Attention and Transformers
Winter 2021
Previously: Sequence-to-Sequence

- **Sequence-to-sequence:**
  - Recurrent neural network for sequences of different lengths.

- **Problem:**
  - All “encoding” information must be summarized by last state \( (z_3) \).
  - Might “forget” earlier parts of sentence.
    - Or middle of sentence if using bi-directional RNN.
  - Might want to “re-focus” on parts of input, depending on decoder state.

Attention

- Attention for language translation:
Attention

- Many recent systems use “attention” to focus on parts of input.
  - Including “neural machine translation” system of Google Translate.

- Many variations, but usually include the following:
  - Each decoding can use hidden state from each encoding step.
    - Used to re-weight during decoding to emphasize important parts.

RNN vs. RNN with Attention Videos

Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION

Je suis étudiant
Not-Very-Practical Attention

- A naïve “attention” method (no one uses this, but idea is similar):
  - At each decoding step, weight decoder state (as usual) and weight all encoder states.

- Another variation on the “residual connection” or “denseNet” trick.
- But this variant is **not practical** since number of decoding weights depends on input size.
  - Practical variations try to summarize encoder information through a “context vector”.

Context Vectors

• A common way to generate the context vector:
  – Take current decoder state.
  – Compute inner product with each encoder state.
    • Gives a scalar for each encoding “time”.
  – Pass these scalars through the softmax function.
    • Gives a probability for each time (what was previously shown in pairwise table).
  – Multiply each encoder state by probability, add them up.
    • Gives fixed-length “context vector”.

• Alternate notation:
  – Input is “queries” and “keys”.
  – Output is “values”.

Context Vectors

- A common way to generate the **context vector**:
  - Take current decoder state.
  - Compute **inner product with each encoder state**.
    - Gives a scalar for each encoding “time”.
  - Pass these scalars through the **softmax function**.
    - Gives a probability for each time (what was previously shown in pairwise table).
  - Multiply each encoder state by probability, add them up.
    - Gives **fixed-length “context vector”**.

- **Alternate notation**:
  - Input is “queries” and “keys”.
  - Output is “values”.

---

Using Context Vectors for Attention

• Context vector is usually appended to decoder’s state.
  – Output could be generated directly from this, or passed through a neural net.
  – Common variation is “multi-headed attention”: can get scores from different aspects.
    • Semantics, grammar, tense, and so on.
    • Each is appended to decoder state.

• Remember that this is being trained end-to-end.
Using Context Vectors for Attention

- Context vector is usually appended to decoder’s state.
  - Output could be generated directly from this, or passed through a neural net.
  - Common variation is “multi-headed attention”: can get scores from different aspects.
    - Semantics, grammar, tense, and so on.
    - Each is appended to decoder state.

- Remember that this is being trained end-to-end.

Attention

• Attention for image captioning:

Figure 3. Examples of attending to the correct object (white indicates the attended regions, underlines indicated the corresponding word)

A woman is throwing a frisbee in a park.
A dog is standing on a hardwood floor.
A stop sign is on a road with a mountain in the background.
A little girl sitting on a bed with a teddy bear.
A group of people sitting on a boat in the water.
A giraffe standing in a forest with trees in the background.

Convolutions for Sequences?

• Should we really be going through a sequence **sequentially**?
  – What if stuff in the middle is really important, and changes meaning?

• Recent works have explored using **convolutions** for sequences.

https://medium.com/@TalPerry/convolutional-methods-for-text-d5260fd5675f
Digression: Dilated Convolutions (“a trous”)

• Best CNN systems have gradually reduced convolutions sizes.
  — Many modern architectures use 3x3 convolutions, far fewer parameters.
• Sequences of convolutions take into account larger neighbourhood.
  — 3x3 convolution followed by another gives a 5x5 neighbourhood.
  — But need many layers to cover a large area.
• Alternative recent strategy is dilated convolutions (“a trous”).

• Not the same as “stride” in a CNN:
  — Doing a 3x3 convolution at all locations, but using pixels that are not adjacent.
  — During upsampling, you can use interpolation to fill the holes.

https://medium.com/@TalPerry/convolutional-methods-for-text-d52601d5675f
https://github.com/vdumoulin/conv_arithmetic
Dilated Convolutions ("a trous")

- Modeling music and language and with dilated convolutions:

Figure 1. The architecture of the ByteNet. The target decoder (blue) is stacked on top of the source encoder (red). The decoder generates the variable-length target sequence using dynamic unfolding.
RNNs/CNNs for Music and Dance

• Music generation:
  – https://www.youtube.com/watch?v=RaO4HpM07hE

• Text to speech and music waveform generation:

• Dance choreography:

• Music composition:
  – https://www.facebook.com/yann.lecun/videos/10154941390687143
Transformer Networks

- CNNs are less sequential, but take **multiple steps to combine distant information**.

- “Attention is all you need”: keep the attention, ditch the RNN/CNN.
  - Constant time to transfer across positions.
  - Uses **self-attention** layers to model relationship between all words in input.
    - Queries/keys/values all come from input in these steps.

- Sequence of representations of words, each depending on all other words.

All words attend to all words in previous layer; most arrows here are omitted

http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture09-transformers.pdf
Transformer Networks

- CNNs are less sequential, but take multiple steps to combine distant information.

- “Attention is all you need”: keep the attention, ditch the RNN/CNN.
  - Constant time to transfer across positions.
  - Uses “self-attention” layers to model relationship between all words in input.
    - Queries/keys/values all come from input in these steps.

- Sequence of representations of words, each depending on all other words.

https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a
Transformer Networks

- Multiple “self-attention” layers in transformers replace RNN/CNN.
  - Has improved on state of the art results in many tasks.

Transformer Networks: Practical Issues

• “Self-attention” layers are basis for transformer networks.
  – Simple idea, but practical systems have a lot of moving pieces.

• Problem: position information is lost (self-attention is unordered).
  – “Position representations” are additional variables added to each layer.

• Problem: information about the future can be visible in the past.
  – During training, prevent decoder from looking ahead.

• Further “standard” tricks to make it work better:
  – Multi-headed attention, residual connections, and layer normalization.
  – Between layers, pass each embedding through a feedforward neural network.

http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture09-transformers.pdf
Transformer Architecture (from paper)

Figure 1: The Transformer - model architecture.

Subsequent Work

• BERT: incredibly-popular model in natural language processing.
  – Transformer model trained on masked sentences to predict masked words.
  – Then fine-tune the architecture on specific applications.

• Transformers also form basis for other advanced language models (GPT).
• Transformers have been adapted to images, music, and so on.
Neural Turing/Programmers

• Many interesting recent variations on memory/attention.
  – A good place to start is here: https://distill.pub/2016/augmented-rnns
Summary

• **Attention:**
  – Allow decoder to look at previous states.

• **Context vectors:**
  – Combine previous states into a fixed-length vector.

• **[Dilated] convolutions for sequences.**
  – Alternative to sequential architectures like RNNs.

• **Transformer networks:**
  – Layers of “self-attention” to build context.
    • “Everything depends on everything”, and you learn how.
    • Lots of implementation details, but excellent performance on many tasks.