Crossing the Channels: Improving PatchTST for Long-Term Forecasting

Younwoo Jeong, Guoyu Zhang, Ziyue Pan

Abstract

Recent transformer-based models achieve great success in multivariate long-term time series forecasting (MLTSF). However, both channel-dependent and independent approaches fail to effectively capture cross-channel dependencies. We propose **C**ross-**C**hannel **C**orrelation transformer (3Cformer) which builds upon the state-of-the-art (SOTA) channel independent model PatchTST by incorporating a graph convolutional network with a cosine similarity-based graph construction approach to capture cross-channel dependencies. We perform evaluation on 5 popular real world datasets to show that our model achieves SOTA accuracy on MLTSF tasks where there are a high number of cross-channel dependencies.

1. Introduction

Time series forecasting (TSF) is a critical component in many real world applications, such as weather forecasting (Shi et al., 2015), energy management (Uremović et al., 2022), disease propagation analysis (Matsubara et al., 2014), and traffic flow forecasting (Guo et al., 2019). Approaches to tackle the TSF problem have evolved from traditional statistical methods (ARIMA (Box & Jenkins, 1968)) to machine learning methods utilising deep learning models such as Recurrent Neural Networks (Lai et al., 2017), Convolutional Neural Networks (Koprinska et al., 2018), and most recently, Transformers (Wen et al., 2022).

Transformers (Vaswani et al., 2017) introduced the self-attention mechanism, whereby pairwise dependencies for each pair of elements in a sequence can be efficiently learned in parallel. This allows long range dependencies within a sequence to be better captured, thus enabling transformers to excel in sequence modelling tasks. Transformers are able to outperform Recurrent Neural Networks (RNN (Rumelhart et al., 1986)), which often suffer from vanishing gradients during the training of long sequences (Pascanu et al., 2012), and Convolutional Neural Networks (CNN (LeCun et al., 1989)), which are only adept at identifying dependencies among adjacent elements.

Transformers have achieved great success in domains such as natural language processing (NLP) (Brown et al., 2020), speech processing (Karita et al., 2019), computer vision (Dosovitskiy et al., 2021), and now TSF (Wen et al., 2022). FEDformer (Zhou et al., 2022), Crossformer (Zhang & Yan, 2023), and Autoformer (Wu et al., 2021) are some of the SOTA models for TSF tasks, especially MLTSF.

Multivariate time series involves a group of dynamically changing interdependent variables, for example, an increase in daily temperature could increase ice-cream sales, and decrease the demand for household heating. The aforementioned transformer-based models address the MLTSF task through channel mixing. Each univariate time series within the multivariate data could be thought of as a channel. Channel mixing is where all interdependent channels at a specific time step is collated as a singular vector and projected to the embedding space. This approach may introduce excessive mixing of information and has been challenged by DLinear (Zeng et al., 2023), a channel independent model which uses simple linear layers instead of a transformer-based architecture.

PatchTST (Nie et al., 2022) is a transformer-based channel independent model which improves on DLinear. Recognizing that individual time steps in a series does not convey semantic meaning in the same way words do in a sentence, PatchTST mainly focuses on the importance of extracting localised semantic information in time series. By aggregating adjacent time steps into subseries-level patches, PatchTST allows the transformer's attention mechanism to focus on patches representing local context, rather than individual time steps. However, the inherent issue with channel independent models is that it is unable to account for correlation between channels of the multivariate time series, unlike the channel-mixing models.

In this paper, we introduce cross-channel dependencies to PatchTST through the use of graph convolutional networks (GCN (Kipf & Welling, 2017)). We choose GCNs as we anticipate that they will be effective in encapsulating cross-channel correlations due to their high capability in capturing relational dependencies (Wu et al., 2020). Graphs are composed of vertices, and edges which represent the connections between the vertices. GCNs essentially mimic the convolution process in CNNs. They work by updating the information of each vertex by considering the information from its neighboring vertices, effectively allowing each vertex to learn from its local neighbours. Our proposed method treats each channel as a vertex, thus enabling information propagation between channels. The structure of the graph is defined by an adjacency matrix, constructed by filtering the cosine similarities between the vector representations of each channel within the embedding space by a specific threshold. We choose cosine similarity given its proven effectiveness in assessing semantic relationships

between words in NLP (Sitikhu et al., 2019), and considering PatchTST's focus on constructing each patch to capture deeper local semantic information, this similarity measure seems particularly apt.

Our proposed approach introduces cross channel dependencies without the excessive information blending that is inherent in channel-mixing approaches. This is because information propagation occurs after individually projecting each channel to its own embedding space, rather than merging all channels into a unified embedding, maintaining the distinctiveness of each channel's information.

In summary, the main contributions of our research are as follows:

- We introduce a graph convolutional network to PatchTST with a cosine similarity based graph construction approach. This enables each channel to propagate information predominantly to channels that are closely related to itself, effectively capturing cross-channel dependencies. We name our approach **C**ross-**C**hannel **C**orrelation transformer (3Cformer).
- We perform experiments on 5 real world datasets and improve on PatchTSTS and recent SOTA models for MLTSF tasks for datasets where there are high correlations between channels. Notably, for the ILI dataset (see Section 4) which contains many correlations, 3Cformer improves the MSE by **13.65%** and MAE by **5.7%** on average across all prediction lengths, compared to PatchTST.

2. Related Work

2.1. Transformers for MLTSF

State-of-the-art transformer models for MLTSF tasks include, as previously mentioned, FEDformer, Crossformer, and Autoformer.

Autoformer (Wu et al., 2021) introduces a decomposition architecture combined with an Auto-Correlation mechanism. Decomposition (Cleveland et al., 1990) is a popular method in time series which breaks down a series into three systematic components: trend-cycle (long-term direction of the time series), seasonal variation (recurring patterns within the time series), and random fluctuations. The model utilises decomposition blocks to aggregate trend-cycles and extract seasonal variation from the series progressively. The Auto-correlation mechanism is introduced to replace self-attention. The mechanism splits the time series into sub-sequences and focuses on the underlying connections among these sub-sequences, effectively capturing periodic dependencies in a series.

In addition to the decomposition architecture of the Autoformer, FEDformer (Zhou et al., 2022) decomposes the time series using Fourier transform to apply attention in the frequency domain, rather than time domain.

Crossformer (Zhang & Yan, 2023) is composed of three main components: Dimension-Segment-Wise (DSW) embedding, Two-Stage Attention (TSA), and Hierarchical Encoder-Decoder (HED). DSW embeds segments of a time series thus capturing temporal context. TSA utilises the attention mechanism to capture not only intra-series dependencies but also inter-series relationships. HED integrates data from various levels of granularity for higher precision forecasting.

These models all adopt channel-mixing and point-wise attention. As previously mentioned, PatchTST (Nie et al., 2022) has been effective in introducing channel independence and patch-wise attention to achieve superior performance in comparison to these SOTA models.

2.2. Graph Neural Networks for Multivariate Time Series Forecasting

Graph neural networks (GNN (Scarselli et al., 2008)) excel in handling relational dependencies. By viewing multivariate time series data from a graph perspective, GNNs are able to achieve great success in multivariate time series forecasting.

Wu et al. introduces a model comprised of three modules: graph learning, graph convolution, and temporal convolution (Wu et al., 2020). The graph learning module extracts relations among data, using this for graph construction. Then, the graph convolution module enables information propagation between each vertex and it's neighbours through a mix-hop propagation layer, effectively capturing inter-series dependencies. Lastly, the temporal convolution module applies convolution filters to the time series to extract inter-series features.

The Spectral Temporal Graph Neural Network (StemGNN) is another GNN which aims to capture both inter and intra series dependencies (Cao et al., 2020). The StemGNN block combines Graph Fourier Transform (GFT) and Discrete Fourier Transform (DFT) to enhance feature representations. The GFT models inter-series correlations and the DFT transfers each univariate time series into the frequency domain to capture intra-series correlations.

These approaches excel in identifying dependencies between different series, but do not utilise the powerful attention mechanism of transformers to capture relationships within individual series.

3. Methodology

3.1. Problem Definition

The multivariate time series forecasting problem can be defined as follows: given a time series sequence $X = \{x_1, x_2, \ldots, x_L \mid x_t \in \mathbb{R}^M\}$, we would like to forecast the future time series sequence $Y = \{y_{L+1}, y_{L+2}, \ldots, y_{L+F} \mid y_t \in \mathbb{R}^M\}$, where each x_t and y_t at time t is of dimension M (representing the number of channels/variables at each data point in the multivariate series), L is the length of the historical sequence, and F is the forecast horizon.

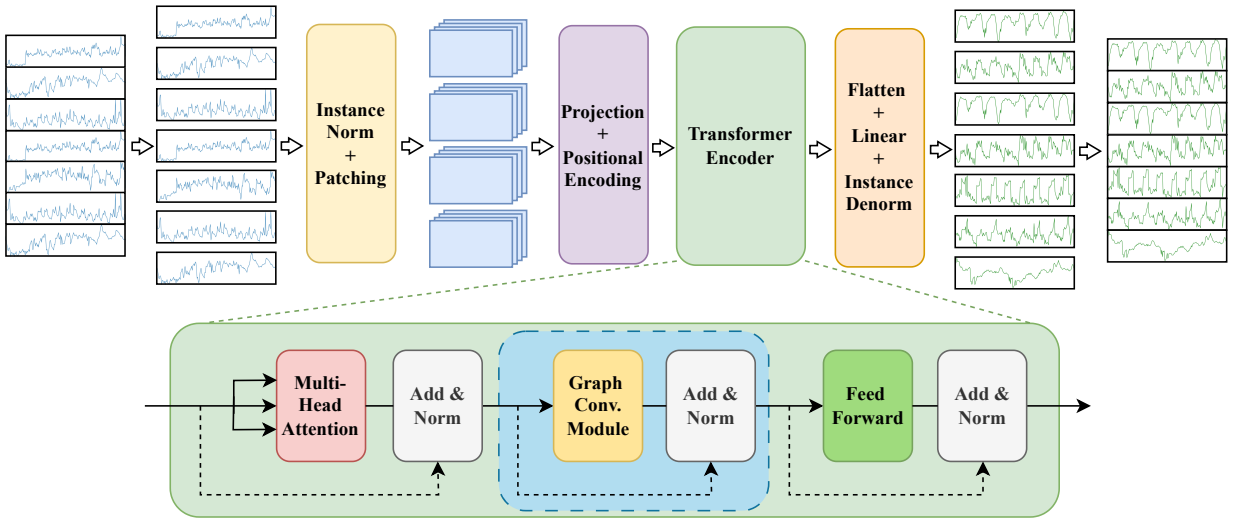

Figure 1. The overall architecture of our proposed approach, building upon PatchTST by incorporating a graph convolution module, highlighted by the blue rectangle, within the transformer encoder to capture cross-channel dependencies.

PatchTST (Nie et al., 2022) splits the multivariate series into univariate sequences/channels defined as follows: $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_L^{(i)} \mid x_t^{(i)} \in \mathbb{R}\}$, where $i = (1, \ldots, M)$. Predictions are made on each univariate sequence/channel independently to get $y^{(i)} = \{y_1^{(i)}, y_2^{(i)}, \ldots, y_L^{(i)} \mid y_t^{(i)} \in \mathbb{R}\}$, then concatenated back to form a multivariate series.

3.2. Model Structure

Our proposed approach introduces a graph convolution module to PatchTST, the overall architecture is shown in Figure 1. Aside from the graph convolution module and it's associated add and norm layers highlighted in the blue box, the remaining is identical to the original PatchTST.

The model takes multivariate data as input, but performs all operations independently for each channel. It first performs instance normalisation (further detailed in Section 3.2.4) and patching (more deatil in Section 3.2.1) to remove distributional shift and transform each series into a collection of patches. The patches are then projected into the embedding space, at which point positional encoding is applied, and fed into the transformer encoder.

The transformer encoder is composed of standard multi-head attention and feed-forward layers, augmented by the insertion of our graph convolution module in-between them to capture cross-channel dependencies. It constructs a graph (further detailed in Section 3.2.2) from the correlation between the channels outputted by the multi-head attention and facilitates information propagation (further detailed in Section 3.2.2) across these channels.

Lastly, the multi-dimensional output from the transformer encoder is flattened, passed through a linear layer, denormalised (further detailed in 3.2.4), and the channels are concatenated to produce the desired multivariate time series data.

3.2.1. Patching

As in the original PatchTST paper, patching is defined as splitting the univariate input $x^{(i)} \in \mathbb{R}$ to $x^{(i)} \in \mathbb{R}^{P \times N}$ where P is the patch length, N is the number of patches, $N = \left\lfloor \frac{L-P}{S} \right\rfloor + 2$, and S is the stride.

3.2.2. Graph Convolution Module

A graph G is defined as an ordered pair $G = (V, E)$ consisting of a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, and a set of edges $E = \{\{v_i, v_j\}, \ldots\}$. The adjacency matrix A of a graph with n vertices is an $n \times n$ matrix where A_{ij} is 1 if there is an edge from vertex v_i to vertex v_j, and 0 otherwise.

By considering each channel of the multivariate series as a vertex and connecting highly correlated channels with edges, we can utilise a graph convolutional network to propagate information between them. The overall approach for our graph convolution module is illustrated in Figure 2.

Graph Construction After the multi-head attention, the output consists of several channels. We utilise the cosine similarity between each pair of channels, as defined in Equation 1, for graph construction. This yields the correlation matrix $A^{cosine} \in [-1, 1]^{M \times M}$, where M is the number of channels. A filter function $f(x)$ with threshold k, as defined in Equation 2, is applied to A^{cosine} to filter for channels with a high cosine similarity (and thus high correlation), obtaining the adjacency matrix $A \in \{0, 1\}^{M \times M}$.

$$\text{Cosine Similarity}(x^{(i)}, x^{(j)}) = \frac{x^{(i)} \cdot x^{(j)}}{\|x^{(i)}\| \|x^{(j)}\|} \tag{1}$$

where $(x^{(i)}, x^{(j)})$ are channel pairs, $i = (1, \ldots, M)$, $j = (1, \ldots, M)$.

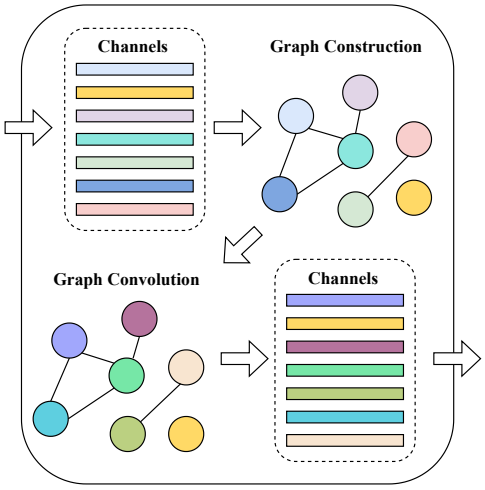

Figure 2. The overall approach for our graph convolution module. Each vertex in the graph represents a single channel. A graph is first constructed by the cosine similarity-based adjacency matrix, information is then propagated between connected vertices and outputted from the module.

$$f(x) = \begin{cases} 1 & \text{if } x > k, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Information Propagation Given the graph structure defined by the adjacency matrix $A \in \{0,1\}^{M \times M}$, we utilise a graph convolutional network for information propagation between the vertices. Due to computational limitations, a single layer graph convolutional network was used.

A single layer graph convolutional network is defined by Equation 3.

$$H = \sigma(\hat{A}XW) \tag{3}$$

where:

- σ is the activation function (ReLU (Fukushima, 1975)).
- $\hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$ is the normalized adjacency matrix, with I being the identity matrix used to introduce a self-connection and D the diagonal degree matrix. Each diagonal element D_{ii} represents the number of connections (or "degree") that vertex i has, including the self-connection. Normalization is necessary to scale the influence of each vertex's connections so that vertices with many connections don't dominate the feature aggregation process.
- X is the feature matrix for vertices, with dimensions $M \times U$, where M is the number of vertices and U is the number of features for each vertex.
- W is the weight matrix.

After information propagation by the graph convolutional network, the output H is the matrix of transformed vertex features, which effectively captures cross channel-dependencies.

3.2.3. Loss Function

As in the original PatchTST paper, we use the mean squared error (MSE) loss to measure the difference between the prediction and ground truth. The MSE loss is defined in Equation 4.

$$\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M}(y^{(i)} - \hat{y}^{(i)})^2 \tag{4}$$

where $\hat{y}^{(i)}$ is the prediction for a univariate channel i, $y^{(i)}$ is the ground truth for the corresponding univariate channel i, and M is the number of channels in the multivariate time series. The MSE loss essentially gets an average loss over all channels for the multivariate predictions.

3.2.4. Instance Normalization

As in the original PatchTST paper, reversible instance normalization (RevIN) (Kim et al., 2021) is adopted to mitigate the distributional shift present in time series data. In real-world situations, the statistical properties such as the mean and variance of a dataset can change over time. RevIN normalises each univariate input time series with zero mean and unit standard deviation, then reverses this process through denormalising the output. These processes correspond to the instance norm and instance denorm layers in Figure 1.

4. Data

4.1. Datasets

We selected the following 5 popular real-world multivariate datasets for model evaluation: ILI, 2 ETT datasets (ETTh1, ETTh2), Exchange, and Weather. These datasets from (Wu et al., 2021) are highly utilized as a benchmark for multivariate time series forecasting.(Nie et al., 2022) We split the datasets into training, validation, and testing sets with a ratio of 7:1:2, respectively. This same split ratio is used across all the SOTA time series models. The number of time steps and features for each dataset are shown in Table 1. Further detail on each dataset is as follows.

ILI[1] (Influenza-Like Illness) contains features of hospital visits related to influenza, such as the number of patients in different age groups, on a weekly basis.

ETT[2] (Electricity Transformer Temperature) records 7 indicators of an electricity transformer, such as power loads and oil temperature. The data is recorded over two years. There are 2 sets of data, the affixes '1' and '2' correspond to the machine number, 'h' corresponds to the time step

[1] https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html
[2] https://github.com/zhouhaoyi/ETDataset

frequency (1 hour).

Exchange (Lai et al., 2017) records the daily exchange rates of eight different countries ranging from 1990 to 2016.

Weather[3] contains meteorological indicators such as temperature, air pressure, and humidity for Germany in 2020.

Dataset	Features	Time steps
ILI	7	966
ETTh1	7	17420
ETTh2	7	17420
Exchange	8	7588
Weather	21	52696

Table 1. Statistics of datasets used for model evaluation.

4.2. Data pre-processing

All the datasets have been pre-processed by the original Autoformer paper (Wu et al., 2021) to be able to test accuracy of the MLTSF. The first column of each dataset is the time step and the other columns are the different features and their values during a specific time. These datasets are used across all MLTSF models including all of the SOTA models introduced in this paper.

5. Experiments and Results

5.1. Motivation and Task

We set out experiments to verify whether introducing cross-channel dependencies would enhance the original channel-independent PatchTST's (Nie et al., 2022) ability to produce more precise predictions in MLTSF tasks. Our task is to predict values across all M channels of each dataset with a fixed historical sequence length L over multiple prediction lengths/forecast horizons F, as defined in 3.1.

We compare the performance of 3Cformer with PatchTST, and with other SOTA and baseline models to assess 3Cformer's performance in the broader landscape.

5.2. Experiment Settings

Our experiment setting mainly follows that of PatchTST, with differences stated otherwise.

5.2.1. Baseline Models

We chose the following five SOTA models in MLTSF tasks as baseline models to compare results: transformer-based PatchTST (Nie et al., 2022), Autoformer (Wu et al., 2021), FEDformer (Zhou et al., 2022), Crossformer (Zhang & Yan, 2023) and one non-transformer based model DLinear (Zeng et al., 2023).

5.2.2. Patching Parameters

We set the patching parameters, as specified in 3.2, including patch length P, and stride S, to facilitate different dataset sizes whilst keeping the number of patches N uniform in all tasks. For ILI which is a smaller dataset, $L = 104$, and $F \in \{24, 36, 48, 60\}$, $P = 24$, $S = 2$, $N = \lfloor \frac{(104-24)}{2} \rfloor + 2 = 42$. For the rest of the datasets, we set $L = 336$, and $F \in \{96, 192, 336, 720\}$. $P = 16$, $S = 8$, $N = \lfloor \frac{(336-16)}{8} \rfloor + 2 = 42$.

5.2.3. Metrics

To evaluate the generalisation ability in various long-term prediction tasks, we use the mean square error (MSE) and the mean absolute error (MAE). The experiment results are present in Table 2. The MSE of the test set is calculated similar to Equation (4):

$$\textbf{MSE} = \frac{1}{T}\sum_{i=1}^{T}\frac{1}{M}\sum_{j=1}^{M}(y_j^{(i)} - \hat{y}_j^{(i)})^2 \tag{5}$$

where T stands for the number of samples in the test set, M is the number of channels, $\hat{y}_j^{(i)}$ is the predicted value and $y_j^{(i)}$ is the ground truth for the j^{th} channel of the i^{th} test sample.

The MAE of the test set is computed by using absolute value between prediction and ground truth instead:

$$\textbf{MAE} = \frac{1}{T}\sum_{i=1}^{T}\frac{1}{M}\sum_{j=1}^{M}|y_j^{(i)} - \hat{y}_j^{(i)}| \tag{6}$$

The squaring of MSE penalise large differences between prediction and ground truth more, which emphasizes outliers, while MAE reflects the average error between prediction and ground truth.

5.2.4. Hyperparameters

There are three core hyperparameters that we need to tune for our model: the cosine similarity threshold k, the learning rate for the overall model, and the learning rate for the graph convolution network. The **cosine similarity threshold** is set to k **= 0.6** for **all experiments**. We choose 0.6 as it creates a correlation for all of the datasets while being high enough to avoid noise. As seen in Table 4 in the Appendix, thresholds 0 to 0.4 all suffer from noise producing bad results. We were also able to see that 0.6 gave the best average MSE result for all metrics. For the overall model learning rate, we mostly followed the original PatchTST paper and set **learning rate** = **1e-4** for **all experiments**. Although, the original paper used a learning rate = 2.5e-3 for ILI, however, we set it to 1e-4 as the model was more stable and had better results during training, as seen in Table 5 of the Appendix. The model produced the best results with a learning rate of 1e-4 for all metrics except 36. In addition to the model's learning rate, we introduced a learning rate for the GCN since we noticed that during training the GCN layer would converge faster than other layers. We found that using 1e-4 for ETT and Exchange led to the models overfitting as not many correlations were being made. As a result, we set **GCN learning rate** = **1e-5** for **ETT** and **Exchange**, and **1e-4** for the **remaining experiments**. The full experiment could be seen in Table 6 of the Appendix and further discussed in Section 6.

[3] https://www.bgc-jena.mpg.de/wetter/

Models		3Cformer		PatchTST*		Dlinear*		FEDformer*		Autoformer*		Crossformer	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	0.157	0.209	**0.152**	**0.199**	0.176	0.237	0.238	0.314	0.249	0.329	0.156	0.218
	192	**0.196**	0.246	0.197	**0.243**	0.220	0.282	0.275	0.329	0.325	0.370	0.198	0.262
	336	**0.249**	**0.283**	0.265	0.319	0.339	0.377	0.351	0.391	0.583	0.543	0.266	0.295
	720	0.325	**0.334**	**0.320**	0.335	0.323	0.362	0.389	0.409	0.415	0.426	0.327	0.363
Exchange	96	0.093	0.214	0.093	**0.213**	**0.091**	0.220	0.148	0.278	0.197	0.323	N/A	N/A
	192	0.192	**0.314**	0.195	0.315	**0.184**	0.323	0.271	0.380	0.300	0.369	N/A	N/A
	336	0.365	0.442	0.354	**0.435**	**0.328**	0.436	0.360	0.500	0.509	0.524	N/A	N/A
	720	N/A	N/A	N/A	N/A	**0.975**	**0.781**	1.195	0.841	1.447	0.941	N/A	N/A
ILI	24	**1.400**	**0.781**	1.516	0.802	2.215	1.081	2.624	1.095	2.906	1.182	3.537	1.215
	36	**1.270**	**0.763**	1.447	0.802	1.963	0.963	2.516	1.021	2.585	1.038	3.559	1.222
	48	**1.583**	**0.860**	1.775	0.890	2.130	1.024	2.505	1.041	3.024	1.145	3.776	1.257
	60	**1.619**	**0.879**	2.128	0.999	2.368	1.096	2.742	1.122	2.761	1.114	3.932	1.285
ETTh1	96	**0.373**	**0.398**	0.375	0.399	0.375	0.399	0.376	0.415	0.435	0.446	0.402	0.418
	192	0.414	0.422	0.414	0.421	**0.405**	**0.416**	0.423	0.446	0.456	0.457	0.469	0.458
	336	**0.429**	0.437	0.431	**0.436**	0.439	0.443	0.444	0.462	0.486	0.487	0.588	0.540
	720	0.453	**0.465**	**0.449**	0.466	0.472	0.490	0.469	0.492	0.515	0.517	0.725	0.610
ETTh2	96	0.278	**0.340**	**0.274**	**0.336**	0.289	0.353	0.332	0.374	0.332	0.368	0.706	0.587
	192	0.344	0.386	**0.339**	**0.379**	0.383	0.418	0.407	0.446	0.426	0.434	0.855	0.689
	336	**0.330**	**0.380**	0.331	**0.380**	0.448	0.465	0.400	0.447	0.477	0.479	1.013	0.767
	720	**0.379**	**0.419**	**0.379**	0.422	0.605	0.551	0.412	0.469	0.453	0.490	1.131	0.800

Table 2. Multivariate results with different prediction lengths for all models on all datasets. Note that data for ILI and Exchange with PatchTST was produced by us and the rest of the models with * are from the PatchTST paper (Nie et al., 2022), data for Crossformer is from the Crossformer paper (Zhang & Yan, 2023). We set the prediction lengths F ∈ {24, 36, 48, 60} for ILI and F ∈ {96, 192, 336, 720} for the others. The best results are in **bold** and the second best are underlined.

5.3. Results

Table 2 shows all models' results on every MLTSF task. We can conclude our model's results in the following:

- **Weather** 3Cformer achieves the lowest MSE and MAE at prediction length 336. Other results either beat or closely trail behind PatchTST.
- **Exchange** 3Cformer achieves the lowest MAE at prediction length 192, but is beat by PatchTST or DLinear in other settings. We are unable to obtain results for 3Cformer and PatchTST for prediction length 720 as the dataset is of insufficient length for patching during training.
- **ILI** 3Cformer achieves exceptional results and surpasses all baseline models, reaching the lowest test MSE and MAE across all prediction lengths. Compared to PatchTST, which attains all second-best results, our model improves the MSE by **13.65%** and the MAE by **5.7%** on average.
- **ETTh1, ETTh2** 3Cformer achieves the best MSE and MAE at prediction length 96 for ETTh1, and predictions lengths 336 and 720 for ETTh2. Other results either beat or closely trail behind SOTA models.

Overall, our model demonstrates exceptional performance for the ILI dataset, attaining lower values in MSE and MAE in comparison to all SOTA models. It also achieves the best or near-best results on other datasets. We analyse these results further in Section 6.

6. Discussion

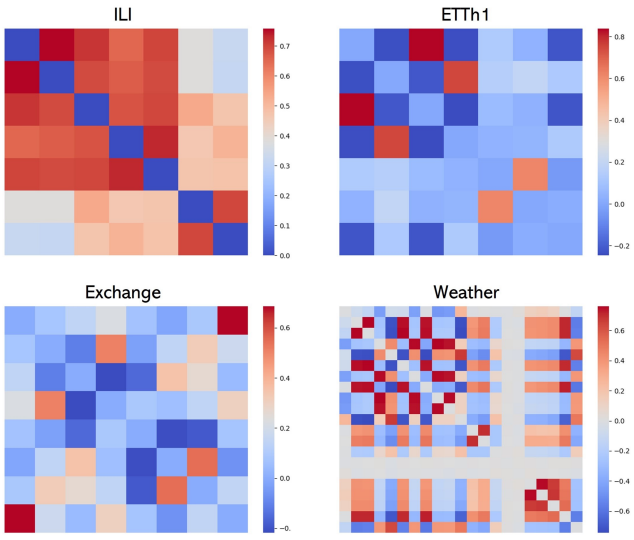

Figure 3. Cosine similarity matrices between the channels of the datasets: ILI, ETTh1, Exchange, Weather.

Model	Position Encoding	Encoder Layer	Decoder Layer
Autoformer	$O(1)$	$O(L \log L)$	$O\left((\frac{L}{2}+F)\log(\frac{L}{2}+F)\right)$
FEDformer	$O(1)$	$O(L)$	$O(F+\frac{L}{2})$
Crossformer	$O(\frac{L}{P})$	$O(M(\frac{L}{P})^2)$	$O(MF\frac{(F+L)}{P^2})$
PatchTST	$O(\frac{L}{P})$	$O(M(\frac{L}{P})^2)$	$O(MF\frac{L_{dim}}{P})$
3Cformer	$O(\frac{L}{P})$	$O(M^2(\frac{L}{P})^2)$	$O(MF\frac{L_{dim}}{P})$

Table 3. The time complexity of different models. L represents the size of the historical look-back window, M is the number of channels, P is the patch length, F is the future prediction horizons, dim is the dimension of the latent state.

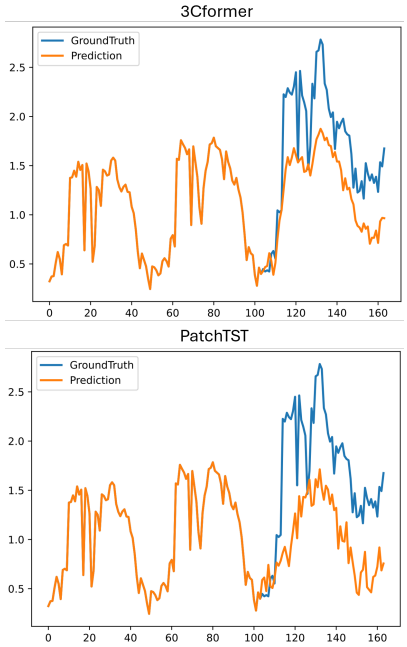

Figure 4. Prediction vs ground truth for the total number of visits inside the ILI dataset for 3Cformer and PatchTST.

6.1. Findings

Our model's main feature is to capture correlations between channels, therefore it is expected to yield higher accuracy on datasets containing highly correlated channels. The ILI dataset is one such dataset, it cotains data that are segmented into overlapping age categories (0-4 and 5-24 years) and parallel ILI measures (weighted and unweighted). These overlapping data points allows dense connectivity between channels to form during graph construction of the GCN, facilitating exchange of information. In contrast, the ETTh1 and ETTh2 datasets contain data that serve as distinct indicators for an electricity transformer's operation, each reflecting independent aspects of its performance, thus the connections within the GCN will be more sparse. Consequently, our model's performance barely improves on that of PatchTST for these datasets. Figure 4 visualises the predictions made for the univariate series: total number of hospital visits, in the ILI dataset for PatchTST and 3Cformer.

Looking at Figure 3, it is clear that the weather data is more interconnected across different features compared to the exchange and ETTh1 datasets, as it has more red squares indicating similarity between channels. However, 3Cformer does not distinctively beat the benchmarks set by PatchTST for this dataset. This could be because the similarity between different parts of the weather data are moderate (with cosine similarity scores ranging from 0.5 to 0.7) rather than very strong (above 0.8). As we set the cosine similarity threshold $k = 0.6$ for graph construction for all experiments, edges constructed between moderately similar channels could add noise to the GCN, making the model less effective. For datasets with a large number of features, where both moderate and strong similarities exist, we could investigate the use of higher thresholds to discount moderate similarities in graph construction to reduce noise.

For the exchange dataset, both 3Cformer and PatchTST are beat by DLinear, which achieves the lowest MSE. Intuitively, each feature of the exchange dataset corresponds to the daily exchange rate of a country over time and does not necessarily have a large influence each other. This is shown in Figure 3 where we see a low number of similar channels, and thus a sparsely connected graph in the GCN. Thus, DLinear may perform better as the forecasting task for exchange data may be more suited to be modelled as a univariate long-term time series forecasting (ULTSF) task, where predictions for each channel is made independently.

In summary, the performance of 3Cformer is closely tied to the underlying correlation patterns within the datasets. 3Cformer works best when there are lots of correlations between different channels, as it propagates information between these channels to introduce cross-channel dependencies.

6.2. Time Complexity

Table 3 shows the time complexity of different models. The patching operation first reduces the size of sequence length to $\frac{L}{P}$, the position encoding time complexity would therefore be O($\frac{L}{P}$). Because of the channel independence, PatchTST's time complexity for one attention layer is the attention weights times the number of channels M. However, because of the GCN, each channel makes connections with the other channels (Blakely et al., 2019) which makes our time complexity ($M^2(\frac{L}{P})^2$). This is PatchTST's complexity multiplied by M. The transformer decoder remains unchanged, therefore it has the same time complexity O($MF\frac{L_{dim}}{P}$) as PatchTST. The disadvantage of our model is

that the time complexity of the encoder is greater than other transformer models, and will scale quadratically with the number of channels M. However, with this additional cost, 3Cformer is able to capture cross-channel dependencies.

7. Future Work

In this section, we present some issues of 3Cformer and propose some measures and alternatives to serve as directions for future work.

As discussed in Section 5.2.4, we observed that the GCN layer converges faster than other layers. We lowered the learning rate of the GCN, while maintaining the overall model learning rate, to alleviate the effect of overfitting. Furthermore, the size of the graph within the GCN varies with the number of highly correlated channels in the data, further complicating the selection of an appropriate learning rate. We can employ a learning rate scheduler (Park et al., 2020) for the GCN, which can adjust the learning rate accordingly to the training and help the GCN converge stably. By adding the scheduler to the layer, the model is able to adapt to different datasets and tasks.

In the graph construction process, we can explore the use of other metrics to compute cross-channel dependencies, such as the Pearson correlation coefficient. By using notations defined in Equation 1, the equation for the Pearson correlation of channel pair $(x^{(i)}, x^{(j)})$ is defined in Equation 7. The Pearson correlation measures both the magnitude and direction of dependency between channels, allowing it to differentiate channels that share a similar linear trend (high magnitude) (Berkhin, 2006). On the other hand, cosine similarity only focuses on the direction between channels and does not consider magnitude. Pearson correlation could capture features that cosine similarity is unable to. To use Pearson correlation, we need to apply it on the dataset before inputting to the model, this also ensures that channels involved in information propagation stays the same for each iteration during training, potentially making the model more stable.

$$r_{x^{(i)}x^{(j)}} = \frac{\sum_k (x_k^{(i)} - \bar{x}^{(i)})(x_k^{(j)} - \bar{x}^{(j)})}{\sqrt{\sum_k (x_k^{(i)} - \bar{x}^{(i)})^2}\sqrt{\sum_k (x_k^{(j)} - \bar{x}^{(j)})^2}} \tag{7}$$

where k denotes the number of dimensions for $x^{(i)}$ and $x^{(j)}$, and $\bar{x}^{(i)}$ and $\bar{x}^{(j)}$ are respectively the means for $x^{(i)}$ and $x^{(j)}$.

Additionally, both cosine similarity and Pearson coefficient values sit within $[-1, 1]$. Our current implementation sets a positive threshold to create the adjacency matrix, disregarding all negative values. However, highly negatively correlated channels may also provide useful information. Future work could explore taking the absolute values of correlation metrics, thereby introducing edges between negatively correlated channels too.

Furthermore, future work could explore the placement of the GCN module. In our implementation, the GCN module is integrated within the transformer encoder, after the multi-head attention layer. The permutation-invariant nature of the self-attention mechanism could cause sequential information loss (Vaswani et al., 2017). Applying similarity calculations ahead of the transformer encoder may better preserve and capture relevant sequential information for finding cross-channel dependencies. A natural location for the GCN would be in-between the projection to embedding space and positional encoding (see Figure 1). In detail, after the multivariate data are transformed to patches and projected to embedding space, the GCN module will compute the adjacency matrix of the patches before they go through positional encoding.

In exploring other GNN architectures for long-term time series forecasting, MixHop (Abu-El-Haija et al., 2019) presents a promising alternative. While maintaining the same graph structure and construction principles as traditional GCNs, MixHop extends the approach to capturing cross-channel dependencies by going beyond only considering direct connections between channels. Instead, MixHop also takes into account the influence of channels that are indirectly connected, broadening the scope to include both direct and indirect dependencies between channels. This method allows for more comprehensive information propagation between channels, potentially enhancing the model's forecasting capabilities for datasets where channels are less correlated. To formalise this approach, we rewrite equation 3 to the following:

$$H^{(l+1)} = \sigma\left(\sum_{p=0}^{P} \hat{A}^p H^{(l)} W^{(l,p)}\right) \tag{8}$$

Comparing to GCN, MixHop adds an additional parameter P denoting the "hop", which means the number of step(s) from the current vertex to another through their connecting edge(s).

8. Conclusions

This paper introduces 3Cformer, a transformer-based model for MLTSF tasks which builds upon the channel independent PatchTST by incorporating cross-channel dependencies. We introduce a GCN module with a cosine similarity-based graph construction, within the transformer encoder of PatchTST. This allows for information propagation between highly correlated channels. Our model outperforms PatchTST for MLTSF on datasets that contain many highly correlated features, but has mixed performance, relative to PatchTST, on datasets with less correlated features.

We also identify aspects that remain to be improved on. Notably, 3Cformer does not achieve meaningful improvements on PatchTST for datasets in which channels represent relatively independent properties. Secondly, the specific implementation of the GCN module could be tweaked, including its placement within the overall architecture and alternative methods for graph construction. We have suggested possible improvements and research directions to guide future work.

References

Abu-El-Haija, Sami, Perozzi, Bryan, Kapoor, Amol, Alipourfard, Nazanin, Lerman, Kristina, Harutyunyan, Hrayr, Steeg, Greg Ver, and Galstyan, Aram. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing, 2019.

Berkhin, Pavel. A survey of clustering data mining techniques. In *Grouping multidimensional data: Recent advances in clustering*, pp. 25–71. Springer, 2006.

Blakely, Derrick, Lanchantin, Jack, and Qi, Yanjun. Time and space complexity of graph convolutional ..., Jun 2019. URL https://qdata.github.io/deep2Read//talks-mb2019/Derrick_201906_GCN_complexityAnalysis-writeup.pdf.

Box, George EP and Jenkins, Gwilym M. Some recent advances in forecasting and control. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 17(2): 91–109, 1968.

Brown, Tom B., Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, Agarwal, Sandhini, Herbert-Voss, Ariel, Krueger, Gretchen, Henighan, Tom, Child, Rewon, Ramesh, Aditya, Ziegler, Daniel M., Wu, Jeffrey, Winter, Clemens, Hesse, Christopher, Chen, Mark, Sigler, Eric, Litwin, Mateusz, Gray, Scott, Chess, Benjamin, Clark, Jack, Berner, Christopher, McCandlish, Sam, Radford, Alec, Sutskever, Ilya, and Amodei, Dario. Language models are few-shot learners, 2020.

Cao, Defu, Wang, Yujing, Duan, Juanyong, Zhang, Ce, Zhu, Xia, Huang, Congrui, Tong, Yunhai, Xu, Bixiong, Bai, Jing, Tong, Jie, et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33: 17766–17778, 2020.

Cleveland, Robert B, Cleveland, William S, McRae, Jean E, Terpenning, Irma, et al. Stl: A seasonal-trend decomposition. *J. Off. Stat*, 6(1):3–73, 1990.

Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, Uszkoreit, Jakob, and Houlsby, Neil. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

Fukushima, Kunihiko. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3): 121–136, 1975.

Guo, Shengnan, Lin, Youfang, Feng, Ning, Song, Chao, and Wan, Huaiyu. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 922–929, 2019.

Karita, Shigeki, Chen, Nanxin, Hayashi, Tomoki, Hori, Takaaki, Inaguma, Hirofumi, Jiang, Ziyan, Someki, Masao, Soplin, Nelson Enrique Yalta, Yamamoto, Ryuichi, Wang, Xiaofei, Watanabe, Shinji, Yoshimura, Takenori, and Zhang, Wangyou. A comparative study on transformer vs rnn in speech applications. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, December 2019. doi: 10.1109/asru46091.2019.9003750. URL http://dx.doi.org/10.1109/ASRU46091.2019.9003750.

Kim, Taesung, Kim, Jinhee, Tae, Yunwon, Park, Cheonbok, Choi, Jang-Ho, and Choo, Jaegul. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Kipf, Thomas N. and Welling, Max. Semi-supervised classification with graph convolutional networks, 2017.

Koprinska, Irena, Wu, Dengsong, and Wang, Zheng. Convolutional neural networks for energy time series forecasting. In *2018 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2018.

Lai, Guokun, Chang, Wei-Cheng, Yang, Yiming, and Liu, Hanxiao. Modeling long- and short-term temporal patterns with deep neural networks, 2017. URL https://arxiv.org/abs/1703.07015.

LeCun, Yann, Boser, Bernhard, Denker, John, Henderson, Donnie, Howard, Richard, Hubbard, Wayne, and Jackel, Lawrence. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

Matsubara, Yasuko, Sakurai, Yasushi, Van Panhuis, Willem G, and Faloutsos, Christos. Funnel: automatic mining of spatially coevolving epidemics. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 105–114, 2014.

Nie, Yuqi, Nguyen, Nam H, Sinthong, Phanwadee, and Kalagnanam, Jayant. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

Park, Jieun, Yi, Dokkyun, and Ji, Sangmin. A novel learning rate schedule in optimization for neural networks and it's convergence. *Symmetry*, 12(4):660, Apr 2020. doi: 10.3390/sym12040660.

Pascanu, Razvan, Mikolov, Tomás, and Bengio, Yoshua. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. Learning internal representations by error propagation. 1986. URL https://api.semanticscholar.org/CorpusID:62245742.

Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, and Monfardini, Gabriele. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Shi, Xingjian, Chen, Zhourong, Wang, Hao, Yeung, Dit-Yan, Wong, Wai-Kin, and Woo, Wang-chun. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

Sitikhu, Pinky, Pahi, Kritish, Thapa, Pujan, and Shakya, Subarna. A comparison of semantic similarity methods for maximum human interpretability. In *2019 artificial intelligence for transforming business and society (AITB)*, volume 1, pp. 1–4. IEEE, 2019.

Uremović, Niko, Bizjak, Marko, Sukič, Primož, Štumberger, Gorazd, Žalik, Borut, and Lukač, Niko. A new framework for multivariate time series forecasting in energy management system. *IEEE Transactions on Smart Grid*, 2022.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wen, Qingsong, Zhou, Tian, Zhang, Chaoli, Chen, Weiqi, Ma, Ziqing, Yan, Junchi, and Sun, Liang. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

Wu, Haixu, Xu, Jiehui, Wang, Jianmin, and Long, Mingsheng. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, 2021. URL https://arxiv.org/abs/2106.13008.

Wu, Zonghan, Pan, Shirui, Long, Guodong, Jiang, Jing, Chang, Xiaojun, and Zhang, Chengqi. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 753–763, 2020.

Zeng, Ailing, Chen, Muxi, Zhang, Lei, and Xu, Qiang. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 11121–11128, 2023.

Zhang, Yunhao and Yan, Junchi. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=vSVLM2j9eie.

Zhou, Tian, Ma, Ziqing, Wen, Qingsong, Wang, Xue, Sun, Liang, and Jin, Rong. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International conference on machine learning*, pp. 27268–27286. PMLR, 2022.

A. Appendix

A.1. Testing of different thresholds for correlation on the ILI dataset

Threshold		0		0.2		0.4		0.6		0.8	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ILI	24	1.425	0.796	1.424	0.796	1.417	0.792	1.400	0.879	**1.371**	**0.778**
	36	1.385	0.799	1.382	0.798	1.386	0.791	1.270	0.763	**1.244**	**0.743**
	48	1.695	0.877	1.694	0.877	1.694	0.877	**1.583**	**0.860**	1.676	0.884
	60	1.748	0.893	1.748	0.893	1.643	0.858	**1.619**	**0.879**	1.704	0.903
Mean		1.563	0.841	1.562	0.841	1.535	0.830	**1.468**	0.845	1.499	**0.827**

Table 4. Threshold experiments on the ILI dataset.

A.2. Testing of different learning rates on the ILI dataset

Learning Rate		2.5e-3		5e-4		1e-4	
Metric		MSE	MAE	MSE	MAE	MSE	MAE
ILI	24	1.694	0.880	1.557	**0.837**	**1.400**	0.879
	36	1.966	0.949	**1.248**	**0.755**	1.270	0.763
	48	2.173	0.996	2.170	1.018	**1.583**	**0.860**
	60	1.805	0.928	1.725	**0.876**	**1.619**	0.879

Table 5. Learning rate experiments on the ILI dataset.

A.3. Testing of 1e-4 GCN learning rate vs 1e-5

GCN Learning Rates		1e-4		1e-5	
Metric		MSE	MAE	MSE	MAE
ETTh1	96	0.379	0.404	**0.373**	**0.398**
	192	0.416	0.423	**0.344**	**0.386**
	336	0.434	0.439	**0.429**	**0.437**
	720	**0.451**	0.468	0.453	**0.465**
ETTh2	96	0.293	0.350	0.278	0.340
	192	0.351	0.391	**0.344**	**0.386**
	336	0.339	0.389	**0.330**	**0.380**
	720	0.402	0.440	**0.379**	**0.419**
Exchange	96	0.099	0.220	**0.093**	**0.214**
	192	0.203	0.324	**0.192**	**0.314**
	336	0.381	0.454	**0.365**	**0.442**
	720	N/A	N/A	N/A	N/A
ILI	24	**1.400**	**0.781**	1.468	0.805
	36	**1.270**	**0.763**	1.454	0.819
	48	**1.583**	**0.860**	1.781	0.913
	60	**1.619**	**0.879**	1.683	0.883

Table 6. GCN learning rate experiments on four datasets.