UNIVERSITY OF BUCHAREST

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

MASTERS OF ARTIFICIAL INTELLIGENCE

**Dissertation Thesis**

**Spatial-Temporal Neural Networks for Traffic Systems**

Coordinator Graduate

Conf. dr. Ichim Bogdan Bomher Sebastian

BUCHAREST,

September 2021

Abstract

Table of Contents

[INTRODUCTION 6](#_Toc82434000)

[THEORY 7](#_Toc82434001)

[2. Traffic Flow 7](#_Toc82434002)

[3. Machine Learning 9](#_Toc82434003)

[3.1. Multiple Linear Regression 10](#_Toc82434004)

[3.2. Artificial Neural Networks 11](#_Toc82434005)

[3.2.1. Learning Process 13](#_Toc82434006)

[3.2.2. Cost Functions 14](#_Toc82434007)

[3.2.3. Backward Propagation 15](#_Toc82434008)

[3.3. Deep Learning 15](#_Toc82434009)

[3.3.1. Regularization 15](#_Toc82434010)

[3.3.2. Optimization 18](#_Toc82434011)

[3.3.3. Convolutional Neural Networks 21](#_Toc82434015)

[3.3.4. Geometric Deep Learning 22](#_Toc82434016)

[3.3.5. Recurrent Neural Network 25](#_Toc82434017)

[RELATED WORK 28](#_Toc82434018)

[METHODOLOGY & EXPERIMENTS 29](#_Toc82434019)

[1. Methodology 29](#_Toc82434020)

[1.1. Technical Implementation 29](#_Toc82434021)

[1.2. Data Preparation 29](#_Toc82434022)

[2. Experiment 29](#_Toc82434023)

[2.2. Dataset 29](#_Toc82434024)

[2.3. Data Visualization 29](#_Toc82434025)

[2.4. Models 29](#_Toc82434026)

[2.4.1. Linear Regression 29](#_Toc82434027)

[2.4.2. Custom Graph Neural Network 29](#_Toc82434028)

[2.4.3. Spatial Temporal Convolution 29](#_Toc82434029)

[2.1. Results 29](#_Toc82434030)

[2.1.1. Node Predictions 29](#_Toc82434031)

[2.1.2. Model Predictions 29](#_Toc82434032)

[2.1.3. Size predictions 29](#_Toc82434033)

[CONCLUSIONS & DEVELOPMENT 30](#_Toc82434034)

[ANNEXES 31](#_Toc82434035)

# INTRODUCTION

# THEORY

## Traffic Flow

Traffic Flow represents the study of interaction between travelers of different types, including vehicles and pedestrians with infrastructure having the goal to create an optimal transport network with efficient and optimal traffic movement such that traffic congestion is reduced to a minimum.

Traffic networks are complex and nonlinear, depending on external factors such as current time of day/year, weather, and also on internal factors such as infrastructure, number of vehicles and their types or unexpected vehicle crashes. Due to the fact that traffic is based on human interaction, or rather individual reactions to other pedestrians or vehicles, the system does not follow any rules and is rather chaotic.

The traffic system is composed of roads/pavement on which the participants in traffic travel at a certain speed to reach their individual goal. The size can vary and is objective, it can range from a neighborhood to a countries entire traveling system. There are traffic systems which are unique to a single type vehicle, for example the railway system in which just the train travels. Traffic flow is focused on city sized traffic systems for cars, which exhibit the biggest congestion.

Participants in traffic are pedestrians and vehicles (bikes, motorbikes, cars or public transportation). Different types of vehicles can be more effective than others. An example is the underground metro system which is supervised by an entity, thus the participants travel restrictively to preserve the entire system efficiency and can take thousands of passengers. Another example is a car which can take up to a maximum of 5 passengers legally. Although a car can travel to any point but a metro system has predetermined stations.

The main variables that are considered in traffic flow theory are density (k, number of vehicles per unit of space), speed (v) and flow (q, number of vehicles per unit of time).

Speed is defined as the distance traveled per unit of time. Due to the high number of vehicles in a traffic system it is impractical to measure the speed of every vehicle, and the average speed is measured. There are 2 main definitions of speed.

The first one is time mean speed, which is the average speed of a vehicle over a period of time, is computed by measuring the distance traveled by a car over a certain reference point. This method is not accurate because the average speed of a vehicle over a wide range does not take into account the difference between the speeds of different vehicles.

m = Number of vehicles passing through a fixed point; = speed of vehicle i

The second method is called space mean speed. This method is measured over the entire segment. Consecutive recordings of passing vehicles over the entire segment are considered to track individual vehicle speed, after which the speed is calculated. It is considered more accurate than the first method

m = Number of vehicles passing through the roadway segment; = speed of vehicle i

Density represents the total number of vehicles per road segment. There are two types of densities: critical density and jam density. Critical density is considered to be the maximum density under free flow, while jam density is considered the maximum density under congestion. Density is measured as:

K = density, L = length of a roadway, = time, m = number of vehicles

Flow is represented as the total number of vehicles passing through a point given a time period. Headway is the inverse flow, which is represented by the time elapsed between the n-th vehicle passing a point and the next (n+1-th) vehicle. Flow is measured as:

q = flow, T= time interval, = point in space, m = number of vehicles



Figure 1 Flow Density Relationship

The goal of traffic flow analysis is to develop a model that will allow vehicles to reach their destinations within the shortest possible time. This is achieved in a four stages process:

* Generation – Calculate how many trips to be generated by the needs of the passengers.
* Distribution – Generate the path between starting point and stopping point based on what has been generated at step 1
* Modal Split/Mode Choice – Decide the distribution of different type of vehicles for the number of passengers.
* Route Assignment – Assign each vehicle its route such as the entire system has a minimum travelling time.

## Machine Learning

Machine Learning is a subset of Artificial Intelligence which creates and develop software which is able to learn and improve from data over a period of time exposed to data (experience) without being explicitly programmed. This method is used for tasks which are mathematically impossible to program iteratively. For example, it is impossible to explicitly program a function which can classify cats and dogs. The process of learning starts with observing data, experience (time observing data) and instruction in order to look for patterns in data for better future decisions. The goal is to let the software learn without human intervention.

There are three categories which divide machine learning algorithms based on the task and based on the nature of our data:

* Supervised learning: Our data is presented with a label and our task is to predict the correct label. An example is classifying an animal inside a photo as a cat or a dog. Our image contains a correct label and the objective is to predict which animal is inside the photo.
* Unsupervised learning: Our data has no label and the task is to structure the data to find similarities or patterns in the given data and label it for future unknown data.
* Reinforcement learning: There exists a dynamic environment with a goal to perform. As the problem space is explored the program receives positive or negative feedback based on its current decision such that positive behavior is reinforced. Self-driving vehicles are an example.

### Multiple Linear Regression

One of the most popular machine learning algorithm is Linear Regression. Regression is a method of learning a target value based on independent predictors. This method is mostly used in forecasting or finding cause and effect relationships between variables. There are more than one method of regression but mostly differ on the number of independent variables and the type of relationships between independent and dependent variables.

Linear Regression is a type of regression where there is a linear relationship between independent and dependent variables. Multiple Linear Regression, or Multiple Regression is when we observe the relationship between more than one independent variables and one dependent variable. For each independent variable is assigned a weight, then they are summed together with a bias to produce a result:

Where Y is the dependent variable, is the independent variables, n is the total number of independent variables and b is the bias.

### Artificial Neural Networks

One of the algorithms used in Supervised learning are called Artificial Neural Networks (ANN’s). They are simply called neural networks (NN’s). Neural Networks are a method based on the anatomy of the nervous system and the brain. These networks are based on the electrical activity of the nervous system [1].

The base element of the neural networks is a neuron, which is based on the fundamental unit of the brain with the same name, which receives an input, multiplies it by a weight, adds a bias and the returns the result for further processing by a software or another neuron. The function for a neuron looks like this :

Where x is the input, w is the weight, b is the bias and y is the output. Normally the neurons are placed in a layer, with the output from the previous layer being the input of the previous layer, thus simulating the synaptic connections of the brain [1].

After computation a scalar function is used to aggregate a layer into a single input value. Once it is calculated a transfer function, or activation function is used to calculate the output of the layer. A few examples are Linear, ReLU (Rectified Linear Unit) or Sigmoid activation. The Linear function is typically used as the last activation function for regression type tasks such as price prediction. while Sigmoid is used for non-linear classification type tasks such as dogs vs. cats.



Figure 2 ReLU Activation Function Figure 3 Linear Activation Function

Generally, the artificial neural network is divided in tree named parts known as the input layer, the hidden layer and the output layer [2].

* The input layer is responsible for receiving the data,
* The hidden layer is composed of neurons that are responsible for extracting patterns and relevant information for the task. This layer has the most of the computational work of the network.
* The output layer is responsible for giving the result of the neural networks.



Figure 4 Neural Network Architecture

There are multiple types of neural networks [2]:

* The single layer feedforward architecture has an input layer and an output layer.
* The multiple layer feedforward architecture is composed of an input layer, an output layer and one or more hidden layer. This type of neural network is also called deep neural networks.
* The recurrent or feedback architecture is using the outputs of the neurons as the feedback input of the same neurons. This feature is good for dynamic information processing, such as timeseries.

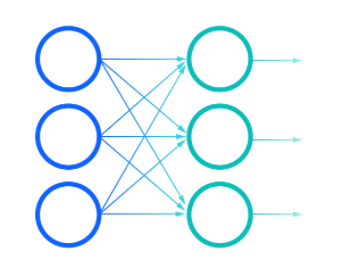


Figure 5 Feed Forward Neural Network

### Learning Process

The process of learning in neural networks is called training. The basis of neural networks is that they are capable of learning from exposure of data samples and retain the patterns. After the training the network is able to create a function which maps the inputs and outputs, such that in future when exposed to unknown data it can generalize solutions which are close if not the desired output for the task given.[2]

The training step is applying the algorithm for tunning the synaptic weights and biases of the neurons with the purpose to generalize a future solution better than the previous training step. In practice our data is split in 2 datasets: the training and test data. The test data is used to measure the performance of the network after the training is finished and is not seen by the neural network during the training phase. For this reason, the training data is the bulk of the information, containing between 60-90% of the data. For each complete exposure of the training dataset for adjusting the weights are biases is called an epoch.

The learning algorithm consists of minimizing the observed errors between the predicted output and the desired output. If the error decreases as the epoch (time) increases then the neural network is considered to be learning. The error rate will never reach 0, no matter how much the network learns, it cannot be perfect. The error rate of the network is given by a cost function which is evaluated at every epoch and depending on an increase or decrease will adjust the weights accordingly.

The learning rate defines the dimension of the correction in the learning process to adjust the weights. A high learning rate makes the training time shorter but can have a smaller accuracy by taking too big a step in correcting the data and “jumping” over the correct output. A smaller learning rate will have a higher accuracy but will lengthen the training time. In practice an adaptive learning rate is used in which at the start of the training the learning rate is higher and after hitting a plateau in the error rate the learning rate is decreased. This process is repeated until a minimum learning rate is reached.

### Cost Functions

There are multiple ways to compute the loss function depending if the prediction is a classification or a regression type. We will denote as n = number of values, y = predicted values and x = correct values. For regression there are several ways of calculating the error:

* The Mean Absolute Error (abbreviated MAE) is measured as the absolute difference between predicted values and correct values. The mean between n predicted values is computed thus MAE is calculated as:
* The Mean Absolute Percentage Error (abbreviated MAPE) is measured as a percentage version of MAE and is calculated as:
* The Root Mean Squared Error (abbreviated RMSE) is measured as the squaring the difference between the predicted and correct values, calculating the mean for n values predicted and then calculating the root squared. It is essentially the root squared of the Mean Squared Error (abbreviated MSE).

### Backward Propagation

Given a cost function and a neural network architecture (also called a model) there is a method which calculates the gradient of the cost function with respect to the model’s weights. This function is called the backpropagation, a short term for backward propagation. The algorithm starts backward, in which the gradient of the final layer is calculated first and finishing with the first layer last. As the gradients are calculated the previous computation of gradients are also reused. This gives this method computational efficiency, as gradients computations are reused rather than calculating each layer individually as a forward propagation.

During the training process of a neural network with gradient descent the gradient of the error function is calculated with respect to the weights and biases. Each iteration of gradient descent updates the weights and biases as:

Where is the cost function, are the weights and biases at an iteration t of the training process and is the learning rate.

### Deep Learning

### Regularization

A common problem in machine learning is to make a model perform well on the training data and also on new, unseen data. This term is also called generalization.[3] There are situations in which the error function in the training step is lower than in the validation/test step. This behavior is called underfitting, a situation in which for the given data the model is not complex enough to capture the relation between the input and target value and the model has too much bias. The opposite, in which the validation/test error function is lower than the training phase error is called overfitting. This makes the model learn “too much” on the training set and is unable to generalize well on unseen data, such as the test set. In this case the model has a high variance and is able to variate much more than it is needed. Overfitting is a recurrent problem in machine learning, because the purpose of supervised machine learning models is to be able to generalize well on unseen data, in a more realistic situation.



Figure 6 Underfitting, Robust and Overfitted

The perfect situation is between overfitting and underfitting, where although the model has some training error it is able to label new data as good as it labels its training data. There are a few solutions to adjust how well a model learns, for underfitting is to increase the model complexity. An overfit model is harder to resolve, but it can be done with regularization. Regularization is any component of the model, training process or prediction procedure which is included to account for limitations of the training data, including its finiteness.[3] There are a few methods of regularization:

* Modifying the loss function, in which the loss function is directly modified to consider the normalization of the learned parameters
* Modifying data sampling. Sometimes the data is not enough to create a relation between input and target label. These methods try to manipulate the little amount of data to create a better representation of the actual input distribution.
* Changing the training approach.

In modifying the loss function, we modify the cost function to include the weighted L1 (Lasso Regression) and L2 (Ridge regression) normalization of the weights that are being optimized. This method prevents the weights becoming to variate and avoids overfitting. For example, the MAE loss function and add the L1 and L2 regularization.

Where n = number of values p = number of neurons, x = correct value, y = predicted value with respect to weight . The L1 regularization adds a squared coefficient as a penalty to the cost function while the L2 regularization adds an absolute value coefficient as penalty.

In modifying the sampling method, we have two options: Data augmentation and Cross-Validation. Data augmentation is done by increasing the quantity of available data by artificially augmenting existing data to create more input data. This is done by adding a random set of functions, differing by the type of data. For image data it can be cropping, dilating or rotating, for sound data it can be noise pitching or time stretching.

In the previous chapter there has been stated that there are two subsets for the data: train data and test data. To implement cross validation a third subset of data is created called validation data. This portion has about 20% of the remaining training data. At the end of each epoch the validation data is used as a test data. This means that the training error will not be taken into account, but the validation error. This prevents overfitting, because the error considered is not on training data but on unseen data.

We can alter the model architecture by adding a regularization layer called Dropout. This type of regularization is frequently used and has the best results in deep learning. At every iteration Dropout randomly selects some nodes with probability p and removes them along with their input and output connection as well. This method reduces the variance which the output can produce since there are less nodes.



Figure 7 Neural Network model without and with Dropout

It is often encountered in machine learning a huge disproportion between features. For example we can have age and income as features. While age usually does not go higher than 100 income can be 100 times bigger than the age. This will cause the income to have a bigger influence on the outcome than the age, although they are equally important.

To overcome this disproportion of features normalization must be applied. Normalization is a technique used in machine learning as a part of data preparation. The goal is to transform numerical features to a common scale without distorting the true value. The values end up in a range between 0 and 1. It is also called Min-Max scaling and is defined as :

Where X is a feature, is the minimum value of a feature, is the maximum value of a feature.

### Optimization

The biggest problem with deep neural networks is the time required for computing the gradients of the cost function with respect to the model’s weight and bias. Due to the fact that deep neural networks have at least one hidden layer they have more gradients to compute rather than single feedforward architecture. The backpropagation algorithm does not specify how we use this gradient to update the weights of the model.[3]

The first method for calculating the error function’s gradients is Gradient Descent. Gradient Descent is the first and most basic gradient algorithm which you can apply to train a deep neural model. It is also called “batch gradient descent” or “deterministic gradient descent”. That is because the parameters are updated after seeing an entire batch of the training data and the gradient is deterministic computed. Gradient descent updates the parameter with a small step towards the minima. The Gradient Descent algorithm looks like this:

Where = learning rate, = model’s weights and biases at time t, = predicted values for value input values and model parameters , = true values, L = loss function for predicted values and true values. Gradient Descent reaches convergence with a fixed learning rate as the gradients get smaller.

There are many algorithms which are based on Gradient Descent to optimize the gradient computation such that convergence can be reached faster. While Gradient Descent will eventually reach convergence it can be pretty slow, which is the reason why it is not used as much in deep learning anymore. There are some situations when the gradients can take a bigger step. For this purpose, Momentum has been created.

Momentum is intended to accelerate the learning process and it is inspired by physics. It is just like an object which gains momentum when it is descending on a slope. The longer the slope has the same inclination our object will gain speed and if the slope becomes steeper our object will accelerate even more. The same logic can be applied to our Gradient Descent:

The velocity v will take into account previous calculated gradients and, if they consistently are the same, the momentum will grow.

The learning rate is one of the hyperparameters that is the most difficult to set because it can have a high impact on the performance of our model. While momentum has mitigated some of the problem the learning rate has, we still have added a new hyperparameter that is just as difficult to set as the learning rate.

For this purpose, adaptive learning rate algorithms has been made. The first algorithm introduced is AdaGrad. Though several variants of Gradient Descent algorithms have evolved, the challenge lies in fine-tuning the hyper-parameters to redefine the algorithm to train the network as per the requirements of the dataset (A. Agnes Lydia, 2019). AdaGrad adaptively scales the learning rate for of all models parameters by scaling them inversely proportional to the sum of squared partial derivates from the training epochs. This will make the parameters with greater gradient have a higher learning rate decrease while the parameters with lower gradient will have a lower learning rate decrease. The downside of AdaGrad is that for training deep neural networks the cumulation of squared gradients will result in a decrease of learning rate’s effectivity.

RMSprop (Hinton, 2012) addresses AdaGrad’s problem by shifting the gradient accumulation into an exponential weighted moving average. Adam (Kingma and Ba, 2014) is another adaptive learning algorithm and is mostly viewed as RMSprop with momentum. The distinction is that Adam is directly integrated into the algorithm as an estimate of the first order momentum of the gradient.

Further optimization can be achieved by using parallel computing. Weight adjusting in a neural network is done by using matrix multiplications in which we don’t have to wait for every element-to-element multiplication and can be done in parallel to speed up the training process. A small deep learning model can take from a few minutes or even an hour to train but a deep learning model with millions of trainable parameters can take days to train.

A graphics processing unit (abbreviated GPU) is specialized to accelerate the creation of images and uses parallelization for efficient computing. Although it is used most commonly in video games it has been recently used in deep learning instead of the Central processing unit (abbreviated CPU). Although it has a smaller memory than the CPU, a GPU computes more efficient the weights for deep learning training.



### Convolutional Neural Networks

For more complex and bigger sized data a simple linear function with a weight and bias is not enough. For this we replace the traditional function for a neuron with a convolution. A Convolution is the simple application of a filter to an input that results in an activation. Since the convolution operation takes multiple dimensions, the CNN is the best choice for image classification, but it cand be used in other areas such as text classification.

The Convolutional neural network (CNN for short) is a type of neural network which contains at least one convolutional layer. In a convolution, the input is a tensor with shape: number of inputs x input height x input width x input channels and the input is transformed into a feature map, also known as activation map with the same number of dimensions. The convolution operation is similar to the response of a neuron in the visual cortex to a certain stimulus (Lab., 2013). A convolutional neuron processes a part of the image, like a receptive field. A convolutional network is able to capture spatial and temporal dependencies in its input through application of filters. Instead of operating on a normal linear function a convolution uses multidimensional filters which “hover” over the image in order to extract information. In the case of a 2d convolution we would have a matrix of a small area, usually 3x3 or 5x5.



Figure 8 Convolutional Neural Network

In real situations images have high resolution, which can make the CNN model heavier with weights. For example, an image with 100 x 100 size will have 10 000 weights for each neuron. We don’t need that much information, we can extract essential information with a down-sized picture such that the CNN model can be deeper and have relevant weights. Information down-sizing can be applied to most data which will be used in a CNN model.

### Geometric Deep Learning

The best way to represent a temporal dependency is with Graphs. A graph is an abstract data type which can be used to represent non-linear relationships between objects and locations. A graph is made up from nodes (sometimes called vertices) which are interconnected with edges (sometimes called arcs, lines or links). There are two types of graphs depending on the type of edges: directed graphs in which the edges have a direction (are asymmetrical) and undirected graphs in which the edges representation do not have a direction (are symmetrical).

A graph can be represented as where V is the set of nodes, E is the set of edges and is a mapping function which attributes an edge for an unordered pair of edges. This representation of is different for a directed graph where is a mapping function which attributes an edge for an ordered pair of edges. In the above definition we can have in the mapping function a situation in which the nodes are identical, which is called a loop. A loop allows an edge to start and end in the same node. We can restrict the graph in permitting loops by adding the in the mapping function.

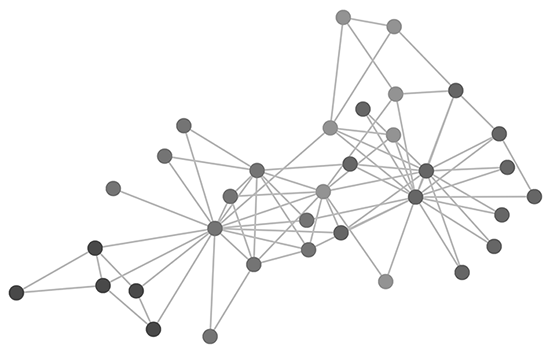


Figure 9 Graph Example

Graphs have a multitude of real-life applications, they can be used to model processes in biological, physical and information systems (Adali & Ortega, 2018). Graphs have been receiving more attention in machine learning due to the great expressive power of graphs. They can be used to describe systems across various domains such as social networks, physical systems, protein interactions systems or even knowledge graphs (Takuo Hamaguchi).

There are multiple types of graph neural networks classifications. There are nodes classification, in which the nodes contain information to be classified or edge classification in which the edges contain information to be classified. There are also two types of graphs from a temporal point of view: dynamic and static graphs. Dynamic graphs may change nodes and edges but static graphs remain the same. As with the signal received in the graph, they can be dynamic or static as well.

Graph neural networks (called GNN’s) are a deep learning method which operate on graphs. To use graph in convolutions the Laplacian matrix and Adjacency matrix. The adjacency matrix A is an n x n matrix of edges where:

Where e is an edge belonging to the graph. In the case of undirected graphs an edge connects both nodes, so the adjacency matrix is symmetrical. In the case of directed graphs an edge comes from a node and goes into another but not necessarily the other way around. We can also define our graph as a weighted graph in which the edges have a weight defined as a number to mark the edge as more or less important. In this case we will redefine our adjacency matrix as:

Where w is the weight of edge e

The Laplacian matrix is defined as L = D – A where D is the degree matrix. The degree matrix is diagonal and its elements are the degree of each node. The degree of a node is given by the number of connecting edges. . The Laplacian matrix usually has three forms:

* Combinatorial Laplacian: L = D – A
* Symmetric normalized Laplacian:
* Random Walk Normalized Laplacian:

To further understand convolution on graph we will define eigenvectors and eigenvalues as well as Fourier transformation

Eigenvectors (sometimes called characteristic vectors) of a linear transformation are described in linear algebra as the non-zero vector which changes with only at a scalar factor when said linear transformation is applied. The factor of which the eigenvector is scaled is called eigenvalue, denoted with . A formal definition can be written as:

v

Where T is a linear transformation, v is the eigenvector of T and is the eigenvalue

A Fourier transformation is a decomposition of a function which depends on either time or space into multiple functions depending on spatial or temporal frequency. The Fourier transform may be formally defined as an improper Riemann integral, making it an integral transform. The Fourier transformation of a function is formally denoted . There are more ways of defining a Fourier transformation of an integrable function, for any real number one of them is:

A key role in graph convolutional networks relied on spectral method is the eigen-decomposition of the Laplacian Matrix (WENMING CAO, 2020). The Laplacian matrix can also be written as: where is a matrix made of eigenvectors sorted by eigenvalues. We also know that U is orthogonal matrix and has

The Convolution Theorem states that under some conditions the Fourier transformation of a convolution of two functions is the pointwise product of their Fourier transformation. We can do, by analogy to the graph and implement the Fourier transforms on graphs. We will define de graph convolution on input signal and filter as:

⊙

Where ⊙ is the element wise product

Graph Convolutional Network (also abbreviated GCN) is based and simplifies ChebNets architecture and is uses graph convolutions. The output of the convolution is:

GCN is spatially localized which combines benefits from spectral based and spatial based methods. Each output is based on weighted aggregation of the node itself and the neighboring nodes. There are 2 disadvantages to this architecture: the weights assigned to a node in a vicinity are the same, thus limiting the architecture ability to capture correlation in spatial information. During training the weights computing increases exponentially with the number of the architecture’s layers.



Figure 10 Graph Convolutional Network

### Recurrent Neural Network

One of the presented types of neural networks architectures are recurrent neural networks. A recurrent neural network (abbreviated RNN) is a special type of neural network which contains neurons that receive prior inputs and influence the current input and output. In short, they are feeding themselves information. They are commonly used for temporal or ordinal problems such as time series prediction, language translation or natural language processing (abbreviated NLP)

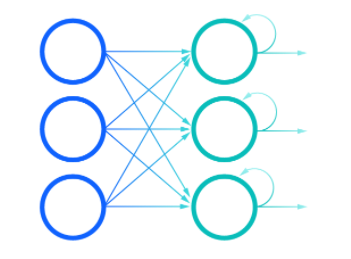


Figure 11 Recurrent Neural Network

There are multiple types of recurrent neural networks, one of the most popular is the Long short-term memory (abbreviated LSTM). LSTM (Sepp Hochreiter, 1997) have been introduced in 1997 as a problem to classic RNN’s problem. Theoretically RNN’s can keep an arbitrary long-term dependency in its input but it has a flaw: when training classical RNN’s with backpropagation the long-term gradients tend to zero or infinity. This is also called the vanishing gradient problem (for zero gradients) and exploding gradient (for infinity gradients). LSTM partially solves this problem by allowing gradients to propagate unchanged and solves the vanishing gradient problem buy can still be affected by exploding gradient problem.

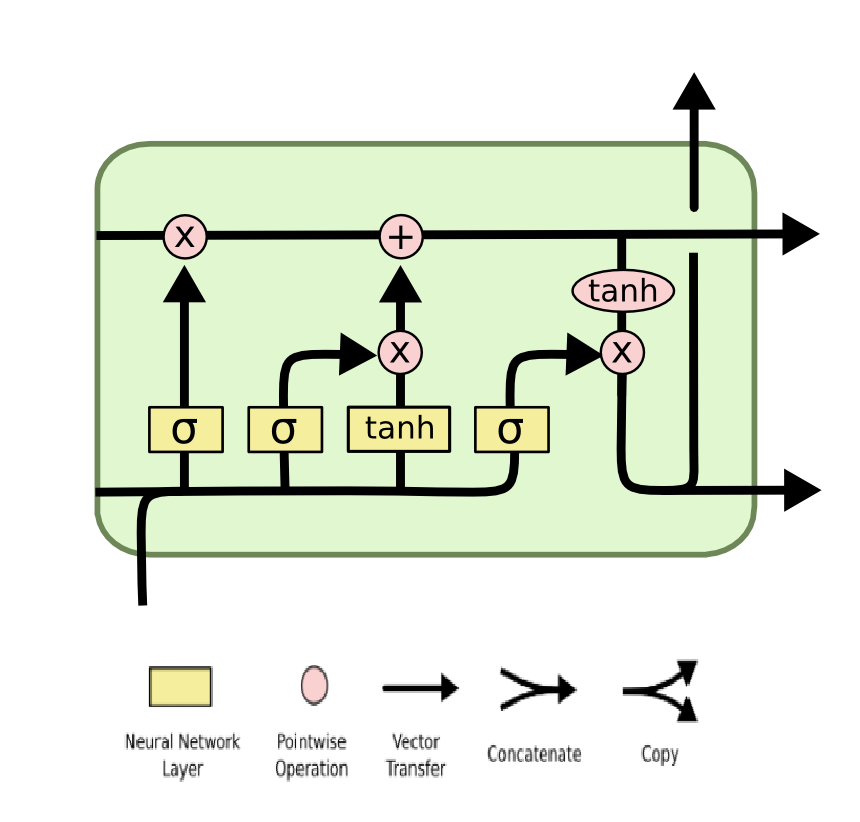


Figure 12 Long short-term memory cell

LSTM operates like classical RNN in chain-like structure but the module has a different structure. A LSTM cell can be observed in figure 12. The mechanism behind LSTM is made up of 4 stages:

The first thing updated in LSTM is the “forget gate” layer. It is the leftmost part of the cell represented in Figure 12. We take at the previous output and and apply a sigmoid layer to have a number between 0 and 1 which will determine the magnitude of how much will be discarded, with 0 being completely discarded. The forget function can be written as:

The second step is to get new information with the “input gate” layer. This represents the middle part of the cell. The input gate takes input and applies a sigmoid layer to it. Next another layer is created   with possible candidate values as a tanh layer of the input. The input gate and candidate values can be expressed as:

The third step is to update the cell state with the previous three values computed earlier. This step represents the top half of the LSTM cell. The forget gate is multiplied with the previous state to discard information. Then the input gate and possible candidate values are multiplied as well. Finally, the two are summed and the function looks like this:

The final step is to create a new output. This is represented by the right part of the cell. On the previous output is applied a sigmoid function. In the newly created cell state a tanh function is applied and is multiplied with our output. The final part looks like this:

# RELATED WORK

# METHODOLOGY & EXPERIMENTS

## Methodology

### Technical Implementation

### Data Preparation

## Experiment

## Dataset

### Data Visualization

### Models

### Linear Regression

### Custom Graph Neural Network

### Spatial Temporal Convolution

### Results

## Node Predictions

### Model Predictions

### Size predictions

# CONCLUSIONS & DEVELOPMENT

# ANNEXES

Steven Walczak, N. C. (2003). *Encyclopedia of Physical Science and Technology (Third Edition).*

Ivan Nunes da Silva, D. H. (2017). *Artificial Neural Networks.* Switzerland: Springer International Publishing.

Jan Kukačka, V. G. (2017). *Regularization for Deep Learning.* Munich.

Zeiler, M. D. (2012). *ADADELTA: AN ADAPTIVE LEARNING RATE METHOD.* New York City.

A. Agnes Lydia, F. S. (2019). *Adagrad - An Optimizer for Stochastic Gradient .*

Lab., L. (2013). *Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation.*

Adali, T., & Ortega, A. (2018). *Applications of Graph Theory [Scanning the Issue].*

Takuo Hamaguchi, H. O. (n.d.). *Knowledge Transfer for Out-of-Knowledge-Base Entities : A Graph Neural Network Approach.*