Well no, not these kind of transformers...

Transformers

Professor Marie Roch

for details on transformers, see chapter 10 in draft:
Transformers

• Architecture which processes blocks of inputs:
  $$[x_1, x_2, ..., x_n] \rightarrow [y_1, y_2, ..., y_n]$$

• Each mapping, $$x_i \rightarrow y_i$$, is
  • computed independently of $$x_j \rightarrow y_j$$ ($$j \neq i$$)
  • has access to other inputs $$x_j$$
  • is said to be causal if $$x_i \rightarrow y_i$$ only has access to inputs $$x_j: j \leq i$$
    (non-causal if we can access future inputs)

• The access to other inputs is used in a self-attention mechanism
  (self-attention, as the attention is within the current block)
Causal self-attention layer

$x_4 \rightarrow y_4$ can attend to the values of $x_1, x_2, x_3$

J&M 2023, Fig 10.1
How do we attend to other inputs?

• Simplest attention mechanism is the dot product
  • $\text{score}(x_i, x_j) = x_i \cdot x_j$
  • remember: $x_i \cdot x_j = |x_i||x_j|\cos(\theta)$ where $\theta = \angle x_i x_j$,
    $\therefore$ larger score implies vector similarity

• We can transform the scores to a distribution
  \[ \alpha_{i,j} = \text{softmax}\left(\text{score}(x_i, x_j)\right) \]

• Output in this simple mechanism (we will do more)
  \[ y_i = \sum_{j=1}^{N} \alpha_{i,j} x_j \] or if causal \[ y_i = \sum_{j=1}^{i} \alpha_{i,j} x_j \]
Key ideas so far

• For each input, we estimate a distribution indicating the relevance of neighboring inputs including the current one.

• The output is a linear combination of the inputs scaled by their importance.

• Attention can be used anywhere in the network, so the inputs are likely to be some type of feature representation.

• From here on, we will use causal or non-causal examples with the understanding that the other can be easily derived.
Building on this idea

Input $x_1, x_2, \ldots, x_N$ will play a variety of roles in the prediction of $y_i$

- **Value $x_{1 \leq j \leq N}$**: will weight these values to compute $y_i$.
- **Query and Key provide the importance of each Value**
  - **Query $x_i$**: focus of attention mechanism.
  - **Key $x_{1 \leq j \leq N}$**: vector to which we compare the query

$$y_i = \sum_{j=1}^{N} \text{softmax}(x_i \cdot x_j)x_j$$

Note: We are showing roles here, there is one more step before we compute actual key, query, value
Query, key, and value

- Weight vectors learn to appropriately transform inputs for their role
  - \(q_i, k_i, v_i: \ q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i\)
  - Learned by standard backpropagation

- Dimension of W matrices
  - For now, we define \(d_k = d_v = d\) each \(W\) is \(d \times d\) where \(d\) is the dimension of the observation \(x_i:\)
    \[ (d \times d)(d \times 1) \rightarrow (d \times 1) \]
  - Later, we will introduce multi-headed attention
    - Will let us learn multiple attention representations.
    - Will permit query & key vectors of length \(d_k\) and value of length \(d_v\) where \(d_k \neq d_v \neq d\)
Query, key, and value

• Scores can now be thought of as the dot product of a query and key:
  \[ \alpha_{i,j} = \text{score}(x_i, x_j) = (W^Q x_i) \cdot (W^K x_j) = q_i \cdot k_j \]
  as these can be quite large, we normalize by the input dimension
  \[ \alpha_{i,j} = \text{score}(x_i, x_j) = \text{softmax} \left( \frac{q_i \cdot k_j}{\sqrt{d_k}} \right) \]

• Output
  \[ y_i = \sum_{j=1}^{N} \alpha_{i,j} v_j \]
Self attention example

Computing $x_3 \rightarrow y_3$
Efficiency

• We can take advantage of highly optimized parallel matrix libraries

• Pack all $x_i$ into $N \times d$ matrix $X$.

• Three multiplications resulting in $N \times d_k$ matrices:

\[
Q = XW^Q \\
K = XW^K \\
V = XW^V
\]
Efficiency

• The score required many multiplications between queries and keys

• We can do this once: $QK^T$

• Matrix on the right show sample $QK^T$
  • evident that attention is quadratic with respect to input length
  • causal transformer example
    • upper triangle set to $-\infty$ in postprocessing
    • why?

\[
\begin{array}{cccc}
q1 \cdot k1 & \infty & \infty & \infty & \infty \\
q2 \cdot k1 & q2 \cdot k2 & \infty & \infty & \infty \\
q3 \cdot k1 & q3 \cdot k2 & q3 \cdot k3 & \infty & \infty \\
q4 \cdot k1 & q4 \cdot k2 & q4 \cdot k3 & q4 \cdot k4 & \infty \\
q5 \cdot k1 & q5 \cdot k2 & q5 \cdot k3 & q5 \cdot k4 & q5 \cdot k5 \\
\end{array}
\]
The transformer block
Residual layer (He et al. 2016, CVPR)

- He et. al. asked: Are deeper networks better?
  - normalization of starting values and intermediate layers (e.g. batchnorm) helps with vanishing/exploding gradients, yet ...
  - deeper networks can start to converge and then saturate or degrade
- Insight: provide skip connections that carry the input forward along with what we learn each layer
Residual Layer (ResNet)

- Introduces “skip” layer
- Feeds information forward to deeper layers
- Layer learns $H(X) - x$
- Operation need not be $wx$ (e.g., convolution)

Classic ResNet block (He et al. 2016)
ResNet helps learn deeper networks

Training (thin lines) and validation (thick lines) curves for CNN (left) vs CNN ResNet (right) on ImageNet training data with 18 and 34 layers.
Layer normalization

• Similar to Z-score normalization
  • Remember, if $x \sim n(\mu, \sigma^2)$, then $\frac{x - \mu}{\sigma} \sim n(0,1)$
  • adds learnable gain and offset
  • Given input vector $x$, we compute mean $\mu$ and standard deviation $\sigma$.
    $$\hat{x} = \frac{x - \mu}{\sigma}$$

• Layer normalization computes $\gamma \hat{x} + \beta$ where $\gamma$ and $\beta$ are learnable.

These ideas are similar to what we have seen with batch normalization layers
The transformer block

\[
y_n = \text{LayerNorm}(z + \text{feedforward}(z))
\]

\[
z = \text{LayerNorm}(x + \text{SelfAttention}(x))
\]
Multihead attention
or multiple heads are better than one...

• The query and key matrices learn a specific type of relationship.

• It might be the case that there is more than one type of relationship to be learned...

• Let $h = \#$ heads. Learn $h$ query, key, and value transforms
  • $W^Q_1, W^Q_2, \ldots, W^Q_h$
  • $W^K_1, W^K_2, \ldots, W^K_h$
  • $W^V_1, W^V_2, \ldots, W^V_h$

• We also relax the requirement that weight layers be square
  • $W^Q_i, W^K_i$ can be $d \times d_k$
  • $W^V_i$ can be $d \times d_v$
Multihead attention layer

• Each head produces a vector of length $d_v$.

• When we concatenate the outputs of the N heads, we end up with a vector of size $1 \times h d_v$: $[o_{h_1} \oplus o_{h_2} \oplus o_{h_3} \oplus ... \oplus o_{h_N}]$

• Project down to $1 \times d$
  • Learn a weight matrix $w^O$ of size $h d_v \times d$
  • So $[o_{h_1} \oplus o_{h_2} \oplus o_{h_3} \oplus ... \oplus o_{h_N}]w^O$ is size: $(1 \times h d_v)(h d_v \times d) \rightarrow 1 \times d$. 
Transformer Block

Layer Normalize
Feedforward Layer
Layer Normalize
MultiHead Attention Layer

Residual connection
Residual connection

\[ y_n \]

\[ x_1 \ x_2 \ x_3 \ \ldots \ x_n \]

Adapted from J&M 10.4
Fruit for thought

Suppose we predict $x_3$ with the following two sequences:

Is one sequence likely to be more likely than the other?

sequence cherry
sequence orange

Adapted from J&M 10.4
Positional encodings

• Supplement the input vector with something that lets us learn relative position

• Active area of research

• Common simple method: Use sinusoids of varying frequency

Keras ≥2.9

• Module `keras.layers.attention` contains attention layers
  • `attention` – classic dot-product single-headed attention
  • `multi_head_attention` – Multi-head attention
    The module is flexible and can be used in ways that we have not talked about.

• Complete example of transformer-based speech recognizer at: https://keras.io/examples/audio/transformer_asr/