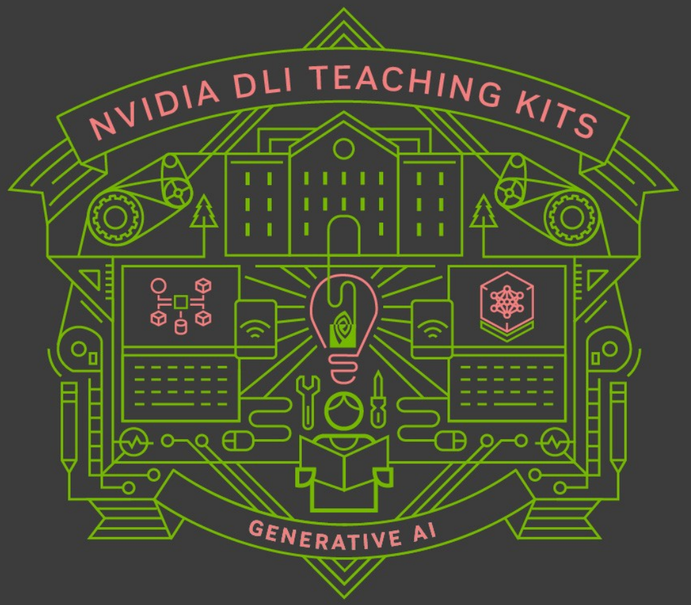
Lecture 3.1 - Language Models and Attention

Generative AI Teaching Kit

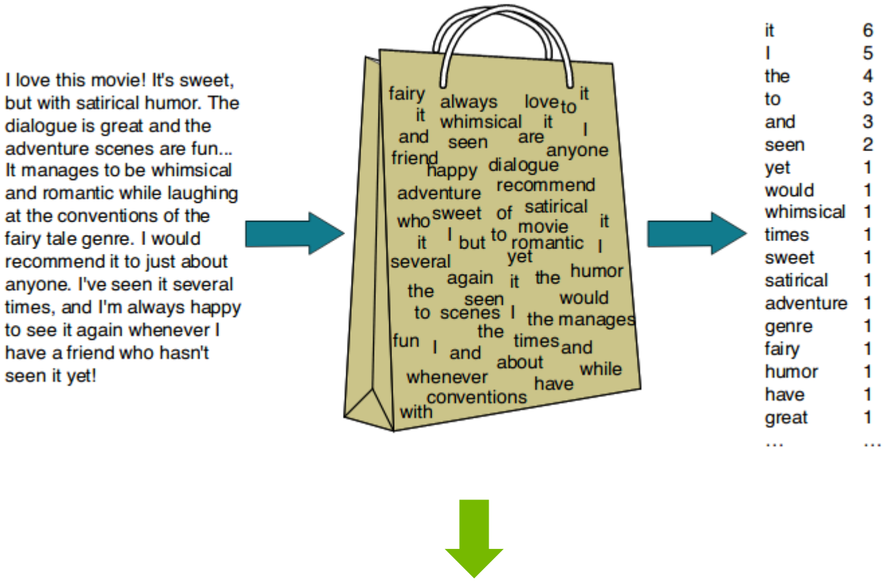


This lecture

- Language Models in Deep Learning
- Evolution of attention mechanisms pre-transformers
- Attention Mechanism

Language Models in Deep Learning

Deep Language Modeling

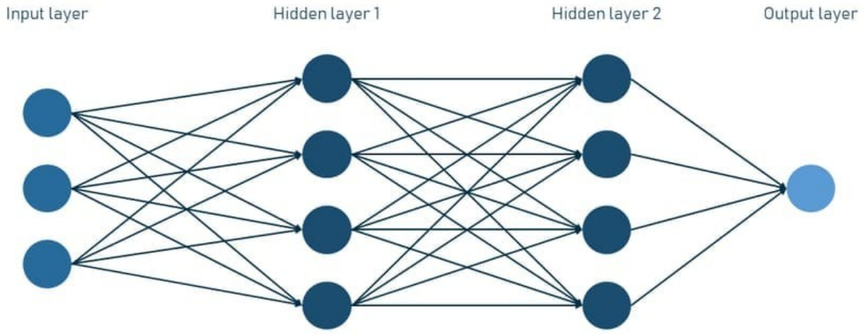


Recall that a goal of a language model is to predict the correct word based on the given input and the available corpus.

We have already seen statistical methods like Bag of Words which use the statistics of the prevalence of words in the dataset to predict the most likely next word.

But deep learning typically allows for more complex features to be encoded into a model.

Using deep learning for language, we can explore more complex relationships and more efficiently make use of the wealth of language data that is present digitally.

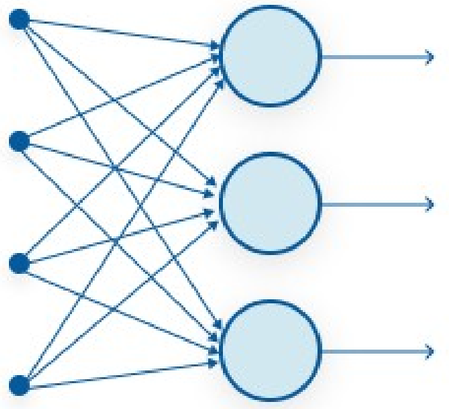


Challenges of Deep Learning Sequences

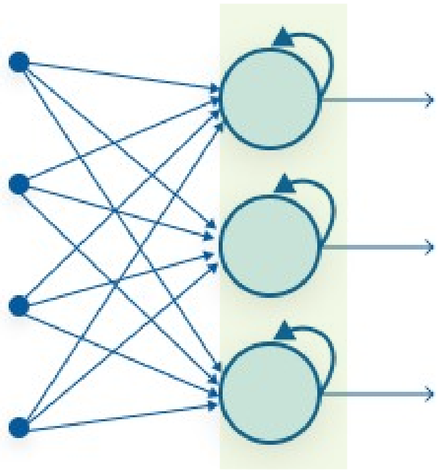
One main issue that standard neural networks run into when attempting to model language is time-dependency.

Language is **causal**, previous words influence future words, the same is true for sentences and paragraphs.
Neural networks don't have an inherent way to "**remember**" previous inputs, or to properly represent sequences.

A newer innovation, the **recurrent** neural network is designed to process sequential data by maintaining a form of memory through its recurrent connections.



Feed-Forward Neural Network



Recurrent Neural Network

Recurrent Neural Networks

Unlike traditional feedforward networks, RNNs are designed to recognize patterns in sequences of data, such as time-series, text, speech, or videos, by maintaining a hidden state that captures information about previous inputs in the sequence.

Key Characteristics of RNNs:

Sequential Processing:

RNNs process input one step at a time, making them well-suited for tasks where data has a temporal or sequential structure.

Recurrent Connections:

At each time step, the hidden state of the network is updated based on the current input and the hidden state from the previous time step. This allows the network to "remember" information from earlier in the sequence.

Shared Weights:

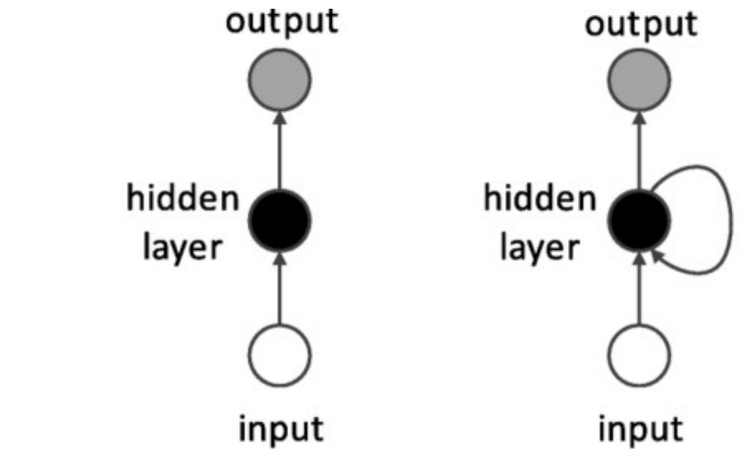
The weights used for processing are shared across time steps, which reduces the number of parameters and helps capture temporal dependencies.

Memory:

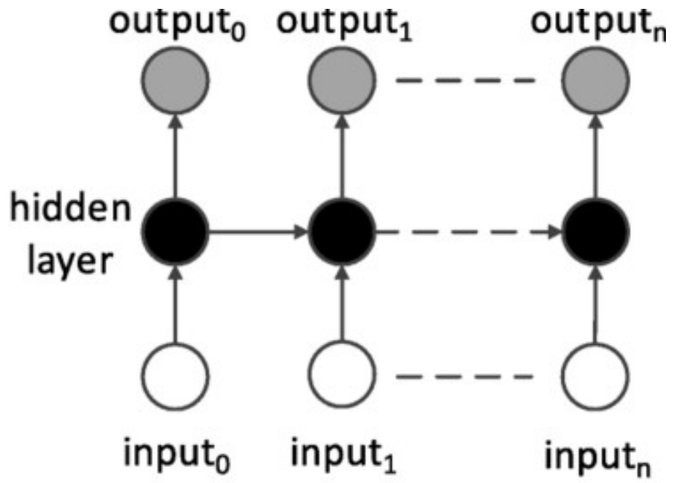
The hidden state serves as a form of memory, enabling the network to use information from earlier inputs to influence later outputs.

Training via Backpropagation Through Time (BPTT):

The gradient is computed for all time steps of the sequence to train the network. However, this can lead to issues like vanishing or exploding gradients.



(a) A traditional neural network (b) RNN



(c) RNN in unfolded form

Recurrent Neural Networks – Code Differences

```python
# ANN: Feedforward Neural Network
class SimpleANN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleANN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)  # Fully connected layer
        self.relu = nn.ReLU()                          # Activation function
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

```python
# RNN: Recurrent Neural Network
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleRNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)  # Recurrent layer
        self.fc = nn.Linear(hidden_size, output_size)  # Output layer

    def forward(self, x):
        out, _ = self.rnn(x)  # The RNN returns all outputs and the final hidden state
        out = self.fc(out[:, -1, :])  # Take the last time step's output for prediction
        return out
```

Key Differences

- **ANN**: Processes inputs without considering temporal relationships; uses a simple Linear layer.
- **RNN**: Processes sequences with recurrent connections; uses an RNN layer and captures sequential dependencies.

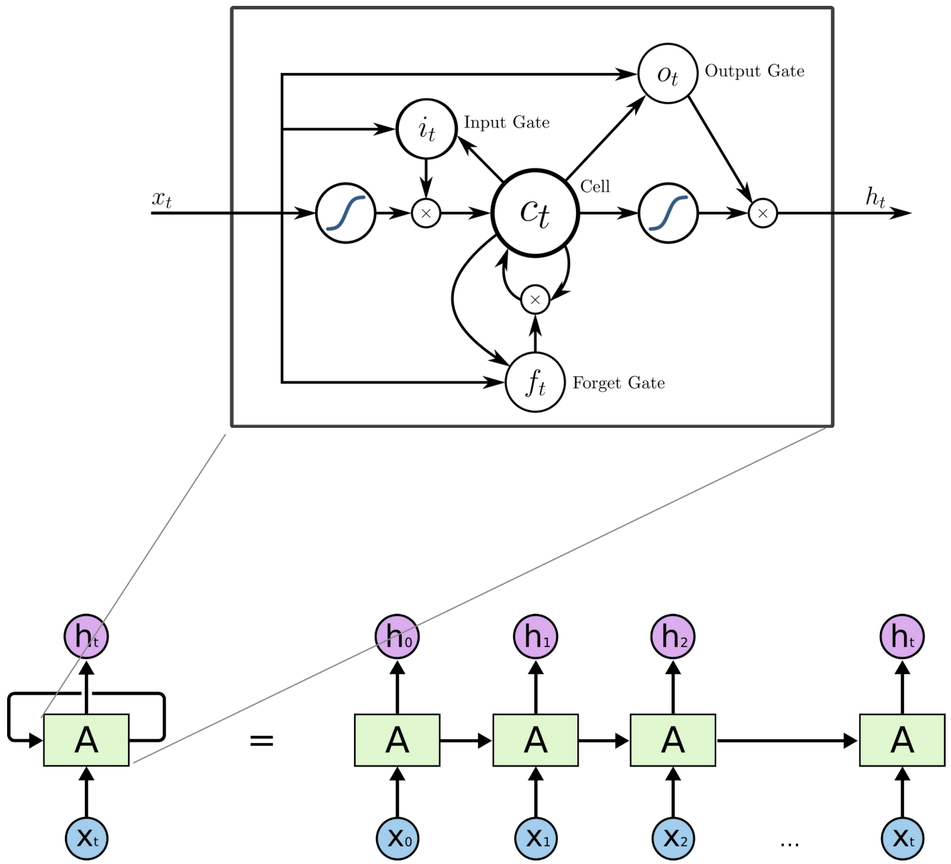
Long-Short Term Memory Models (LSTM)

Challenges with RNNs

- Struggle with long input sequences: Information from earlier words gets diluted as it propagates through the sequence.
- **Vanishing Gradient Problem**: Gradients shrink over time, reducing the ability to learn long-term dependencies.

Solution: LSTMs

- Introduce a "cell state" to selectively retain or discard information.
- Effectively capture long-range dependencies in sequences.



Limitations of RNNs and LSTMs in Language Modeling

Sequential Processing Bottleneck

- RNNs and LSTMs process inputs step-by-step, making training and inference slow for long sequences.

Vanishing Gradients

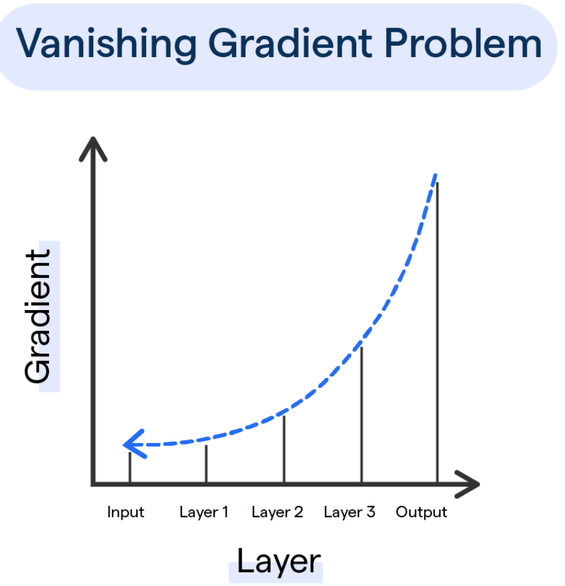
- Gradients diminish as they backpropagate through many time steps, limiting the network's ability to learn relationships across long sequences.

Challenges with Long-Range Dependencies

- Even with LSTMs, retaining information from distant parts of a sequence is difficult, leading to a loss of context over time.

Focus on Local Context

- RNNs and LSTMs prioritize immediate neighboring words but struggle to model relationships across the entire input effectively.



The solution?

A new mechanism that would enable parallel process, dynamically focusing on relevant sequence parts, and capturing long-range dependencies without the limitations of step-by-step computing or vanishing gradients…

Evolution of attention mechanisms pre-transformers

Adding Attention

What is Attention?

Attention is a mechanism that enables a model to focus on the most relevant parts of the input while making predictions.

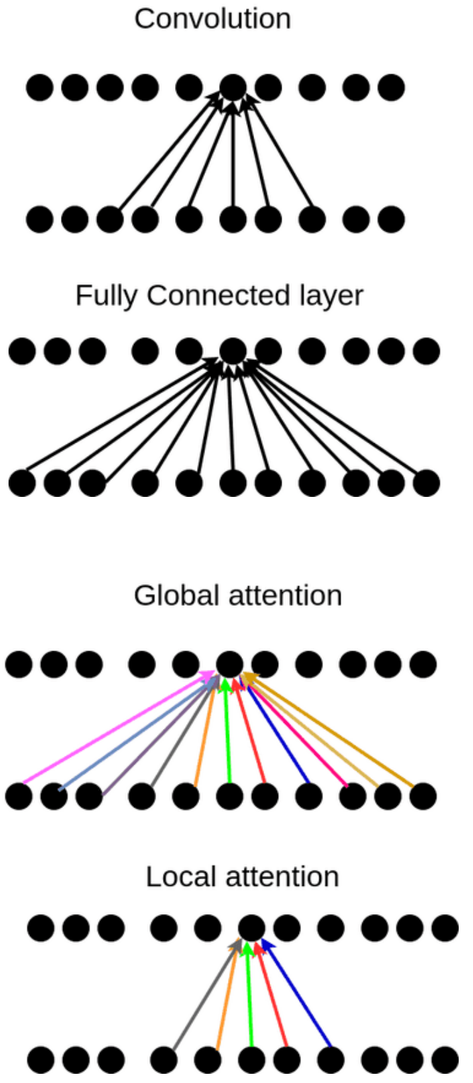
Instead of treating all input information equally, it assigns varying levels of "importance" (weights) to different parts based on the task.

<u>Key Idea</u>:

When processing sequences, attention computes a weighted sum of the input elements, where the weights represent how much "attention" each element deserves.

Different to other Layer types?

- *<u>Fully Connected</u>*: Listens to everyone equally, regardless of relevance.
- *<u>Convolution</u>*: Pays attention only to nearby neighbors.
- *<u>Attention</u>*: Actively decides who to listen to, whether they are nearby or far away.



Early Attention Attempts – Bahdanau et.al 2014

Bahdanau et.al introduced a mechanism to dynamically focus on specific parts of the input sequence while generating each element of the output sequence.

- The encoder produces a set of **context vectors** (hidden states) for each input token.
- For each output token, the decoder calculates an **attention score** for each input token based on its relevance to the current decoding step.
- The scores are normalized (using SoftMax) to produce attention weights, which are used to compute a **weighted sum of encoder hidden states** (context vector).
- This context vector is then used by the decoder to produce the next output token.

Core Formula

Attention weight ($\alpha_{t,i}$):

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})}$$

Where $e_{t,i} = \text{score}(s_t, h_i)$ is a learned scoring function (additive).

Context vector:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$$

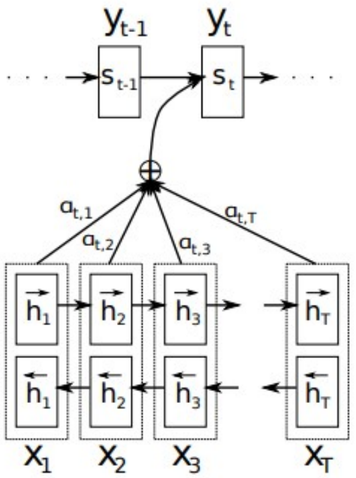


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Improved Attention – Luong et.al 2015

Luong et.al introduced in the paper *“Effective Approaches to Attention-based Neural Machine Translation”* by Minh-Thang Luong et al., this innovation focused on improving the computational efficiency and flexibility of attention mechanisms.

Multiplicative Attention (Dot-Product Scoring):

- Replaced Bahdanau's additive scoring function with a simpler **dot-product** or **scaled dot-product** function to calculate attention scores.
- Resulted in faster computations while maintaining strong performance.

Global vs. Local Attention:

Proposed two types of attention mechanisms:

1. **Global Attention**: Attends to all input tokens for each output token (like Bahdanau et.al).
2. **Local Attention**: Focuses on a subset of the input sequence for efficiency, using a predicted alignment window.

Improved Training Efficiency:

By simplifying the attention computation, Luong et.al allowed larger models and longer sequences to be trained more effectively.

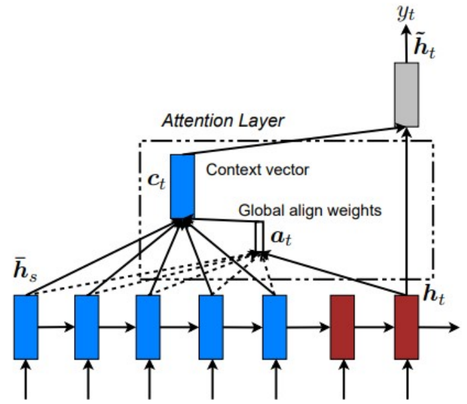


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector a_t based on the current target state h_t and all source states $\bar{h}_s$. A global context vector c_t is then computed as the weighted average, according to a_t, over all the source states.

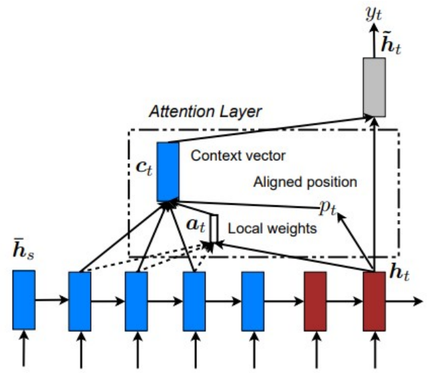


Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector c_t, a weighted average of the source hidden states in the window. The weights a_t are inferred from the current target state h_t and those source states $\bar{h}_s$ in the window.

Early Self-Attention – Lin et.al 2017

Lin et al. introduced a self-attention mechanism to create structured, multi-aspect sentence embeddings, enabling models to focus on different parts of a sentence simultaneously.

- **Self-Attention for Sentence Embeddings:**
Dynamically assigns attention weights to input tokens, emphasizing their importance to the sentence representation.

- **Multi-Aspect Representations:**
Captures diverse aspects of a sentence by generating **multiple attention vectors (hops)**, enabling richer embeddings.

- **Regularization for Diversity:**
Introduced a **penalty term** to ensure attention focuses on distinct sentence parts, reducing redundancy.

Key Formula for Attention:

Attention scores α_i:

$$\alpha_i = \text{softmax}(w_a^\top \tanh(W_h h_i))$$

Where:

- h_i: Hidden state of the word.
- w_a: Learnable weight vector.
- W_h: Learnable weight matrix.

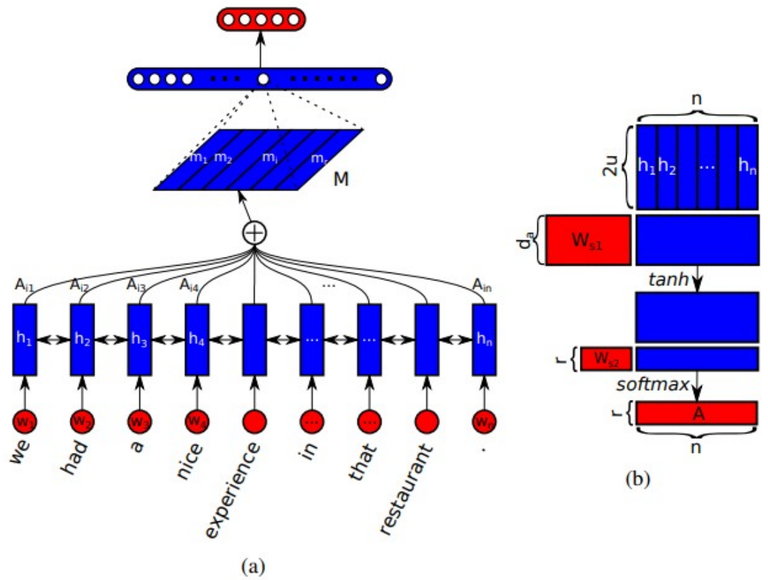


Figure 1: A sample model structure showing the sentence embedding model combined with a fully connected and softmax layer for sentiment analysis (a). The sentence embedding M is computed as multiple weighted sums of hidden states from a bidirectional LSTM ($\mathbf{h_1}, ..., \mathbf{h_n}$), where the summation weights ($A_{i1}, ..., A_{in}$) are computed in a way illustrated in (b). Blue colored shapes stand for hidden representations, and red colored shapes stand for weights, annotations, or input/output.

Remaining Limitations with Attention Methods

Despite the novel innovations of attention methods, these approaches still suffered from some general limitations, preventing their widespread use.

Dependence on Recurrence

- Attention mechanisms (e.g., Bahdanau, Luong, Lin et al.) were tightly integrated with RNNs/LSTMs, which process sequences sequentially.
- Gradient Issues: The reliance on recurrence also made them susceptible to vanishing or exploding gradients, limiting their ability to model very long dependencies.

Lack of Scalability

- RNN/LSTM-based models with attention were computationally expensive and struggled with large datasets or sequences.
- Memory Usage: Maintaining hidden states for long sequences was resource-intensive.

Inefficient Training

- Training LSTM-based models with attention was slow because of sequential dependencies and the need to process data step-by-step.

The Self-Attention Mechanism

Attention is all you need – Vaswani et.al 2017

One of the most seminal papers of machine learning to date. “Attention is all you need” introduced a new concept of how to make use of attention and completely removed the reliance on recurrence to process sequences. In the next lesson we will dive deeper into the Transformer model, but now we will focus on how self-attention is presented in that paper.

Self-Attention: What Does It Actually Mean?
Self-attention is a mechanism that allows a model to dynamically focus on the most relevant parts of a sequence when computing representations for each token.

Key Concept:
Every token in the sequence can *attend* to every other token, including **itself**, to understand its relationship and importance in the context of the entire sequence.

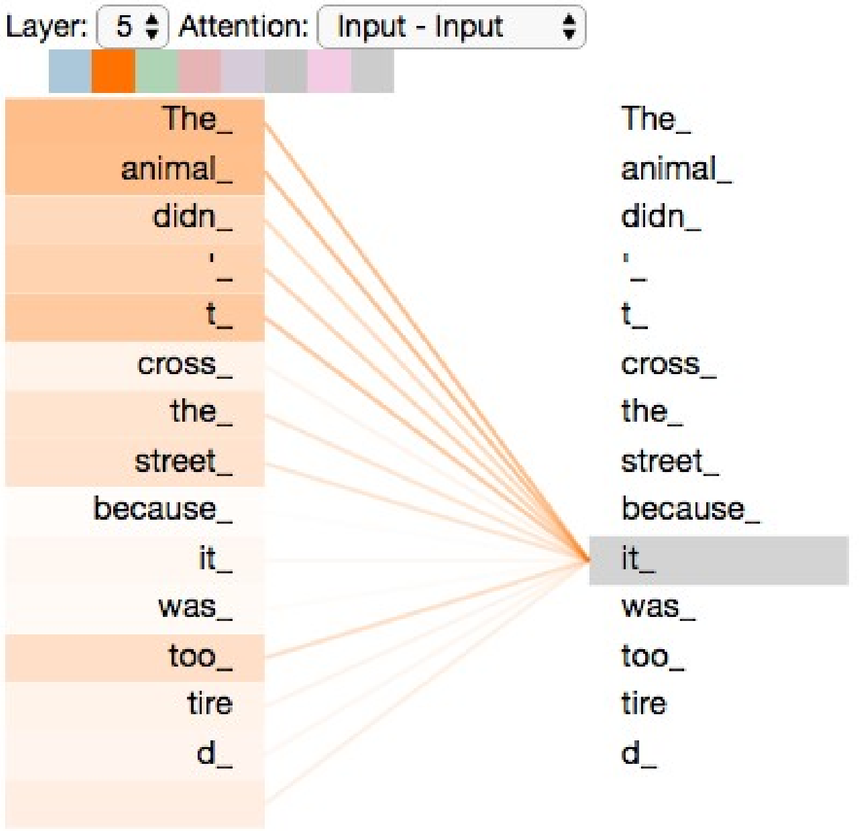
E.g.
In the sentence: "**The cat chased the mouse**,"

Self-attention helps the model understand that "*the mouse*" is what "*the cat*" **chased**, by focusing on the semantic relationship between "chased" and "the mouse."

How Self-Attention Mechanisms Work

The implementation of self-attention can vary, but the essence is to:

- **Compare:** Each token is compared to every other token in the sequence to compute relevance scores.
- **Weight:** Assign weights to other tokens based on their relevance to the current token.
- **Aggregate:** Combine information from all tokens, weighted by their relevance, to compute a new representation for the current token.



Global Context
Captures relationships between tokens across the entire sequence, not just local neighbors.

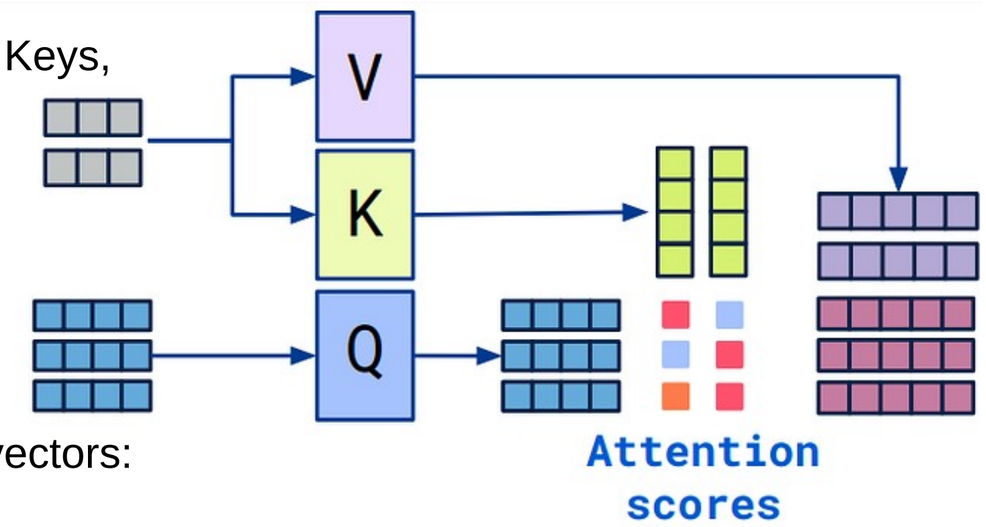
Dynamic Focus
The model decides what to focus on for each token, rather than relying on fixe patterns (e.g., sliding windows in convolution).

Flexibility
Works for variable-length sequences and tasks requiring both short- and long-range dependencies.

Queries, Key, and Values

In the Attention is all you need paper, a novel algorithm, based on Queries, Keys, and Values is presented to handle the attention calculations:
QKV allows each token to decide:

- What it wants to know (**Query**).
- What information it can provide (**Key**).
- The actual data it contributes to the result (**Value**).



Step 1: Create Q, K, V
Each input token (e.g., word embedding) is linearly transformed into three vectors: Query (Q), Key (K), and Value (V).

Step 2: Compute Attention Scores:
Compare the Query of a token with the Keys of all tokens in the sequence to measure relevance. Use the dot product to capture similarity.

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

Step 3: Normalize Scores (SoftMax):
Apply SoftMax to the attention scores to ensure they sum to 1, producing attention weights.

Step 4: Aggregate Values:
Multiply the attention weights by the corresponding Values and sum them to produce the output representation for each token.

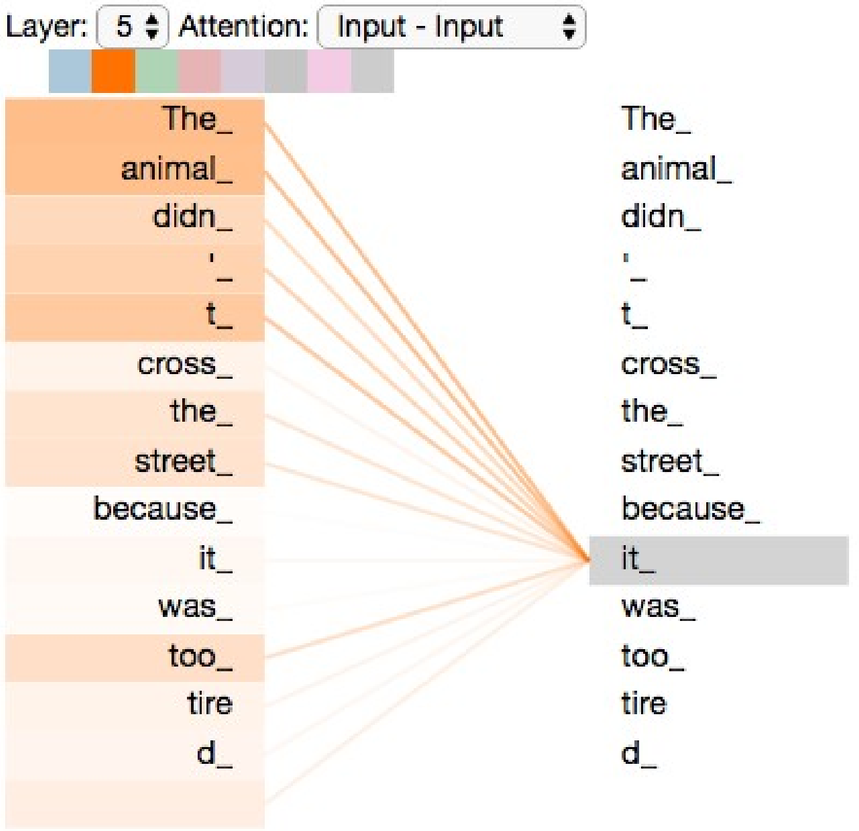
Wrap Up

Language Models and Attention

- Today we introduced the concepts of deep learning in the context of language models.
- We saw some of the older model architectures like recurrent neural networks that were used to model sequence data as an improvement over traditional feedforward neural networks.
- The concept of attention mechanisms was also discussed as a means to add extra information between parts of sequences
- As a precursor to the introduction of the full transformer model, we looked back at the evolution of attention models in deep learning.
- Finally, we presented the self-attention mechanism that provided the breakthrough needed for modern LLMs and Transformer-based models

---

In the next class we will explore the transformer model.



Thank you!