Grasping Deep Learning from Fundamentals to Applications

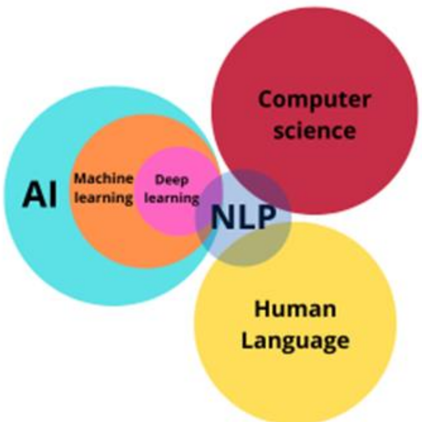
June 15, 2023

Lecture 3 – Introduction to Natural Language Processing

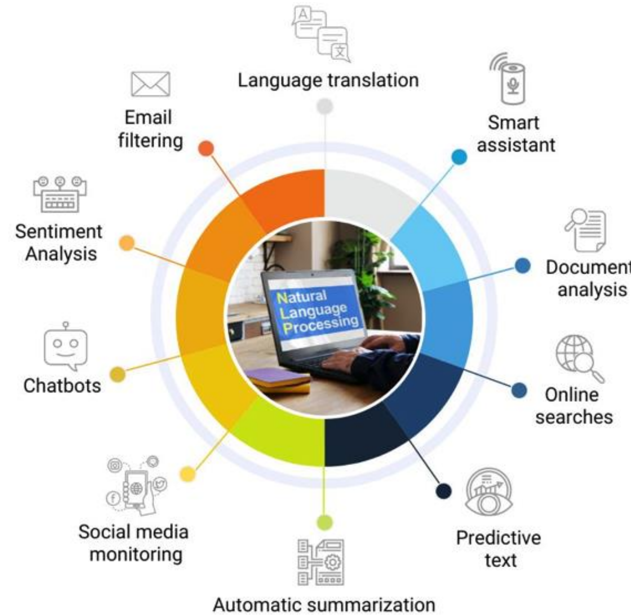
Instructors: **Yufei Huang**, PhD; **Arun Das**, PhD

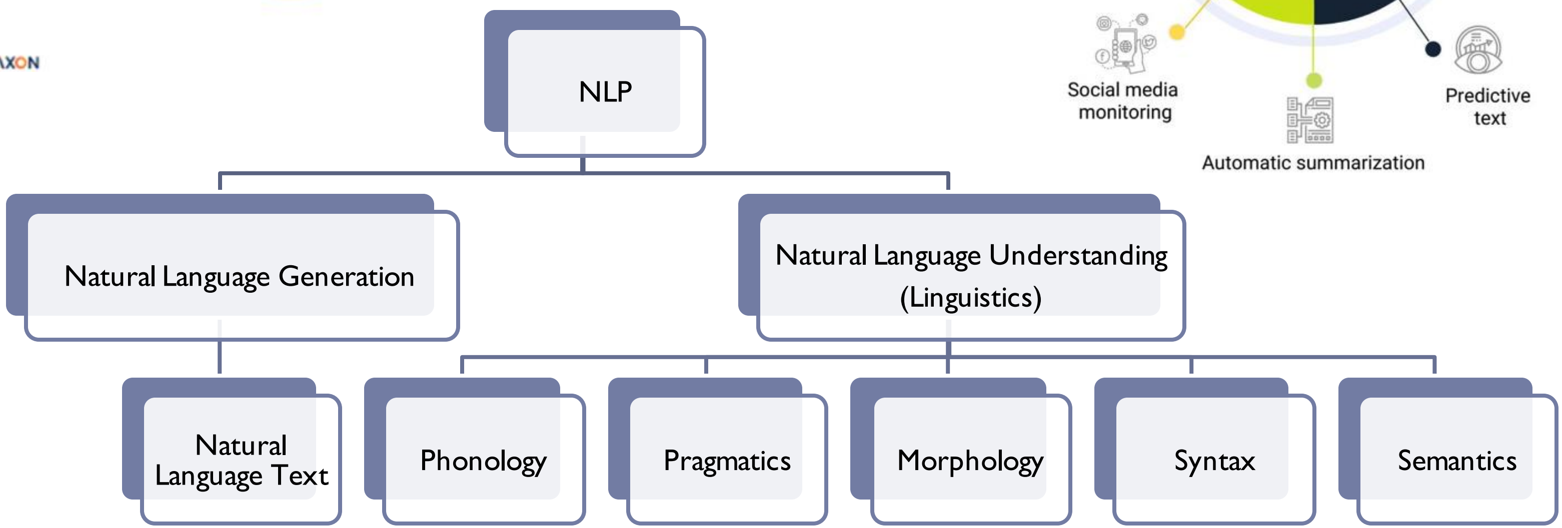
University of Pittsburgh

Evolution of Natural Language Processing (NLP)

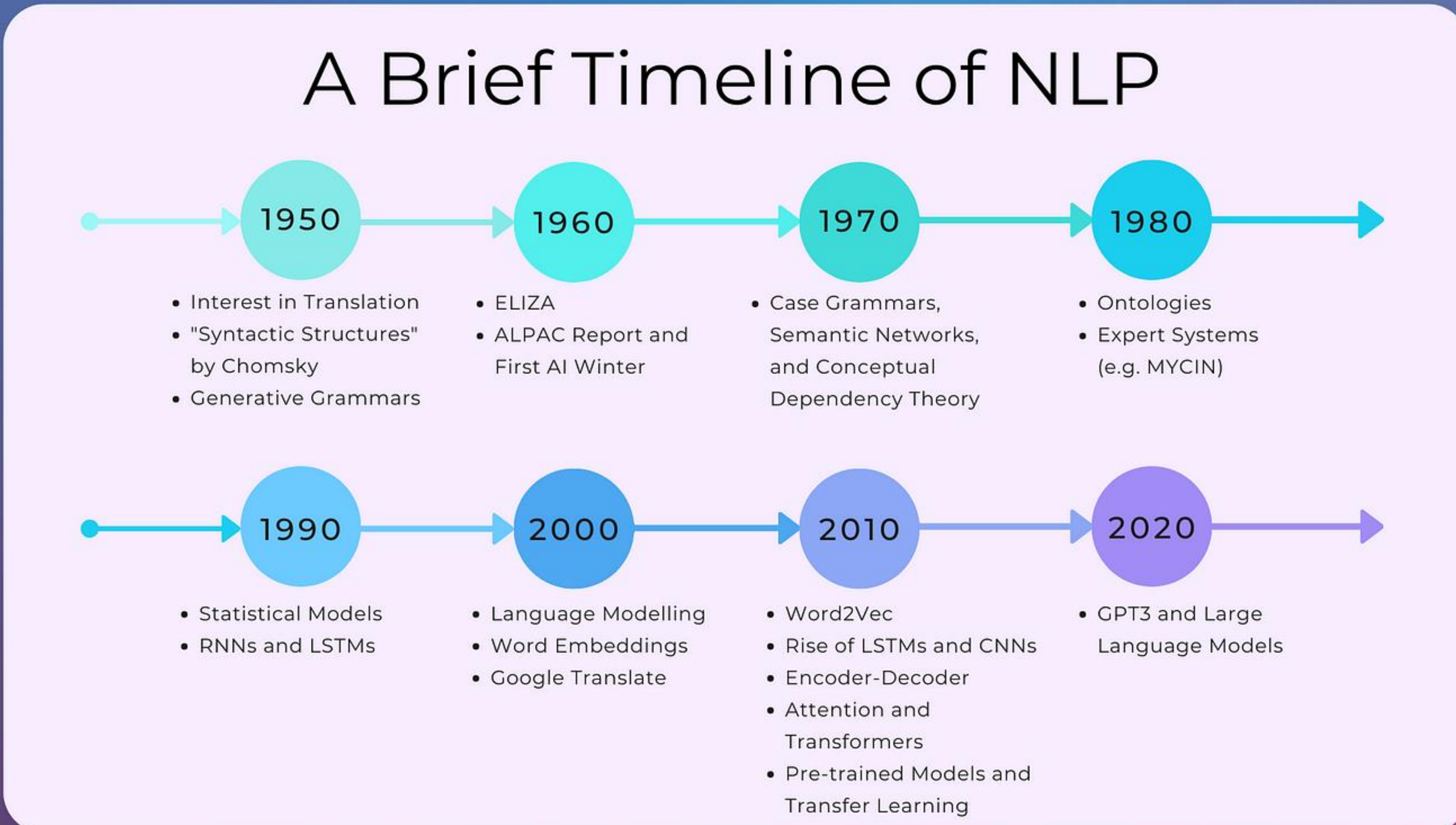


SAXON



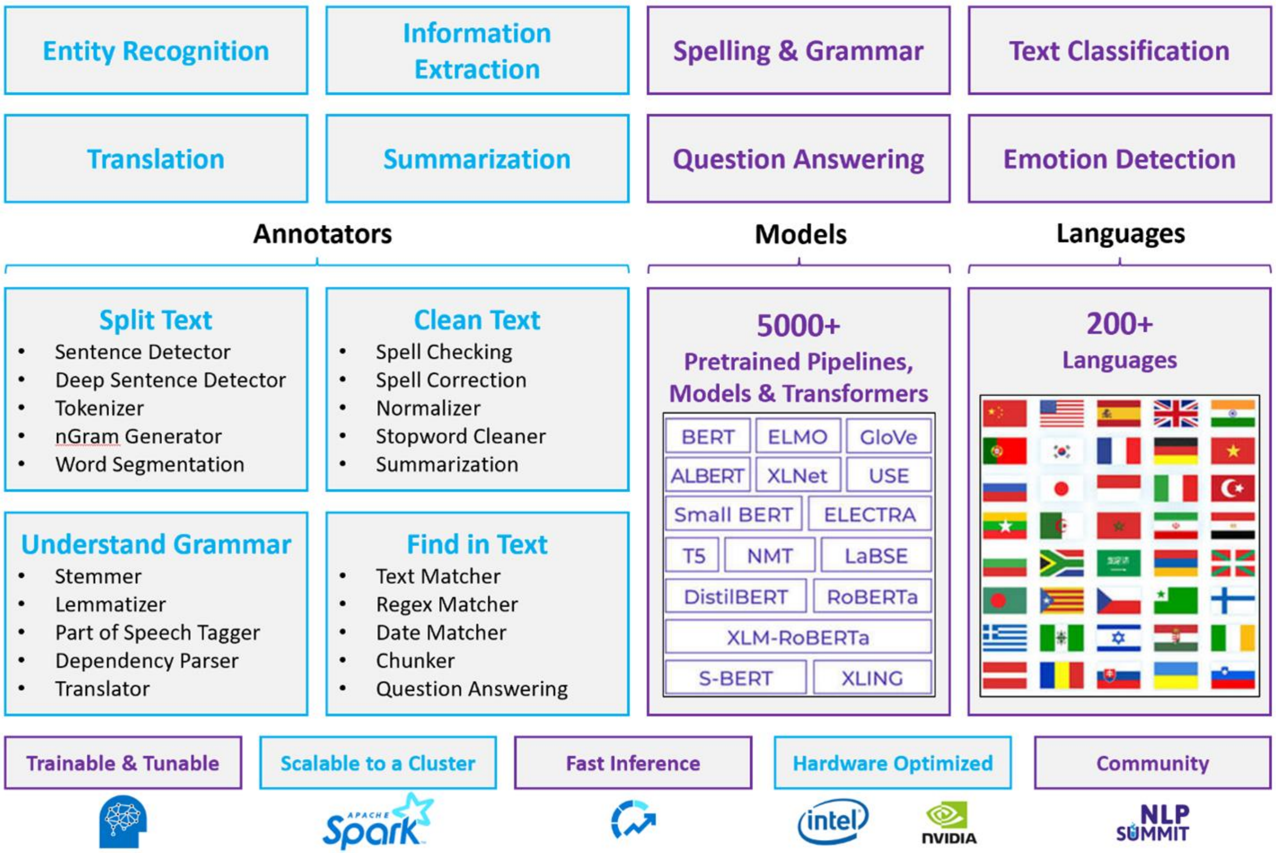


University of Pittsburgh



https://medium.com/nlplanet/a-brief-timeline-of-nlp-bc45b640f07d

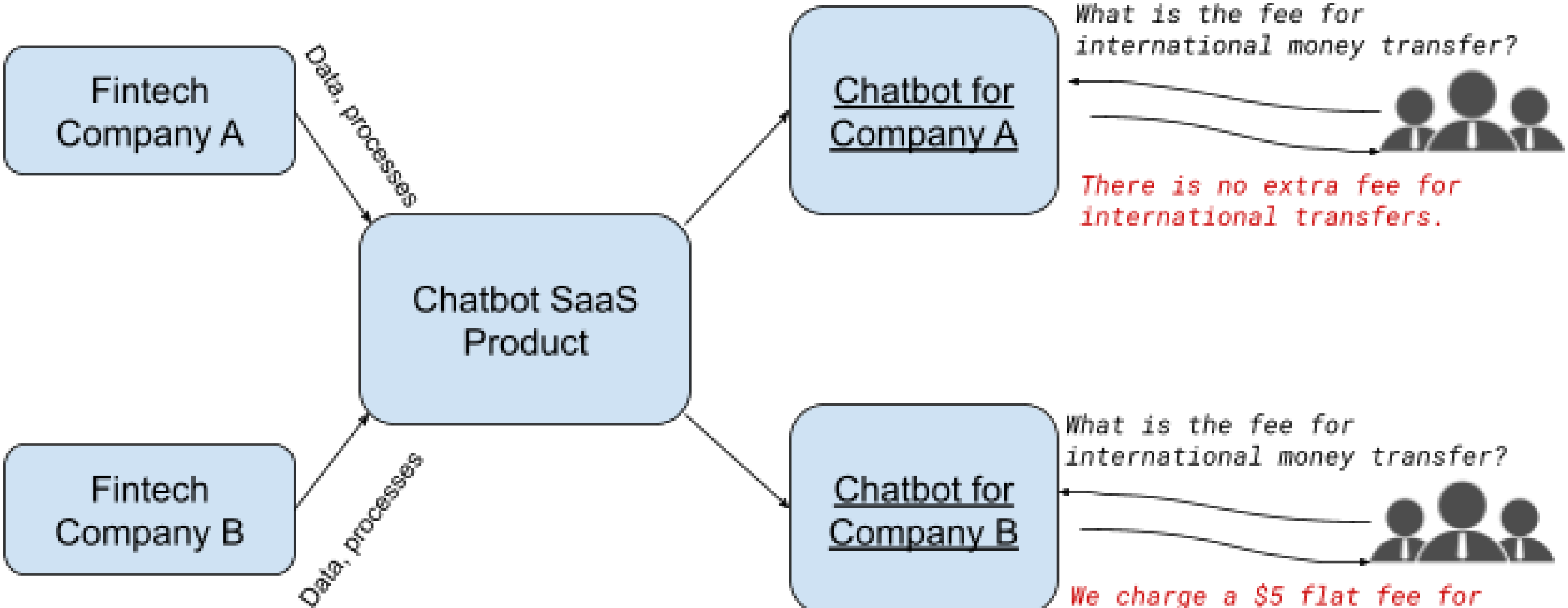
Heavy Investments!



ChatGPT

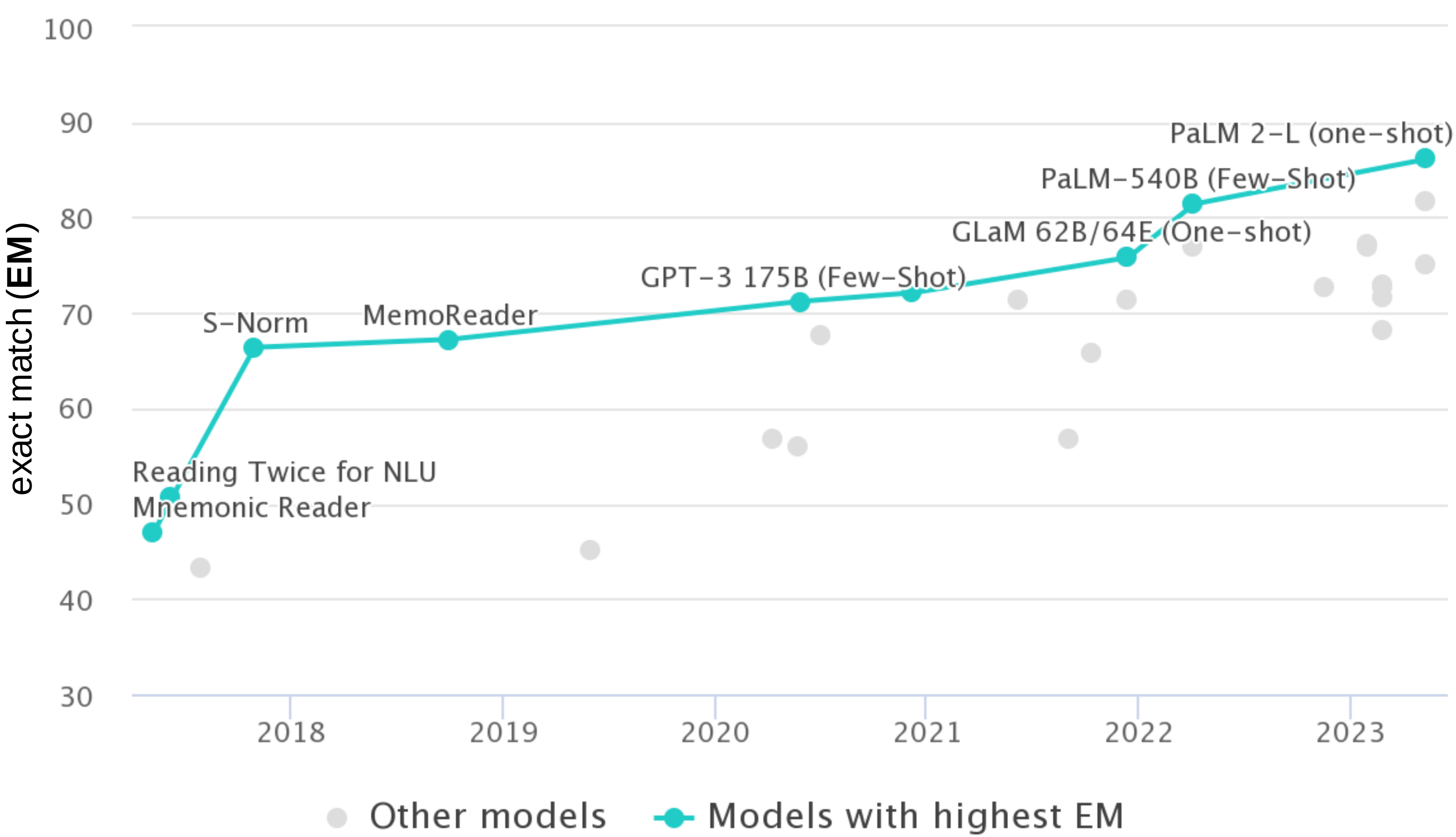
Microsoft Azure

University of Pittsburgh

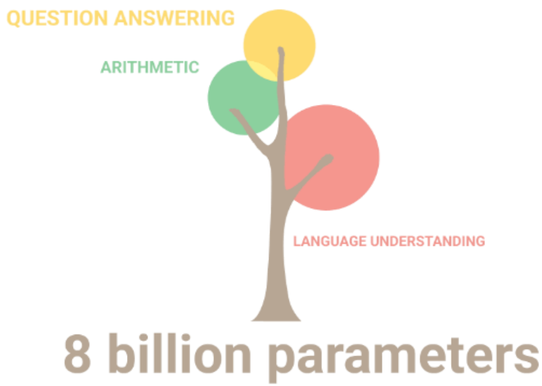


Fintech Company A
Data, processes
Fintech Company B
Data, processes
Chatbot SaaS Product
Chatbot for Company A
What is the fee for international money transfer?
There is no extra fee for international transfers.
Chatbot for Company B
What is the fee for international money transfer?
We charge a $5 flat fee for international transfers.

State-of-the-Art in Question Answering

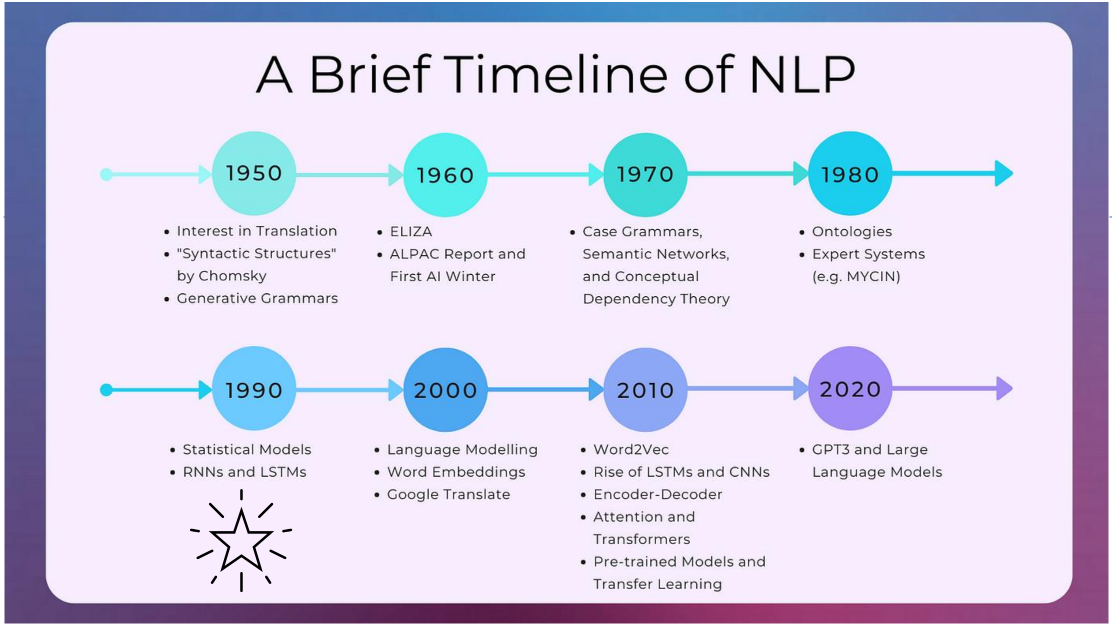


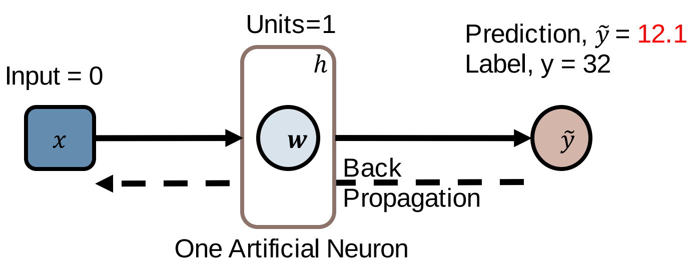
University of Pittsburgh



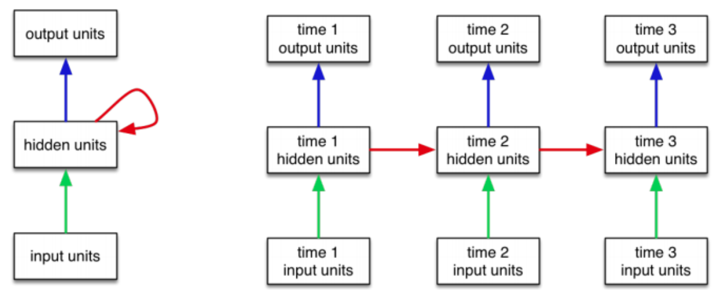
University of Pittsburgh

https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

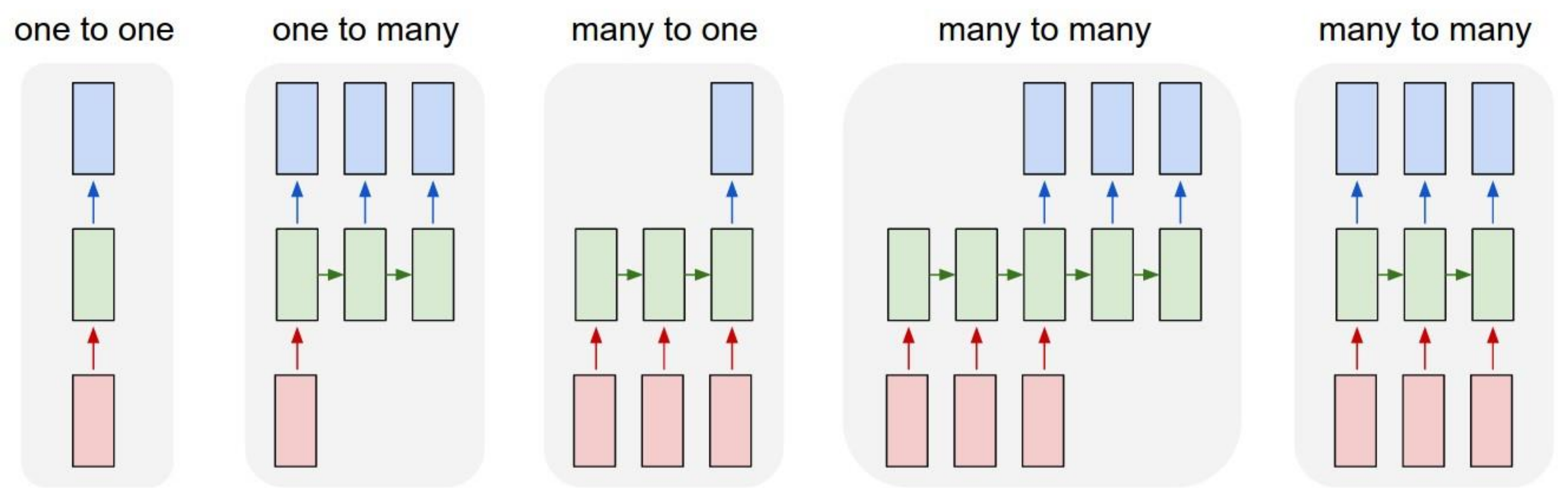


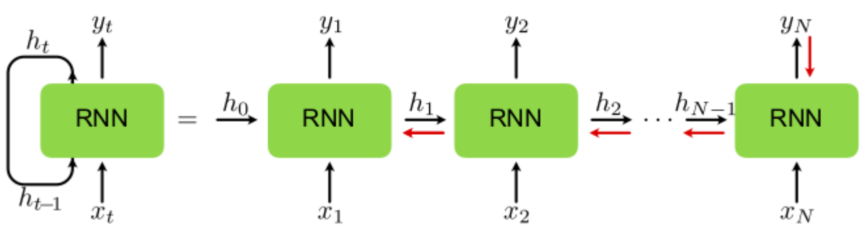


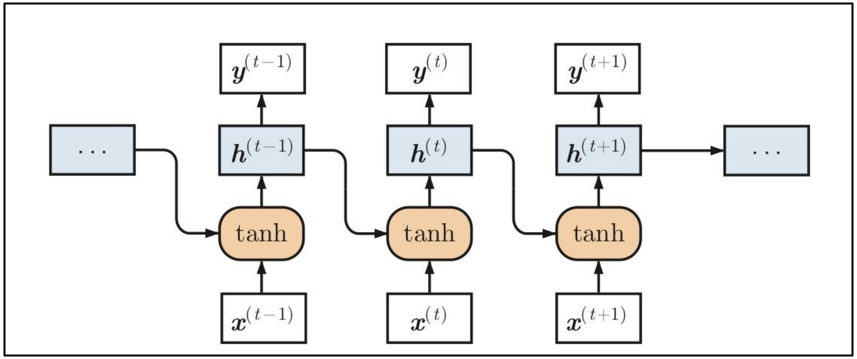
Recurrent Neural Network (RNN)

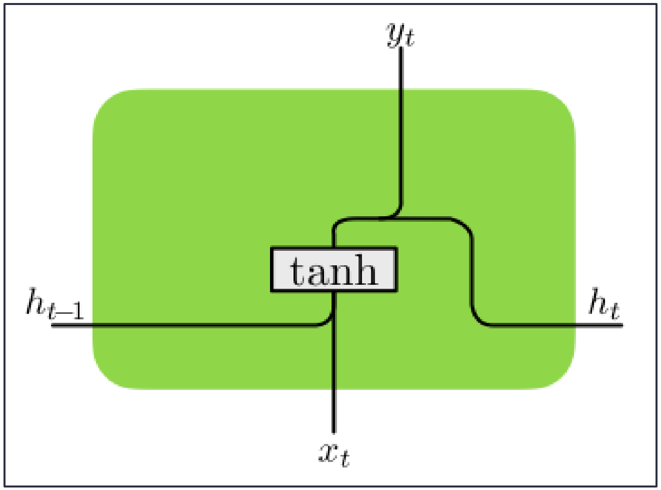


*good for learning temporal associations in the input or input sequences







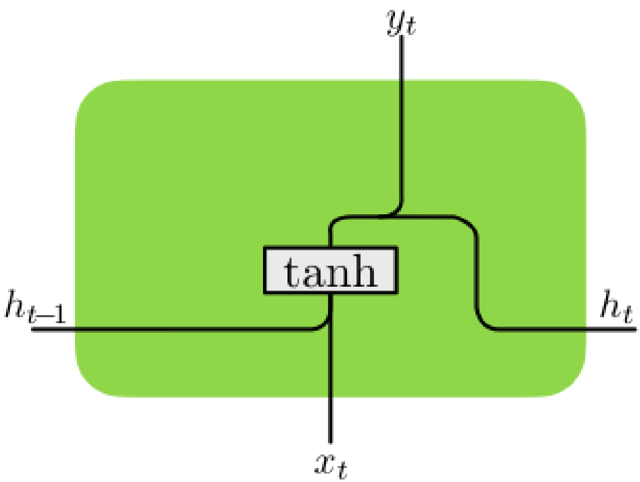


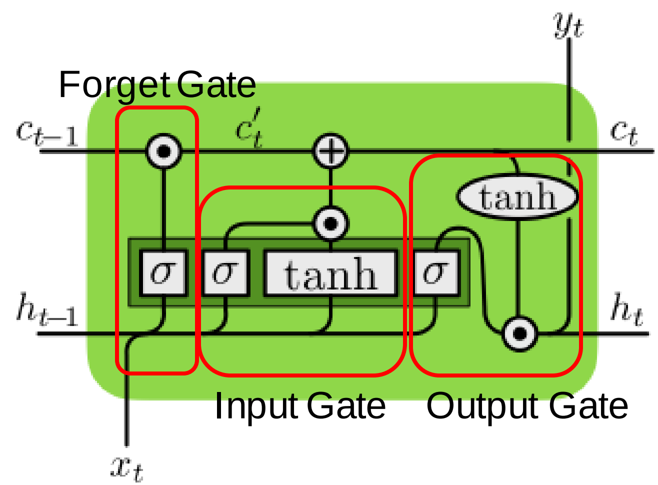
$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_{t-1} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}_t = \boldsymbol{W}_{ho}\boldsymbol{h}_t$$

Long Short-Term Memory (LSTM)

- Adds cell state to the RNN.
- Adds four gates to control the flow of information.
- Carries computation sequentially in three steps.





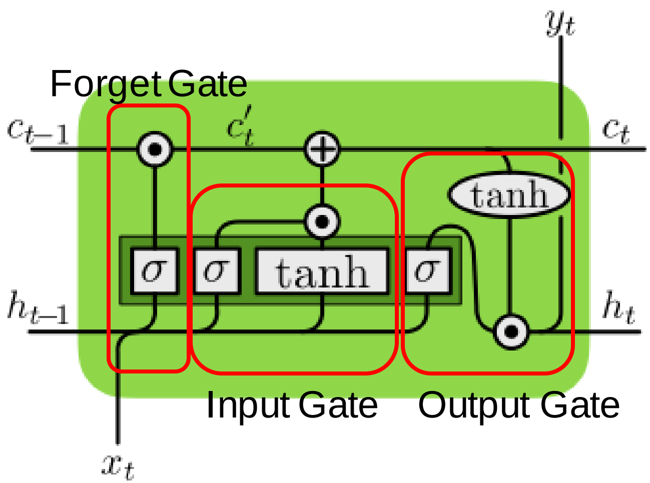
Long Short-Term Memory (LSTM)

- Forget Step: $c_t' = \sigma(W_{hf} h_{t-1} + W_{xf} x_t + b_f) \odot c_{t-1}$
- Input and Update Step: $g_t = \tanh(W_{hu} h_{t-1} + W_{xu} x_t + b_u)$
  $i_t = \sigma(W_{hi} h_{t-1} + W_{xi} x_t + b_i)$
  $c_t = c_t' + i_t \odot g_t$
- Output Step: $h_t = \sigma(W_{ho} h_{t-1} + W_{xo} x_t + b_o) \odot \tanh(c_t)$

* Forgets specific information of the cell state.

* Decide what and how much to add to the cell state.

* Decide how much of the information stored in the cell state should be written to the new hidden state.



Text Data Processing

•*Standardization* refers to preprocessing the text, typically to remove punctuation or HTML elements to simplify the dataset.
•*Tokenization* refers to splitting strings into tokens (for example, splitting a sentence into individual words by splitting on whitespace).
•*Vectorization* refers to converting tokens into numbers so they can be fed into a neural network.

Textual data is usually preprocessed using the following 5 tasks:

1. Standardize each example (usually lowercasing + punctuation stripping)
2. Split each example into substrings (usually words)
3. Recombine substrings into tokens (usually ngrams)
4. Index tokens (associate a unique int value with each token)
5. Transform each example using this index into a vector of ints or a dense float vector.

Original Sentence:
Yes it was a little low budget, but this movie shows love!

Standardization:
yes it was a little low budget but this movie shows love

Tokenization:
[yes, it, was, a, little, low, budget, but, this, movie, shows, love]

Vectorization:
[414 9 10 192 20 25 200 200 250 300 0 0 0 0 …]

Subword Tokenizers

Original Sentence:
but they did n't test for curiosity .

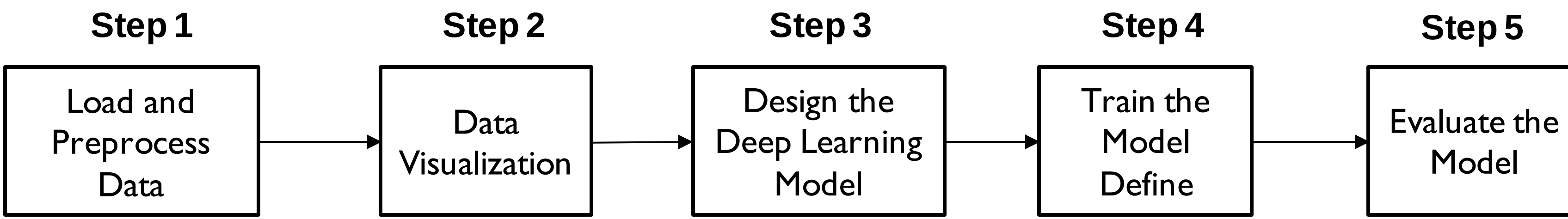
Tokenization:
[b'[START]', b'but', b'they', b'did', b'n', b"'", b't', b'test', b'for', b'curiosity', b'.', b'[END]’]

Vectorization:
[2, 87, 83, 149, 50, 9, 56, 664, 85, 2512, 15, 3]

* Skipped standardization step in this example.

University of Pittsburgh

Typical DL Pipeline



Deep Learning Training Process

You can try this service from here: [Semantic Analysis]

- Weights are randomly initialized at the beginning.
- We know the actual labels (supervised).

1. Input data -> Batch of data.
2. We find the predictions.
3. We pass the predictions to the optimizer (the optimizer already knows the actual labels.)
4. We find the loss between predicted labels and actual labels.
5. We tune the “learnable” parameters according to the loss.
6. Go back to step 1.

Input sentence label

`Yes it was a little low budget, but this movie shows love!` 1

Text Encoder

Vocabulary (** 1000 words) – VOCAB SIZE

`[414 9 10 192 20 25 200 200 250 300 0 0 0 0 …]`

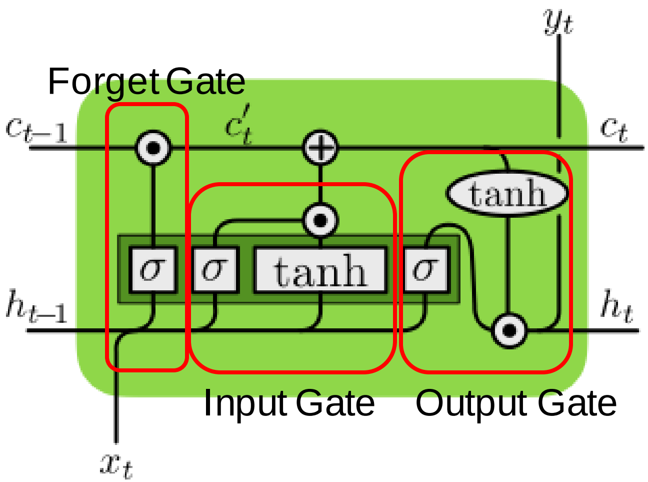
Vocabulary

```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U14')
```

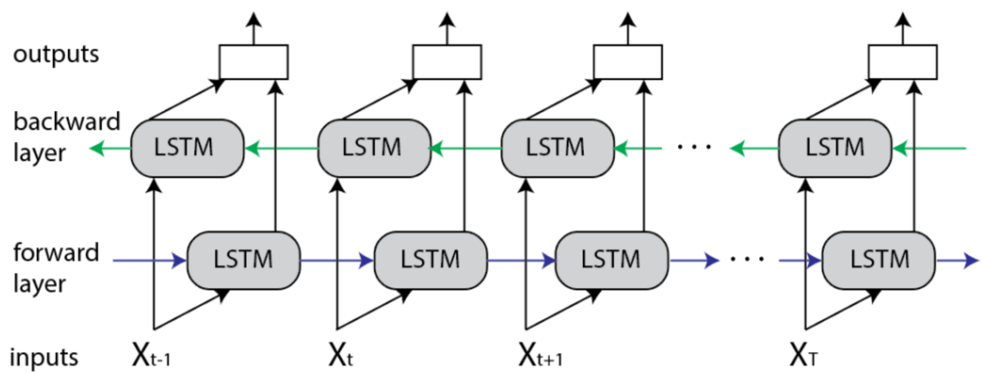
Encoded sentence

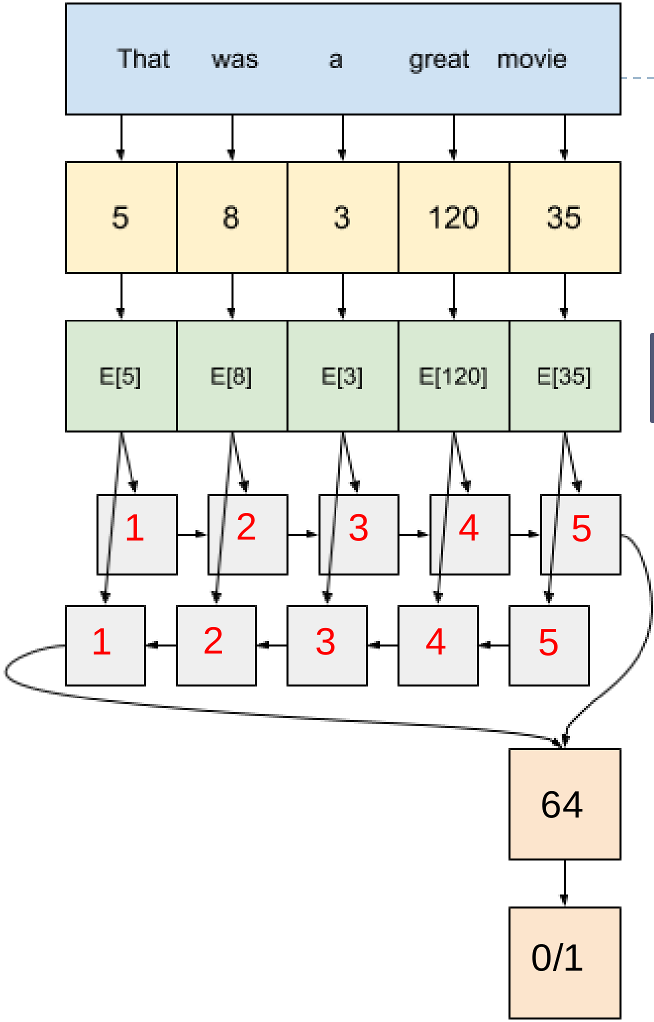
```
array([[ 10, 540,   4, ...,   0,   0,   0],
       [ 10, 120,  11, ...,   0,   0,   0],
       [414,   9,  14, ...,   0,   0,   0]])
```

* UNK token for unknown words that didn’t fit in the set vocabulary size.
* Multiple words represented by same vector encoding.

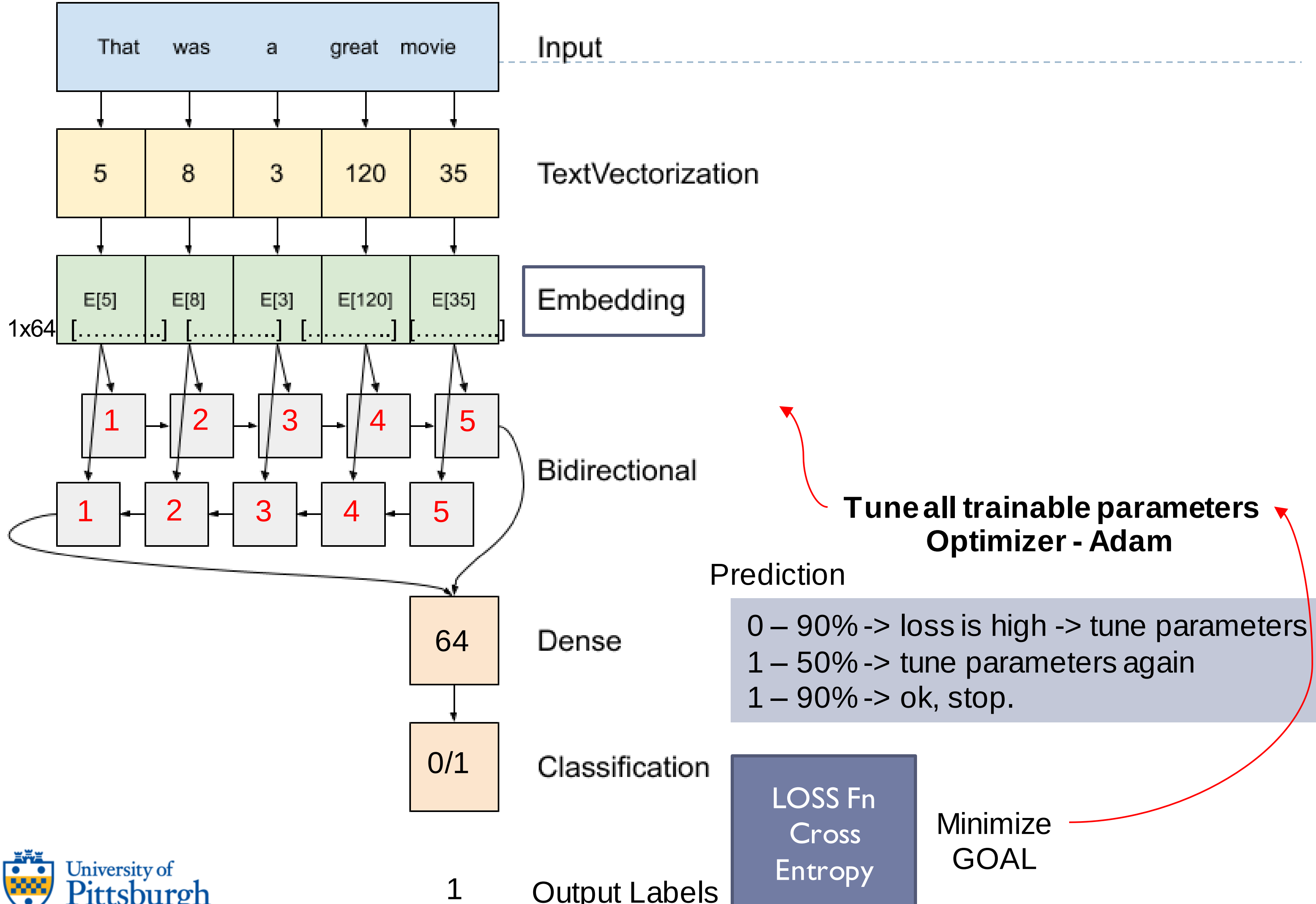


Bidirectional LSTM

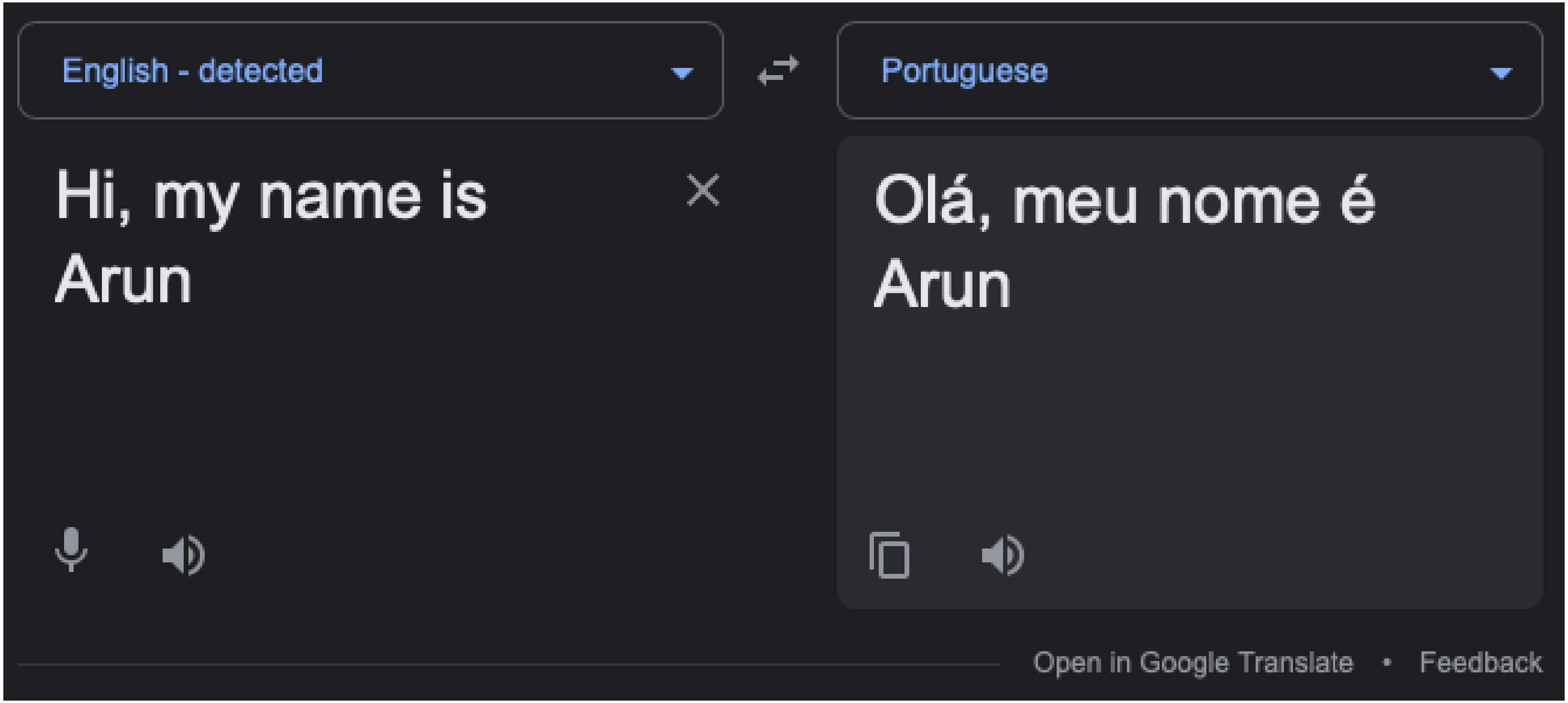




Input	
TextVectorization	
Embedding	"Learnable" Parameter or "trainable"
Bidirectional	Learnable Parameter
Dense	Learnable Parameter
Classification	Learnable Parameter



That was a great movie
Input
5 8 3 120 35
TextVectorization
E[5] E[8] E[3] E[120] E[35]
1x64 [……….] [……….] [……….] [……….]
Embedding
1 2 3 4 5
1 2 3 4 5
Bidirectional
Tune all trainable parameters
Optimizer - Adam
Prediction
0 – 90% -> loss is high -> tune parameters
1 – 50% -> tune parameters again
1 – 90% -> ok, stop.
64
Dense
0/1
Classification
LOSS Fn
Cross
Entropy
Minimize
GOAL
1
Output Labels

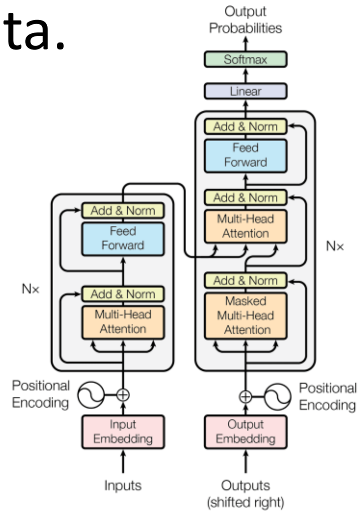


- Complex!
- May not have an exact match for phrases or words.
- Grammar/humor/context + cultural differences.

University of Pittsburgh

Transformers

- Transformers are parallelizable.
- Transformers can capture distant or long-range contexts and dependencies.
- Transformers make no assumptions about the temporal/spatial relationships across the data.



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 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.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

University of Pittsburgh

Portuguese

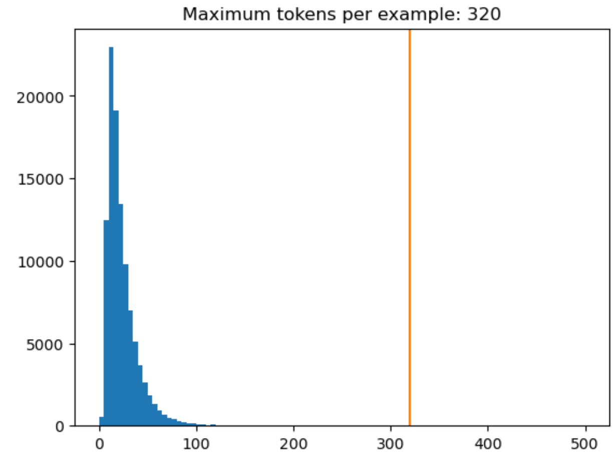
mas eles não tinham a curiosidade de me testar .

English

but they did n't test for curiosity .

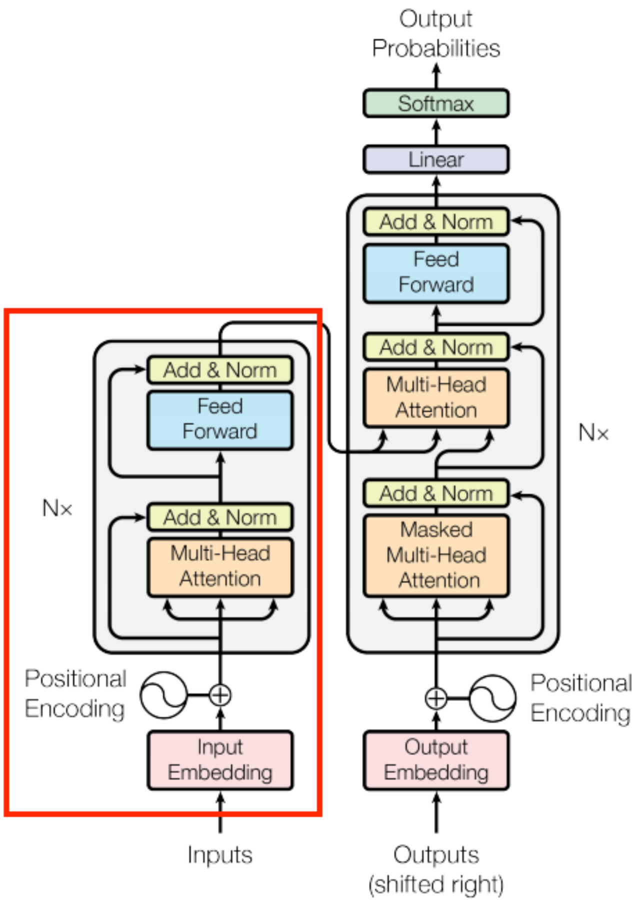
```
[2, 87, 83, 149, 50, 9, 56, 664, 85, 2512, 15, 3]
[b'[START]', b'but', b'they', b'did', b'n', b"'", b't', b'test', b'for', b'curiosity', b'.', b'[END]’]
```

> Subwords: the word 'searchability' is decomposed into 'search' and '##ability', and the word 'serendipity' into 's', '##ere', '##nd', '##ip' and '##ity'.

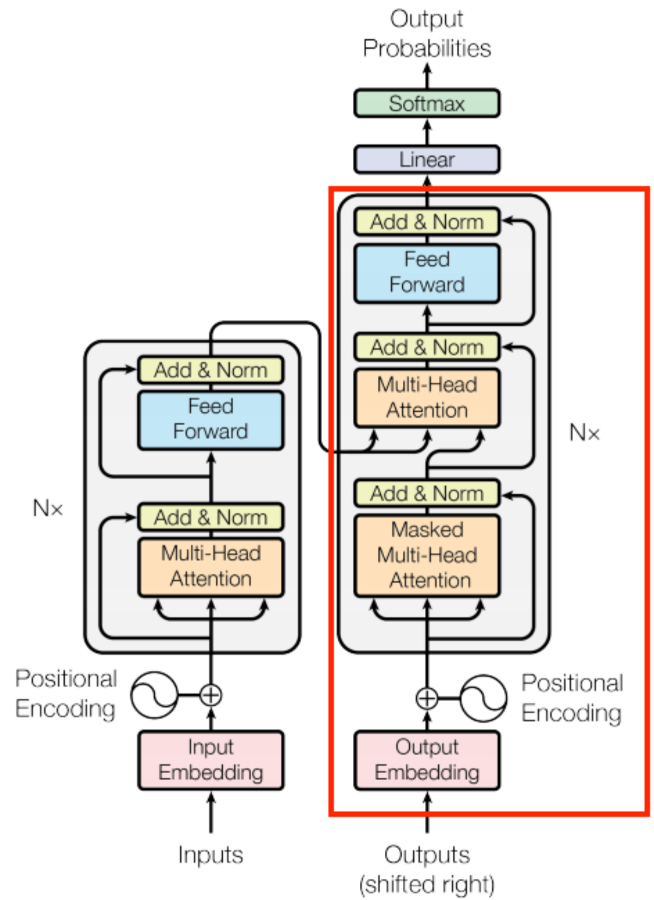


University of Pittsburgh

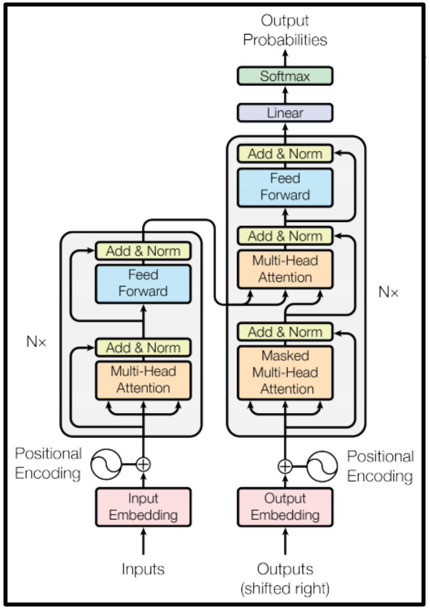
Encoder

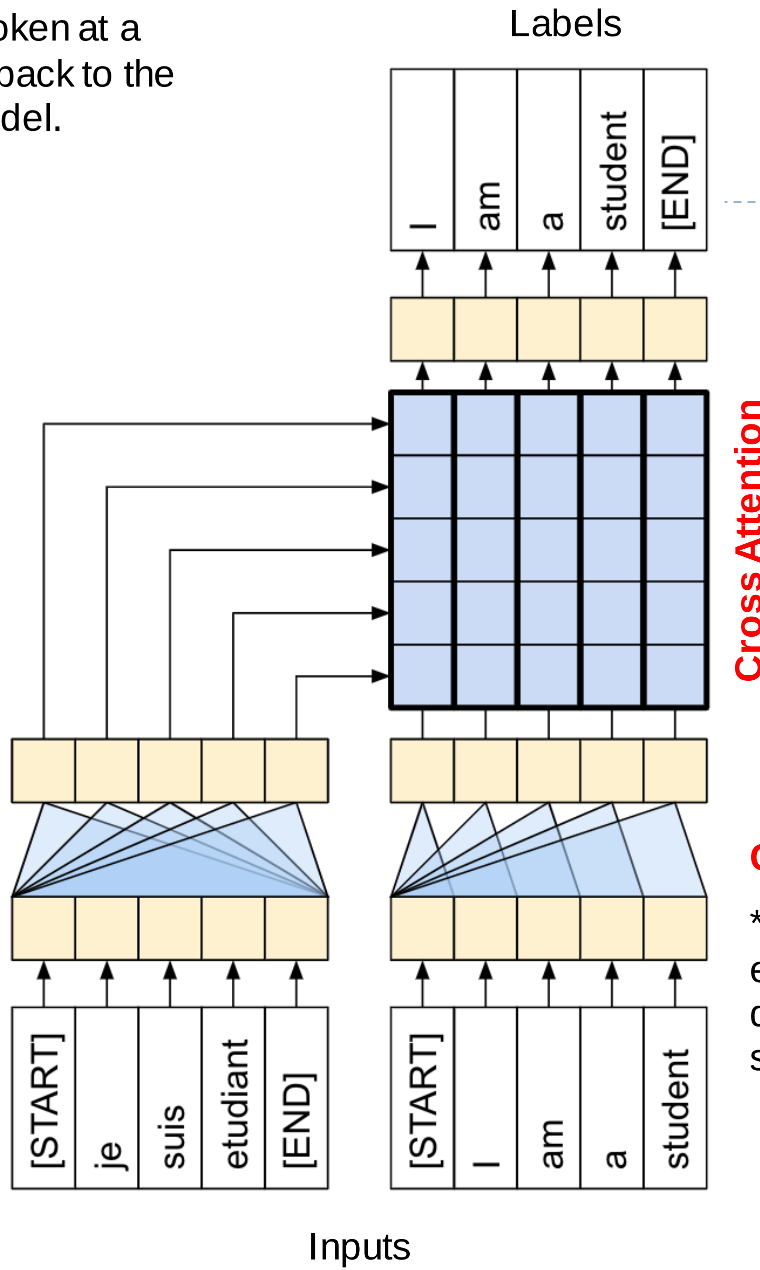


Decoder



* Generates the text one token at a time and feeds the output back to the input – Autoregressive model.





* Inputs and Labels are shifted by 1.

* Lets the decoder access the information extracted by the encoder.

* It computes a vector from the entire context sequence, and adds that to the decoder's output.

Global Self-Attention

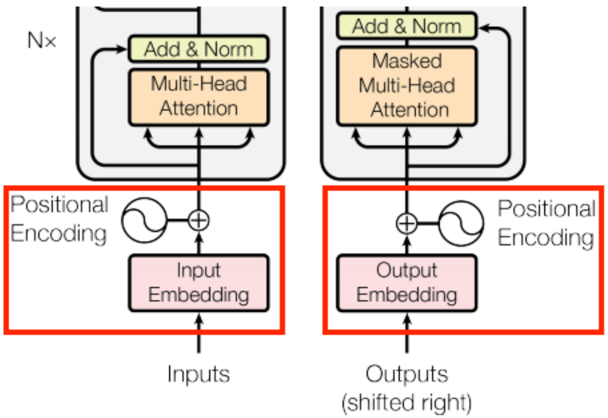
* Responsible for processing the context sequence, and propagating information along its length.

Causal Self-Attention

* Makes sure output for each sequence element depends on the previous sequence elements.

University of Pittsburgh

Positional Encoding

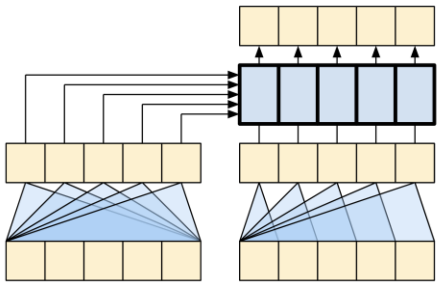


* A stack of sines and cosines that vibrate at different frequencies depending on their location along the depth of the embedding vector.

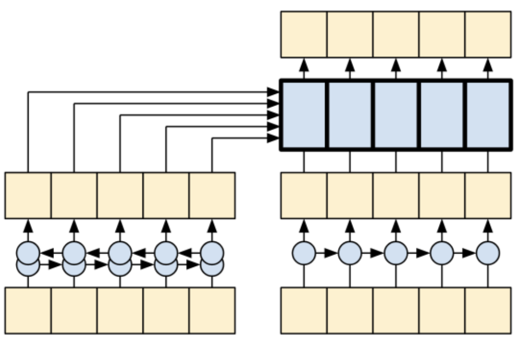
* The attention layers see their input as a set of vectors, with no order.

* So, we add a positional encoding to the embeddings to force near-by elements to have similar positional encodings.

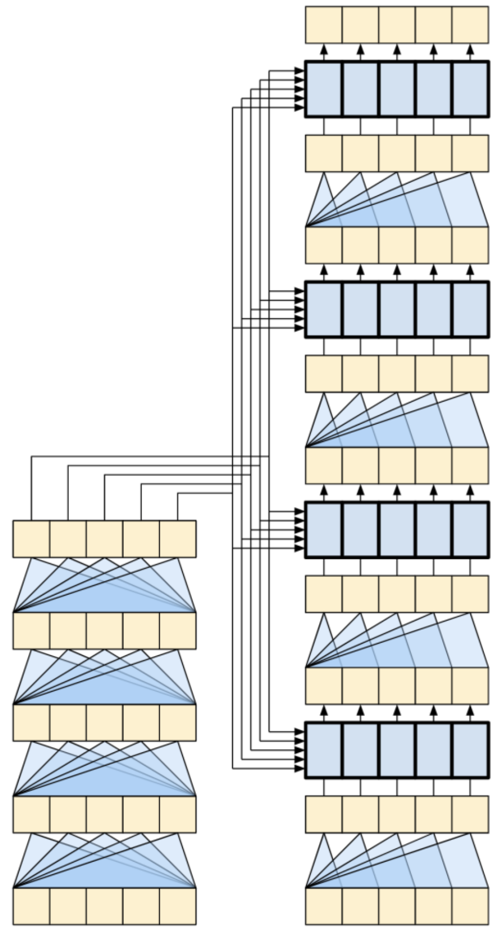
1-layer Transformer

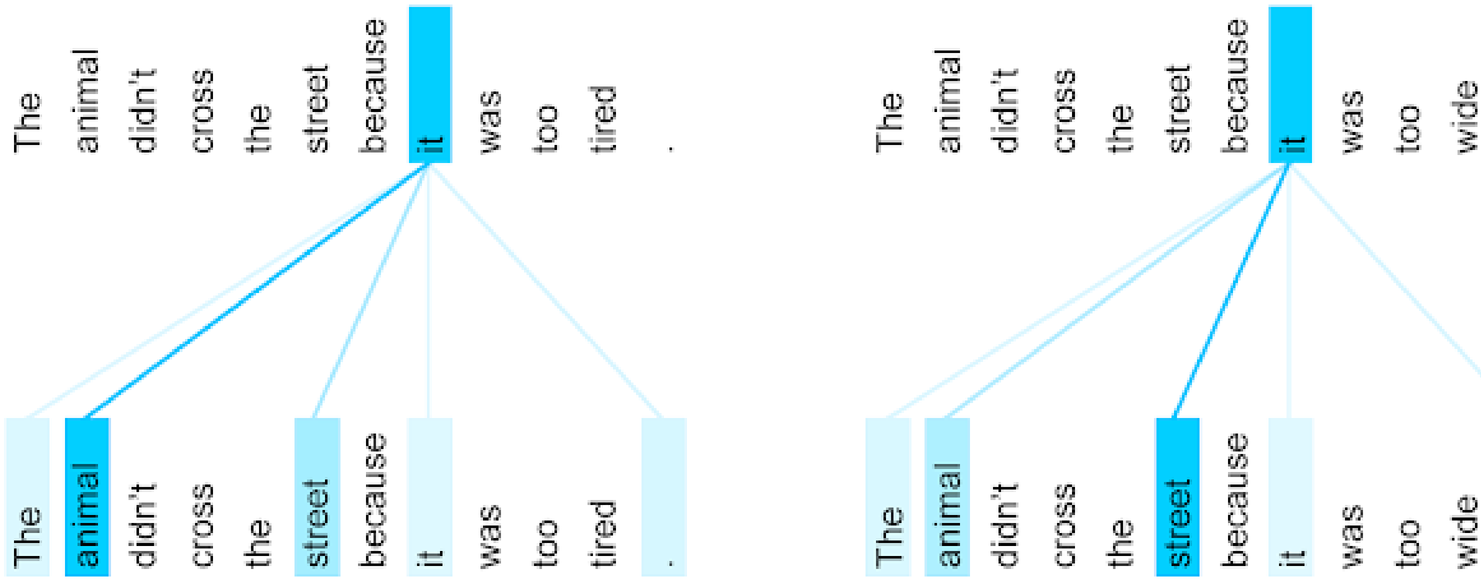


RNN+Attention



4-layer Transformer





The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English-to-French translation (one of eight attention heads). Source: Google AI Blog.

Code

Step 1: Load and Preprocess Data

```
VOCAB_SIZE = 1000
encoder = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label: text))
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U14')
```

```
encoded_example = encoder(example)[:3].numpy()
encoded_example
```

```
array([[ 10, 540,   4, ...,   0,   0,   0],
       [ 10, 120,  11, ...,   0,   0,   0],
       [414,   9,  14, ...,   0,   0,   0]])
```

University of Pittsburgh

Step 2: Visualize the dataset

[33]

```
num_words = 15
words_in_the_sentence = str(example[n].numpy()).split(' ')[:num_words]
encodeded_id_of_the_words = encoded_example[n][:num_words]

print("Encoding\tWord")
for word, encoded_id in zip(words_in_the_sentence, encodeded_id_of_the_words):
  print(encoded_id, "\t\t", word)
```

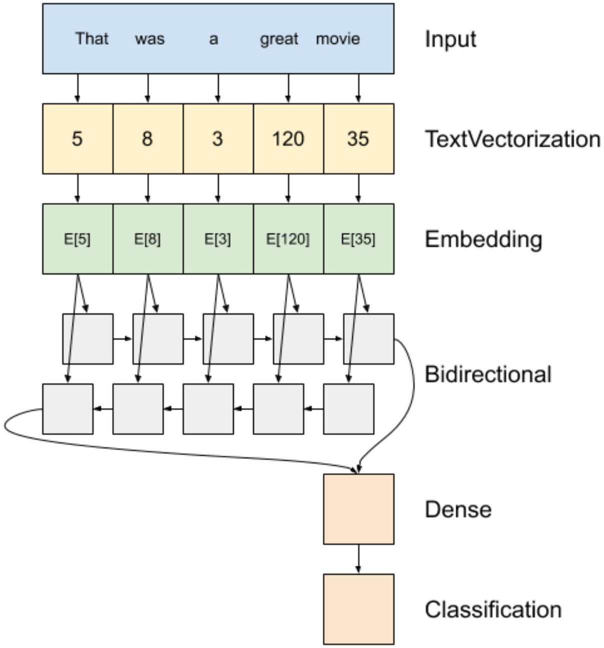
```
Encoding        Word
10               b'I
86               first
1                encountered
11               this
120              show
51               when
10               I
14               was
1                staying
8                in
1                Japan
16               for
1                six
1                months
226              last
```

1000 VOCAB SIZE

Word Count or Bag of Words

University of Pittsburgh

Step 3: Design the NLP Model




```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

The output of the Bidirectional LSTM is passed to a Dense layer with 64 nodes, and then further passed to the output layer for final binary classification.

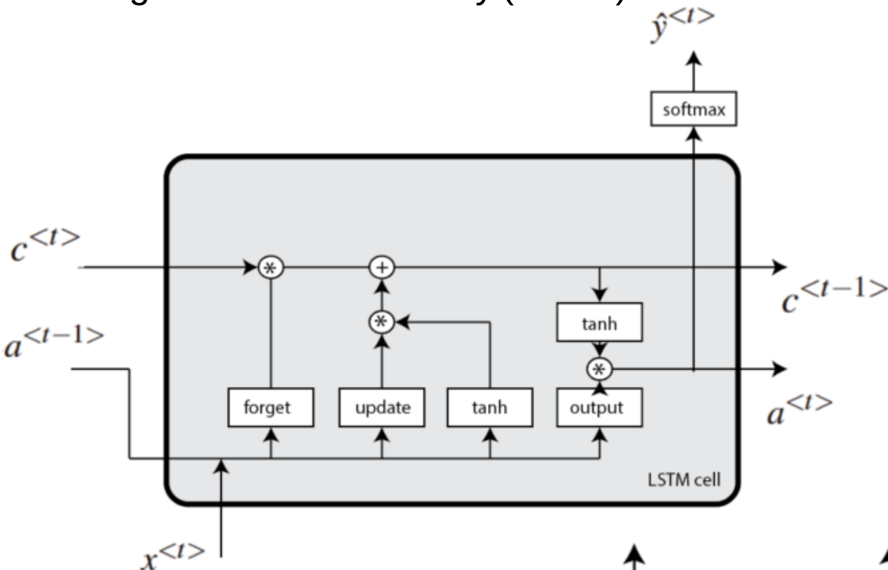
2nd word is processed based on the embedding at 2nd location in the sentence as well as the output of the first word.

3rd word is processed based on the embedding at the 3rd loc in the sentence as well as the output of the second word.

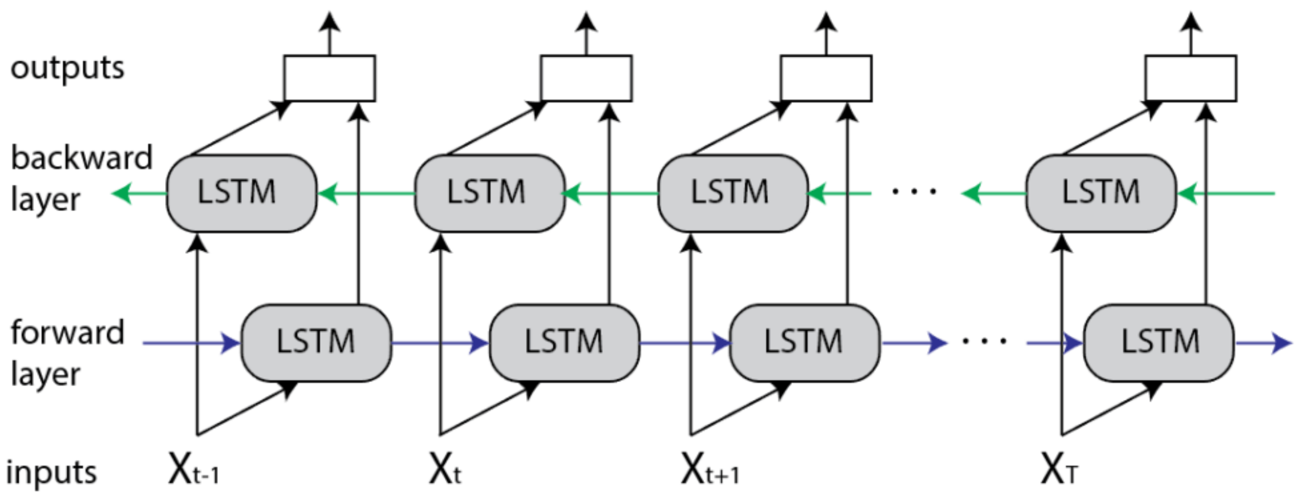
Word2Vec – Package by Google to create Embeddings.

University of Pittsburgh

Long Short-Term Memory (LSTM) Module



Bidirectional LSTM



University of Pittsburgh

Step 4: Train the NLP Model

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

Once we have defined the model, now we can compile the model with the loss and optimizer functions just like we did for the DNN and CNN examples last week. We can then fit the model on the train dataset to train the embedding layer, RNN, and dense layers. Note that the RNN layer has multiple layers inside which enables the temporal or sequential nature of learning. The overall parameters of the model is thus dependent on the embedding size, number and size of RNN layers, and the number and size of dense layers.

```
history = model.fit(train_dataset, epochs=5)
```

Step 5: Evaluate the Trained Model

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

We can run evaluate method on the model to find the test loss and accuracy.

Now, given a new input, we can understand if a movie review is positive or negative.

```
Question:  This is a fantastic movie.
Predicted label:  Positive
Question:  This is a bad movie.
Predicted label:  Negative
Question:  This movie was so bad that it was good.
Predicted label:  Negative
Question:  I will never say yes to watching this movie.
Predicted label:  Negative
Question:  Skip this movie.
Predicted label:  Negative
Question:  Don't waste your time.
Predicted label:  Negative
```

We can now experiment by adding multiple RNN layers to the network and trying out different types of RNN layers.