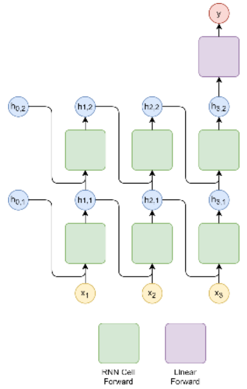
11-785 Spring 2020
Recitation 9

Attention Networks

Ken, Yuying

What we could do with RNN so far



- “Many-to-One” Architecture
  - (HW3P1) Sequence Classification: sequence -> label
- Streaming “Many-to-Many” Architecture
  - (HW3P2) speech frame sequence -> phoneme sequence
    - Order-correspondence
    - Each output corresponds to a small segment of input sequence

How about these tasks?

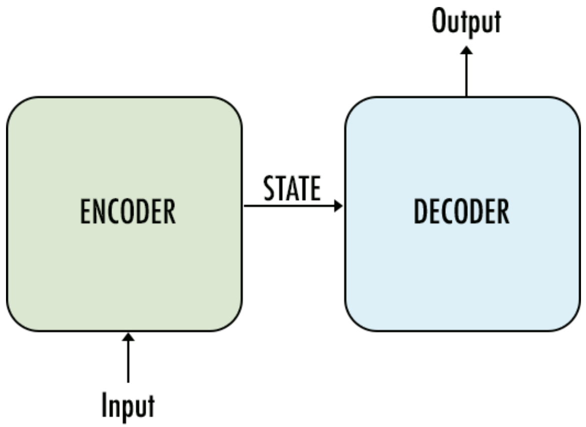
Text → Translation of that text

Image → A caption describing the image

Document, Question → Answer selected from the document

“Generative” Architecture

- The output sequence can only be built **after seeing the entire input sequence**
- The output is itself a sequence, **generated** from the input sequence



Decoder = Conditional Generator

In Math:

$$P(y_t \mid x_1, \ldots, x_T, y_1, \ldots, y_{t-1})$$

Each item in the output sequence must be conditioned on:

- The entire input sequence
- All the past output items

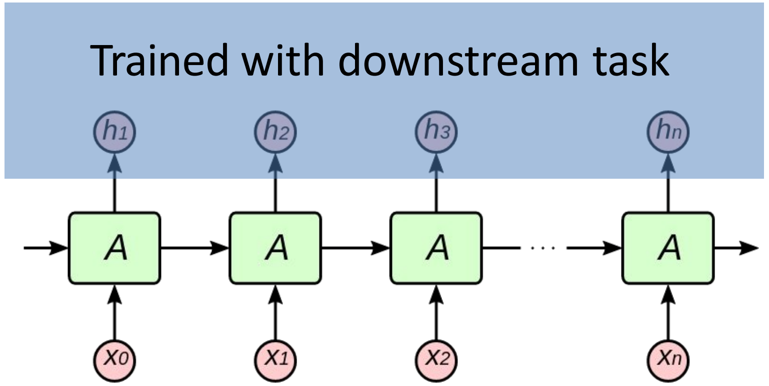
In Deep Learning:

The decoder must have access to:

- Some kind of encoding of the entire input sequence
- The past states of the decoder

How to encode the input sequence

Recall the “many-to-many” Architecture (HW3P2):



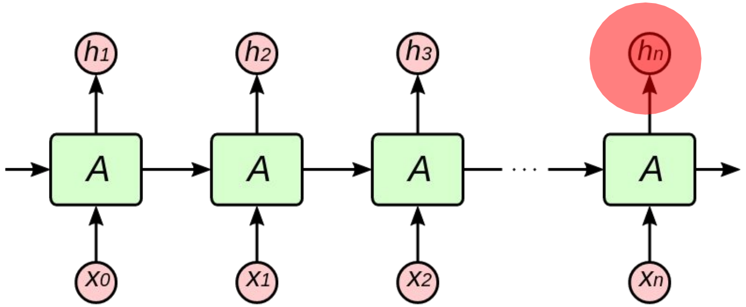
How to encode the input sequence

Now remove the output targets:

Hidden states only encode information about the history of inputs

Ideally the **last hidden state** is an encoding of the entire input sequence

Let's consider the last hidden state first



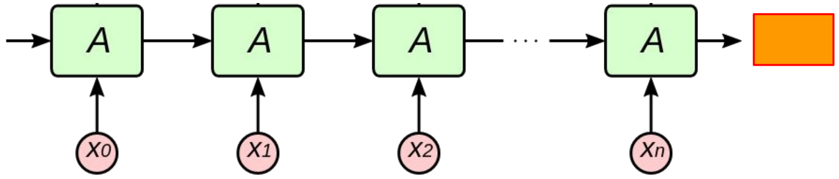
How to inform the decoder of the input encoding

- Pass the last hidden state of the input sequence at:
  - the first time step (to be forgotten later?)
  - every time step

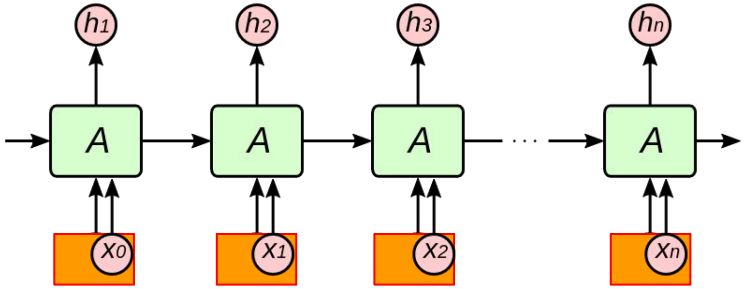
- Pass a more flexible input encoding at every time step
  - How flexible? Determined by the current decoder state

Network Prototype 1

Produce an encoding of the entire input.



Repeatedly pass the encoding to the output network.



Problems?

- Using one fixed vector to encode an entire sequence, hoping that the last hidden state could compress all the information
    - Hard to train. Input encoding vector is overloaded with information, and earlier inputs tends to get forgotten
    - Hard for the decoder to focus. Each time it's seeing the same thing

How to inform the decoder of the input encoding

- Pass the last hidden state of the input sequence at:
  - the first time step (to be forgotten later?)
  - every time step

- Pass a more flexible input encoding at every time step
  - How flexible? Decided by the current decoder state

Let the decoder decide the input encoding

Intuition:
At each time step, the decoder **focuses** on a specific segment of the input sequence to produce the current output

Formulation:

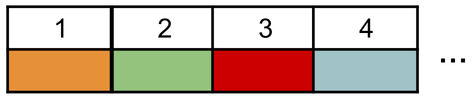
- Compute a time-varying input encoding that focuses on the part of input that matters to the current time step in the output
- Therefore, this input encoding should be a function of:
    - The decoder hidden state at the current time step
    - The encoder hidden states at each input time step

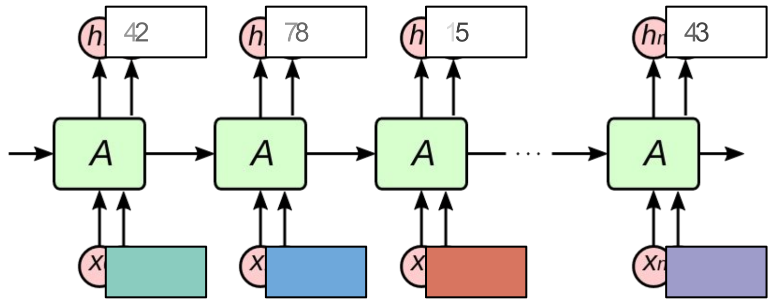
General Attention Mechanism

- Construct a **query** $\mathbf{q}_i$ from the decoder state $\mathbf{h}_i^{dec}$
  - Represents the decoder's interest
- Construct a **key** $\mathbf{k}_j$ from the encoder state $\mathbf{h}_j^{enc}$
- Calculate an **attention score** $\mathbf{att}(\mathbf{q}_i, \mathbf{k}_j)$
  - Tells how much at output time step i the decoder should focus on the j-th input item
- Construct a **value** $\mathbf{v}_j$ from the encoder state $\mathbf{h}_j^{enc}$
- Then construct the encoding by computing a weighted sum of values using attention scores as weights:
  - $\sum_{j=1}^{T} \mathbf{att}(\mathbf{q}_i, \mathbf{k}_j)\ \mathbf{v}_j$

Network Prototype 2

Encode each element of the input sequence into a vector.
For each time step, generate a query, compute an attention on this sequence.
Generate a linear combination of the input items using the computed attention values as weights.
Pass this combination to the output RNN.





Variation: Dot Product Attention

- **Query** $\mathbf{q}_i = \mathbf{h}_i^{dec}$
- **Key** $\mathbf{k}_j = \mathbf{h}_j^{enc}$
- **Value** $\mathbf{v}_j = \mathbf{h}_j^{enc}$
- **Attention score** att$(\mathbf{q}_i, \mathbf{k}_j)$ = softmax$(\mathbf{q}_i \cdot \mathbf{k}_j)$ (over all j)
  - Simplest similarity calculation (but works well in practice)
  - Does not introduce new parameters

Variation: Dot Product Attention

- **Query** $\mathbf{q}_i = \mathbf{h}_i^{dec}$
- **Key** $\mathbf{k}_j = \mathbf{h}_j^{enc}$
- **Value** $\mathbf{v}_j = \mathbf{h}_j^{enc}$
- **Attention score** att$(\mathbf{q}_i, \mathbf{k}_j)$ = softmax$(\mathbf{q}_i \cdot \mathbf{k}_j)$ (over all j)
  - Simplest similarity calculation (but works well in practice)
  - Does not introduce new parameters

Variation: Bilinear Attention

- **Query** $\mathbf{q}_i = \mathbf{h}_i^{dec}$
- **Key** $\mathbf{k}_j = \mathbf{h}_j^{enc}$
- **Value** $\mathbf{v}_j = \mathbf{h}_j^{enc}$
- **Attention score** $\text{att}(\mathbf{q}_i, \mathbf{k}_j) = \text{softmax}(\mathbf{q}_i^T \, W \, \mathbf{k}_j)$ (over all j)
  - Queries and keys do not have to be in the same space
  - Introduces new parameters

Variation: Additive Attention

- **Query** $\mathbf{q}_i = \mathbf{h}_i^{dec}$
- **Key** $\mathbf{k}_j = \mathbf{h}_j^{enc}$
- **Value** $\mathbf{v}_j = \mathbf{h}_j^{enc}$
- **Attention score** $\text{att}(\mathbf{q}_i, \mathbf{k}_j) = \text{softmax}(\mathbf{W}_a^T \tanh(\mathbf{W}_q \mathbf{q}_i + \mathbf{W}_k \mathbf{k}_j))$ (over all j)

Variation: Scaled Dot Product Attention

- **Query** $\mathbf{q}_i = \mathrm{MLP}_\mathrm{q}(\mathbf{h}_i^{dec})$
- **Key** $\mathbf{k}_j = \mathrm{MLP}_\mathrm{k}(\mathbf{h}_j^{enc})$
- **Value** $\mathbf{v}_j = \mathrm{MLP}_\mathrm{v}(\mathbf{h}_j^{enc})$
- **Attention score** $\mathrm{att}(\mathbf{q}_i, \mathbf{k}_j) = \mathrm{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{H}}\right)$ (over all j)
  - Use the dot product to calculate similarity for projected key value representation
  - Scaled by the sqrt of hidden size in order not to saturate the gradient of softmax

RNN-based Attention is great.. but

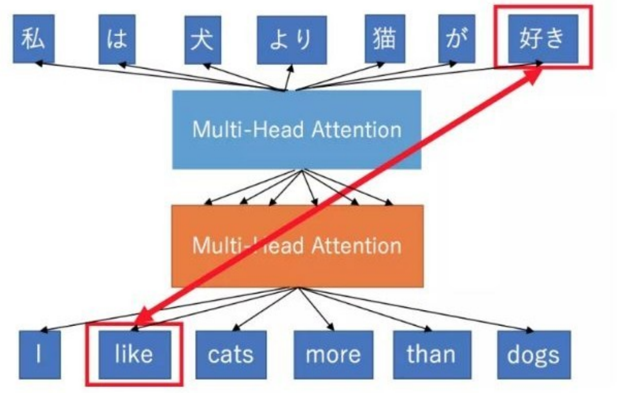
- Sequential nature of RNNs make them impossible to fully parallelize
  - Step-by-step computation relies on the output of the previous time-step
  - Cannot leverage those hard-core GPUs

- They struggle with long-term dependencies
  - What about LSTMs? Still can't hold information across very long sequences..
  - In NLP tasks, the same word can mean very different things based on context

Maybe Attention is All You Need

Transformer Nets

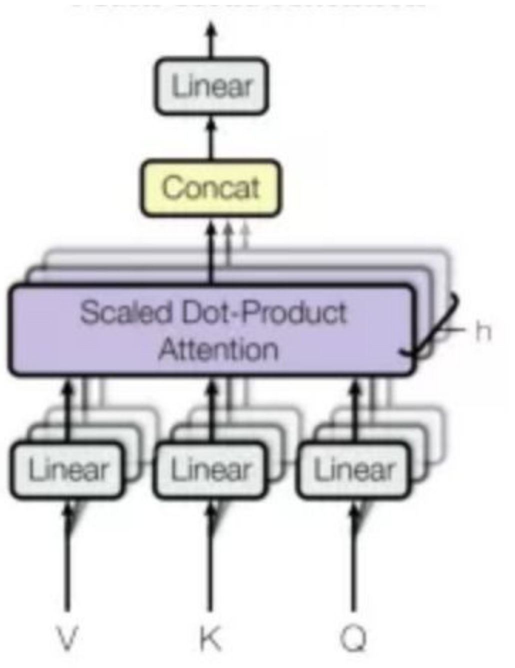
- Revolutionary machine-translation (**sequence to sequence**) architecture from Google
- Forget about RNNs, **capture dependencies across the sequences using attention**
- This lets the encoder and decoder see the entire sequence at once
- Also allows more parallelism than RNNs



The dependency that the Transformer has to learn. Now the path length is independent of the length of the source and target sentences.

Multi-Head Attention

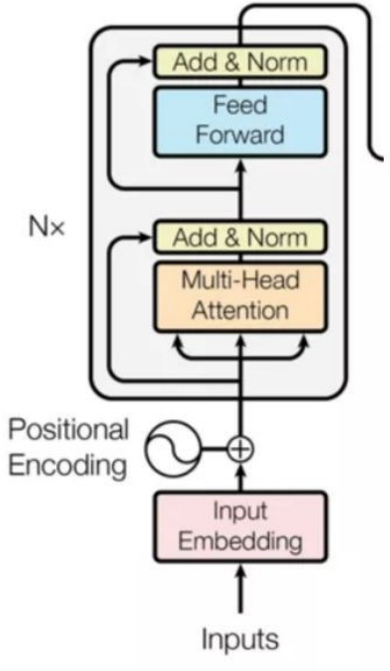
- Attention can be interpreted as a way of computing the relevance of a set of **values**, based on some **keys** and **queries**.
- Attention is applied multiple times to capture more complex input dependencies
- Each attention ‘head’ has unique weights
- Each ‘head’ can focus on different parts of the input sequence (and probably serves different purposes)



The Multi–Head Attention block

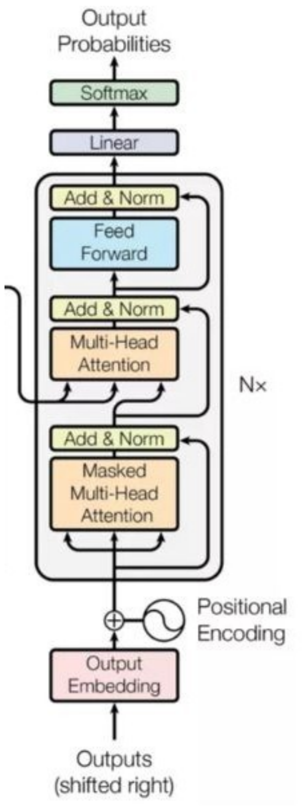
Encoder

- Contains multiple ‘blocks’ (~6 blocks)
- Residual connections between the multi-head attention blocks
- **Positional encodings explicitly encode the relative and absolute positions of the inputs as vectors**
- These encodings are then added to the input embeddings
- Without them the output for *“I like 11-785 more than 10-707”* would be identical to the output for *“I like 10-707 more than 11-785”*



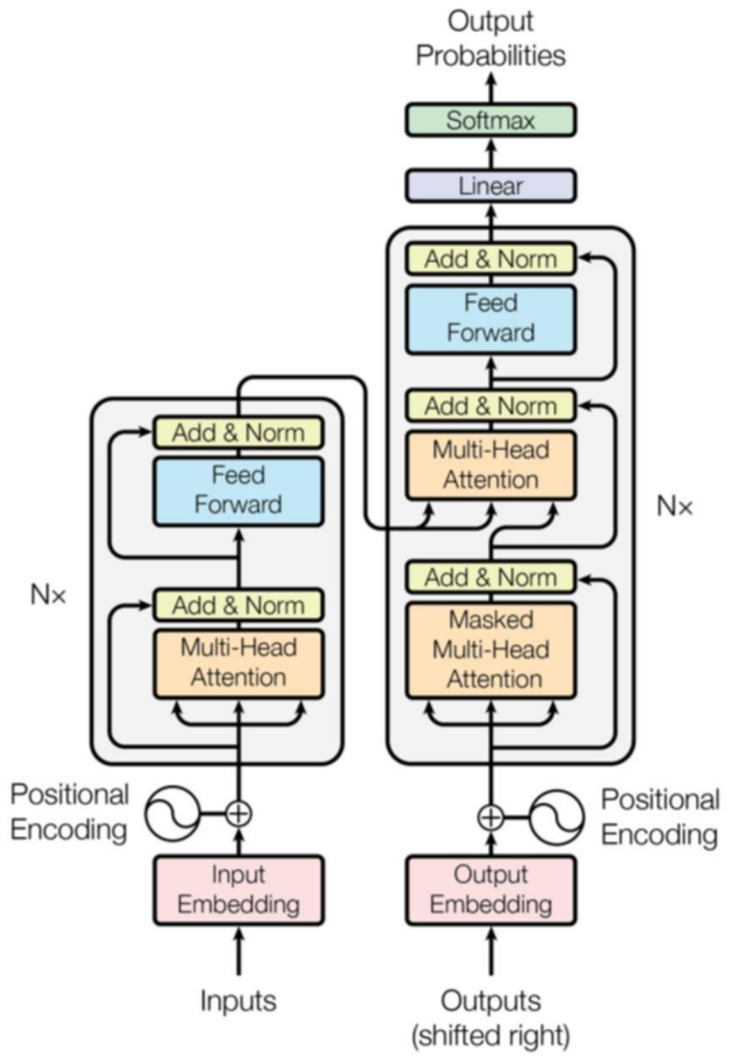
Decoder

- Very similar to the encoder
- ‘Masked’ Multi-Head Attention block to hide future output values during training
- The query from the decoder is used with the keys/values from the encoder
- Final output probabilities are computed using a projection layer followed by a softmax



Big Picture

- Input sequence is used to compute the **keys** and **values** in the encoder
- Masked-attention blocks in the decoder transform the output sequence until the current time-step into the **queries**
- Multi-head attention in the decoder combines the keys, queries and values
- The result is projected into output probabilities for the current time step



More detailed explanation