Identifying Deep Contrasting Networks from Time Series Data: Application to Brain Network Analysis

John Boaz Lee* Xiangnan Kong* Yihan Bao* Constance Moore†

Abstract

The analysis of multiple time series data, which are generated from a networked system, has attracted much attention recently. This technique has been used in a wide range of applications including functional brain network analysis of neuroimaging data and social influence analysis. In functional brain network analysis, the activity of different brain regions can be represented as multiple time series. An important task in the analysis is to identify the latent network from the observed time series data. In this network, the edges (functional connectivity) capture the correlation between different time series (brain regions). Conventional network extraction approaches usually focus on capturing the connectivity through linear measures under unsupervised settings. In this paper, we study the problem of identifying deep nonlinear connections under group-contrasting settings, where we have two groups of time series samples, and the goal is to identify nonlinear connections that are discriminative across the two groups. We propose a method called GCC (Graph Construction CNN) which is based on deep convolutional neural networks for the task of network construction. The CNN in our model learns a nonlinear edge-weighting function to assign discriminative values to the edges of a network. Experiments on a real-world ADHD dataset show that our proposed method can effectively identify the nonlinear connections among different brain regions. We also demonstrate the extensibility of our proposed framework by combining it with an autoencoder to capture subgraph patterns from the constructed networks.

1 Introduction

The task of inferring the structure of an underlying graph from a given set of time series is a problem with many practical applications. For instance, the activities of different users over time in an online social network can be explained by an influence graph which is unobserved [14]. From a marketing perspective, the ability to infer this graph from the given user activities is useful since it allows companies to target influential individuals on the network. Another example of network construction can be found in the study of functional brain networks [4, 5, 18] which has become quite popular recently. These networks can be constructed by measuring the correlation of the activity of different brain regions in a functional magnetic resonance imaging (fMRI) scan. Analyzing the networks gives researchers the ability to identify "clinically significant connectivity patterns of brain regions."

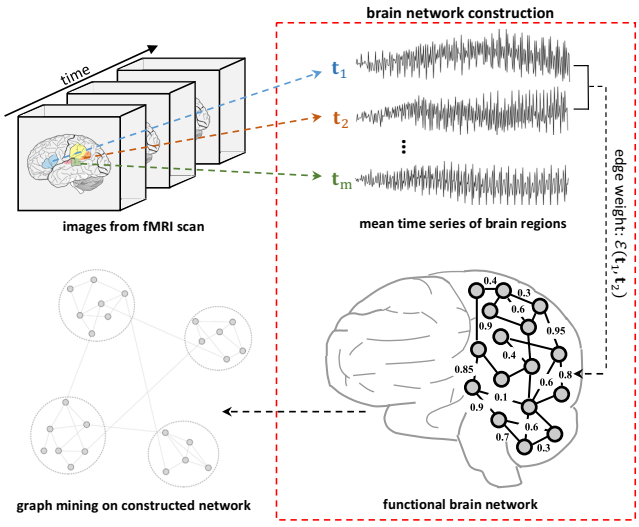


Figure 1: An example of constructing a functional brain network from the fMRI data of one human subject.

Network construction is an important step in network analysis. Figure 1 illustrates the steps involved when doing brain network analysis on fMRI data. It is clear that the network construction step is a crucial one. The quality of the results that one can hope to get by analyzing the brain network is dependent on the quality of the brain construction technique. In other words, the ability to find interesting and useful information from an inferred network is highly dependent on the quality of the construction process.

The problem of constructing a graph from time series data has attracted much attention recently [7, 8, 15, 20, 23]. Many conventional approaches use a linear measure of correlation to capture the relationship between different nodes in a network [2, 8]. However, in many real-world cases the correlation between different

*Worcester Polytechnic Institute
†University of Massachusetts Medical School

time series are highly nonlinear. Figure 2 shows a case when a linear measure like Pearson's correlation may assign the same correlation value to two pairs of time series that have different nonlinear relationships. A nonlinear measure, on the other hand, can give different values and is hence capable of capturing the difference. Other works [20] treat the problem as a supervised link prediction problem where the labels (*i*.e. whether there is a link or not) for each pair of time series is known. However, in many real-world cases the link labels are quite difficult to acquire while the label for the entire set of time series is relatively easy to get. For instance, in neuroimaging datasets the ground truth labels for the edges in a functional network is unknown but the label for each sample (*e*.g., whether the fMRI scan belongs to an ADHD patient or not) is given.

In the group-contrasting setting, only the set of time series and their labels are given. The network construction problem under this setting then corresponds to learning a discriminative nonlinear edge-weighting function with the following properties: (1) *Inter-group:* the weighting function tends to extract different edge weights for samples in different groups; and (2) *Intra-group:* the weighting function tends to extract similar edge weights for samples in the same group. The problem is challenging because the relationship between different time series can be highly nonlinear. Also, this relationship has to be learned from pairs of time series whose labels are not given.

In this paper, we tackle the important problem of group-contrasting network construction by proposing a technique, called GCC, based on convolutional neural networks (CNN). CNNs have proven to be a particularly suitable model for a wide variety of tasks [1, 10, 12]. In our architecture, the CNN tries to learn a discriminative nonlinear edge-weighting function for network construction. We combine multiple CNNs in a unified architecture so that training can be done via backpropagation on the error of the predicted group label. To the best of our knowledge, this is the first work that attempts to use deep learning to solve the problem of group-contrasting network construction.

The main contributions of this work can be summarized as follows:

- We propose a method based on CNNs that learns the nonlinear correlations between pairs of time series. The goal is to discover an edge-weighting function which can assign discriminative values to the edges of positive and negative networks.
- To combat the problem of overfitting, we propose an architecture that uses a technique similar to Dropout [17] to help the model generalize. We also demonstrate the general nature of the proposed framework by testing a variation of it where it is coupled with an autoencoder layer.
- We perform experiments on a real-world ADHD dataset. We show that it outperforms several baseline methods in terms of accuracy when the derived brain networks are used as input for classification.

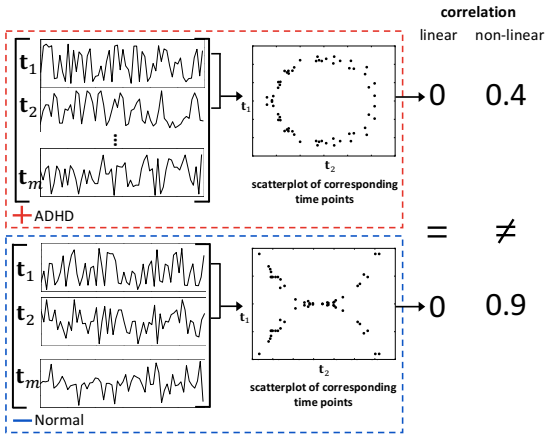


Figure 2: An example of a nonlinear group-contrasting measure on two sets of time series (for two human subjects over the same brain regions). The regions t_1 and t_2 have different nonlinear correlations.

2 Problem Definition

In this section, we briefly define the problem of network construction from multiple time series under the group-contrasting setting. Given a set of labeled time series, $\mathcal{D} = \{(\mathbf{T}^{(i)}, y^{(i)})\}$, $\mathbf{T}^{(i)} = \left(\mathbf{t}_1^{(i)}, \cdots, \mathbf{t}_m^{(i)}\right) \in \mathbb{R}^{n \times m}$ corresponds to the i-th example in $\mathcal{D}$ (*e*.g., the i-th human subject), which contains m time series (*e*.g., the activities of m brain regions). Here $\mathbf{t}_j^{(i)} \in \mathbb{R}^n$ denotes the j-th time series of the i-th sample. $y^{(i)} \in \{-1, 1\}$ denotes the group assignment of $\mathbf{T}^{(i)}$. For example, in neuroimaging applications, a sample of time series (*e*.g., the fMRI scans of one human subject) will be labeled as positive (*i*.e., $y^{(i)} = 1$), if the human subject has a certain neurological disorder, such as ADHD. Otherwise, the sample of time series $\mathbf{T}^{(i)}$ will belong to the normal control group, *i*.e., $y^{(i)} = -1$.

The goal of network construction is to infer a weighted undirected graph $G^{(i)} = \left(V, E^{(i)}, \mathcal{E}^{(i)}\right)$ from each time series sample $\mathbf{T}^{(i)}$. Here the set of vertices $V = \{v_1, \cdots, v_m\}$ corresponds to the set of m variables in the time series (*e*.g., the m brain regions in neuroimaging applications). Following previous works [11], we assume that V is shared across all samples in $\mathcal{D}$. The edge set $E^{(i)} \subseteq V \times V$ is the set of links between these variables (brain regions), and $\mathcal{E}^{(i)} \colon E^{(i)} \mapsto \mathbb{R}$

is a weighting function that maps each edge in the graph to a weight. The weight associated with each edge $\mathcal{E}^{(i)}(v_p, v_q)$ represents the nonlinear correlations between the two time series $\mathbf{t}_p^{(i)}$ and $\mathbf{t}_q^{(i)}$ under the group-contrasting setting. In group-contrasting settings, we want to learn a *discriminative* edge-weighting function which has the following properties: (1) *Inter-group*: the weighting function tends to extract different edge weights for a pair of samples $\mathbf{T}^{(i)}$ and $\mathbf{T}^{(j)}$, if they belong to different groups, *i*.e., $y^{(i)} \neq y^{(j)}$; (2) *Intra-group*: If $y^{(i)} = y^{(j)}$, the edge weights on the two samples should have similar values.

3 Methodology

3.1 CNN Background Our proposed framework is based on CNNs, which are built by stacking one or more pairs of convolution and max-pooling layers together [12]. A convolution layer learns a set of filters, whose receptive fields are each relatively small. Each filter, however, is replicated across the entire input space with the replicated units sharing parameters with the original filter. The output of a convolution between the input and a particular filter is a feature map; when multiple filters are defined we get multiple feature maps. A convolution layer is usually connected to a max-pooling layer, which produces down-sampled versions of the feature maps by taking only the maximum value in subregions of a feature map.

Figure 3(a) shows an architecture with a convolution layer followed by a max-pooling layer. Here, the input is a single time series. In this architecture, each filter learns to identify a particular temporal feature and activates when this is found. When a CNN is given time sequences, the filters have been shown to identify activation peaks, periodic changes, and zero-crossing frequencies in a time-window [20].

3.2 Proposed Framework

3.2.1 CNN Component In our proposed architecture, the main component is a CNN which takes as input a pair of time signals, corresponding to two nodes in the network being constructed. The CNN then learns to output a value which is the edge-weight between these nodes. Given two time signals $\mathbf{t}_i$ and $\mathbf{t}_j$, we concatenate these to form a $2 \times n$ matrix

$$\mathbf{X} = \left[\mathbf{t}_i,\ \mathbf{t}_j\right]^\top$$

where n here is the length of the time series.

Since the dimension of the input data is constrained and a CNN in our framework always takes a pair of time series, we define the two kinds of convolution that can be performed.

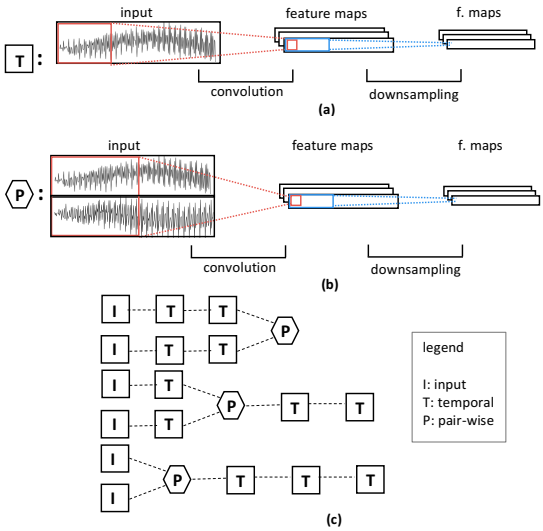


Figure 3: (a) A temporal convolution followed by downsampling; (b) Pair-wise convolution over two time series; (c) Three different CNN architectures.

Temporal: specifically, the feature maps corresponding to filters whose dimensions are $1 \times w$ can be obtained by

$$\begin{aligned} \mathbf{M}_i^k &= \sigma\left((\mathbf{W}^k * \mathbf{t})_i + b_k\right) \\ &= \sigma\left(\sum_{x=0}^{w-1} \mathbf{W}_x^k \mathbf{t}_{i+x} + b_k\right) \end{aligned} \tag{3.1}$$

where $*$ is the convolution operator, $\mathbf{M}^k$ is the k-th feature map, $\mathbf{W}^k$ represents the weights for the corresponding filter, b_k is the bias term, $\mathbf{t}$ is a time series in the input, and w is an arbitrary filter width. This first type of filter learns to identify the salient temporal features on the different input time series separately.

Pair-wise: similarly, feature maps corresponding to filters that have dimension $2 \times w$ can be obtained by

$$\begin{aligned} \mathbf{M}_{ij}^k &= \sigma\left((\mathbf{W}^k * \mathbf{X})_{ij} + b_k\right) \\ &= \sigma\left(\sum_{x=0}^{1}\sum_{y=0}^{w-1} \mathbf{W}_{xy}^k \mathbf{X}_{i+x,j+y} + b_k\right) \end{aligned} \tag{3.2}$$

where $\mathbf{M}^k$, $\mathbf{W}^k$, $\mathbf{X}$, and b_k represent the same things. In this second kind of convolution, the filter compares the values in both time series and squashes the two time series down to a single dimension. In both equations, σ is a nonlinearity, for instance it may be a *tanh* function.

CNNs built with these two kinds of filters can be arbitrarily deep and can have an arbitrary number of filters at each layer. There is only one constraint, at some stage in the network, a pair-wise convolution has to be applied so that the time signals are compared but this can be specified at any layer. Also, a pair-wise

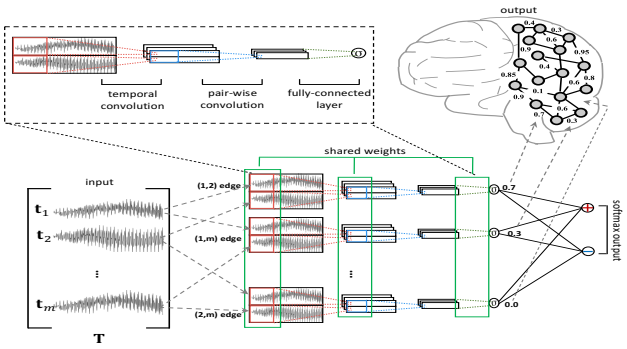


Figure 4: Multiple CNNs integrated into one network. All the CNN components share the same parameters. The final layer is a classification layer.

convolution layer only appears once whereas one can choose to have an arbitrary number of temporal convolution layers. Figure 3(a) and 3(b) show illustrations of the two kinds of convolution layers.

Three possible CNN configurations based on the convolutions we defined are shown in Figure 3(c). Here, it is assumed that the convolution layers are each followed by pooling layers. In the first case, the network extracts salient temporal features and summarizes (via the pooling layers) the two time signals separately before they are compared via the pair-wise convolution layer. Initially, the network in the second example works much like the one that was shown before. However, here we demonstrate that additional temporal layers can be added after a pair-wise convolution to extract features from the time series created by joining the two original series. In the last case, we compare the two time series immediately before more temporal layers are used to learn features from the joint time series.

A CNN always has as its last layer a fully-connected layer which outputs a single value. This layer summarizes all the information learned by the earlier layers and outputs a single edge-weight value.

3.2.2 Integrating Multiple CNNs The CNN that we have shown so far only compares two time signals. In our setting, it is assumed that we do not have the ground truth data on the edges of the network we are trying to infer, thus it is not clear how supervised training can be done. To do training on a network with m regions, we can combine multiple CNNs in parallel, with each one tasked to compute the edge-weight for one of the $\frac{m(m-1)}{2}$ undirected edges. Figure 4 shows an illustration of this.

The output of each of the CNNs can then be stacked together and fed as input to a Logistic Regression (LR) layer which predicts the label of the network. Since the architecture is quite general, we can easily replace the LR layer with an arbitrary set of layers. Taking the output of the CNN layers (*i*.e. the edge-weights of the inferred network) and passing this as input to an LR layer with *softmax* activations can be formulated as

$$\begin{aligned} \mathrm{P}(Y=i|\mathbf{W},\mathbf{x},\mathbf{b}) &= \mathrm{softmax}_i(\mathbf{W}\mathbf{x}+\mathbf{b}) \\ &= \frac{e^{\mathbf{W}_i\mathbf{x}+b_i}}{\sum_j e^{\mathbf{W}_j\mathbf{x}+b_j}} \end{aligned} \tag{3.3}$$

where $\mathbf{W}_i$ and b_i are the weights and bias terms of the LR layer corresponding to the *softmax* output associated with label i, and $\mathbf{x}$ is the output from the CNN layer. The predicted value can then simply be the label which obtained the maximal probability:

$$\hat{y} = \arg\max_i \mathrm{P}(Y=i|\mathbf{W},\mathbf{x},\mathbf{b})$$

An advantage of this architecture, where the classifier layer is coupled with the network construction layer, is that the error is back-propagated during training to the earlier layers which can help the CNN learn a discriminative measure of correlation.

An important thing to note here is that all of the CNNs in the model share the same set of weights. This makes sense since we are trying to learn a single edge-weighting function. Intuitively, the network attempts to learn a discriminative nonlinear edge-weighting function that will allow the LR layer to better discriminate between positive and negative samples. It is also important to note here that even though the model can be quite large, the number of parameters that the neural network learns is actually relatively small since all the CNN components share the same set of filters and, in practice, each filter is quite small.

3.2.3 Dropout Because neuroimaging experiments are usually quite costly to conduct, most datasets only contain a limited number of samples. For instance, the dataset we use in this study only has 776 samples. Even fewer samples - a total of 77 - were used in [24]. One thing that is problematic when attempting to learn from a small sample is the increased chances for the model to overfit the data. This is further compounded by the fact that brain data is known to be quite noisy [24].

To help the model generalize, and to speed up the training process, we introduce a parameter called the dropout rate $\delta = [0, 1]$. Dropout [17] is a technique used in deep architectures which randomly excludes nodes (and their connections) from a neural network during training to prevent the nodes from being too reliant on each other. This has been shown to be a powerful regularization technique and in many cases dropout has outperformed standard ℓ_1 and ℓ_2 regularization.

To implement dropout in our scenario, we do training on a smaller version of our original neural network where only a fraction $(1-\delta)$ of the CNNs are present. In other words, we try to learn an edge-weighting function given only a fraction of the time signals. Before every training forward-pass, we randomly select $1-\delta$ of the time signals and we swap the corresponding LR weights from the larger network to the smaller network used for training. The weights in the CNN components of both networks are shared so there is no need to swap those. During testing, we simply use the large network for prediction. Also, we make sure to rescale the input values to the LR layer by $1-\delta$ during the testing phase since the weights that were learned were based on a model that had access to only a fraction of the input.

3.2.4 Training via Backpropagation We use a negative log-likelihood cost function, its formulation is

$$\mathcal{L}(\theta, \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \log \Big(\mathrm{P}(Y = y^{(i)} | \mathbf{X}^{(i)}, \mathbf{W}, \mathbf{b}) \Big) \tag{3.4}$$

where $\theta = \{\mathbf{W}, \mathbf{b}\}$ is the set of model parameters, $\mathcal{D}$ is our training set, and $\mathbf{X}^{(i)}$ and $y^{(i)}$ are the set of signals and the label, respectively, of the i-th training sample. With regularization, the cost function can be written as

$$\mathcal{C}(\theta, \mathcal{D}) = -\mathcal{L}(\theta, \mathcal{D}) + \lambda_1 ||\mathbf{W}||_1 + \lambda_2 ||\mathbf{W}||_2^2 \tag{3.5}$$

where the second and third terms are the ℓ_1 and ℓ_2 regularization terms. The parameters λ_1 and λ_2 allow us to specify the amount of regularization to apply.

We can then use gradient descent after each forward-pass to update each of the parameters in our model as follows

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{C}(\theta, \mathcal{D}) \tag{3.6}$$

where α is the learning-rate. This can be done efficiently using backpropagation. We just have to sum the gradients for the parameters over all the connections where they are defined since these are shared.

4 Experiments and Results

4.1 Dataset In this study, we use the ADHD-200[1] dataset from the 1000 Functional Connectomes Project, which was made available by the Neuro Bureau[2]. The dataset contains the resting-state fMRI scans of 776 subjects, 491 of whom were typically developing individuals and the rest of whom were diagnosed with ADHD.

[1] http://fcon_1000.projects.nitrc.org/indi/adhd200/

[2] http://www.neurobureau.org/

Table 1: Classification accuracy of the various methods.

Method	Accuracy
GLASSO	53.04
CORR without thresholding	58.04
CORR with thresholding	58.21
GCC	**64.11**

The fMRIs, which were acquired in at least the $3 \times 3 \times 3$mm space, were preprocessed using AFNI[3]. During preprocessing, the first 4 echo-planar imaging volumes were removed and slice timing correction was applied. The scans were reoriented into the Right Posterior Inferior orientation, and smoothed with a 6mm Full Width at Half Maximum Gaussian kernel. A band-pass filter (0.009-0.08 Hz) was also applied to remove frequencies not implicated in resting state functional connectivity. A set of sequences of the average activation values averaged over voxels in 116 regions of interest (ROI), as specified by the the Anatomical Automatic Labeling [19] atlas, was then derived from the fMRI scan. The time series length we used was 74. Z-score normalization was applied to the time sequences.

We explicitly chose a dataset where the ground truth for the underlying network is unavailable because we wanted to work on the task of network construction in the group-contrasting setting. In many real-world cases, the network ground truth is unavailable but network construction is still useful for discovering differences between networks from different groups.

4.2 Compared Methods We evaluated the performance of the following methods:

- CORR: One popular technique for constructing a brain network involves calculating the Pearson's correlation [2] for pairs of time signals corresponding to the different regions. Since the calculated correlation matrix is usually quite dense, thresholding can be applied to remove edges that fall below a certain threshold.
- GLASSO: More recently, considerable attention has been given towards estimating a sparse inverse covariance matrix which represents the connections in the brain. This technique is a step up over other models based on simpler measures because it only retains edges between brain regions whose time signals are conditionally dependent on each other. The Graphical Lasso is a good method for this [6, 8] and it has been demonstrated to be a suitable method in many domains.

[3] https://afni.nimh.nih.gov/

Table 2: Best performance of GCC under various CNN architectures. The parameter setting that achieved the highest accuracy is shown. Here "conv" indicates the convolution filter dimension, "pool" indicate the pooling filter dimension, and "size" means the number of filters.

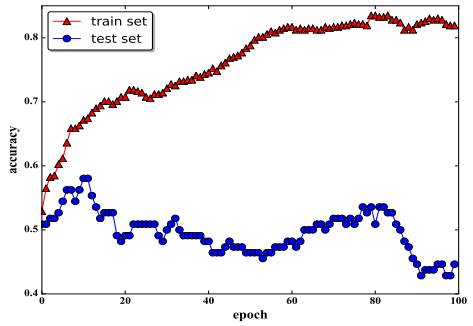
	GCC Architecture															Ideal Parameters			Ave.
	layer 1			layer 2			layer 3			layer 4			layer 5						
ver.	conv	pool	size	conv	pool	size	conv	pool	size	conv	pool	size	conv	pool	size	ℓ_1	ℓ_2	α	Acc.
1	(2,1)	(1,1)	4	(1,7)	(1,2)	3	(1,5)	(1,3)	3	(1,10)	(1,1)	1	–	–	–	0.0	0.0	0.01	64.11
2	(2,1)	(1,1)	4	(1,6)	(1,3)	3	(1,4)	(1,2)	2	(1,3)	(1,2)	2	(1,4)	(1,1)	1	0.0	0.0	0.01	61.61
3	(1,9)	(1,3)	2	(1,5)	(1,2)	3	(1,4)	(1,2)	3	(2,1)	(1,1)	1	(1,3)	(1,1)	1	0.0001	0.0001	0.01	62.32
4	(2,3)	(1,3)	5	(1,5)	(1,4)	8	(1,5)	(1,1)	1	–	–	–	–	–	–	0.0001	0.0	0.05	60.54
5	(2,6)	(1,3)	2	(1,4)	(1,2)	2	(1,3)	(1,2)	2	(1,4)	(1,1)	1	–	–	–	0.0	0.0	0.10	57.86

- GCC: This is our proposed method which uses a CNN to learn a nonlinear discriminative edge-weighting function. We evaluate multiple versions of the proposed method by trying different architectures for the CNN.

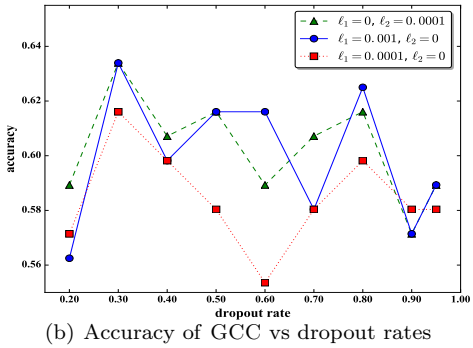
Since we do not have the ground truth connectivity information for each of the brain networks, one way to evaluate the quality of the constructed networks is to test the accuracy of a classifier when the constructed networks are given as input in a classification task. For a fair comparison, we use the same classifier (Logistic Regression) to test each of the models. We performed 5-fold cross validation on the dataset, where both the training and the test sets have been balanced, and report the average accuracy of each method. We do a grid search on the parameters of each method to identify the parameter settings that yield the highest accuracy; this is done using cross validation. For GLASSO, this is the regularization parameter, and for CORR it is the threshold value. Additionally, we do a grid search on various values for the ℓ_1 and ℓ_2 penalties for the LR. For our model, we also do a grid search on the values for α, δ, and the number of training epochs.

4.3 Performance Evaluation Table 1 lists the average accuracy of the different methods. It is interesting, and a little surprising, to note that in this case, the method based on Graphical Lasso only performs a little better than random guessing. For the correlation-based method, when thresholding (at 0.25) is applied the performance increases slightly. Our proposed method, however, clearly outperforms the different baselines. It can be noted that the performance of the different methods are all quite low, this is due to the fact that the dataset is particularly noisy[4].

We also performed tests on various architectures of the CNN component in our network. Table 2 lists the performance of the top architectures we tested with their ideal parameter settings. Each of the convolution



(a) Accuracy of GCC without regularization



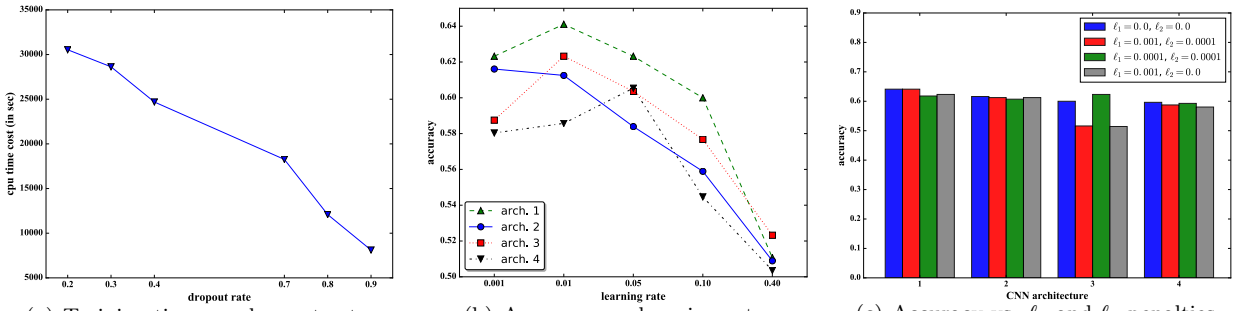
(b) Accuracy of GCC vs dropout rates

Figure 5: Performance under various dropout settings.

layer in the models we tested is followed by a pooling layer. From the results, it seems that doing a pair-wise convolution on the two signals first, which is performed by the $(2,1)$ convolution filter in the first layer, before summarizing the joint time signal works best. Also, it is interesting to note that the best results come from models that do not perform max-pooling after the two signals are compared. The $(2,1)$ convolution layer is followed by a pooling layer of dimension $(1,1)$, which is to say no pooling is performed. Also, architectures that compared multiple values during the pair-wise convolution – in our case, the models with the convolution filters of dimension $(2,3)$ and $(2,6)$ – fared worse than the others that we tested. Their result were much lower.

4.4 Parameter Study In this sub-section, we examine the different parameters used in our proposed model.

[4] http://fcon_1000.projects.nitrc.org/indi/adhd200/results.html



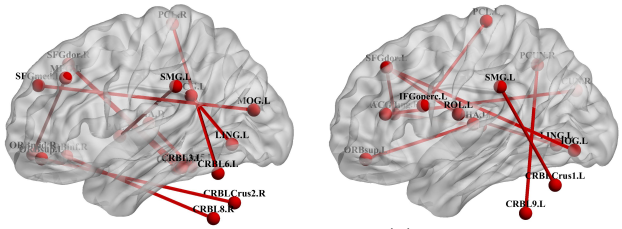
(a) Training time vs. dropout rates (b) Accuracy vs. learning rates (c) Accuracy vs. ℓ_1 and ℓ_2 penalties

Figure 6: The performance of our proposed method GCC under various parameter settings.

Figure 5(a) shows the performance of our model (with an LR layer) when doing classification on the dataset when no regularization is applied. It is clear to see that the model quickly overfits. In the first 10 epochs, the accuracy on the test set increases together with the accuracy on the training set which seems to suggest that the CNN is learning a general edge-weighting function that helps the LR layer discriminate between positive and negative samples. However, beyond that, the accuracy on the test set drops and remains pretty constant while the accuracy on the training set continues to increase steadily as the model starts to become overly complex, overfitting the data. This shows us that regularization is important to improve the generalization power of our model.

Dropout rate: The dropout rate δ controls the fraction of the input that is revealed to the model during training. Figure 5(b) shows the accuracy of our top architecture (with other parameters fixed and over a single fold) when different dropout rates are used. As expected, the performance is low when a very high dropout rate is applied ($90\% - 95\%$) since the data given to the model is too limited. Similarly, the accuracy is low when the dropout rate is too small since the model easily overfits. Surprisingly, the performance of the model at $\delta = 0.8$ is consistently high and it almost matches the performance of the model when the dropout is at 30%. This suggests that a moderately high amount of dropout is good and will actually help the network generalize better. In fact, the model that achieved the highest accuracy overall in our tests had 0.8 dropout. An intuitive way to look at why a a large dropout rate is good is to realize that since the weights for all the CNN components are shared, the network does not actually have to look at all pairs of time signal to learn a discriminative edge-weighting function.

The dropout rate also affects the speed of the algorithm. Since a fewer number of the CNN components are active when the dropout rate is high this speeds up the training process. Figure 6(a) shows the average time in seconds it takes for the training algorithm to execute 100 epochs on a machine with a 2.5GHz Intel Xeon CPU and 16GB of memory. It can be seen that there is around a 3-time speedup in training time when the dropout is increased from 0.2 to 0.8. In practice, we expect the dropout rate to be relatively high since the weights in the CNN layer are shared so learning from a subset of the available time series is a reasonable.



(a) Subnetwork 1 (b) Subnetwork 2

Figure 7: The two most discriminative sub-networks discovered from GCC + autoencoder.

Learning rate: Figure 6(b) shows the accuracy of the top-4 architectures listed in Table 2 under different learning rates. With the exception of the fourth architecture, which performed best with $\alpha = 0.05$, the architectures we tested worked well with $\alpha = 0.01$.

Other parameters: Figure 6(c) show the accuracy of the various architectures when different ℓ_1 and ℓ_2 penalties are applied. In most cases, the best results are achieved when both ℓ_1 and ℓ_2 are set to zero. This seems to suggest that there is no longer a need to apply ℓ_1 or ℓ_2 regularization if dropout is being applied. Finally, we found the ideal number of epochs for training to be typically between 25 and 80.

4.5 Case Study: Autoencoder for discriminative sub-network mining An autoencoder is a type of fully-connected neural network which has been found to be particularly useful as a model for feature learning [21]. It is able to perform well in this task because it is trained to reconstruct its own input by decoding it from a lower-dimensional encoding. Hence, it learns to identify a low-dimensional embedding that can faith-

fully represent the original input.

In the case when there is only a single hidden layer, the encoder part of the autoencoder takes the input $\mathbf{x} \in \mathbb{R}^n$ and maps it onto $\mathbf{z} \in \mathbb{R}^m$, where typically $m < n$. Formally, the output of the encoder is

$$\mathbf{z} = \sigma_1(\mathbf{W}_e\mathbf{x} + \mathbf{b}_e)$$

where $\mathbf{W}_e$ is the set of weights for the encoder, and $\mathbf{b}_e$ is the bias vector. The decoder then takes $\mathbf{z}$ and maps it back to $\mathbb{R}^n$. The output of the decoder is

$$\mathbf{x}' = \sigma_{\mathbf{2}}(\mathbf{W_d}\mathbf{z} + \mathbf{b_d})$$

here $\mathbf{W}_d$ and $\mathbf{b}_d$ are defined similarly, and $\mathbf{x}'$ is the reconstructed input. The model is trained to minimize the reconstruction error, for instance the squared difference between the original and reconstructed values.

Recently, it has been shown that an autoencoder can discover discriminative sub-network patterns [24] from a given brain network. When a set of links in a brain network are given as input to an autoencoder, each node in the first hidden layer of the autoencoder captures a discriminative sub-network pattern from the given input. The weights on the edges corresponding to the first hidden layer capture the strength of each link in the sub-network pattern.

For this test, instead of connecting our CNN outputs to an LR layer, we connect them to an autoencoder. The average activations for each node in the first hidden layer of the autoencoder can then be calculated separately for positive and negative inputs. The absolute difference between the averaged activations for the two groups of input indicate the ability of a particular node to discriminate between positive and negative samples. This allows us to identify discriminative sub-network patterns from the given samples. We trained our model and selected the two neurons corresponding to the most discriminative sub-networks. We visualized the most significant connections from the two sub-networks using BrainNet Viewer [22]; this is shown in Figure 7.

It can be observed that the two most discriminative brain sub-network patterns are quite different, this shows that there is potential for the model to discover interesting discriminative sub-network patterns from the data which may be useful during further analysis. We have also shown here that it is possible to combine a brain network construction layer with a subgraph-mining layer in a unified framework. Here the errors in the subgraph-mining layers can be back-propagated to help the CNNs learn a better edge-weighting function.

5 Related Work

Much research utilizing neuroimaging data has been published in recent years. One active area of research is the automatic diagnosis of brain disease from these images and many technique, some of which use deep learning, have been proposed. Plant et al. proposed techniques that use Support Vector Machines, Bayes classifiers, and Voting Feature Intervals for the prediction of patients with Alzheimer's disease [16]. More recently, a number of papers have been published which used deep learning models [3, 13]. In [3], a CNN that learns a low-dimensional manifold of the original neuroimage was employed to improve the accuracy of Alzheimer's diagnosis. The method in [13] proposed a deep learning model that predicted missing multi-modal information to generate more features to aid in classification. For instance, given a patient's MRI data, the proposed model attempted to predict the corresponding PET patterns. However, all of the above-mentioned techniques only considered the raw neuroimaging data and analysis was not performed on a brain network of the data.

Another body of work focuses on functional connectivity modeling which constructs a brain network where the weight of edges capture the covarying pattern of the signals associated with the brain regions [2, 7, 8, 15, 23]. The earlier works [2] use simpler measures like correlation to build the brain network while, more recently, the focus has been to learn a sparse network representation of the brain. One method that does the latter is [23] which uses the Graphical Lasso method to learn a sparse inverse covariance matrix which has the nice property guaranteeing that non-zero entries in the matrix will be between variables that are conditionally dependent. The only assumption is that the observations are drawn from a multivariate Gaussian distribution. The majority of the work in this area, however, assume that functional relation between brain regions can be captured by linear measures. In our framework, we attempt to learn a nonlinear function using a deep learning approach.

Deep learning models have been used to predict the link between two brain regions given the signal corresponding to these regions. A notable example can be found in [20]. One assumption, however, of the model in [20] is that the ground truth labels for each edge in the network is known and the problem is thus reduced to a supervised link prediction problem. This is not a realistic assumption to make since we do not have the capability to exhaustively map the functional (and even neuronal) connections of the brain in most cases. In contrast, in this work we only assume that the actual label of the brain network is known (e.g. whether the data belongs to an Alzheimer's patient or not).

Recently, there has also been a growing body of work whose focus is on mining brain networks [4, 9]. In [4], for instance, a deep learning architecture was designed that uses multiple side-view information in guid-

ing the model in selecting discriminative subnetwork features from a brain network. One difference between their work and ours is their use of side-view information, which may not always be available. The interested reader is referred to [9] for a survey of the field.

6 Conclusion

In this paper, we studied the problem of learning a nonlinear edge-weighting function for network construction of time series data in a group-contrasting setting. The model uses components based on CNNs which are constructed with two different kinds of convolution. Experiments on a real-world dataset show the effectiveness of the approach. We also demonstrate the extensibility of our proposed method by testing a variation of it that has an autoencoder component which was able to identify discriminative sub-network patterns from the data.

References

[1] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM TASLP*, 22(10):1533–1545, 2014.

[2] N. P. Azari, S. I. Rapoport, C. L. Grady, M. B. Schapiro, J. A. Salerno, A. Gonzales-Aviles, and B. Horwitz. Patterns of interregional correlations of cerebral glucose metabolic rates in patients with dementia of the Alzheimer type. *Neurodegeneration*, 1(1):101–111, 1992.

[3] T. Brosch and R. Tam. Manifold learning of brain MRIs by deep learning. In *MICCAI '13*, pages 633–640.

[4] B. Cao, X. Kong, J. Zhang, P. S. Yu, and A. B. Ragin. Mining brain networks using multiple side views for neurological disorder identification. In *ICDM '15*, pages 709–714.

[5] I. Davidson, S. Gilpin, O. Carmichael, and P. Walker. Network discovery via constrained tensor analysis of fMRI data. In *KDD '13*, pages 194–202.

[6] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[7] K. J. Friston. Functional and effective connectivity in neuroimaging: a synthesis. *Human brain mapping*, 2(1-2):56–78, 1994.

[8] S. Huang, J. Li, L. Sun, J. Ye, A. Fleisher, T. Wu, K. Chen, and E. Reiman. Learning brain connectivity of Alzheimer's disease by sparse inverse covariance estimation. *NeuroImage*, 50(3):935–949, 2010.

[9] X. Kong and P. S. Yu. Brain network analysis: a data mining perspective. *ACM SIGKDD Explorations Newsletter*, 15(2):30–38, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS '12*, pages 1097–1105.

[11] C. Kuo, X. Wang, P. Walker, O. Carmichael, J. Ye, and I. Davidson. Unified and contrasting cuts in multiple graphs: Application to medical imaging segmentation. In *KDD '15*, pages 617–626.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] R. Li, W. Zhang, H.-I. Suk, L. Wang, J. Li, D. Shen, and S. Ji. Deep learning based imaging data completion for improved brain disease diagnosis. In *MICCAI '14*, pages 305–312.

[14] Y. Li, W. Chen, Y. Wang, and Z.-L. Zhang. Influence diffusion dynamics and influence maximization in social networks with friend and foe relationships. In *WSDM '13*, pages 657–666.

[15] A. R. McIntosh, F. L. Bookstein, J. V. Haxby, and C. L. Grady. Spatial pattern analysis of functional brain images using partial least squares. *NeuroImage*, 3(3):143–157, 1996.

[16] C. Plant, S. J. Teipel, A. Oswald, C. Böhm, T. Meindl, J. Mourao-Miranda, A. W. Bokde, H. Hampel, and M. Ewers. Automated detection of brain atrophy patterns based on MRI for the prediction of Alzheimer's disease. *NeuroImage*, 50(1):162–174, 2010.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[18] L. Sun, R. Patel, J. Liu, K. Chen, T. We, J. Li, E. Reiman, and J. Ye. Mining brain region connectivity for Alzheimer's disease study via sparse inverse covariance estimation. In *KDD '09*, pages 1335–1344.

[19] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot. Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *NeuroImage*, 15(1):273–289, 2002.

[20] V. Veeriah, R. Durvasula, and G.-J. Qi. Deep learning architecture with dynamically programmed layers for brain connectome prediction. In *KDD '15*, pages 1205–1214.

[21] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML '08*, pages 1096–1103.

[22] M. Xia, J. Wang, and Y. He. Brainnet viewer: A network visualization tool for human brain connectomics. *PLoS ONE*, 8(7):1–15, 2013.

[23] S. Yang, Q. Sun, S. Ji, P. Wonka, I. Davidson, and J. Ye. Structural graphical lasso for learning mouse brain connectivity. In *KDD '15*, pages 1385–1394.

[24] J. Zhang, B. Cao, S. Xie, C.-T. Lu, P. S. Yu, and A. B. Ragin. Identifying connectivity patterns for brain diseases via multi-side-view guided deep architectures. In *SDM '16*, pages 36–44.