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

$$Y_j = \sum_i \frac{\exp(Q_j K_i^\top)}{\sum_r \exp(Q_j K_r^\top)} V_i,$$

or

$$Y = \text{softmax}(Q K^\top) V.$$

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.
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

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

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.
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.

\[
\text{batch\_size} = 100
\]

\[
\text{optimizer} = \text{torch.optim.Adam(model.parameters(), lr = 1e-3)}
\]

\[
\text{mse\_loss} = \text{nn.MSELoss()}
\]

\[
\text{mu, std} = \text{train\_input.mean()}, \text{train\_input.std()}
\]

for \( e \) in range(args.nb\_epochs):

    for \( \text{input}, \text{targets} \) in zip(train\_input.split(batch\_size),
                        train\_targets.split(batch\_size)):
        \[
        \text{output} = \text{model((input - mu) / std)}
        \]
        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

Input
Output

Input
Output

Input
Output
Transformers
Scaled Dot-Product Attention

Multi-Head Attention

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 $d_{\text{model}}$ and is of the form

$$PE_{t, 2i} = \sin \left( \frac{t}{10,000 \cdot 2^i} \right)$$

$$PE_{t, 2i+1} = \cos \left( \frac{t}{10,000 \cdot 2^{i+1}} \right)$$

(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 \(1 \leq t \leq T\)

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

\[
P_{i,2j+1}^t = \cos \left( \frac{i}{10,000 \text{ } d_{\text{model}}} \right) + \cos \left( \frac{t}{10,000 \text{ } d_{\text{model}}} \right)
\]

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

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

![Diagram of non-local block](image)

**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)

![Examples of non-local block behavior](image)

**Figure 3.** Examples of the behavior of a non-local block in $	ext{res}_3$ 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.

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

(Convolution)

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

(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.

Figure 3: An example of a local attention layer over spatial extent of \( k = 3 \). (Ramachandran et al., 2019)

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)
References


