EE-559 – Deep learning

13.2. Transformer Networks

François Fleuret

https://fleuret.org/ee559/

May 12, 2020
The most powerful language models (as of 03.07.2019) take the form of a sequence of attention-based revisions of the representation (Vaswani et al., 2017).

The original sequence structure of the signal is secondary and provided indirectly to the processing through additional inputs.
Vaswani et al. (2017) use the terminology of Graves et al. (2014): attention is an averaging of **values** associated to **keys** matching a **query**.

With \( Q \) the tensor of row queries, \( K \) the keys, and \( V \) the values,

\[
Q \in \mathbb{R}^{T \times C} \quad K \in \mathbb{R}^{T' \times C} \quad V \in \mathbb{R}^{T' \times D},
\]

the result of the attention operation is

\[
Y_j = \sum_i \frac{\exp(Q_j K_i^T)}{\sum_r \exp(Q_j K_r^T)} V_i,
\]

or

\[
Y = \text{softmax} \left( Q K^T \right) V.
\]

The queries and keys have the same dimension \( C \). There are as many keys as there are values \( T' \). The result has as many rows \( T \) as there are input rows and they are of same dimension \( D \) as the values.
\( K^T \)

\[
Q \quad QK^T \quad A \quad Y = AV
\]
In the currently standard models for sequences, the queries, keys, and values are linear functions of the features and $T = T'$.

Hence given three matrices $W_Q \in \mathbb{R}^{C \times D}$, $W_K \in \mathbb{R}^{C \times D}$, and $W_V \in \mathbb{R}^{D \times D'}$, and an input sequence $X \in \mathbb{R}^{T \times D}$, we have

\[
\begin{align*}
Q &= X W_Q^\top \\
K &= X W_K^\top \\
V &= X W_V^\top
\end{align*}
\]

This is **self-attention** since the sequence is “looking at itself”.
To illustrate the behavior of attention layers, we consider a toy problem with 1d sequences composed of two triangular and two rectangular patterns. The objective is to change the height of the patterns to the average of their pair.
To illustrate the behavior of attention layers, we consider a toy problem with 1d sequences composed of two triangular and two rectangular patterns. The objective is to change the height of the patterns to the average of their pair.
To illustrate the behavior of attention layers, we consider a toy problem with 1d sequences composed of two triangular and two rectangular patterns. The objective is to change the height of the patterns to the average of their pair.
To illustrate the behavior of attention layers, we consider a toy problem with 1d sequences composed of two triangular and two rectangular patterns. The objective is to change the height of the patterns to the average of their pair.
To illustrate the behavior of attention layers, we consider a toy problem with 1d sequences composed of two triangular and two rectangular patterns. The objective is to change the height of the patterns to the average of their pair.
Some training examples.
We test first a 1d convolutional network, with no attention mechanism.

```
Sequential(
    (0): Conv1d(1, 32, kernel_size=(5,), stride=(1,), padding=(2,))
    (1): ReLU()
    (2): Conv1d(32, 32, kernel_size=(5,), stride=(1,), padding=(2,))
    (3): ReLU()
    (4): Conv1d(32, 32, kernel_size=(5,), stride=(1,), padding=(2,))
    (5): ReLU()
    (6): Conv1d(32, 32, kernel_size=(5,), stride=(1,), padding=(2,))
    (7): ReLU()
    (8): Conv1d(32, 1, kernel_size=(5,), stride=(1,), padding=(2,))
)
```

nb_parameters 15809
Training is done with the MSE loss and Adam.

```python
batch_size = 100

optimizer = torch.optim.Adam(model.parameters(), lr = 1e-3)
mse_loss = nn.MSELoss()

mu, std = train_input.mean(), train_input.std()

for e in range(args.nb_epochs):
    for input, targets in zip(train_input.split(batch_size),
                               train_targets.split(batch_size)):
        output = model((input - mu) / std)
        loss = mse_loss(output, targets)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```
The poor performance of this model is not surprising given its inability to channel information from “far away” in the signal. Using more layers, global channel averaging, or fully connected layers could possibly solve the problem. However it is more natural to equip the model with the ability to combine information from parts of the signal that it actively identifies as relevant.

This is exactly what an attention layer would do.
We use a classical $N \times C \times T$ representation to allow standard convolutions, and to implement the attention mechanism with the products by $W_Q$, $W_K$, and $W_V$ as convolutions.

To manipulate more clearly the dimensions we use `torch.permute()` that allow to reorder them arbitrarily.

To compute $QK^\top$ and $AV$ we need a batch matrix product, which is provided by `torch.matmul()`.
```python
>>> a = torch.rand(11, 9, 2, 3)
>>> b = torch.rand(11, 9, 3, 4)
>>> m = a.matmul(b)
>>> m.size()
torch.Size([11, 9, 2, 4])

>>> m[7, 1]  
tensor([[0.8839, 1.0253, 0.7473, 1.1397],
        [0.4966, 0.5515, 0.4631, 0.6616]])

>>> a[7, 1].mm(b[7, 1])  
tensor([[0.8839, 1.0253, 0.7473, 1.1397],
        [0.4966, 0.5515, 0.4631, 0.6616]])

>>> m[3, 0]  
tensor([[0.6906, 0.7657, 0.9310, 0.7547],
        [0.6259, 0.5570, 1.1012, 1.2319]])

>>> a[3, 0].mm(b[3, 0])  
tensor([[0.6906, 0.7657, 0.9310, 0.7547],
        [0.6259, 0.5570, 1.1012, 1.2319]])
```
class AttentionLayer(nn.Module):
    def __init__(self, in_channels, out_channels, key_channels):
        super(AttentionLayer, self).__init__()
        self.conv_Q = nn.Conv1d(in_channels, key_channels, kernel_size = 1, bias = False)
        self.conv_K = nn.Conv1d(in_channels, key_channels, kernel_size = 1, bias = False)
        self.conv_V = nn.Conv1d(in_channels, out_channels, kernel_size = 1, bias = False)

    def forward(self, x):
        Q = self.conv_Q(x)
        K = self.conv_K(x)
        V = self.conv_V(x)
        A = Q.permute(0, 2, 1).matmul(K).softmax(2)
        x = A.matmul(V.permute(0, 2, 1)).permute(0, 2, 1)
        return x

    def __repr__(self):
        return self._get_name() + ' (in_channels={}, out_channels={}, key_channels={})'.format(
            self.conv_Q.in_channels,
            self.conv_V.out_channels,
            self.conv_K.out_channels
        )
Sequential(
(0): Conv1d(1, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(1): ReLU()
(2): Conv1d(32, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(3): ReLU()
(4): AttentionLayer(in_channels=32, out_channels=32, key_channels=32)
(5): Conv1d(32, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(6): ReLU()
(7): Conv1d(32, 1, kernel_size=(5,), stride=(1,), padding=(2,))
)

nb_parameters 13729
Without attention
With attention

MSE vs Number of epochs for models with and without attention.
Francois Fleuret

EE-559 – Deep learning / 13.2. Transformer Networks
Transformers
3.2.1 Scaled Dot-Product Attention

We call our particular attention “Scaled Dot-Product Attention” (Figure 2). The input consists of queries and keys of dimension $d_k$, and values of dimension $d_v$. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix $Q$. The keys and values are also packed together into matrices $K$ and $V$. We compute the matrix of outputs as:

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

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of $d_k$ the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of $d_k$ [3]. We suspect that for large values of $d_k$, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

3.2.2 Multi-Head Attention

Instead of performing a single attention function with $d_{\text{model}}$-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values $h$ times with different, learned linear projections to $d_k$, $d_k$ and $d_v$ dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding $d_v$-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

$$(\text{Vaswani et al., 2017})$$

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

$$\text{MultiHead}(Q, K, V) = \text{Concat} \left( H_1, \ldots, H_h \right) W^O$$

$$H_i = \text{Attention} \left( QW_i^Q, KW_i^K, VW_i^V \right), \; i = 1, \ldots, h$$

with

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \; W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \; W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \; W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$. 

(Vaswani et al., 2017)

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

$$\text{MultiHead}(Q, K, V) = \text{Concat} \left( H_1, \ldots, H_h \right) W^O$$

$$H_i = \text{Attention} \left( QW_i^Q, KW_i^K, VW_i^V \right), \; i = 1, \ldots, h$$

with

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \; W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \; W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \; W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$. 

(Vaswani et al., 2017)
Attention disregards positioning in the sequence, Vaswani et al. provide this information through a **positional encoding**, added to the original input.

It has the same dimension as $d_{model}$ and is of the form

\[
PE_{t,2i} = \sin \left( \frac{t \cdot 2i}{10,000 \cdot d_{model}} \right)
\]

\[
PE_{t,2i+1} = \cos \left( \frac{t \cdot 2i+1}{10,000 \cdot d_{model}} \right)
\]
3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection \( \text{LayerNorm}(x + \text{Sublayer}(x)) \), where \( \text{Sublayer}(x) \) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position $i$ can depend only on the known outputs at positions less than $i$.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

(Vaswani et al., 2017)
The Universal Transformer (Dehghani et al., 2018) is a similar model where all the blocks are identical, resulting in a **recurrent model that iterates over consecutive revisions of the representation instead of positions**.

The positional embedding is expended with the block index $1 \leq t \leq T$

$$
P_{i,2j}^t = \sin \left( \frac{i}{10,000 \frac{2j}{d_{model}}} \right) + \sin \left( \frac{t}{10,000 \frac{2j}{d_{model}}} \right)
$$

$$
P_{i,2j+1}^t = \cos \left( \frac{i}{10,000 \frac{2j}{d_{model}}} \right) + \cos \left( \frac{t}{10,000 \frac{2j}{d_{model}}} \right)
$$
The Universal Transformer (Dehghani et al., 2018) is a similar model where all the blocks are identical, resulting in a **recurrent model that iterates over consecutive revisions of the representation instead of positions**.

The positional embedding is expended with the block index $1 \leq t \leq T$

\[
\begin{align*}
P_{i,2j}^t &= \sin \left( \frac{i}{10,000 \frac{2j}{d_{model}}} \right) + \sin \left( \frac{t}{10,000 \frac{2j}{d_{model}}} \right) \\
P_{i,2j+1}^t &= \cos \left( \frac{i}{10,000 \frac{2j}{d_{model}}} \right) + \cos \left( \frac{t}{10,000 \frac{2j}{d_{model}}} \right)
\end{align*}
\]

Additionally the number of steps is modulated per position dynamically.
Attention in computer vision
Wang et al. (2018) proposed an attention mechanism for images, following the model from Vaswani et al. (2017).

\[ y = \text{softmax} \left( (W_\theta x)^T (W_\phi x) \right) W_g x. \]
Wang et al. insert “non-local blocks” in residual architectures and get improvements on both video and images classification.

Figure 2. A spacetime **non-local block**. The feature maps are shown as the shape of their tensors, e.g., $T \times H \times W \times 1024$ for 1024 channels (proper reshaping is performed when noted). “$\otimes$” denotes matrix multiplication, and “⊕” denotes element-wise sum. The softmax operation is performed on each row. The blue boxes denote $1 \times 1 \times 1$ convolutions. Here we show the embedded Gaussian version, with a bottleneck of 512 channels. The vanilla Gaussian version can be done by removing $\theta$ and $\phi$, and the dot-product version can be done by replacing softmax with scaling by $1/N$.

(Wang et al., 2018)
Figure 3. Examples of the behavior of a non-local block in res3 computed by a 5-block non-local model trained on Kinetics. These examples are from held-out validation videos. The starting point of arrows represents one \( x_i \), and the ending points represent \( x_j \). The 20 highest weighted arrows for each \( x_i \) are visualized. The 4 frames are from a 32-frame input, shown with a stride of 8 frames. These visualizations show how the model finds related clues to support its prediction.

(Wang et al., 2018)
Ramachandran et al. (2019) replaced convolutions with local attention.

\[
y_{i,j} = \sum_{(a,b) \in \mathcal{N}(i,j)} W_{i-a,j-b} x_{a,b} \quad \text{(Convolution)}
\]

\[
y_{i,j} = \sum_{(a,b) \in \mathcal{N}(i,j)} \text{softmax}_{a,b} \left( (W_Q x_{i,j})^T (W_K x_{a,b}) \right) v_{a,b} \quad \text{(Local attention)}
\]

![Figure 2: An example of a 3 × 3 convolution. The output is the inner product between the local window and the learned weights.](image1)

![Figure 3: An example of a local attention layer over spatial extent of \( k = 3 \).](image2)

(Ramachandran et al., 2019)
<table>
<thead>
<tr>
<th></th>
<th>ResNet-26</th>
<th>ResNet-38</th>
<th>ResNet-50</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>4.7</td>
<td>13.7</td>
<td>74.5</td>
</tr>
<tr>
<td><strong>Conv-stem + Attention</strong></td>
<td>4.5</td>
<td>10.3</td>
<td>75.8</td>
</tr>
<tr>
<td><strong>Full Attention</strong></td>
<td>4.7</td>
<td>10.3</td>
<td>74.8</td>
</tr>
</tbody>
</table>

### Table 1: ImageNet classification results for a ResNet network with different depths.

Baseline is a standard ResNet, Conv-stem + Attention uses spatial convolution in the stem and attention everywhere else, and Full Attention uses attention everywhere including the stem. The attention models outperform the baseline across all depths while having 12% fewer FLOPS and 29% fewer parameters.

Figure 5: Comparing parameters and FLOPS against accuracy on ImageNet classification across a range of network widths for ResNet-50. Attention models have fewer parameters and FLOPS while improving upon the accuracy of the baseline.

(Ramachandran et al., 2019)
The end
References


