Summary of Deep Learning Lecture 6: Attention Mechanisms

Malak Yehia

1 Introduction

This lecture explores advanced models for sequence-to-sequence prediction, focusing on the evolution from RNNs to attention mechanisms, highlighting their significance in handling sequential data efficiently.

2 Limitations of RNNs

Recurrent Neural Networks (RNNs) face several challenges that can limit their performance and applicability:

- **Continuous Stream of Data**: RNNs struggle with processing continuous data streams because they require fixed-size input and output vectors.
- **Parallelization**: Due to their sequential nature, RNNs cannot fully utilize modern parallel computing resources, leading to slower training and inference times.
- **Long-Term Dependencies**: While theoretically capable of handling long-range dependencies, in practice, RNNs often fail to maintain long-term state information due to vanishing or exploding gradient problems during training.

3 Causal Convolutional Neural Networks (CNNs)

Causal CNNs address the limitations of RNNs in handling sequential data, offering a parallelizable and efficient alternative.

3.1 Causal CNNs Overview

Causal CNNs ensure that the output at any given time is influenced only by the present and past inputs, not the future. This property is crucial for tasks where future data points are unavailable or irrelevant.

3.2 How Causal CNNs Work

- **Kernel, Stride, and Dilation**: These networks utilize kernels to process data points within a sequence. A stride determines the step size of the kernel across the input, while dilation allows the network to cover a larger input area without

increasing the kernel size, improving the network's ability to capture long-range dependencies.

- **Causality**: Achieved by manipulating the padding to only consider past and present data, ensuring predictions do not depend on future input.
- **Receptive Field**: The portion of the input sequence that affects a particular output. Dilated convolutions expand the receptive field, enabling the model to consider longer sequences.

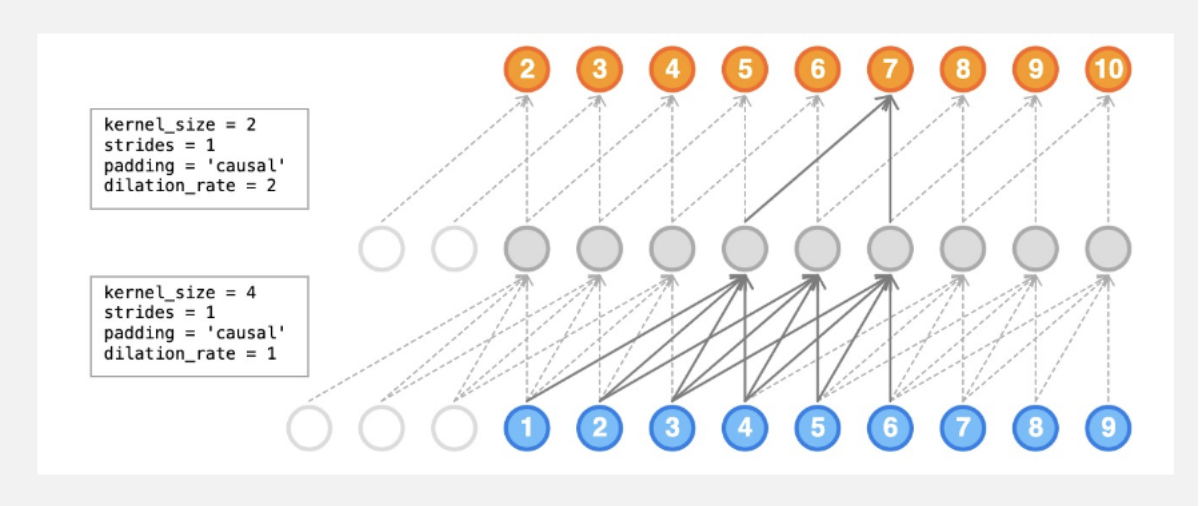

Figure 1: Causal CNNs with Different Kernel Sizes and Dilation Rates

3.3 Challenges and Limitations

- **Parameter Efficiency**: Large kernels increase the model's complexity and computational cost, making it impractical for long sequences.
- **Fixed Receptive Field**: The network's architecture must be defined in advance, limiting its flexibility in adapting to various sequence lengths.
- **Receptive Field Size**: The receptive field size can only be as deep as the depth of the network. Potential solutions:
  - Increase kernel size.
  - Pooling layers.
  - Dilated convolution
- **Sparsity**: Dilation introduces sparsity in the connections, potentially missing important intermediate inputs.

Causal CNNs represent a significant advancement in processing sequential data, offering solutions to some of the challenges faced by RNNs, despite having their own set of limitations.

4 Attention Mechanism Overview

Attention mechanisms allow models to focus on specific parts of the input sequence, improving the model's ability to remember long-term dependencies.

4.1 Self-Attention

Self-attention is a specific form of attention mechanism, it assigns weights to different parts of the input data. It involves transforming input sequences into Query (Q), Key (K), and Value (V) matrices to calculate attention scores.

4.2 The Math behind Self-Attention

Given Q, K and V matrices, we compute the matrix of outputs of self-attention by the dot-product attention, which is represented as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{KQ^T}{\sqrt{d_k}}\right)V \tag{1}$$

Where d_k represents the dimension of key vectors.

Let's break down the operation:

- How did we get Q, K, and V?
  These matrices are derived by multiplying the input X (word embeddings) with learned weight matrices W_q, W_k, and W_v, transforming the input X into three matrices:

$$Q = W_q X \tag{2}$$
$$K = W_k X \tag{3}$$
$$V = W_v X \tag{4}$$

- Similarity calculation $\frac{QK^T}{\sqrt{d_k}}$:

  The expanded form:

$$K \times Q^T = W_k \times X \times X^T \times W_q^T \tag{5}$$

  - W_k and W_q are weight matrices that transform X into the key and query spaces, respectively.
  - From 2 and 3 above, we get K, and Q.
  - The dot product KQ^T is then computed between the transformed query and key matrices.
  - Dividing the attention scores by the square root of the dimension of the key vector $\sqrt{d_k}$ is a normalization technique, and helps in controlling the magnitude of the attention scores and stabilizing the learning process.

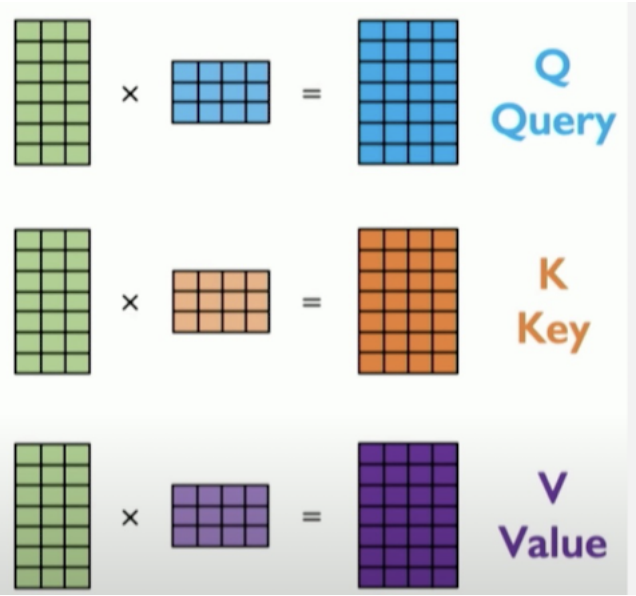

Figure 2: Q, K, and V

- Attention Weights: softmax $\left(\frac{QK^T}{\sqrt{d_k}}\right)$:

  The similarity scores go through softmax operation. Softmax normalizes the scores so they're all positive and add up to 1. Softmax scores determines how much each word will be expressed at this position, words with higher scores mean that they are most relevant words.

- Weighted Sum: softmax $\left(\frac{QK^T}{\sqrt{d_k}}\right)V$:

  After calculating the attention scores, we adjust each word's importance in the sequence by multiplying its values by the scores. This emphasizes relevant words while minimizing others. Summing these adjusted values gives us a new representation for each word, enriched by context from the whole sequence. This method ensures every word's output reflects not just its own meaning but also its relationship with every other word.

5 Properties of Self-Attention

- Equivariance: This means that if the input sequence's order changes, the output changes in the same way. It ensures the model's output accurately reflects alterations in the input sequence.

- Global Context: Self-attention allows the model to consider the entire input sequence simultaneously. This global perspective ensures each word's representation benefits from understanding its relationship with every other word, capturing long-distance dependencies effectively.

- Scalability: Thanks to parallelization, self-attention scales well with longer sequences. Unlike RNNs that process data sequentially, self-attention mechanisms

can compute attention for all elements in parallel, significantly reducing training times, and making it best for processing long sequences of data.

6 Compute Considerations

Each token in the input sequence attends to every other token, which means computing similarity scores between all pairs of tokens, requiring the calculation of dot products for the Q, K, and V matrices across the sequence.Which leads to a computational complexity of $O(T^2 \times d)$, where T is the sequence length and d is the dimension of the tokens.

7 Self-Attention Code Implementation

Self-Attention in PyTorch:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SelfAttention(nn.Module):
    def __init__(self, input_dim):
        super(SelfAttention, self).__init__()
        self.input_dim = input_dim
        self.query = nn.Linear(input_dim, input_dim)
        self.key = nn.Linear(input_dim, input_dim)
        self.value = nn.Linear(input_dim, input_dim)
        self.softmax = nn.Softmax(dim=2)

    def forward(self, x):
        queries = self.query(x)
        keys = self.key(x)
        values = self.value(x)
        scores = torch.bmm(queries, keys.transpose(1, 2)) / (self.input_dim ** 0.5)
        attention = self.softmax(scores)
        weighted = torch.bmm(attention, values)
        return weighted
```

Torch.bmm performs a batch matrix-matrix product of matrices

Figure 3: Implementation of Self-Attention in PyTorch

- `self.softmax = nn.Softmax(dim=2)`: Initializes the softmax function to be applied over the last dimension of the input, ensuring that softmax is applied to the feature or hidden unit dimension (dim=2) across each sequence position.
- `queries = self.query(x)`: Transforms the input x into a set of query matrices Q.
- `keys = self.key(x)`: Transforms the input x into a set of key matrices K.
- `values = self.value(x)`: Transforms the input x into a set of value matrices V.
- `scores = torch.bmm(queries, keys.transpose(1, 2))`: Computes the raw attention scores by performing a batch matrix-matrix product of queries and the transpose of keys.

- `scores = scores / (self.input_dim ** 0.5)`: Normalizes the attention scores by the square root of the input dimension (d_k).
- `attention = self.softmax(scores)`: Applies the softmax function to the normalized scores to obtain the attention weights.
- `weighted = torch.bmm(attention, values)`: Computes the weighted sum of the values using the attention weights, to get the final output of the self-attention layer.

8 The Bahdanau Attention Mechanism

The Bahdanau attention mechanism addresses the limitations of classical seq2seq models by selectively focusing on parts of the input sequence that are most relevant to the current prediction. This selective focus is used to dynamically update the decoder's state at each time step.

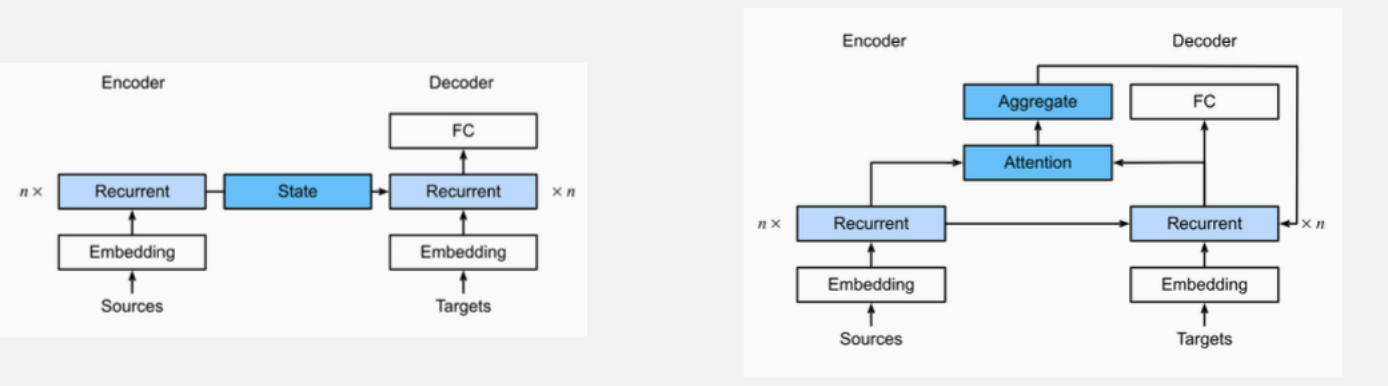

Figure 4: Comparison of Encoder-Decoder Architecture: Traditional vs. Bahdanau Attention Mechanism

8.1 Architecture Overview

- **Context Vector**: Denoted as C_t, it is calculated at each decoder time step to provide a snapshot of the input sequence, tailored for the current prediction.
- **Decoder Hidden State**: Represented by s_{t-1}, it is the state of the decoder at the previous time step, carrying information used to generate the next output token.
- **Encoder Annotations**: These are encoded information of the input sequence, which the attention mechanism uses to compute how much focus each word should get during the decoding.

The architecture involves a bidirectional RNN as an encoder and an RNN decoder, with the attention mechanism between them allowing the model to weigh the importance of each input token dynamically.

8.2 Computing the Context Vector

The context vector is a weighted sum of annotations, where the weights are determined by an alignment model scoring how well the inputs around position i and the output at position t match. This score is computed as follows:

$$e_{t,i} = \text{alignment}(s_{t-1}, h_i) \tag{6}$$

where $e_{t,i}$ is the attention score between the decoder state and each encoder hidden state.

Then we apply softmax function to each attention score to obtain the corresponding weight value:

$$\alpha_{t,i} \ = softmax(e_{t,i}) \tag{7}$$

The context vector C_t is then computed by:

$$C_t = \sum_{i=1}^{T} \alpha(s_{t-1}, h_i) h_i \tag{8}$$

$$C_t = \sum_{i} \alpha_{t,i} h_i \tag{9}$$

Bahdanau et al. tested their architecture on the task of English-to-French translation. They reported that their model significantly outperformed the conventional encoder-decoder model, regardless of the sentence length.

9 Multi-Head Attention

Instead of performing a single attention function, the multi-head attention projects the queries, keys, and values multiple times with different, learned linear projections. This allows the model to capture various aspects of the input data.

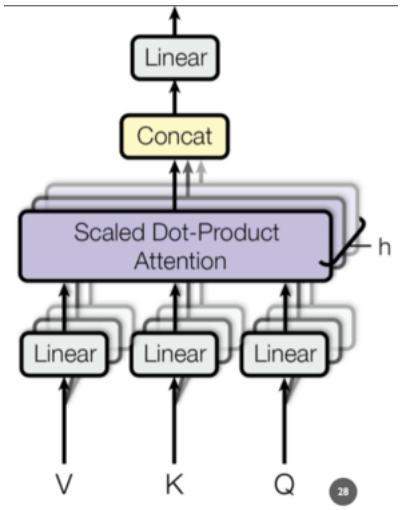

Figure 5: Multi-Head Attention Mechanism

- The model employs multiple attention "heads" — in the original paper, the authors used 8 heads ($h = 8$).

- With multi-headed attention, we maintain separate query, key and value weight matrices for each head.
- For each head, the input embedding X is multiplied by different sets of weight matrices W^Q, W^K, and W^V to produce separate Q, K, and V matrices.
- Since the subsequent feed-forward layer expects a single matrix, the h output matrices are concatenated and then linearly transformed using an additional weights matrix W^O, condensing them into a single output matrix.

10 Resources

- The original paper on the attention model: `https://arxiv.org/pdf/1706.03762.pdf`
- A video of causal CNNs `https://www.youtube.com/watch?v=rT77lBfAZm4&ab_channel=DLVU`
- An overview of the Bahdanau attention mechanism `https://machinelearningmastery.com/the-bahdanau-attention-mechanism/`