CNN for Time Series Forecasting

General

The aim of this work is to

- train and make predictions using the original btrotta (M4 competitor) convolutional neural models for time series forecasting (https://github.com/btrotta/m4) . The basic functions for training and predicting are taken from the above link and are modified, extended accordingly.
- understand the models and the implementation
- design our own implementation of convolutional neural networks and use them to make predictions on the M4 provided dataset
- evaluate the results and provide with some notes – potential explanations.
- Design some additional functions like plot functions to monitor the loss functions and visualization functions to plot the predictions.Those can be found here : (https://github.com/mdarm/m4-competition-approaches/tree/07ccfccd4afa770a25e9f40e93e5570ce93e466b/CNN-Forecasting/visualizations)

Analysis of the original implementation

The author of the original models has followed a general approach, regarding the numerous time series provided for every frequency. In particular instead of trying to manually extract characteristics like trend and seasonality from all the time series (which is not feasible since in total the dataset of the M4 competition contains 100.000 time series from different domains), the task of modeling the time series is assigned to a number of different convolutional neural networks, which are designed to retrieve the trend and seasonalities of the time series for each different frequency.
In particular the author suggests that the convolutional kernels can capture the seasonality patterns and uses 3 filters for each convolutional layer. Different convolutional models are built for each frequency as well as for each horizon time step but also for each training length. The training length is the number of observations used for each frequency to predict the horizon. The horizon time step is a time step in the future window the model tries to predict. The author provides each model with a list of fixed training lengths, which the neural networks receive as input. From the above we can see that for a horizon of 14 days(in the example of the daily series) and a list of training lengths {16,24} we would end up with 14 models (one for each horizon step) for each training length , so in total 14 * 2 = 28 models. This logic is applied for all the other frequencies as well.
For the training the author suggests that the latest time series observations are the most important and thus suggests picking the above training lengths for each frequency from the end of the training data and going backwards If we want to extend the training length. In essence the small training lengths correspond to the latest short term historic data whereas going backwards means we take into account a history of older data. The training lengths are picked to be multiples of all the frequencies that the series are expected to exhibit. This decision is grounded on the fact that if the filter length divides exactly the input length the filter will slide and align an integer number of times over the input. For the final predictions of a particular frequency the author combines the results of the models of the different training lengths by calculating the mean of them . For series with not enough data, synthetic data are constructed by filling each time step t with the observation at the (T+ t) time step (same position at the next period), in order to maintain periodic patterns. For the standardization the typical approach of subtracting the mean and dividing by the standard deviation is used. Also for the series with not enough samples the series are copied and shifted backwards 1 step creating a same length series. In this case, in total N such series are created and addied to the dataset, where N is chosen as the minimun between some variables defined in the code involving the training lengths and the horizon (num_extra = min(num_to_add, len(row) -series_length – horizon).

Model Description

Regarding the author’s Model design for each frequency, more information can be found in (https://github.com/btrotta/m4) but in essence in each model : With the exception of the yearly series in all the other frequencies , first averages are created by use of average pooling . For example in the quarterly series, applying average pooling with stride 4 and pool size 4 means we are collecting the yearly averages (4 quarters = 1 year). Then 2 operations follow : (1) Some dense layers flatten those averages into a scalar output, let’s call it A. (2) The averages are upsampled N times in order to align to the initial input. For example for the aforementioned example for a series of 2 years we would have 16 quarters and by applying the average pooling we would get 2 yearly averages. Then each average would be upsampled to 4 , so we would get 4 x 1st year average , 4 x 2nd year average such that each quarter of the first year could align with the first year average and each quarter of the second year could align with the second year average. Then from each quarter it’s correspoding average is subtracted and if we generalize this for each frequency, the periodic differences are created. Those differences are then transformed into 3d tensors so we can apply the subsequent convolutional layers. The author uses a fixed number of filters (3) and the kernel size is the periodic frequency. For example for the querterly series, the kernel will be 4, so a kernel of size 4 will slide over the total , training_length/4 groups packed into the tensor. Consequently , dense layers flatten the output of the convolutional layers into a scalar output . This output is added to A and the final output is produced.

Testing The code and Extensions
Regarding the already existent models , we first downloaded the dataset and trained the models , while also collecting the mase and smape metrics and then we made some predictions , included in the github page , in the directory tree original_trained_models/results
We also created some new functions :

1. a function to plot the losses for each time step a model is trained on
2. a function to plot/visualize the final model’s predictions. For the visualization we picked only a sample series for each frequency to show but the code can be easily extended to make multiple observations. For the visualizations included in the github page , those depict the predictions on the horizon after the end of the training data (the future values) made with the original model vs the predictions made with our model vs the provided `y_test` observations.

Experimentation/Creating New Models
After studying and experimenting with the initial models we decided to implement our own 1 step convolutional models drawing from the author’s designs.

Yearly Frequency

1. For the yearly frequency the author’s model did not include any periodic subtractions so we decided to :
   1. apply average pooling of 2 in order to get 2 year aggregates , using valid padding in order to stay within the range of the input
   2. upsample the averages
   3. add some dense layers similar to those of the author’s model , using 20 nodes in each dense layer and feed the aforementioned averages into those dense layers, giving a scalar output A . This way we try to express any potential history that exists within 2 years.
   4. subtract from each year it’s corresponding average to create some periodic differences. The decision to apply 2 year averages was grounded on the idea that there could potentially exist a 2-3 year periodicity.
   5. Using the same logic as the author the series is transformed into a 3d tensor with 2 year groups where the convolution filter will slide . A kernel of 2 and a stride of 2 is used. 4

filters were used since in the experiments we observed better performance .A learning rate of 0.001 was used.

6. 2 dense layers give the scalar output B.
7. We add A and B to get the final output.

Quarterly Frequency

For the quarterly series we :

1. apply averaging pooling of 4 , capturing the yearly average like the author does. Those pass through some dense layers to give a scalar output A.
2. We follow the same logic as mentioned in “Model description “ ,upsampling the averages constructing the subtractions and passing them through the convolutional layer but besides the average pooling, the upsampling and the convolutional layer ,
3. We add the naive forecast and pass it through 3 dense layers to get a scalar output B.We try in other words to exress the influence that a potential naive forecast could have in the result.
4. After the subtractions we transform them in typical fashion into 3d tensors of groups of 4 and apply a convolutinal layer of kernel 4 and stride 4.We then pass this output into 3 dense layers to get another scalar output C. 4 convolutional filters were used.
5. Finally the output is given as the average of the 3 scalars A,B,C. We experimented with simple additions as well as weighted additions but simple average provided with the best performance. A learning rate of 0.001 was used .

Monthly frequency

For the monthly series we :

3. use the same model as the author but
4. additionally calculate again the naive output and pass it through 3 dense layers to get a scalar B like in the quarterly frequency.
5. The ouput is given again as the average of the outputs A, B ,C , where A is the output from the dense layers we feed the yearly averages, B is the naive output and C is the output from the convolutional filters we feed the subtractions.The convolutional layers has 6 filters of kernel size 12 and stride 12 .

Weekly frequency

For the weekly series the process implemented is as follows :

6. If the training length is 52 we follow the same process as before calculating the yearly averages , upsampling , calculating the subtractions and passing through the convolutional layer with a kernel of 52 and a stride of 52.We used 4 filters. We also again calculate the naive output and pass it through the dense layers and take as output the addition of the naive scalar with the the output from the convolutional layer. In other words the upsampled averages do not pass through the dense layer in this case.
7. If the training length is not 52 we apply the same process as the author and take as the output the addition of the periodic output (the subtractions through some dense layers to produce a scalar) with the output from the convolutional layer passed through some dense layers (to produce a second scalar). For the dense layers we used 52 nodes.For each average pooling we use a pool size of 52.We used a learning rate of 0.01.

Daily frequency

For the daily series we :

8. Apply the same model as the author but again add the naive output like it was described in the quarterly model to get 3 scalars in similar fashion. The total output is given as the

average of the 3 scalars. For the convolutional layer we used a kernel of 7 to try to collect the weekly averages and a stride of 7 and applied 3 filters. The nodes used in the dense layers are 20.We used a learning rate of 0.01.

Hourly Frequency

For the hourly series we used the same model as the author in which :

9. we calculate the weekly averages by use of average pooling of 168 (24 * 7) and then passing them through some dense layers we get one scalar A.
10. we upsample the averages and calculate the subtractions from every hour. Then we create the 3d tensor of N/7 groups of 7 and apply 3 convolutional filters of kernel size 7 and stride 7. Passing the output through some dense layers we get a second scalar B.
11. Then we calculate the daily averages by average pooling with a size of 24 and stride of 24 , we upsample , calculate the subtractions from the hours and pass those trhough a convolutional layer of 3 filters with kernel size of 24 and stride of 24. The output passes through some dense layers and we get another scalar C.
12. We add A, B, C to get the final output.For the intermediate dense layers we used 20 units.

In all the previous models the dense layers consist of the provided number of nodes except if this is missing .This implies simple dense layer of output dimension of (1) that produces a scalar output. Also when the reader sees the phrase “some dense layers” this means in most cases 2-3 dense layers followed by a final dense layer of output dimension of 1. More information can be found by examining the model architecture in the code. For optimizing training Adam optimizer is used and MSE is chosen as the error to minimize.

Metrics used

<u>(Add the 2 equations from M4 competition paper)</u>

For the evaluation of the models we used the mase and smape metrics , as defined in the script provided (metrics.py) , the metrics used in the M4 competition:

- Smape is the forecast error as a proportion of the midpoint between the actual obsevation and the prediction. Smape is defined as the sum of the absolute differences of the actual and the predicted values in each time step, divided by the sum of the actual and the predicted values, with this sum being divided by 2 . The sum of the actual and predicted values divided by 2 represents the midpoint(average) of the actual and the predicted value. Since the error in each time step is divided by this midpoint, in essence we calculate how much of this average the error represents. In the M4 competition this is multiplied with 100 to get a percentage error : <u>(add link of M4 competition paper , where smape is described).</u>
- Mase is also another usual forecasting metric and it tries to express the performance of a forecasting model in comparison to the naive method performance. In the Naive forecast each time step is shifted forwards one step , thus in each time step the forecast value is the previous time step’s value. With the Mase metric each method is evaluated against the naive method as a fraction of it. If the method produces lower error than the naive forecast then the Mase value is smaller than 1 , otherwise if the method leads to higher error than the naive forecast this value is bigger than 1. For the naive method Mase gives a value of 1. (add link for handyman and kohler paper mase definition).Mase is defined as the sum of the errors of all the time steps with each error divided by the mean absolute error of the entire forecast horizon.This sum is divided then by the number of time steps .Specifically for the M4 competition , instead of calculating the mean absolute error between time step t and time step t-1 , the mean absolute error between time step t and t-m is calculated, where m is the time interval between

successive observations, which differ for each frequency.(link for M4 competition , paragraph 3.2).

Results and Comparisons :
The results of the author’s models (trained and tested by us of course) along with the predictions for the horizon values at the end of the training set, can be found in the github page. (https://github.com/mdarm/m4-competition-approaches/tree/07ccfccd4afa770a25e9f40e93e5570ce93e466b/CNN-Forecasting/original_trained_models/results) whereas our model’s results and predictions are here: (https://github.com/mdarm/m4-competition-approaches/tree/07ccfccd4afa770a25e9f40e93e5570ce93e466b/CNN-Forecasting/predictions). For the predictions, we produce different different csv files per frequency, since we argue it is easier to process the files this way and focus on the frequency of interest.

Since it is not possible to compare every single result of every frequency and every training length we will briefly mention that in most cases our models’ results are similar and in some cases better than the original model, which could mean that indeed in many series the naive forecast could model some of the series underlying patterns.Regarding the periodicity and the potential seasonal patterns in the yearly frequencies the original models used only dense layers whereas we applied pooling as with the other frequencies and convolutional filters. The results are similar and in many cases better , suggesting that patterns can be found in the yearly series as well. For the quarterly series we observe similar performances. For the monthly frequency, the use of the naive forecasts in the result as part of an average produces similar and in many cases better results in most cases , suggesting the importance of the latest – short tem history .Similar performance is observed in the weekly frequencies and the hourly frequences.For the daily frequency we decided to add the training length of 364 days and the performance seems to benefit from the longer training lenth suggesting that more history can help find potentially periodic events.

Multi Step Models

As additional experimentation we proceeded to create some multi step convolutional models . In essence we do not construct a model for every single time step but one for the whole horizon and try to predict the whole horizon at the same time. Regarding the architecture , stacked convolutional layers were used. Those models were trained on different training lengths and they did not perform well.On the other hand they were trained and tested in order to try to observe the efficiency of cnn for time series forecasting using generally shorter history.You can find their code (many basic functions are based again on the btrotta implementation) here (https://github.com/mdarm/m4-competition-approaches/tree/07ccfccd4afa770a25e9f40e93e5570ce93e466b/CNN-Forecasting/multi_step).