ATTENTION IN NLP AND CV

COLIN NI

Abstract. Attention is an important concept in modern ML. The Transformer architecture makes use of multi-head attention layers, and in NLP pre-trained Transformer models like BERT and GPT have taken over the SOTA. In CV, researchers have long effectively applied attention in parallel with convolutions to core problems, but recently researchers even have found success with purely attention-based Transformer models.

Contents

1. Review of NLP 1
2. Review of LSTM’s 2
3. Attention function 3
4. Positional embeddings 3
5. The Transformer architecture 4
6. BERT and GPT 6
7. Attention in CV 8
8. Attention versus convolutions 10
9. Transformers in CV 11
Acknowledgments 15

1. Review of NLP

One fundamental tool in NLP is sensibly representing words as vectors. Let us describe one way of doing this for concreteness.

Technique. Given a large corpus of text containing \( w \) different words, one-hot encode each occurring word in \( \mathbb{R}^w \). Then train a shallow two-layer neural net

\[
\mathbb{R}^{kw} \xrightarrow{\text{linear}} \mathbb{R}^h \xrightarrow{\text{linear}} \mathbb{R}^w \xrightarrow{\text{softmax}} \mathbb{R}^w
\]

with a hidden layer of size \( h \) to predict any word in the corpus given \( k \) of its surrounding words. To obtain the embedding of a word, first embed into the first \( k \) coordinates of the input layer, then take the hidden features: \( \mathbb{R}^k \xrightarrow{} \mathbb{R}^{kw} \xrightarrow{} \mathbb{R}^h \).

This context-based method yields intuitive results; things like

\[ \text{king} - \text{male} + \text{female} = \text{queen} \]

tend to hold when activations are normalized correctly.

Common tasks in NLP such as classification, generating and summarizing content, and answering questions can be viewed as taking as input a sequence of words and outputting a sequence of words. Models which perform these tasks are thus
called *seq2seq* models. Typically they involve an encoder which understands the input and a decoder which determines the answer from the encoding. For example, for a question-answer task, the model may look like this:

```
embedded input sequence
↓encoder
encoding
↓decoder
embedded output sequence
↓token-wise argmax
output sequence.
```

A core challenge in NLP, less so in CV, is modeling long-range dependencies. Understanding text requires recalling specific far-away details, such as in the following example:

**Example 1.1.** In the following sentence, in order to understand the word *it*, we must pay attention to the words *bunny*, *river*, and *wide*:

```
The bunny couldn’t cross the river because it was too wide.
```

2. **Review of LSTM’s**

LSTM’s (and RNN’s) approach the problem of tracking long-range dependencies using recurrence and memory. An LSTM maintains a cell state $c_t$ through time, at each step outputting a value $h_t$ and using $h_{t-1}$ and the current input $x_t$ to update $c_{t-1}$. To do this, the LSTM learns four linear maps which do the following things:

- forget information in the cell state: $c_{t-1} \leftarrow c_{t-1} \odot \sigma(\text{linear}[x_t, h_{t-1}])$
- determine what new information is relevant: $i_t \leftarrow \sigma(\text{linear}[x_t, h_{t-1}])$
- update the cell state: $c_t \leftarrow c_{t-1} + i_t \odot \text{tanh}(\text{linear}[x_t, h_{t-1}])$
- output a value: $h_t \leftarrow \sigma(\text{linear}[x_t, h_{t-1}]) \odot \text{tanh}(c_t)$

Diagramatically:

![LSTM Diagram](image)

**Figure 1.** An LSTM cell as illustrated on Wikipedia. Four trained networks regulate flow of information in and out of the cell state.
There are several problems with LSTM’s (and RNN’s in general):
- exploding gradients
- vanishing gradients, despite this being the problem they aimed to solve
- not parallelizable due to its sequential nature, hence slow to train in practice
- (uni-)directional.

3. Attention function

The language of queries, keys, and values is often used when formally defining attention.

**Intuition and Nomenclature.** Suppose we want to query a database for values that have keys which are similar to our query. In SQL pseudocode, this may look like

```sql
SELECT values
FROM database
WHERE key SIMILAR TO query
```

In the context of attention, the queries \( Q = (q_i)_i \), keys \( K = (k_i)_i \), and values \( V = (v_i)_i \) are all vectors in \( \mathbb{R}^d \). Thus a first approach would be to return a sum of all the values weighted by dot-product similarity:

\[
QK^T V = \sum_i (q_i k_i^T) v_i.
\]

Improving on this, we softmax the dot products to normalize the output but scale them beforehand by

\[
\sqrt{d} = \text{Var} \left( \prod_i \mathcal{N}(0,1) \right)
\]

to account for variance as \( d \) increases.

**Definition.** The (scaled dot-product) attention function takes

\[
(Q, K, V) \mapsto \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right) V.
\]

4. Positional embeddings

In practice attention is applied to spatial data, for example things like pixels in feature maps and tokens in a sequence of word embeddings, but the attention function is insensitive to spatial structure because it is invariant up to permutation of rows. Thus in practice we must encode any spatial information of our data into the data itself or give our models room to learn this information themselves.

For concreteness, we will describe one encoding method in the fundamental setting of 1D sequences, as explained in [Attention is All You Need](#).
Figure 2. A plot of our positional embedding for $n = 50$ and $d = 128$ [cf Definition 4.1]. These values are incorporated into input data to add spatial information.

Let $\mathbf{x} = (x_p)_{p=1}^n \subset \mathbb{R}^d$ be a 1D sequence of length $n$. We will construct a sequence $\mathbf{e} = (e_p)_{p=1}^n \subset \mathbb{R}^d$ that encodes the positional information of $\mathbf{x}$ via the sum $\mathbf{x} + \mathbf{e}$.

**Definition 4.1.** For $i = 1, \ldots, d$ and $p = 1, \ldots, n$, set

$$\omega_i = 10000^{-\frac{2i}{d}},$$

and define

$$e_{pi} = \begin{cases} 
\cos(\omega_ip) & i \text{ even} \\
\sin(\omega_ip) & i \text{ odd}.
\end{cases}$$

Intuitively, we imagine a one-handed clock for every pair of dimensions with hand-speed geometrically increasing from $(1 \cdot 2\pi)^{-1}$ radians per position to $(10000 \cdot 2\pi)^{-1}$, and as we traverse the positions in the sequence, the clock hands rotate according to their hand-speeds. The slower hand speeds at higher dimensions are analogous to the slowing rate of bit flips in more significant binary digits. Moreover, $e_{p+k}$ is just $e_p$, roughly speaking, rotated by $e_k$ in each pair of coordinates.

In summary, this method is nice for the following reasons.

**Properties.**
- It's flexible with respect to the length $n$ and the ambient dimension $d$
- Relative position is easy to learn since for any translation $k$, the encoding $e_k$ fits into a matrix with Jordan block sizes $2, \ldots, 2$ mapping $e_p$ to $e_{p+k}$
- Unseen positions are interpolatable

5. **The Transformer Architecture**

Google revolutionized NLP in 2017 by introducing the Transformer architecture, which uses only attention mechanisms to learn dependencies. It is non-recurrent, hence does not suffer from the vanishing gradient problem, and also non-directional. As illustrated by the success of the BERT model, the Transformer architecture can also be used for transfer learning, which at the time was commonplace in CV but ineffective in NLP.

The Transformer architecture uses the attention function in its multi-head attention layers. A multi-head attention layer has multiple heads which project $Q$,
$K$, and $V$ to lower-dimensional spaces, apply attention, then concatenate their attention vectors:

![Diagram of Transformer's multi-head attention mechanism]

**Figure 3.** The Transformer’s multi-head attention mechanism, as illustrated in *Attention is All You Need*. Learning multiple heads is analogous to learning multiple convolutional filters.

The input and output of a multi-head attention layer thus share the same shape. A learned projection is analogous to a convolutional filter, and thus learning multiple heads is analogous to learning multiple filters.

These multi-head attention layers fit into a block along with a feed-forward network, again preserving shape, and these blocks are repeated $N = 6$ times to form the encoder. The queries $Q$, keys $K$, and values $V$ are themselves learned linear embeddings of the input. Appending each layer is a residue and layer norm.
The decoder shares the same structure but with two key differences:
- the attention layer attending its output masks future tokens during training
- there’s an extra attention layer to attend to the encoder output.

6. BERT and GPT

BERT (Bidirectional Encoder Representations from Transformers) is a Transformer language model released in 2018 by Google a year after the *Attention is All You Need* paper. BERT’s role in NLP is analogous to those of pretrained models like VGG, ResNet, Inception in CV. It transfers well to other tasks, often taking only a handful of epochs to train, and at its release obtained SOTA results on many NLP tasks:

<table>
<thead>
<tr>
<th>System</th>
<th>MNL1-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT base</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT large</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>92.7</td>
<td>94.9</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>70.1</td>
<td>82.1</td>
</tr>
</tbody>
</table>

**Figure 5.** BERT’s performance on GLUE, a collection of benchmarks which tests general understanding and transfer ability, as tabulated in the *BERT* paper. It greatly outperformed the SOTA.

BERT was pre-trained on unlabeled text in an semi-supervised fashion to perform the following two tasks:
- **Masked language model (MLM):** given a (WordPiece embedded) sentence with 15% of the tokens masked, predict the masked tokens.
- **Next sentence prediction (NSP):** given two sentences, predict whether the second one follows the first.

It uses several techniques which are now standard:

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{bert_input.png}
\caption{BERT’s input format, as illustrated in the \cite{BERT} paper. The segment embeddings differentiate between types of input, and the \texttt{CLS} and \texttt{SEP} tokens indicate the locations of the separation.}
\end{figure}

**Techniques 6.1.**
- To indicate the start of a sentence, prepend a special \texttt{CLS} symbol.
- Similarly, to indicate separation of two pieces of text (e.g. the two sentences in NSP or question versus answer), insert a special \texttt{SEP} symbol and add learned segment embeddings to the initial input, as in with positional embeddings.

GPT, GPT-2, and GPT-3 are also Transformer language models that have been released over the past two years by OpenAI. GPT-3 exceeds in no- or few-shot learning and meets the SOTA when trained in a supervised fashion.

The GPT models were trained using a mix of unsupervised and supervised methods:
- **Unsupervised:** given a large corpus of text and a size $k$ of a context window, maximize the likelihood of each token given the $k$ tokens before it.
- **Supervised:** given labeled datasets, attach an output layer to the final transformer block, and fit the labels.

Each GPT model improves upon the last by using new training techniques and datasets, and importantly the parameter numbers grow extremely fast from 117M to 1.5B to 175B, as do the numbers of layers and heads, from 12 to 48 to 96. The progression has shown no sign of saturation. Moreover, the later models demonstrate strong one- and few-shot learning capabilities.
Figure 7. Performance of GPT models on an in-context learning task, i.e., one that uses natural language to describe the task to perform, as plotted in the GPT-3 paper [Language Models are Few-Shot Learners]. The researchers behind this paper concluded that the larger models make increasingly efficient use of in-context information.

7. Attention in CV

There have been many applications of attention to CV. Let us describe some notable or interesting ones.

In [Attention Augmented Convolutional Networks], researchers used attention and convolutions in parallel by concatenating the outputs. Despite being a straightforward application of attention, they improved the performance of SOTA CNN models for basic tasks such as classification.

In [Stand-Alone Self-Attention in Vision Models], researchers at Google experimented with a purely attention-based model. They replaced all convolutions in a ResNet model with self-attention layers. To make attention more effective in early layers, they locally transformed early input values to be spatially aware; roughly this boils down to manually convolving the inputs according to the location of a query pixel. The researchers found that their purely attention-based model outperformed the original model despite having fewer parameters and taking less time to train. Moreover, their ablation studies implied that attention was most effective in later layers because minute spatial information is typically more important in early layers:
ATTENTION IN NLP AND CV 9

Figure 8. Performance of the model in \textit{Stand-Alone Self-Attention in Vision Models} as compared to a ResNet-50 baseline, as plotted in the paper. The C-stem attention model uses convolutions for early layers.

In \textit{Generative Image Inpainting with Contextual Attention}, researchers used an attentive generative deep-learning approach to the inpainting problem. Their model consists of two networks, one for initial coarse reconstruction and another for final refinement:

Figure 9. An overview of the inpainting model, as illustrated in \textit{Generative Image Inpainting with Contextual Attention}. The refinement network consists of two parallel decoders.

The early network is trained with only an $\ell_1$ reconstruction loss, and the later network uses the same reconstruction loss as well as global and local WGAN-GP (Wasserstein GAN Gradient Penalty) adversarial losses. The later network is itself comprised of two parallel encoders. Intuitively, the bottom encoder hallucinates contents for the holes using dilated convolutions, and the top encoder focuses on details by attending to background regions with similar features:
8. Attention versus convolutions

Attention has the potential to replace convolutions in CV. Just as RNN’s suffer from problems with tracking long-range dependencies, CNN’s are often limited by its receptive field. In particular, only the fully-connected layers in a typical CNN can relate any area of the image to any other area. Attention mechanisms in CV thus aim to encourage longer-range connections without using expensive fully connected layers.

Theoretically, multi-head attention is at least as representative as convolutions: researchers made this intuition explicit in [On the Relationship between Self-Attention and Convolutional Layers]

Theorem. A multi-head self-attention layer with $N_h$ heads of dimension $D_h$, output dimension $D_{out}$ and a relative positional encoding of dimension $D_p \geq 3$ can express any convolutional layer of kernel size $\sqrt{D_h} \times \sqrt{D_h}$ and $\min(D_h, D_{out})$ output channels.

In the same paper, the researchers found that in practice attention in early layers in fact resemble convolutions. They trained a model with 6 layers and 9 heads using learned relative positional encoding and content-content based attention. In the following plot, they display the averaged attention of the indicated (black) query pixel over 100 images:
Figure 11. Averaged attention plots, as displayed in "On the Relationship between Self-Attention and Convolutional Layers." Heads in 2 and 3 resemble convolutions.

Visibly, the attention in layers 2 and 3 are focused in regions a fixed distance away from the query pixel, similar to the learned filters of a CNN. Here is their plot for a single image of a frog rather than the average over 100 images:

Figure 12. A single image attention plot, as displayed in "On the Relationship between Self-Attention and Convolutional Layers."

9. Transformers in CV

Inspired by the success of the Transformer architecture in NLP, researchers have experimented with Transformer-based models in CV. Most of these ideas seem straightforward but turn out to be quite effective.

In "An Image is Worth 16x16 Words" researchers split images into 16 × 16 pixel patches and trained a Transformer model to classify the images based on the resulting sequences. Similar to the CLS token in BERT [cf Techniques 6.1], they prepend a learned starting patch whose output receives a classification head, and they embed positional information into the patches.
They trained the model on the huge JFT-300M dataset. Their models exceed the SOTA on both ImageNet (Noisy Student) and other datasets (BiT-L) while taking less resources to train:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ours-JFT (ViT-H/14)</th>
<th>Ours-JFT (ViT-L/16)</th>
<th>Ours-121K (ViT-L/16)</th>
<th>BiT-L (ResNet152x4)</th>
<th>Noisy Student (EfficientNet-L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet</td>
<td>88.55 ± 0.04</td>
<td>87.76 ± 0.03</td>
<td>85.30 ± 0.02</td>
<td>87.54 ± 0.02</td>
<td>88.4/88.5*</td>
</tr>
<tr>
<td>ImageNet RealL</td>
<td>90.72 ± 0.05</td>
<td>90.54 ± 0.03</td>
<td>88.62 ± 0.05</td>
<td>90.54</td>
<td>90.55</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>99.50 ± 0.06</td>
<td>99.42 ± 0.03</td>
<td>99.15 ± 0.03</td>
<td>99.37 ± 0.06</td>
<td>—</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>94.55 ± 0.04</td>
<td>93.50 ± 0.05</td>
<td>93.25 ± 0.05</td>
<td>93.51 ± 0.08</td>
<td>—</td>
</tr>
<tr>
<td>Oxford-IIT Pets</td>
<td>97.56 ± 0.03</td>
<td>97.52 ± 0.11</td>
<td>94.67 ± 0.15</td>
<td>96.62 ± 0.23</td>
<td>—</td>
</tr>
<tr>
<td>Oxford Flowers-102</td>
<td>99.68 ± 0.02</td>
<td>99.74 ± 0.00</td>
<td>99.61 ± 0.02</td>
<td>99.63 ± 0.03</td>
<td>—</td>
</tr>
<tr>
<td>VTAB (19 tasks)</td>
<td>77.65 ± 0.23</td>
<td>76.28 ± 0.46</td>
<td>72.72 ± 0.21</td>
<td>76.29 ± 1.70</td>
<td>—</td>
</tr>
</tbody>
</table>

| TPUv3-core-days       | 2.5k                | 0.68k               | 0.23k               | 9.9k                | 12.3k                           |

**Figure 13.** Performance of the model in *An Image is Worth 16x16 Words* as tabulated in the paper. The ViT-L/16 models in particular meet the SOTA while taking a fraction of the time to train.

It also exceeds in transfer learning, especially with respect to pre-training time:

**Figure 14.** Transfer learning capacity of the model from *An Image is Worth 16x16 Words*, presented in the paper. The model reaches SOTA performance on transfer tasks with a fraction of the pre-training time.

Qualitatively, they found sensible convolution-like patch embeddings, positional embeddings, and increasingly strong attentiveness over depth:
Figure 15. A visualization of various components of the model from *An Image is Worth 16x16 Words*, as presented in the paper. Image sizes are $224 \times 244$. The patch embedding plots resemble convolutions, the positional embedding similarities demonstrate spatial understanding, and the attention distance plot illustrates the large receptive field.

In *Image Transformer*, researchers fed local areas of images directly into a Transformer at the pixel and channel level to evaluate its generating ability, e.g. inpainting and super-resolution. They experimented with attending to local 1D areas versus 2D areas:

As expected, the latter was more successful, yielding a lower loss and fooling human observers at a higher rate of 36%, compared to the 10% SOTA at the time. The obvious improvement involves sparsely attending to non-local areas.

In *End-to-End Object Detection with Transformers*, researchers fed CNN features into a Transformer and trained the decoder to perform object detection. They named their framework DETR, for detection transformer. The layers in the decoder produce $N$ positional embeddings or *object queries*, where

$$N \gg \text{maximum number of objects in an image.}$$

These positional embeddings are analogous to RoI’s and are inputted into the decoder as zeros. The decoder as a whole predicts the class and bounding boxes corresponding to the embeddings:
Figure 17. An overview of the object detection model in *End-to-End Object Detection with Transformers* as illustrated in the paper. The number of object queries is set to be much higher than the maximum number of objects in an image.

To force unique predictions, this system uses a bipartite matching loss.

While the decoder performs a similar function to the RPN in Faster RCNN, one key difference is that the slots for the object queries learn to detect different things and in different areas of the image. To illustrate this, the researchers plotted the box prediction centers on all images from the COCO 2017 validation set for 20 out of the $N = 100$ prediction slots:

Figure 18. Bounding box centers predicted by the object detection model in *End-to-End Object Detection with Transformers*, as plotted in the paper. Green corresponds to small boxes, red to large horizontal boxes, and blue to large vertical boxes. The distinct patterns illustrate that the slots in the decoder learn different things.

Though DETR is mostly analogous to Faster RCNN, it in fact performs better, indicating the power of attention. The following table lists the AP of DETR on the COCO validation set compared to Faster RCNN models with various backbones and training methods:
Figure 19. Performance of DETR compared to the SOTA, as tabulated in [End-to-End Object Detection with Transformers]. DETR uses mainly attention-based layers and outperforms Faster RCNN.

Acknowledgments

These notes were written for the Dec 11 biweekly seminar at Novateur Research. Thanks to Jon Amazon, Jacob Hultman, Mark Tenzer, and Yuxiang Zhao for insightful discussions. Thanks also to Zeeshan Rasheed and Khurram Hassan-Shafique for their support.