Nic Kuo
u6424547@anu.edu.au

Discussing TCN: On Residual Connections and Causal Convolutions

This document is meant to cover the implementation details of residual connections and causal convolutions. In particular, we will look at jerrybai1995's official Github repository at lines 15 – 45 of TCN / TCN / tcn.py.

The code

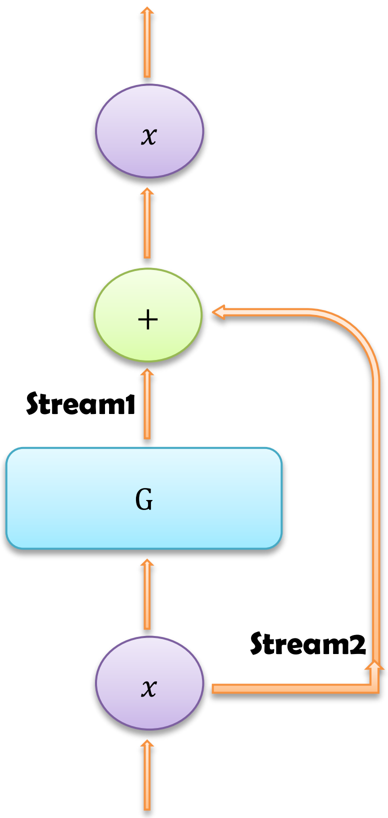
```
class TemporalBlock(nn.Module):
    def __init__(self, n_inputs, n_outputs, kernel_size):
        super(TemporalBlock, self).__init__()
        #===
        self.C1   = Causal_Conv1d(n_inputs, n_outputs, kernel_size)
        self.C2   = Causal_Conv1d (n_inputs, n_outputs, kernel_size)
        #===
        if n_inputs != n_outputs:
            self.Sc    = nn.Conv1d(n_inputs, n_outputs, 1)
        else:
            self.Sc    = None

    def forward(self, x):
        Stream1  = torch.relu(self.C2(
                      torch.relu(self.C1( x) ) )
        Stream2       = x if self.downsample is None else self.Sc(x)

        out = torch.relu(Stream1 + Stream2)
        return out
```

Starting from next page, we will look at the shaded parts more closely.

Residual Connections



The code implements a residual connection at <u>line 20</u>, where

out = torch.relu(Stream1 + Stream2)

or equivalently as

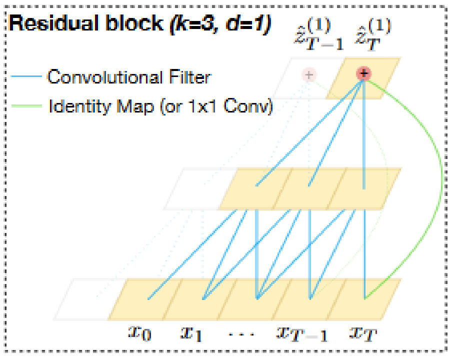
$$out = T(x) \qquad \text{where}$$
$$x = x + G(x)$$

with x as some input and that T and G as some (non-linear) transformation functions. For jerrybai1995's code, both T and G took the form of ReLUs.

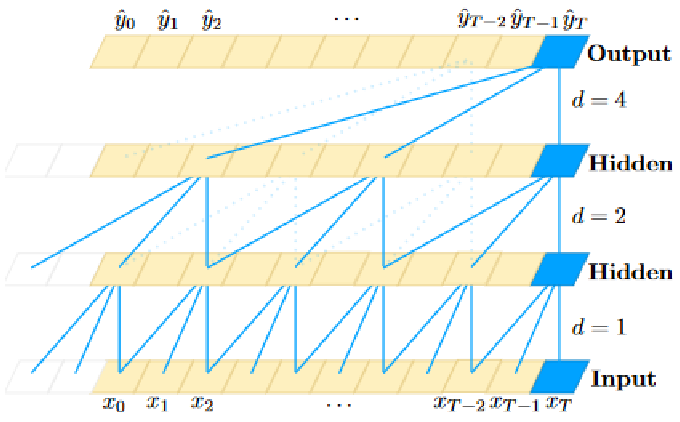
The purpose of a residual connection is to learn an update package $G(x)$ to adjust the original input of x. Furthermore, the plus sign in $x + G(x)$ is commonly referred as a shortcut connection.

But is **Stream2** really x though? Well … more or less. In deep learning, it is common practice to project a low dimensional input to a much higher hidden dimensional latent space, and then learn the structure of the said latent space to represent features of the input. Hence, in order to account of the potential dimensional mismatch, we have the conditional shortcut convolution in <u>lines 10 – 13</u>.

The following figure is taken from Bai's original paper from Figure 1 (c) of page 4. The so-called identity mapping is the same as **Stream2**; whereas all blue lines contribute to the function $G(x)$ of **Stream1**. This is conducted via causal convolutions and we will dive into this a bit deeper in the next page.

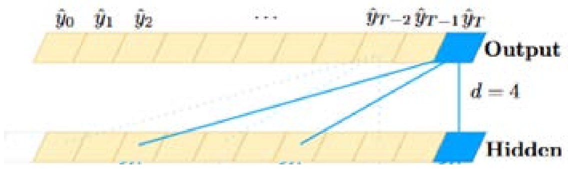


Causal Convolutions

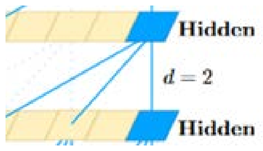

$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\cdots$ $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 $\cdots$ x_{T-2} x_{T-1} x_T

This image can be found in Figure 1 (a) of page 4 of the original paper and it describes the causal convolution. Below, we will discuss how and why the network purposely avoids information from the future as inputs.

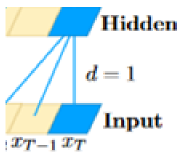
The said avoidance can be seen here:


$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\cdots$ $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden

where the current instance $\hat{y}_T$ is marked by the blue shaded block and that its inputs (the blue lines) are either from distinct past or from directly below of an earlier layer. The same happens for the neural information transmissions in



and in



.

This specific architecture is adopted because the TCN aims to replace RNNs and that (one of) its main goal(s) is to be able to process past data in order to predict future values. Hence the future values cannot be leaked into the inferential process. However, such a desideratum directly conflicts with the inner workings of a standard convolutional network, and we will discuss this overleaf.

Say that we have an input

$$\tilde{x} = [x_1, x_2, x_3, x_4] = [0.1, 0.2, 0.3, 0.4]$$

and we process it with convolution

$$\tilde{w} = [w_1, w_2, w_3] = [9, 33, 2],$$

then the output would be

$$\begin{aligned} \tilde{y} &= [y_1, y_2] \\ &= [w_1 x_1 + w_2 x_2 + w_3 x_3, \\ &\quad w_1 x_2 + w_2 x_3 + w_3 x_4] \\ &= [8.1, 12.5]. \end{aligned}$$

That is, when we are processing for output y at instance 1, a standard convolutional network computes that value with future inputs of instances 2 and 3. Hence in order to prevent this from happening, we would need to redefine convolution instead of using the **nn.Conv1d** function directly from PyTorch.

We need 3 steps in order to utilise past values only. First, we will need to pad $\tilde{x}$ with 2 zeros (2 because $\tilde{w}$ has a convolutional kernel of size 3). That is, we make

$$[x_1, x_2, x_3, x_4] \leftarrow [0, 0, x_1, x_2, x_3, x_4, 0, 0] = [0, 0, 0.1, 0.2, 0.3, 0.4, 0, 0].$$

Second, we will apply the kernel as usual

$$\begin{aligned} \tilde{y} &= [y_1, y_2, y_3, y_4, y_5, y_6] \\ &= [w_1 0 + w_2 0 + w_3 x_1, \\ &\quad w_1 0 + w_2 x_1 + w_3 x_2, \\ &\quad w_1 x_1 + w_2 x_2 + w_3 x_3, \\ &\quad w_1 x_2 + w_2 x_3 + w_3 x_4, \\ &\quad w_1 x_3 + w_2 x_4 + w_3 0, \\ &\quad w_1 x_4 + w_2 0 + w_3 0] \\ &= [0.2, 3.7, 8.1, 12.5, 15.9, 3.6] \ . \end{aligned}$$

However, not everything in this new $\tilde{y}$ is required. We have only padded our input in order to process $w_1 0 + w_2 0 + w_3 x_1$ but our original input remains 4 element in length. Hence the third step is to remove the last 2 elements and make $\tilde{y}$

$$[0.2, 3.7, 8.1, 12.5, 15.9, 3.6] \leftarrow [0.2, 3.7, 8.1, 12.5].$$

Steps 1, 2, and 3 corresponds respectively to lines 4, 9, and 10 in the code overleaf.

```
class Causal_Conv1d (nn.Module):
    def __init__(self, n_inputs, n_outputs, kernel_size, padding):
        super(Causal_Conv1d, self).__init__()
        self.chomp_size = kernel_size - 1
        self.C = nn.Conv1d(n_inputs, n_outputs, kernel_size,
                                        padding = self.chomp_size)

    def forward(self, x):
        out  = self.C(x)
        out  = out[:, :, :-self.chomp_size].contiguous()
        return out
```