

CS167: Machine Learning

RNN (continued)
Transformers

Tuesday, April 30th, 2024



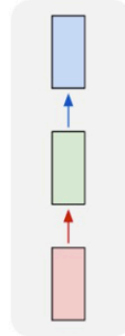
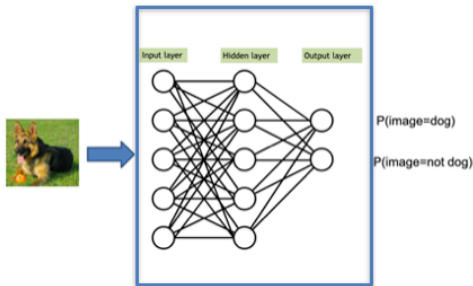
Announcements

- **Project#2**
 - Released and due on **05/12 (Sunday) by 11:59pm**
- **Quiz#3**
 - Will be released early next week
- **Group activity on RNN**
 - Please promptly submit your work if you haven't completed it yet

Recap: mappings of different tasks

- we input one training/testing example and make one prediction.

one to one



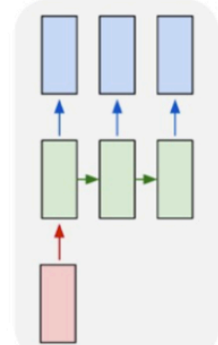
- we input one training/test example, and output many predictions

one to many



"A Dog catching a ball in mid air"

Image caption generation



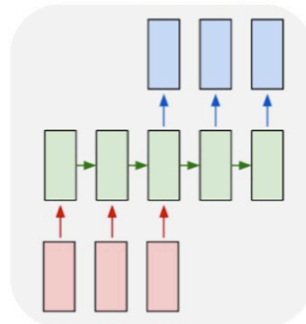
- we input multiple things, and make multiple predictions from it
- the input and output size do not need to be the same length

many to many



Machine Translation (translating from one language to another)

"Hello my name is" -> "Hola me llamo"

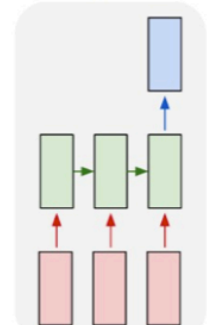


- we input multiple things, and make one prediction from it

many to one

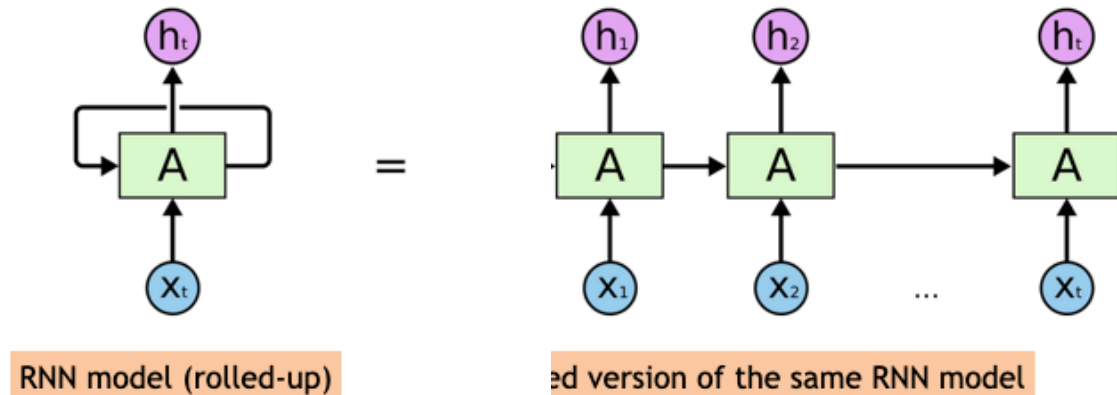
Review (X)	Rating (Y)
"This movie is fantastic! I really like it because it is so good!"	★★★★☆
"Not to my taste, will skip and watch another movie"	★★☆☆☆
"This movie really sucks! Can I get my money back please?"	★☆☆☆☆

Product review prediction



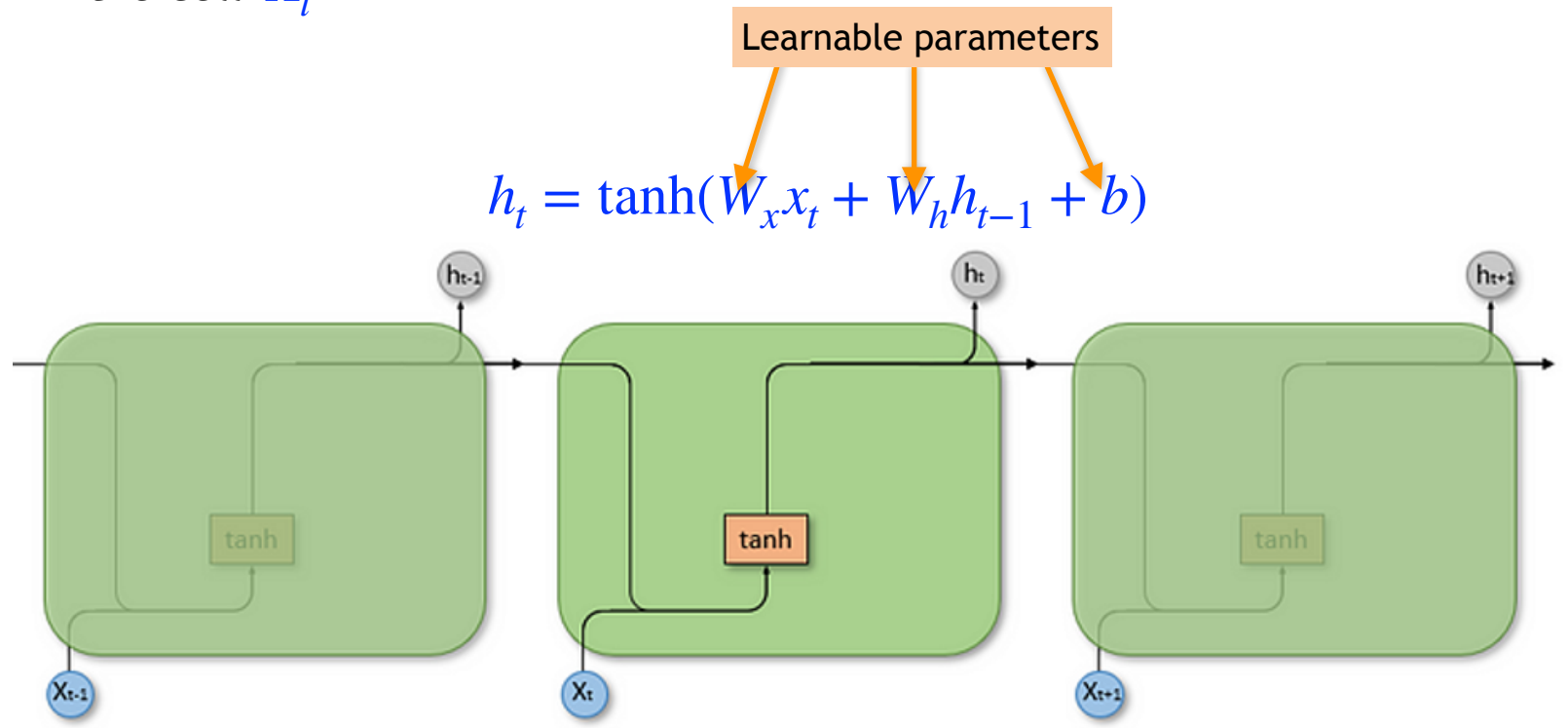
Recap: Recurrent Neural Network (RNN)

- Recurrent neural networks (RNN) include “feedback loops”
 - Vanilla RNN
 - Long Short-Term Memory (LSTM)
 - Gated Recurrent Unit (GRU)
- RNNs are useful for learning and predicting sequences, e.g.
 - translations from one language to another
 - Sentiment prediction of a given text (eg, predicting positive or negative reviews)



Recap: Simple Recurrent Neural Network (RNN)

- Each cell has 2 inputs:
 - the past and the present
 - the output from the previous cell h_{t-1} and the current input into the cell X_t



- This figure represents an unfolded RNN to help understand what is going on. The length of the cell is equal to the number of time steps of the input sequence.

Recap: Long Short-Term Memory (LSTM)

- Each LSTM cell has **3 gates**:
 - **forget gate**: decides how much memory shall be kept
 - **input gate**: decides which information shall be added to the long-term memory
 - **output gate**: decides which parts of the cell state build the output, i.e. it's responsible for the short term memory

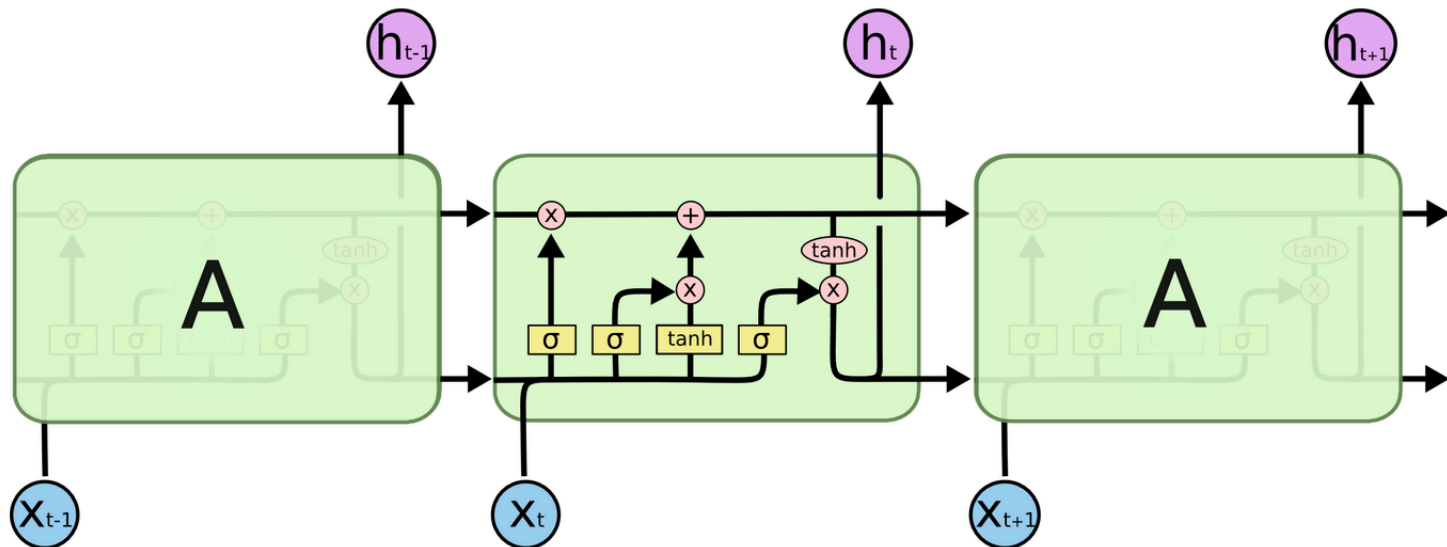
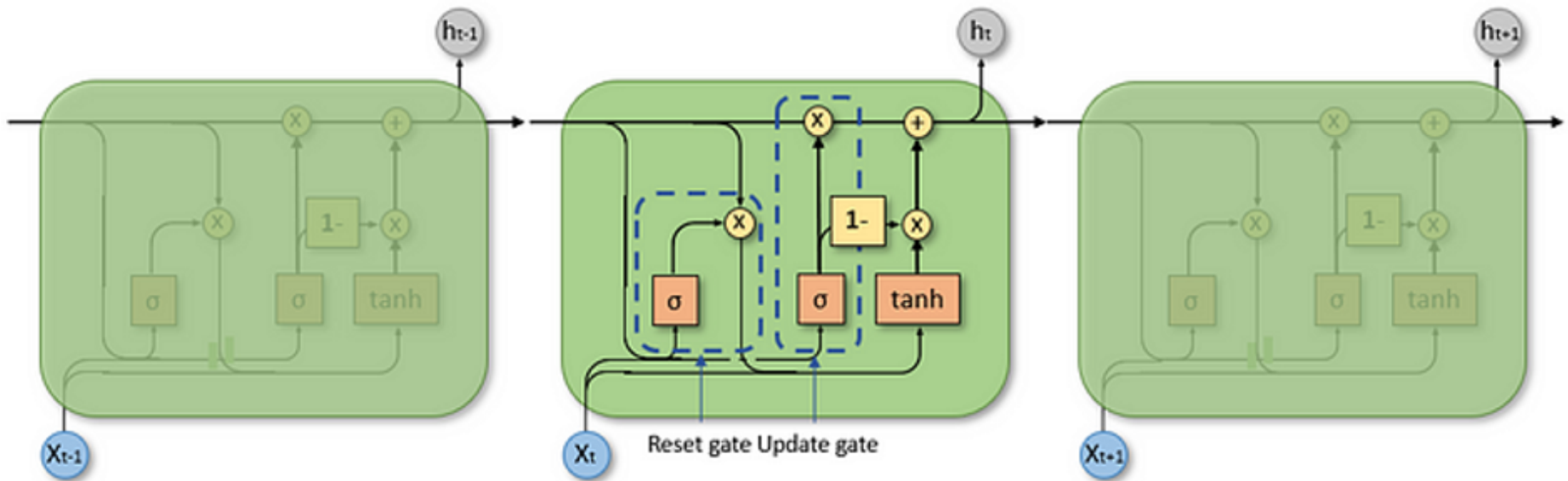


Figure credit: colah.github.io

Recap: Gated Recurrent Unit (GRU)

- Similar to LSTMs, the GRU solves the vanishing gradient problem, but they do so using fewer gates – making them effective, and fast.
- GRUs have 2 gates:
 - a **reset gate** which is responsible for the short-term memory as it decides how much past information is kept and disregarded
 - an **update gate** which is responsible for the long-term memory and is comparable to the LSTMs forget gate



Today's agenda

- In-depth exploration of the caveats of vanilla RNN and LSTM
 - Vanishing gradient in vanilla RNN
 - Transfer learning is not possible with LSTM (or RNN)

Caveats: Vanishing Gradients in Vanilla RNN

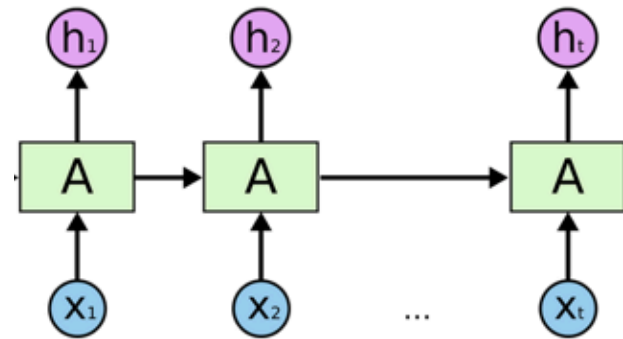
- Each cell in Vanilla RNN has **2 inputs**:
 - At any time step t , it computes the output from
 - the previous cell h_{t-1} and the current input into the cell x_t

$$h_1 = \tanh(W_x x_1 + W_h h_0 + b)$$

$$h_2 = \tanh(W_x x_2 + W_h h_1 + b)$$

...

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



- Let's compute the outputs from time step 0 until time step t for input sequence

$x_1, x_2, x_3, x_4 \dots, x_{t-1}, x_t$

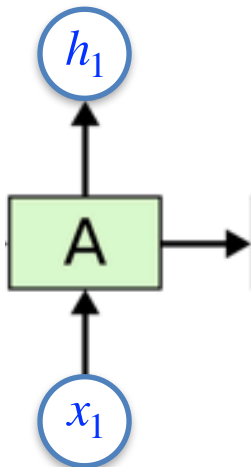
↓ ↓ ↓ ↓
Reza lives in Virginia

Caveats: Vanishing Gradients in Vanilla RNN

- Each cell in Vanilla RNN has **2 inputs**:
 - At any *time step* $t=1$, it computes the output from
 - the previous cell h_0 and the current input into the cell X_1

$$h_1 = \tanh(W_x x_1 + W_h h_0)$$

Leaving out the bias term for the simplification and clarity



Reza

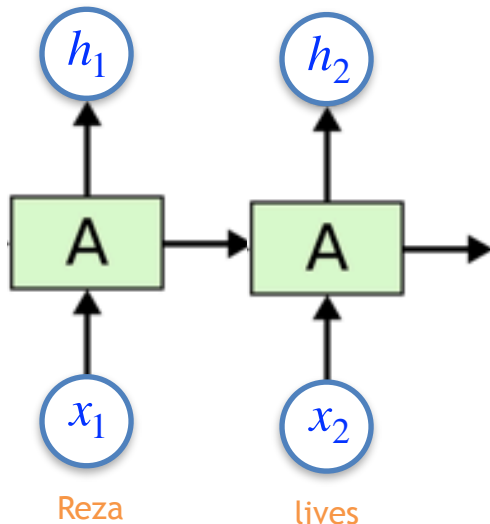
Caveats: Vanishing Gradients in Vanilla RNN

- Each cell in Vanilla RNN has **2 inputs**:
 - At any time step $t=2$, it computes the output from
 - the previous cell h_1 and the current input into the cell x_2

$$h_2 = \tanh(W_x x_2 + W_h h_1)$$

$$= \tanh(W_x x_2 + W_h (\tanh(W_x x_1 + W_h h_0)))$$

substituted the value of h_1
from previous slide



Caveats: Vanishing Gradients in Vanilla RNN

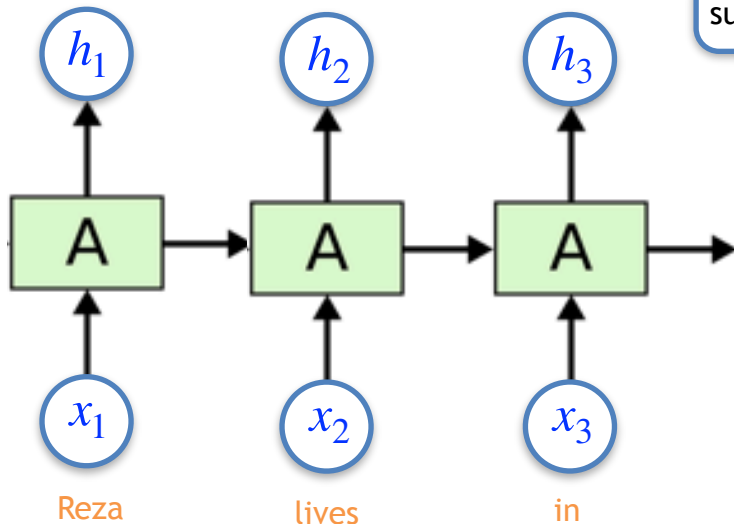
- Each cell in Vanilla RNN has **2 inputs**:
 - At any time step $t=2$, it computes the output from
 - the previous cell h_1 and the current input into the cell x_3

$$h_3 = \tanh(W_x x_3 + W_h h_2)$$

Longest repeated matrix-matrix multiplications term

$$= \tanh(W_x x_3 + W_h (\tanh(W_x x_2 + W_h (\tanh(W_x x_1 + W_h h_0))))))$$

substituted the value of h_2 from previous slide

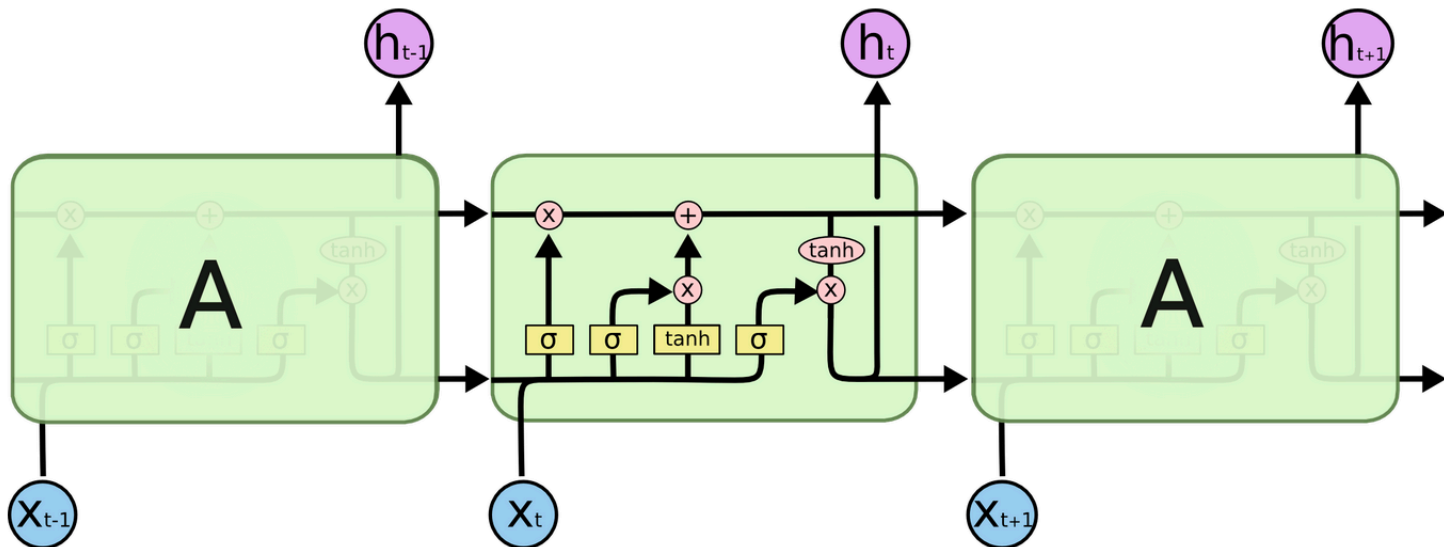


- For a longer sequence of inputs: $x_1, x_1, x_1, \dots, x_t$
- It needs to multiply a lot of matrices together
- Performing repeated matrix-matrix multiplications with small values inside will result in extremely tiny numbers

Eg, $0.10 * 0.10 * 0.10 * 0.10 = 0.0001$

No Vanishing Gradients in LSTM

- **LSTM** is still a sophisticated RNN that uses a clever mechanism with 3 gates
- It prevents the generation of small values resulting from the repetitive multiplication of matrices along with certain additive terms.
 - Hence it avoids the vanishing gradient



Advantages and disadvantages of various RNNs

Vanilla RNN Advantages

- they can also handle inputs of varying lengths.

Vanilla RNN disadvantages

- short term memory
- suffers from the **vanishing gradient** problem:
 - forgets what is seen in longer sequences
 - this problem gets worse with the more layers the network has

LSTM Advantages

- solves vanishing gradient problem
- can capture both the short and long term patterns of a sequence.

LSTM disadvantages

- because LSTMs add complexity, they also are computationally more expensive, leading to longer training times.

GRU Advantages

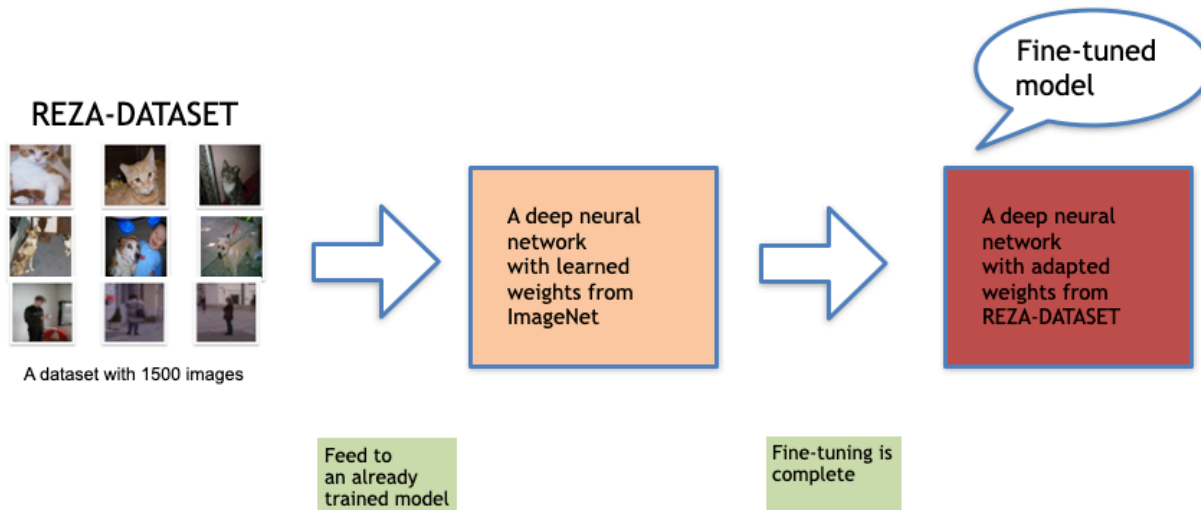
- solves vanishing gradient problem
- less computationally expensive than LSTMs, which makes them faster to train

GRU disadvantages

- do not have a separate hidden and cell state, so they might not be able to consider observations as far into the past as the LSTM

Caveats in LSTM

- LSTMs with added complexity, they also are computationally more expensive, leading to **longer training times**
- **Transfer learning** (on a new dataset with limited samples) **never worked** with LSTM
 - we did transfer learning in CNN when we fine-tuned a pretrained AlexNet on a new datasets such as BCDP
 - we quickly achieved excellent accuracy of over 90% within a few epochs of training
 - you will also fine-tune two other CNNs for your Project 2: i) a pretrained VGG and ii) a pretrained ResNet



Transformers



Today's agenda

- In-depth exploration of the caveats of vanilla RNN and LSTM
 - Vanishing gradient in vanilla RNN
 - Transfer learning is not possible with LSTM (or RNN)
- Transformers
 - Transfer learning is possible
 - New type of network architecture

Transformers



Attention Is All You Need

- In 2017, a new mechanism is introduced for context learning called **attention mechanism**
 - more precisely, **self-attention**
- It takes less time to train **advantage**
- Transfer learning on a new task is **possible** **advantage**
- In subsequent years, it revolutionized the field of AI

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

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

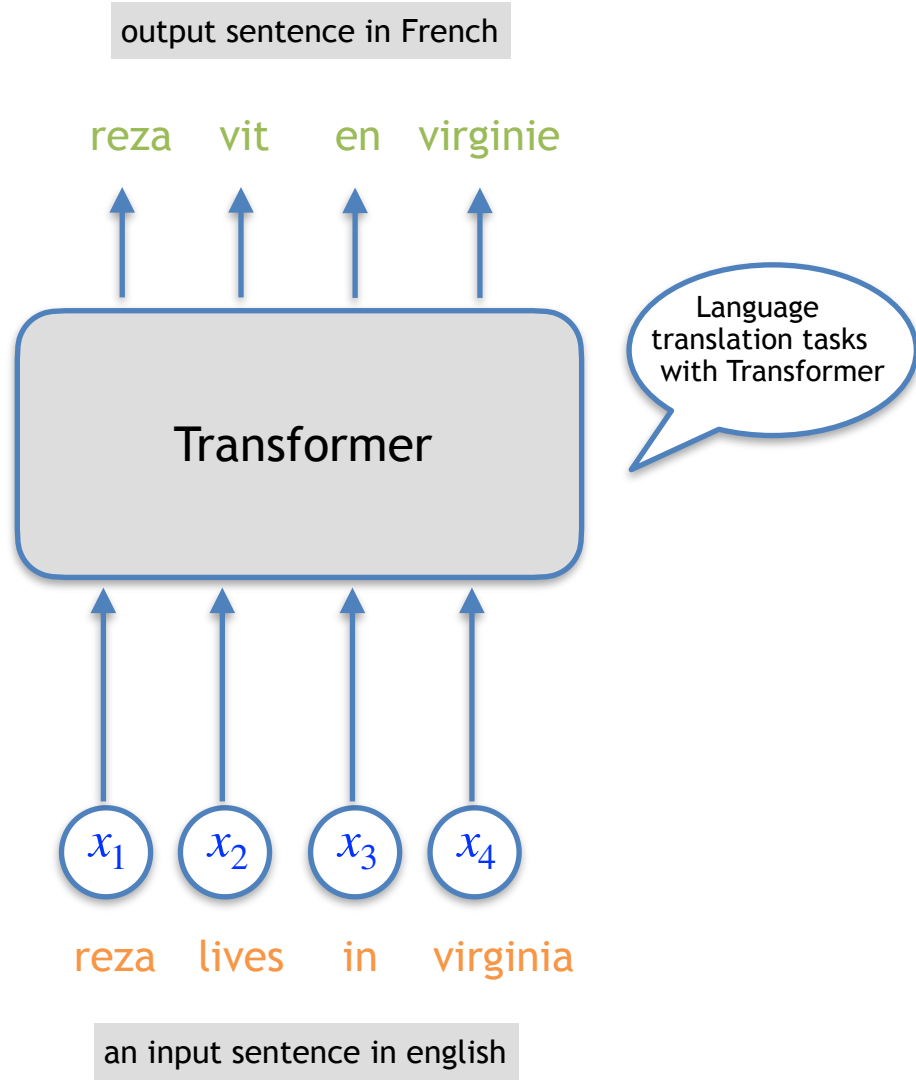
Illia Polosukhin* †
illia.polosukhin@gmail.com

Abstract

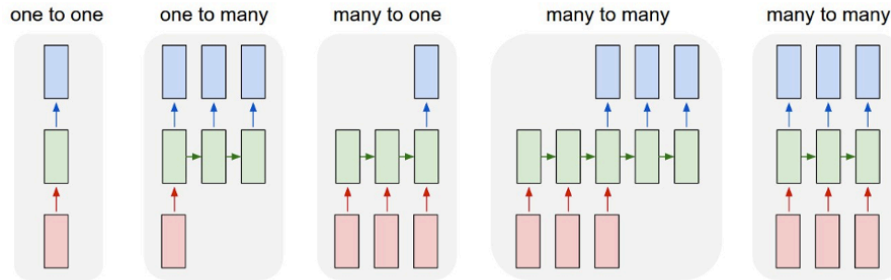
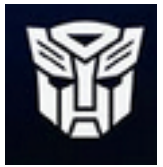
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, 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 ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

[Attention is all you need - NeurIPS'2017](#)

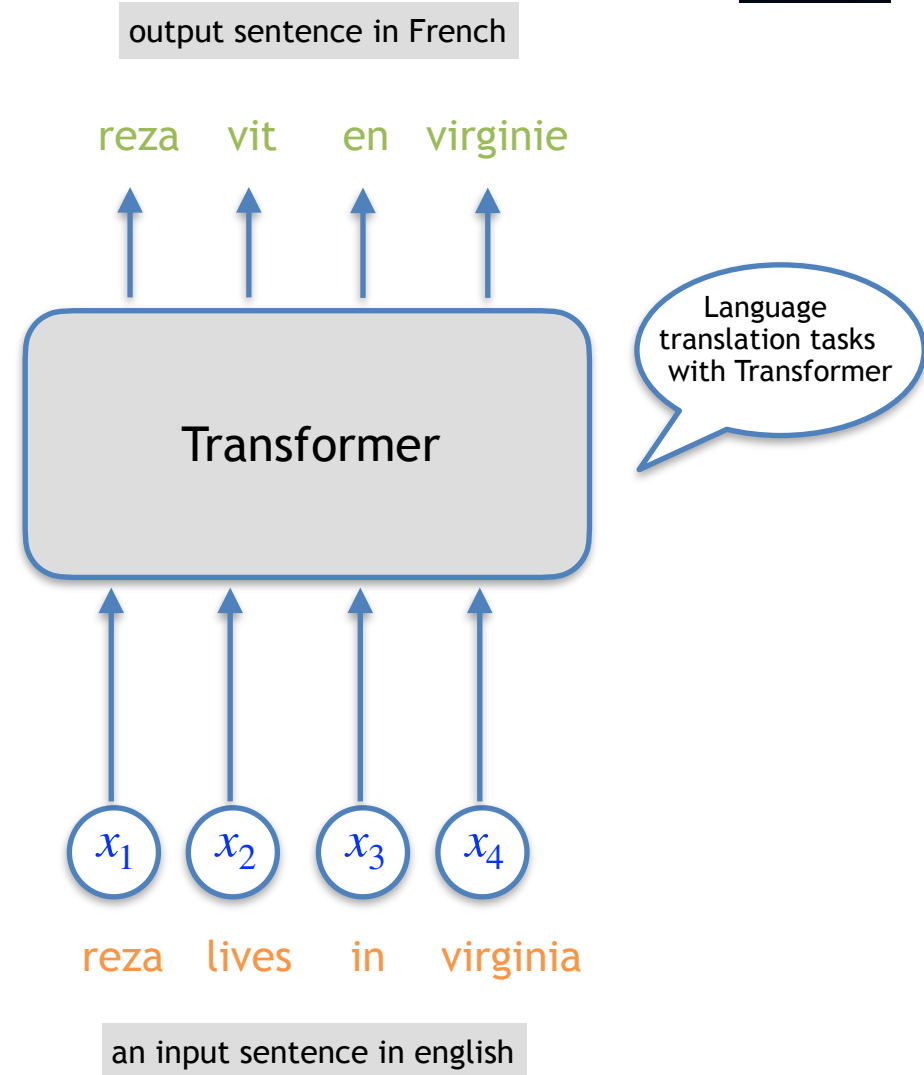
Transformers



Transformers

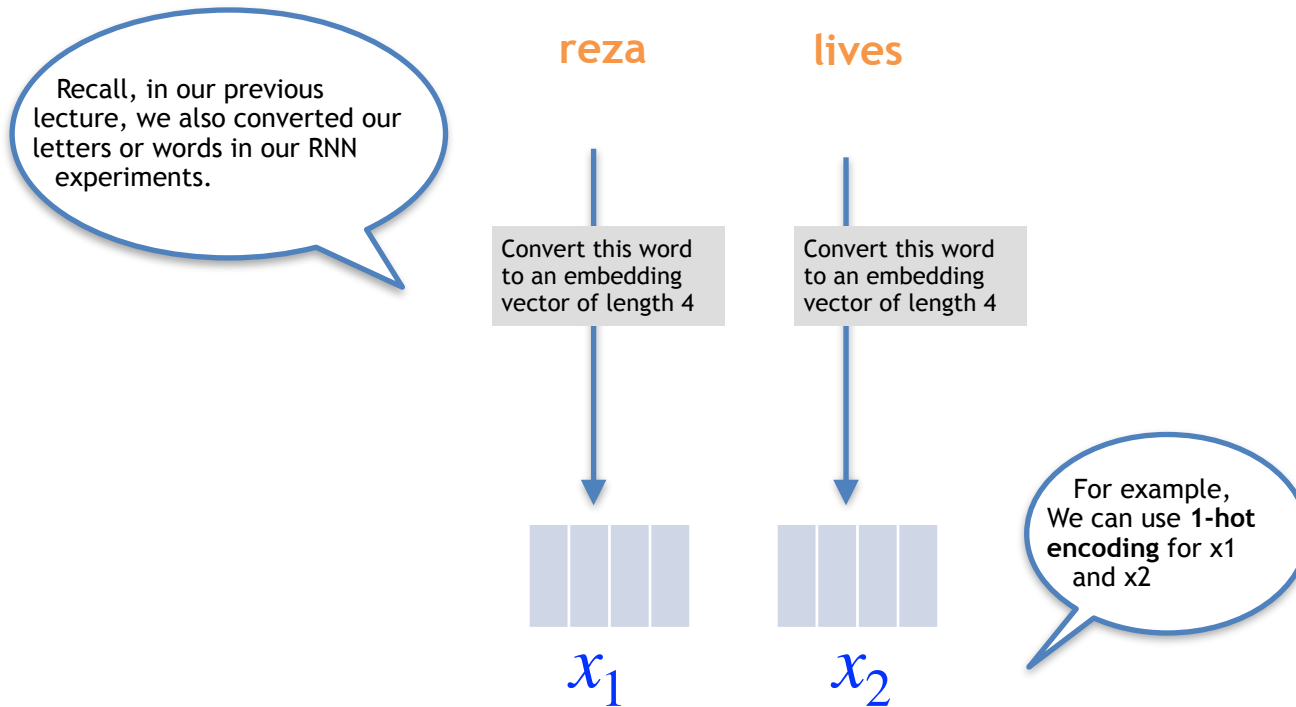


What type of task is this?



Attention

- Let's find out how to calculate the **attention mechanism** in a toy example
- Let's calculate attention with first two words of our sentence: “reza lives”



Attention

CS167-sp24-notes / Day23_Recurrent_Neural_Network_RNN.ipynb

Blame 1410 lines (1410 loc) · 81.6 KB  Code 55% faster with GitHub Copilot

In [6]:

```
# Step 1: create a mapping between the characters in our vocabulary to a set of numeric indices
def convert_vocab_to_index(vocab):
    vocab_to_index_dict = {}
    for index, char in enumerate(vocab):
        vocab_to_index_dict[char] = index
    return vocab_to_index_dict
```

```
def convert_index_to_vocab(vocab):
    index_to_vocab_dict = {}
    for index, char in enumerate(vocab):
        index_to_vocab_dict[index] = char
    return index_to_vocab_dict
```

```
vocab_to_index_dict = convert_vocab_to_index(text_vocab)
index_to_vocab_dict = convert_index_to_vocab(text_vocab)
```

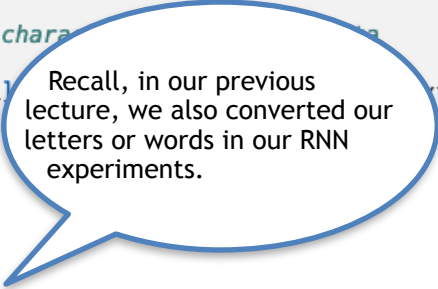
```
# Step 2: convert the text_data to numeric numbers using the above conversion method (this mapped data will be used)
text_data_numeric_values = np.zeros(text_data_size)
```

```
for i in range(text_data_size):
    cur_character = text_data[i].lower()
    text_data_numeric_values[i] = vocab_to_index_dict[cur_character]
```

```
# Step 3: visualize the first few characters
```

```
for i in range(6):
    print("character: ", text_data[i], " encoded as: ", text_data_numeric_values[i])
```

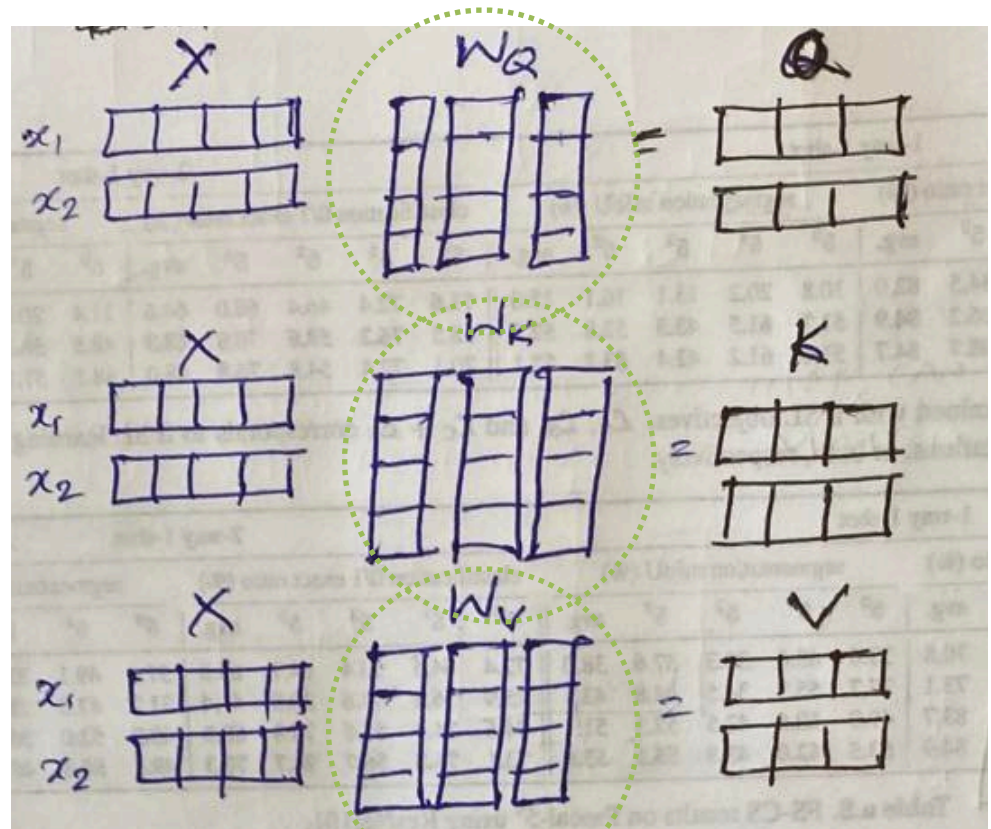
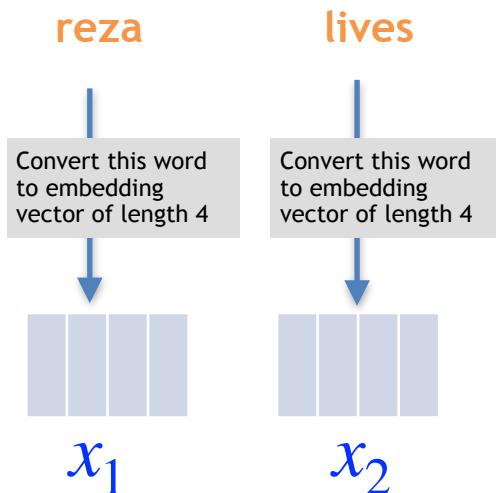
```
character: f encoded as: 18.0
character: i encoded as: 21.0
character: r encoded as: 30.0
character: s encoded as: 31.0
character: t encoded as: 32.0
```



Recall, in our previous lecture, we also converted our letters or words in our RNN experiments.

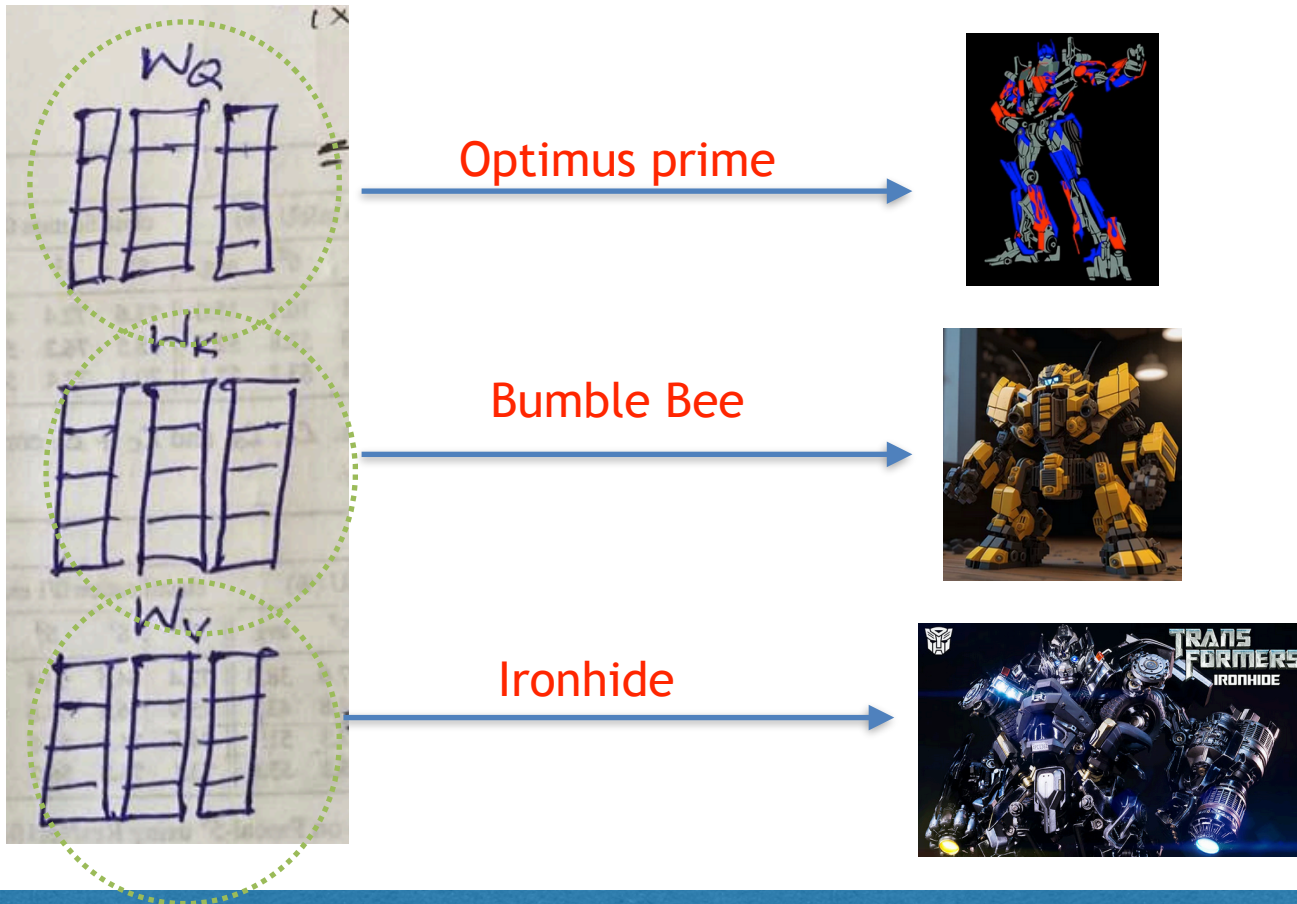
Attention

- It calculates three new matrices Q , K , and V with the help of three weight matrices W_Q , W_K , and W_V
- These three matrices (W_Q , W_K , and W_V) are learned during training



Attention

- It calculates three new matrices Q , K , and V with the help of three weight matrices W_Q , W_K , and W_V
- These three matrices (W_Q , W_K , and W_V) are learned during training



So why do we need this complicated attention?

Recall our “context” discussion

- Consider a language model trying to predict the next word based on the previous ones. Let's predict the last word to this sequence:
 - "The clouds are in the sky"

- We can guess from recent information that it will be the name of a language; we need the context of France to make a prediction.
 - "I grew up in France, I speak fluent french"

Attention

- Finally, attention is calculated using Q, K, and V matrices using the following equation:

$$\text{softmax} \left(\frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \end{matrix} \right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

=

$$\begin{matrix} \text{Z} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

[Reference: Illustrated Transformer](#)

Attention

My hand-notes

SELF-ATTENTION IS COMPUTED AS: $Z = A(X) = \text{softmax} \left(\frac{QK^T}{\sqrt{\text{dim}}} \right) V$

IT ENCODES

OF HIDDEN STATE CONTEXT FEATURE

$\text{softmax} \left(\frac{\begin{matrix} Q & K^T \\ 2 \times 3 & 3 \times 2 \end{matrix}}{\sqrt{3}} \right) = \begin{matrix} \text{WORD}_1\text{'S IMPORTANCE WITH RESPECT TO WORD}_1 & \text{WORD}_1\text{'S IMPORTANCE WITH RESPECT TO WORD}_2 \\ \text{WORD}_2\text{'S IMPORTANCE WITH RESPECT TO WORD}_1 & \text{WORD}_2\text{'S IMPORTANCE WITH RESPECT TO WORD}_2 \end{matrix}$

$Z = \text{ATTENTION} \left(\frac{QK^T}{\sqrt{\text{dim}}} \right) V$

$\begin{matrix} \begin{matrix} \text{WORD}_1\text{'S IMPORTANCE WITH RESPECT TO WORD}_1 & \text{WORD}_1\text{'S IMPORTANCE WITH RESPECT TO WORD}_2 \\ \text{WORD}_2\text{'S IMPORTANCE WITH RESPECT TO WORD}_1 & \text{WORD}_2\text{'S IMPORTANCE WITH RESPECT TO WORD}_2 \end{matrix} & \begin{matrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{matrix} & \begin{matrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{matrix} \end{matrix}$

My hand-notes

$\begin{matrix} \text{WORD}_1 & \text{WORD}_2 & v_{11} & v_{12} & v_{13} \\ \begin{matrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{matrix} & \begin{matrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{matrix} & \begin{matrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{matrix} \end{matrix}$

$\begin{matrix} H_{11}v_{11} + W_{12}v_{21} & H_{11}v_{12} + W_{12}v_{22} & H_{11}v_{13} + W_{12}v_{23} \\ H_{21}v_{11} + W_{22}v_{21} & H_{21}v_{12} + W_{22}v_{22} & H_{21}v_{13} + W_{22}v_{23} \end{matrix}$

$\begin{matrix} W_{11} \times \begin{matrix} v_{11} & v_{12} & v_{13} \end{matrix} & \begin{matrix} \oplus \\ \text{ELEMENT-WISE} \\ \oplus \end{matrix} & W_{12} \times \begin{matrix} v_{21} & v_{22} & v_{23} \end{matrix} \\ W_{21} \times \begin{matrix} v_{11} & v_{12} & v_{13} \end{matrix} & \begin{matrix} \oplus \\ \text{ELEMENT-WISE} \\ \oplus \end{matrix} & W_{22} \times \begin{matrix} v_{21} & v_{22} & v_{23} \end{matrix} \end{matrix}$

Reference: Illustrated Transformer

Going Back to the Transformer Idea



Attention Is All You Need

- This new mechanism for context learning, called the **attention mechanism** is only one part—of course, the central one.
- There are other components. Let's examine those.

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 ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

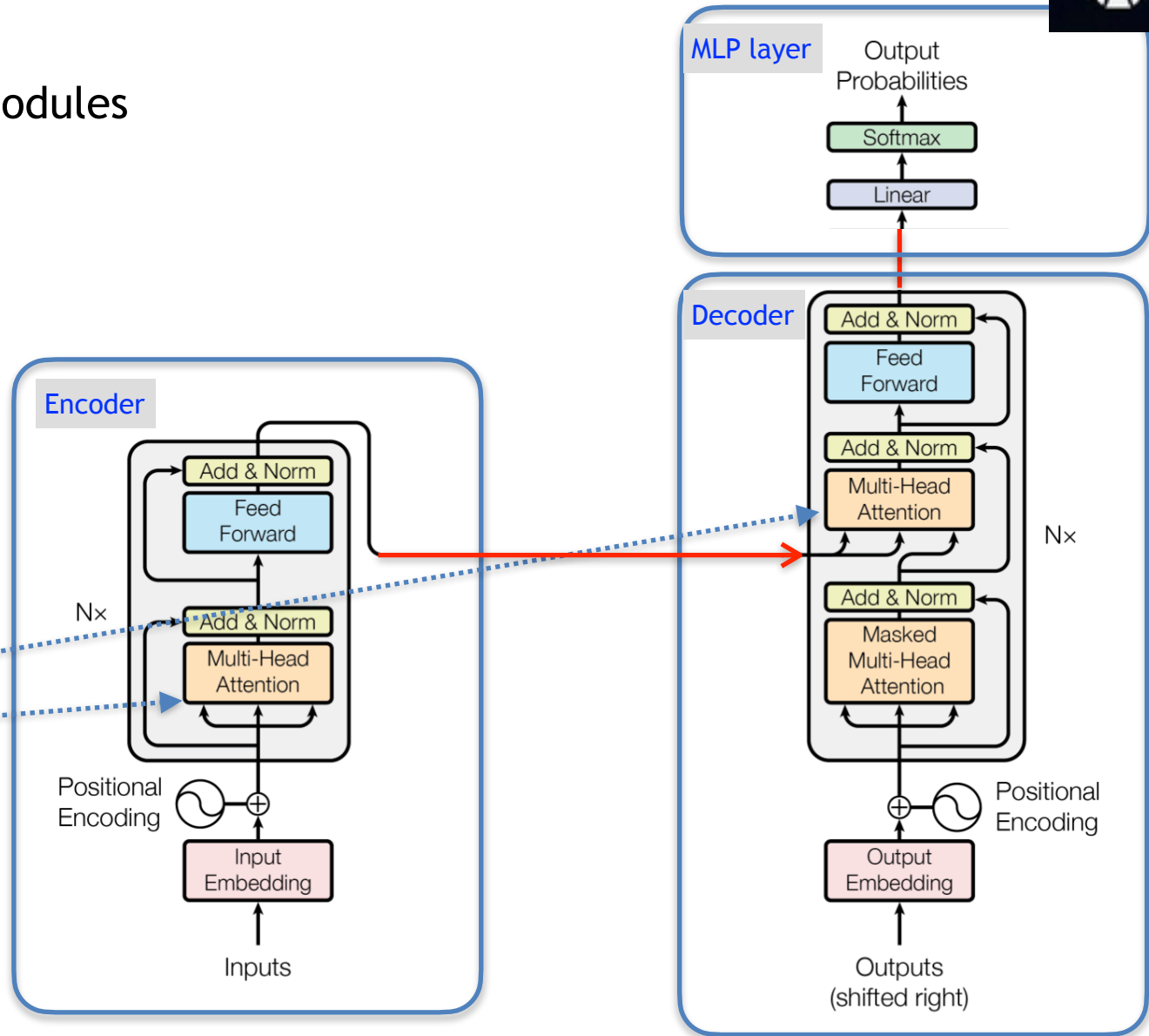
[Attention is all you need - NeurIPS'2017](#)

Transformers



- It has three modules
 - Encoder
 - Decoder
 - MLP layer

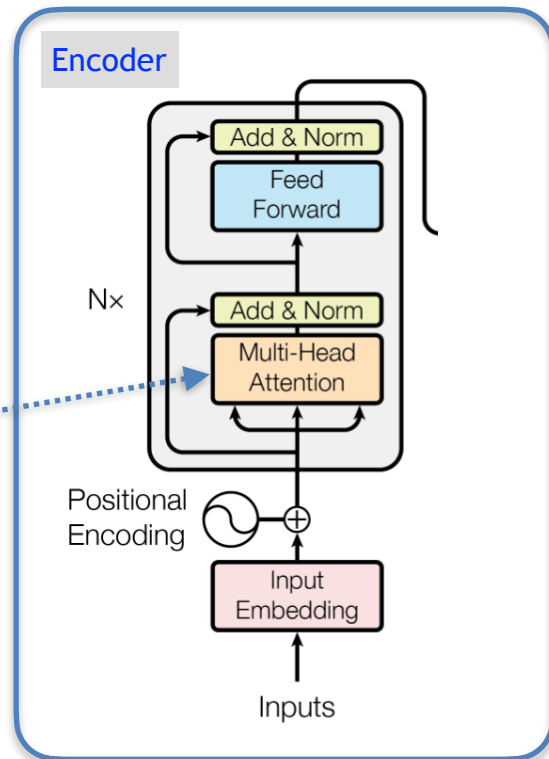
The driving force behind transformer is **attention mechanism**



Transformers: Encoder



- Lets focus on the encoder to understand what is this [attention mechanism](#)

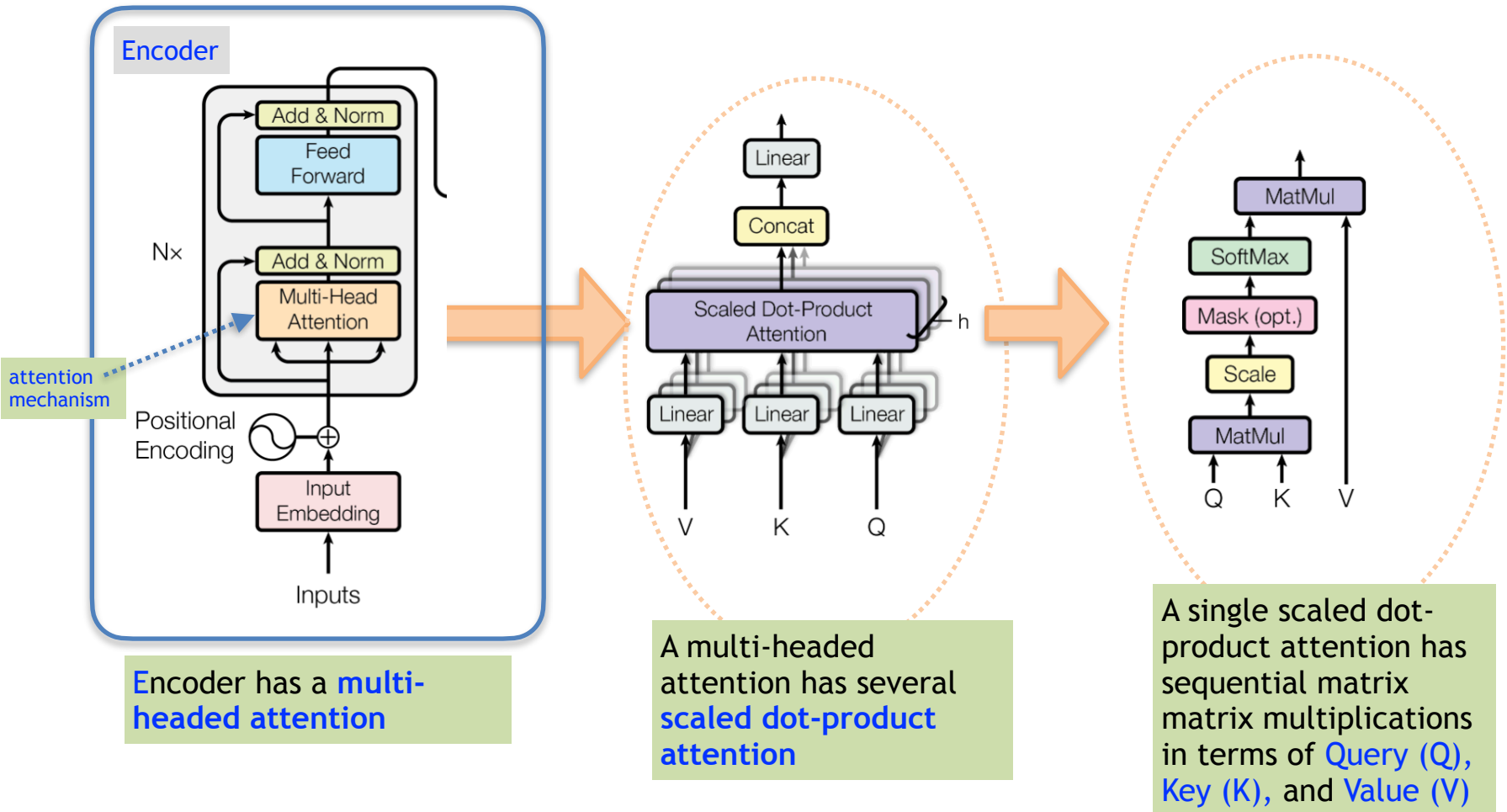


The driving force behind transformer is [attention mechanism](#)

Transformers: Encoder



- Lets focus on the encoder to understand what is this **attention mechanism**



Transformers



Today's agenda

- In-depth exploration of the caveats of vanilla RNN and LSTM
 - Vanishing gradient in vanilla RNN
 - Transfer learning is not possible with LSTM (or RNN)
- Transformers
 - Transfer learning is possible
 - New type of network architecture
- Transformers Implementation in PyTorch

Transformers Implementation in PyTorch

- PyTorch official website has an excellent tutorial demonstrating how to train a Transformer for language modeling

[Open this notebook: transformer.ipynb](#)

✓ Language Modeling with `nn.Transformer` and `torchtext`

This is a tutorial on training a model to predict the next word in a sequence using the [nn.Transformer](#) module.

The PyTorch 1.2 release includes a standard transformer module based on the paper [Attention is All You Need](#). *Compared to Recurrent Neural Networks (RNNs), the transformer model has proven to be superior in quality for many sequence-to-sequence tasks while being more parallelizable. The `nn.Transformer` module relies entirely on an attention mechanism (implemented as [nn.MultiheadAttention](#)) to draw global dependencies between input and output. The `nn.Transformer` module is highly modularized such that a single component (e.g., [nn.TransformerEncoder](#)) can be easily adapted/composed.*