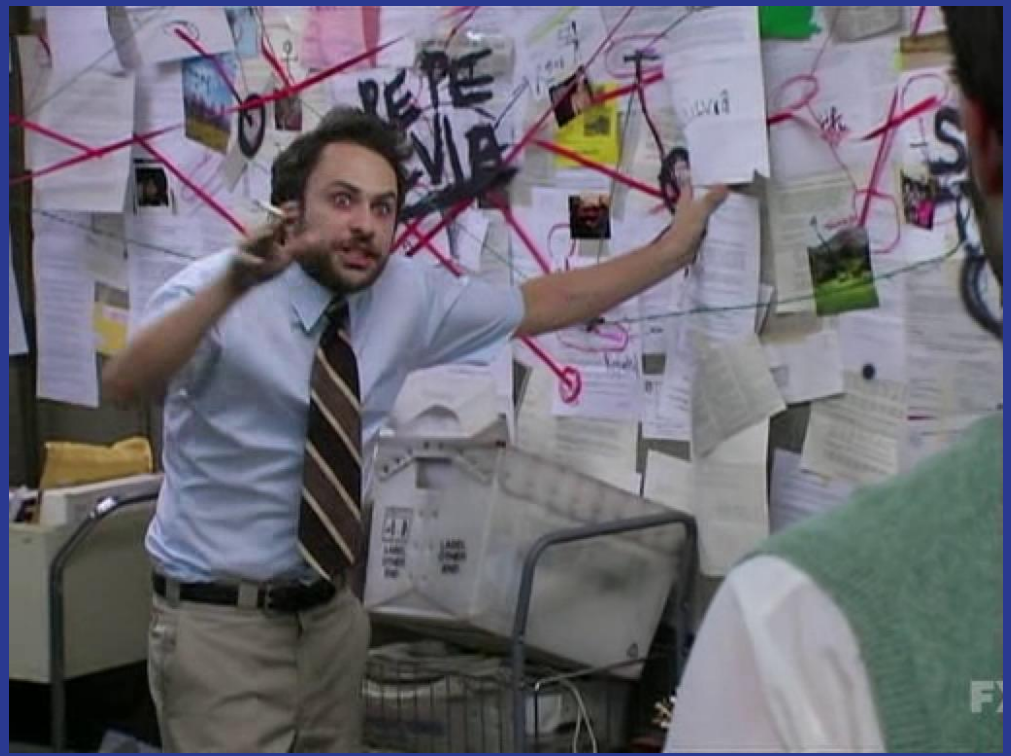
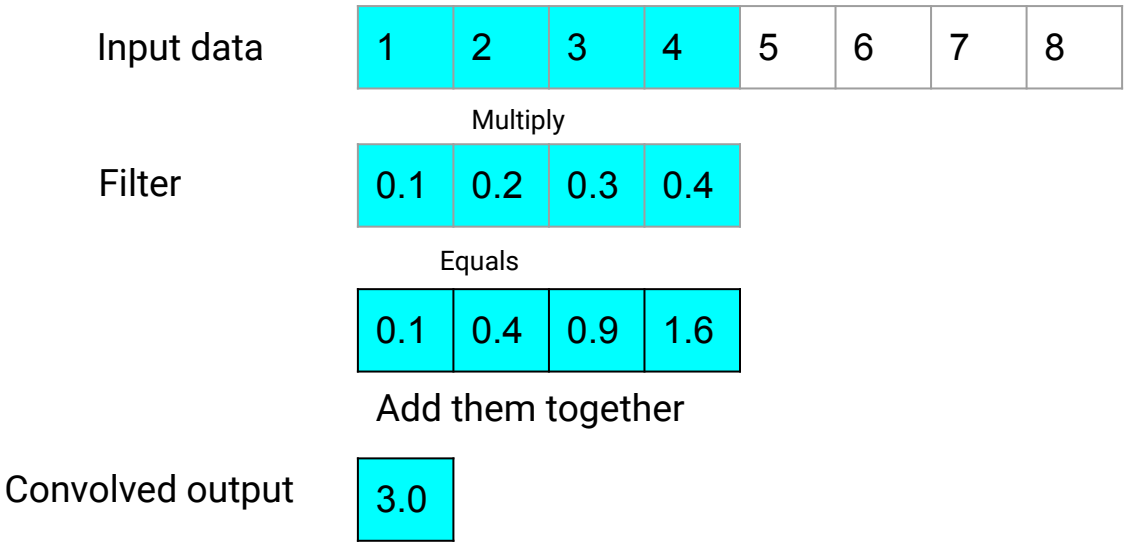
Convolutional Time Series Prediction



What is a 1D convolution?

A convolution is a mathematical operation in which you apply a filter to a set of data.

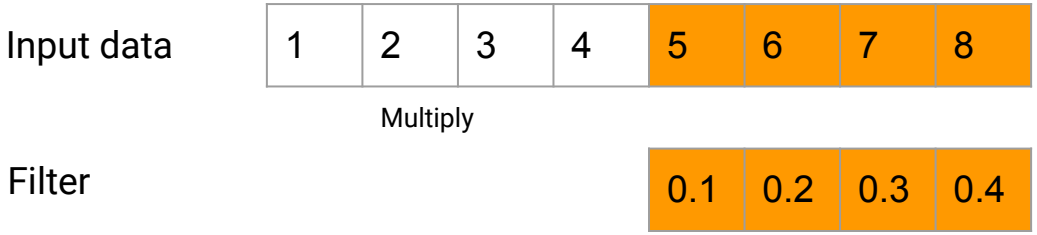


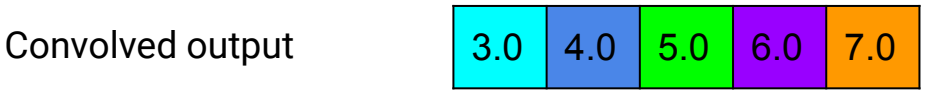
Step 2 in 1D convolution

Then you scoot the filter over and do it again

Loop until done

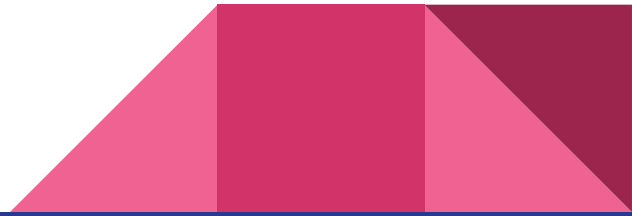
Continue until your window hits the end of the input data





Activations

Technically there is one additional step, the activation function. In the previous slides, the activation was linear or no activation function, but if you set activation=”relu”, you can send the convolved vector through a ReLU activation function. This introduces that idea of nonlinearity, and if your network doesn’t seem to be learning anything, try introducing this nonlinearity to your algorithm. You have the whole set of functions to choose from, so tanh, sigmoid, etc.



Padding

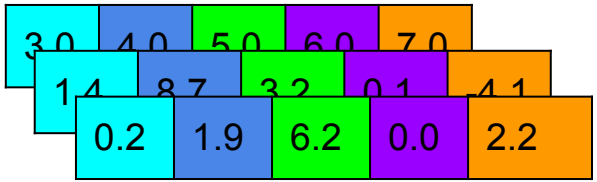
To force the convolved output to be the same shape as the input, you can also do padding = same to pad your input data with zeros. If you use padding = valid, then your output will shrink relative to the size of the input.

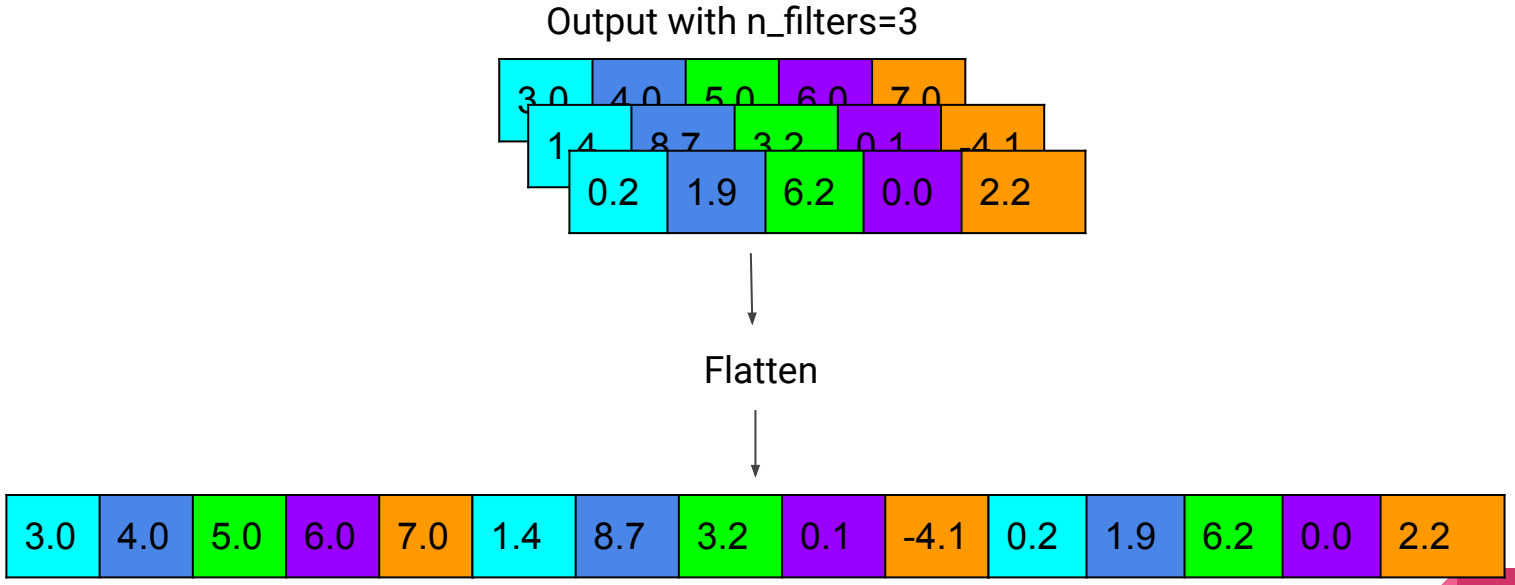
Input data	0	1	2	3	4	5	6	7	8	0

Output shape

The output of your conv1D will have as many layers as you have filters, and each filter will be independently trained. The filters will wind up acting like automatic feature detectors. Some will activate only in an upward curve, some will deliver just the most recent value in the series, some will deliver a moving average. You don't have to decide what all of them will do, because in the training process they will figure out the most useful thing to do. Unfortunately, this leaves you with a 1D input and possible 2D output.

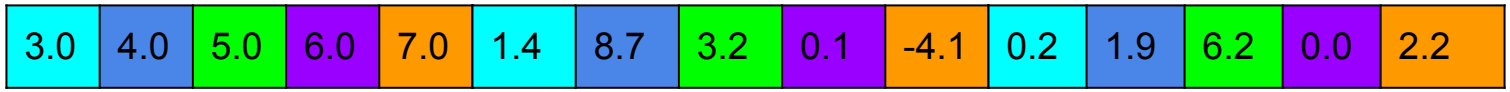
Output with n_filters=3




Output with n_filters=3
0.2
1.9
6.2
0.0
2.2
Flatten
3.0
4.0
5.0
6.0
7.0
1.4
8.7
3.2
0.1
-4.1
0.2
1.9
6.2
0.0
2.2

Flatten

If your conv1D layer is producing an output which is 2D and you feed that into a Dense(), the Dense() will also try to produce a 2D output. If we're trying to produce a 1D output in the end, like a stock price estimate or a trading signal, this can be problematic. So we can force it back down from 2D to 1D with flatten, which does this:



Does it work?

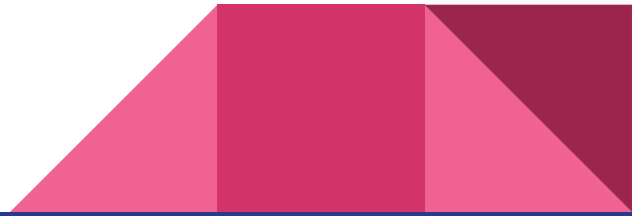
Input: GEFCOM dataset

Target: Next hourly RTDemand

Basic conv1D net: Conv1D(10 filters, each 5-wide) -> Flatten ->Dense(1800)->Dense(100)->Dense(1)

Vs

LSTM 100 -> Flatten -> Dense(1500) -> Dense(100) -> Dense(1)

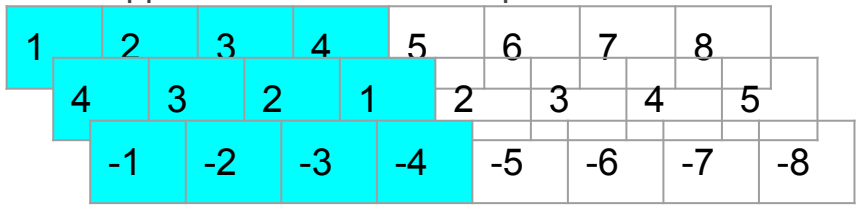


Does it work?

Name	MSE	RMSE	R^2
Dumb model	411,577	642	0.9436
Conv1D basic	921,088	959	0.8786
LSTM basic	1,174,830	1,084	0.8452

Multivariate input

What happens if I have several input series?



Filter

0.1	0.2	0.3	0.4

Equals

0.1+0.4+-0.1	0.4+0.6+-0.4	0.9+0.6+-0.9	1.6+0.4+-1.6

Add them together

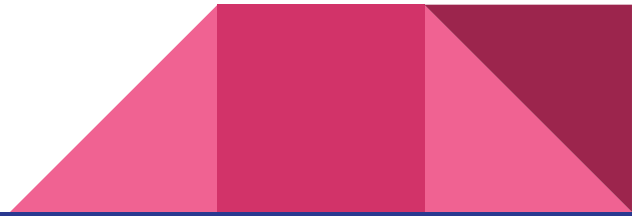
Convolved output

2.0

The problem with that

Because the convolution window is only 1D, adding more data points in the 2nd dimension isn't that helpful because we are not able to multiply each datapoint times a unique number, but instead we multiply them all by the same number. Thus, we can't individually train the convolution window on a per-feature basis, or to say that another way, adding another data point only serves to 'blur out' other data points.

If this hypothesis is correct, adding more data points to our 1D model would result in worse outcomes. Did it?

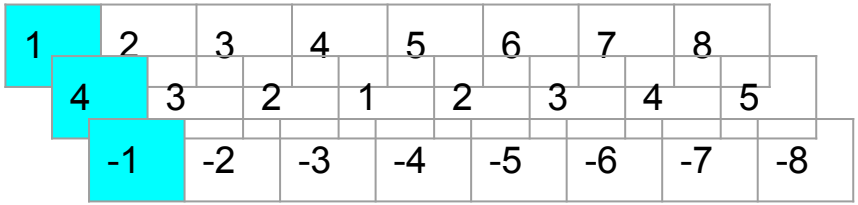


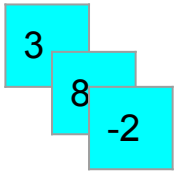
Does it work?

Name	MSE	RMSE	R^2
Dumb model	411,577	642	0.9436
Conv1D basic	921,088	959	0.8786
LSTM basic	1,174,830	1,084	0.8452
Conv1D w/ 12 features	1,190,440	1,091	0.8431
LSTM w/ 12 features	1,439,643	1,200	0.8103

The solution to this problem

We don’t want to convolve over the first elements but with the same filter for all of them, we want a unique filter position for each of our inputs, like this:

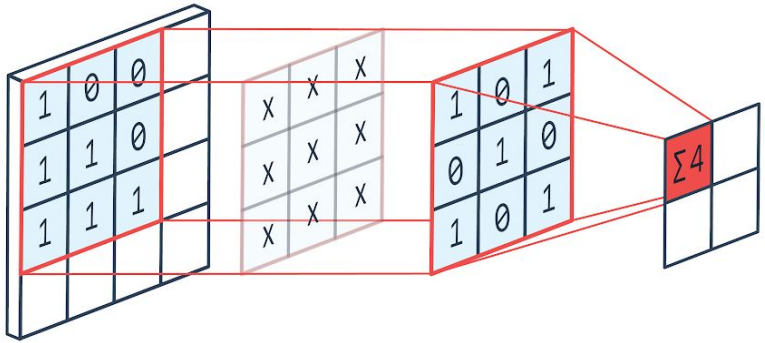


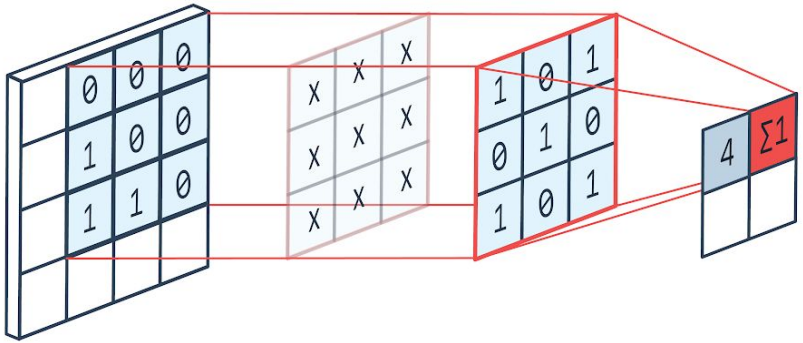


We can do this with a 2D convolution window

2D Convolutions

Normally, a 2D convolution is used in image processing to convolve over squares, like this:





2D Convolutions

In our case, we make the filter size 1, filter_features so it will convolve 1 timestep over filter_features features.

But does it work?

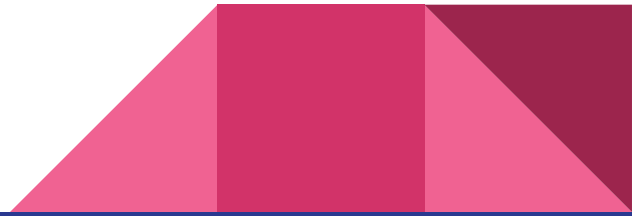
Does it work?

Name	MSE	RMSE	R^2
Conv 2D	68,627	262	0.9910
Dumb model	411,577	642	0.9436
Conv1D basic	921,088	959	0.8786
LSTM basic	1,174,830	1,084	0.8452
Conv1D w/ 12 features	1,190,440	1,091	0.8431
LSTM multi-variable	1,439,643	1,200	0.8103

Deeper still

Our output is going to be a set of convolutions across our features, but I would also like to try to detect patterns across time, as we've seen our data has such patterns. To do this, we'll flip our 2-D window around to (hours, 1) so we don't mix 2 observations (hours) together.

BUT DID IT WORK?



Does it work?

Name	MSE	RMSE	R^2
Conv 2D -> Conv2D	47,929	219	0.9937
Conv 2D	68,627	262	0.9910
Conv1D basic	921,088	959	0.8786
LSTM basic	1,174,830	1,084	0.8452
Conv1D w/ 12 features	1,190,440	1,091	0.8431
LSTM multi-variable	1,439,643	1,200	0.8103

The final payoff: Did it REALLY work?

In order to determine if this really worked, we also need to see what we'd get from the 'just guess the last answer' model. The results:

Final comparison

Name	MSE	RMSE	R^2
Conv 2D -> Conv2D	47,929	219	0.9937
Conv 2D	68,627	262	0.9910
Dumb model	411,577	642	0.9436
Conv1D basic	921,088	959	0.8786
LSTM basic	1,174,830	1,084	0.8452
Conv1D w/ 12 features	1,190,440	1,091	0.8431
LSTM multi-variable	1,439,643	1,200	0.8103

Future improvements

Some ideas for future improvement include:

- Use all the data: I have 10x more data than was used here
- Maybe train a little longer: Many loss functions were still declining pretty well when they ended
- Build a forecast for all my features individually, then composite them together to forecast more than 1 step into the future
- Sell it to Adam for $1 million, and he can sell it to Duke for $1 billion