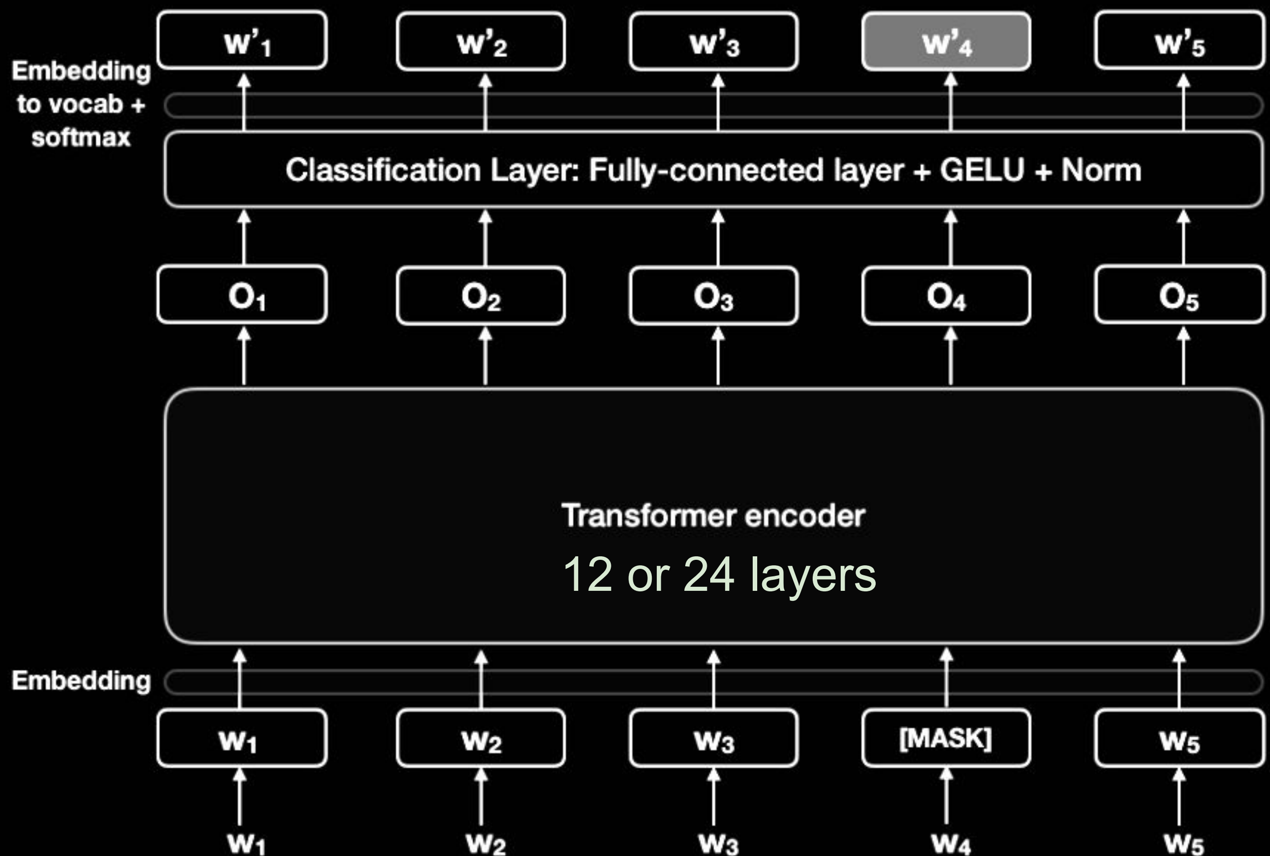
(Vig, 2019)

BERT Performance: e.g. Question Answering

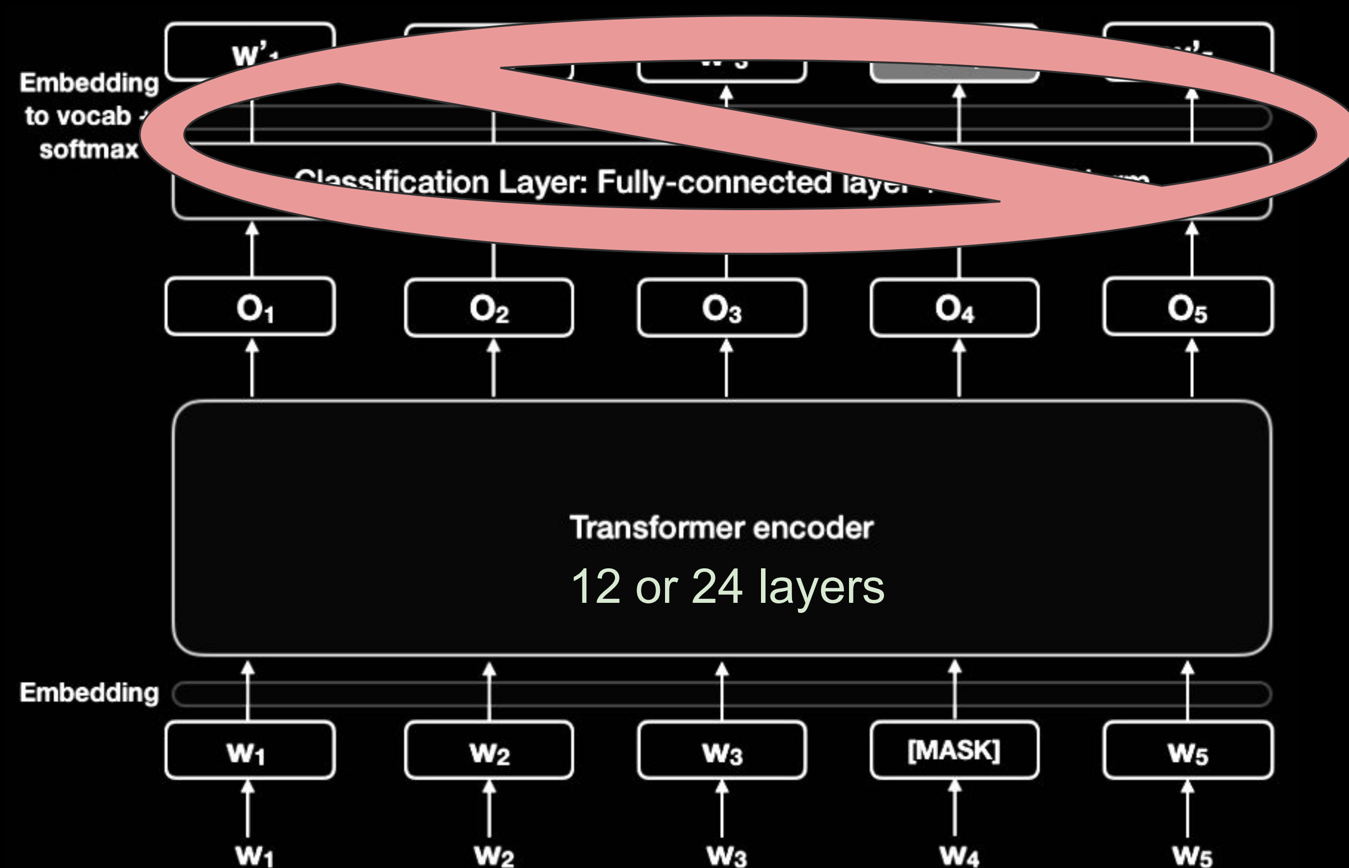


https://rajpurkar.github.io/SQuAD-explorer/

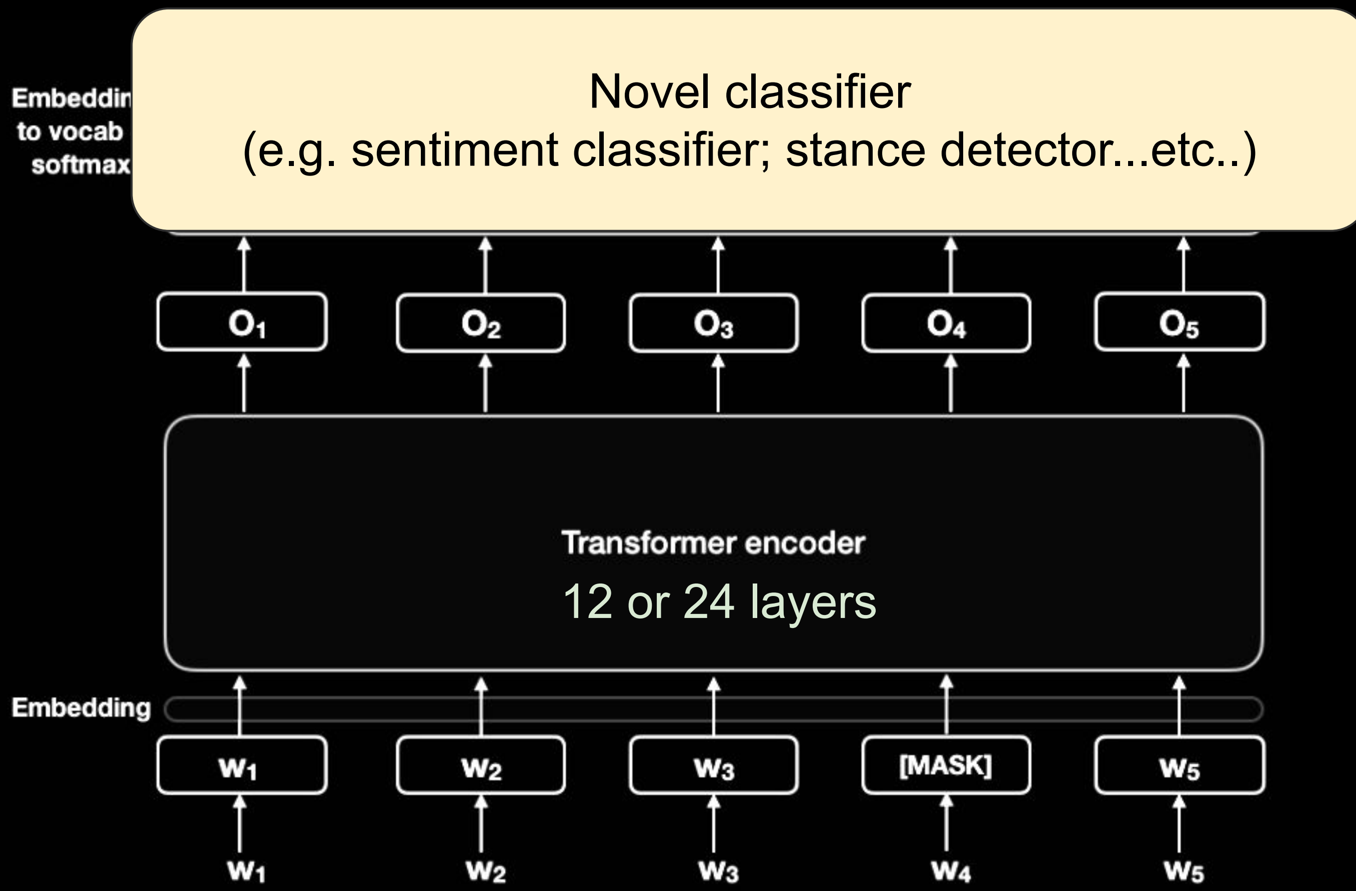
BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



The Transformer: Motivation

Challenges to sequential representation learning

- Capture long-distance dependencies
  Self-attention *treats far away words similar to those close.*
- Preserving sequential distances / periodicity
  Positional embeddings *encode distances/periods.*
- Capture multiple relationships
  Multi-headed attention *enables multiple compositions.*
- Easy to parallelize -- don’t need sequential processing.
  Entire layer can be computed at once. Is only matrix multiplications + standardizing.

Extra Material

When L2, L1 work well?

Depends on data, but generally we find...

k: number of features; N: number of observations

	$k \ll N$,	$k < N$	$k == N$	$k > N$	$k \gg N$
None	Often	Sometimes	Almost Never	Almost Never	Almost Never
L2	Sometimes	Often	Often	Sometimes	Almost Never
L1	Almost Never	Sometimes	Often	Sometimes	Almost Never