Squeeze-Excitation Networks
An easy to implement attention mechanism

Jacques Nel

September 20, 2020

1 Introduction

The convolutional operator, used in CNNs, has an essentially local receptive field. Recently, there have been some spectacular results achieved by neural network architectures that rely on attention mechanisms. The transformer, for example, consists entirely of an attention mechanism, and manages to achieve state-of-the-art results on NLP tasks.[1]

The *Squeeze-and-Excitation (SE)* block allows the model to model complex inter-channel dynamics, and allows a global flow of information, similar to an attention mechanism.[2] SE blocks are simple to implement, and can simply be added to any pre-existing model architecture. SE blocks can significantly improve a model's performance, at a marginal additional computational cost.

2 SE architecture

The SE block is build around a convolutional operator. Let $\boldsymbol{X} \in \mathbb{R}^{H' \times W' \times C'}$ denote the input signal, with height, width, and channel depth being H', W', and C' respectively.

$$\boldsymbol{F}_{tr} : \boldsymbol{X} \in \mathbb{R}^{H' \times W' \times C'} \rightarrow \boldsymbol{U} \in \mathbb{R}^{H \times W \times C}$$

Take $\boldsymbol{F}_{tr}$ to be a convolutional operator with learned filter kernels $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_C]$. In other words

$$\boldsymbol{u}_c = \boldsymbol{v}_c * \boldsymbol{X} = \sum_{s=1}^{C'} \boldsymbol{v}_c^s * \boldsymbol{x}^s.$$

2.1 Squeeze operation

Each filter map operates with a local receptive field and can not utilize information outside of its region. First, a global channel statistic is generated by a *squeeze* operation, consisting of an average pooling layer.

[1] Vaswani et al., "Attention Is All You Need", NIPS 2017.
[2] Hu et al., "Squeeze-and-Excitation Networks".

$$z_c = \boldsymbol{F}_{sq}(\boldsymbol{u}_c) = \frac{1}{HC}\sum_{i=1}^{H}\sum_{j=1}^{W} u_c(i,j).$$

2.1.1 Excitation

Next, an *excitation* layer allows channel-wise dependence to be learned on the output of the *squeeze* operation. This simply consists of a simple gating mechanism with two fully connected layers, a sigmoid, and ReLu activation function.

$$\boldsymbol{s} = \boldsymbol{F}_{ex}(\boldsymbol{z}, \boldsymbol{W}_1, \boldsymbol{W}_2) = \sigma(\boldsymbol{W}_2 \mathrm{ReLu}(\boldsymbol{W}_1 \boldsymbol{z})),$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{\frac{C}{\rho} \times C}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{C \times \frac{C}{\rho}}$. The reduction ratio ρ, preferably a divisor of C, allows one to control the number of parameters. The *excitation* operator acts as a bottleneck. In effect, the SE block acts as a self-attention mechanism, acting on a wider domain than the receptive field of the individual convolution filters.

Finally, the output of the SE block is produced by scaling $\boldsymbol{U}$ with the signal $\boldsymbol{s}$.

$$\tilde{\boldsymbol{x}}_c = \boldsymbol{F}_{scale}(\boldsymbol{u}_c, s_c) = s_c \boldsymbol{u}_c.$$

We also use a skip connection, that bypasses the SE block entirely.

3 Implementation

The idea of an SE block is extremely simple to implement.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam
from torch.utils.data.dataloader import DataLoader


class SeBlock2d(nn.Module):

    def __init__(self,
                 channels,
                 ratio=2):

        super().__init__()

        self.conv = nn.Conv2d(in_channels=channels,
                              out_channels=channels,
                              kernel_size=3,
                              padding=1)

        self.gate_1 = nn.Linear(in_features=channels,
                                out_features=channels // ratio)
        self.gate_2 = nn.Linear(in_features=channels // ratio,
                                out_features=channels)

    def forward(self, x):

        # Generate feature maps U.
        u = self.conv(x)
        h, w = x.shape[-2:]

        # Apply global average pooling to calculate channel descriptor z.
        z = 1. / (h + w) * torch.sum(u, dim=(-2, -1))

        # Pass signal through two fully connected layers of excitation operator.
        s = torch.sigmoid(self.gate_2(F.relu(self.gate_1(z))))
        s = s.unsqueeze(-1).unsqueeze(-1)

        # Scale by activation s.
        x_out = s * u

        return x_out
```

Figure 1: Squeeze-and-Excitation architecture