TCN – Model

In the past, Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been the architecture of choice for sequence modelling. There has been results put forward, to prove that convolutional networks should be considered in the conversation as the primary model for predicting sequential data (S.Bai , 2018).

These results showed that convolutional networks outperformed the forementioned RNNs in many tasks, while simultaneously avoiding the vanishing gradient problem, a problem that plagues RNN’s. As the backpropagation gradient flow is not dependent over time, this brings additional benefits such as the ability to train in parallel for much faster training using GPU optimization.

Temporal Convolutional Networks (TCNs) are a class of time-series models that capture long-range patterns using the order of temporal convolutional filters.

A TCN is a variation of a CNN with 2 main features. Firstly, each hidden layer is the same length as the input layer. This is achieved using zero padding.

Secondly, the network only uses information from past time steps. This is implemented by using casual convolutions, both features can be seen in Figure 1.

The last thing to consider for TCN models is the receptive field. The receptive field is the outputs dependencies from the original input. The larger the receptive field the greater the chance of capturing long range dependencies. With casual convolutions it is not feasible to capture long range dependencies without your network growing too large from adding too many layers. It is not recommended to use larger kernel sizes either as this adds weights in order to describe the kernel, this is not parameter efficient as most of the time we have multiple filters per layer, this could quite quickly lead to overfitting.

The solution is to use dilation casual convolutions, we stretch the kernel position out by a dilation factor so as a result, the receptive field is also extended by a dilation factor. We stack these dilated casual convolutional layers, so we use all inputs as a receptive field as seen in Figure 2.

( The network I used had a ….. )

The formula’s below are to calculate the number of layers in the network and the padding for each dilated layer.

A picture containing text, clock

Description automatically generated

Figure to calculate the number of layers needed in the network, l = input\_size, b = dilation\_base, k = kernel\_size. Roundup the result.

Text

Description automatically generated with low confidence

Figure to calculate padding for each layer, b = dilation\_base, I = number of layers beneath , k = kernel\_size

Timeline

Description automatically generated

Figure 1. Here we can see 24 timesteps, padded at each convolutional layer to keep input same length as output. The output of timestep 1 is only ever dependent on timesteps that come on and before timestep 1.

Chart, bubble chart

Description automatically generated

Figure 2. Here we can see that less layers are needed using dilated convolutional layers.