Machine Learning with Knowledge Graphs and Neural Networks

by

Nicholas Sterling

(Under the Direction of John Miller)

Abstract

Graph data is being produced and collected at an accelerating pace in numerous state of the art applications, and the structure and node properties of the graph data offer fertile ground for data mining and machine learning. Numerous machine learning models have already been applied to graph data and new models are being developed to specifically exploit it. In this work we examine distinct approaches for machine learning over graph data; we provide an in depth examination of applying a modern machine learning framework - probabilistic soft logic - to the problem of graph node label prediction and compare the results to novel neural network architectures applied to the same problem. We also examine the application of a novel knowledge graph based neural network architecture applied to the problem of vehicle traffic flow prediction and compare those results with well established neural network architectures for time series forecasting.

Index words: [Knowledge Graph, Long Short Term Memory, Temporal Convolutional Neural Network, Time Series, Probabalistic Soft Logic]

Machine Learning with Knowledge Graphs and Neural Networks

by

Nicholas Sterling

B.S., University of Georgia, 2014

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree

Master of Science

Athens, Georgia

2021

Machine Learning with Knowledge Graphs and Neural Networks

by

Nicholas Sterling

Major Professor: John Miller
Committee: Budak Arpinar
Fred Mayer

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
May 2021

Dedication

For Baby E.

Acknowledgments

I would like to thank my major professor, Dr. John Miller, for his instruction, guidance, and encouragement. I would also like to thank my committee members: Dr. Budak Arpinar for his compassion and availability and Dr. Fred Mayer for stimulating my intellectual curiosity.

Finally, I would like to thank my wife, Kyla, without whose love and support this would all be meaningless if it weren't impossible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graph data is currently being generated in huge quantities by several state of the art applications: social media, data mining over the Internet, etc. With the proliferation of technologies for harvesting and storing graph data, a great deal of research has focused on developing technologies for representing, storing, retrieving and reasoning over the collected data. Early research into graph data technology focused on the Resource Description Framework (RDF) - described by the World Wide Web Consortium (W3C) as the standard model for data interchange on the Web ("Resource Description Framework - RDF", 2021). The abstract syntax for RDF consists of resources - or things in the universe of discourse - related to one another by predicates which describe the relationship between the resources. Two resources - a subject and an object - related through a predicate form a triple - `(subject,predicate,object)` - which describes a fact about the universe. For instance, a resource representing Joe Biden and another representing the United States of America can be connected through a predicate describing a head of state relationship to express the fact that Joe Biden is the head of state of the United States. A collection of triples forms an RDF graph, a graphical representation about some subset of the facts of the universe of discourse. A persistent database of RDF graphs is often referred to as a triple store.

Many technologies have been developed by the W3C to utilize the abstract RDF syntax, including the TURTLE and JSON-LD syntaxes for serializing RDF, the SPARQL query syntax for data retrieval over RDF triple stores, and RDF Schema (RDFS) and OWL vocabulary extensions which allow for a

richer expression of data models than the basic RDF abstract syntax. In addition, several independent technologies have been developed to make use of the RDF abstract data model. For instance, Apache Jena ("Apache Jena", 2021) provides an RDF API for creating, reading and serializing RDF graphs, a SPARQL compliant query language (ARQ), a high performance triple store (TDB) for persisting RDF data on disk , and an RDFS and OWL compliant reasoner and inference API for reasoning over the RDF graphs. Similar independent technology includes efforts from Oracle ("Oracle Graph Database", 2021) and several others.

Regardless of the early success of the RDF framework and the wide adoption of the RDF data model in academia, the Property Graph Model (PGM) - a much simpler and more user friendly graph data model - has become the more dominant industry tool for representing, storing, retrieving and reasoning over graph data. Popularized by the wildly successful Neo4j ("Neo4J", 2021), the property graph model includes nodes with an optional set of key/value pairs describing their properties, as well as labeled (i.e. - typed) edges between the nodes which also may be annotated with a set of key/value pairs describing their properties. One of the main distinctions between PGM and RDF is that RDF is much more verbose than PGM; the data in an RDF graph is essentially completely atomized to its most basic elements. However, the PGM allows for a much more compact representation of the graph data. Thus, the learning curve for working with a PGM vs. an RDF model is much less steep.

The wealth of data collected, compact representation of the PGM model, and the robust literature on graph theory in the computer science literature means that PGM data is a fertile ground for inference with machine learning models, including modern "black box" models as well as models built specifically to exploit graph data and provide a high degree of lucidity in their structure and mechanisms. For instance, a graph described in PGM can be described, structurally, as an edge matrix while the properties of the graph can be vectorized so that both are suitable as input to a neural network, the quintessential black box model.

While their performance on specific learning tasks is frequently impressive, nonetheless "black box" is often a pejorative term used to describe certain machine learning models, i.e. - a model whose internal

mechanisms are obscured from rigorous inspection and evaluation. Artificial neural networks (ANN) are a typical example of a highly performant model widely condemned as a "black box": they achieve state of the art results on a wide variety of learning tasks at the expense of an extremely opaque mechanism of action. While their has been some progress in looking "under the hood" (so to speak) of some neural network models - for instance a la Bruckner, Rosen, and Sparks, 2013 - nonetheless the challenge of understanding the decision mechanism of most neural network models is a persistently stubborn problem.

On the other hand, models which rely on the domain expertise and intuition of the designer (i.e. - knowledge) offer a great degree of perspicacity, though the lucidity is not without a steep cost of its own: first there must exist a suitable language for representing the knowledge of the domain expert programatically (knowledge representation) and secondly these models are often not as performant as the black box models. Hence, research into efficient and effective Knowledge Representation and Reasoning (KRR) has flourished among computer scientists.

Given the unique advantages of both modeling paradigms, the effort to unify them is accelerating, with some efforts bearing real fruit. For instance, Statistical Relational Learning (SRL) models - an application of statistical machine learning over data in highly relational domains - leverage our understanding of the relationships between objects to achieve impressive results in tasks such as collective classification, link based clustering, and link prediction (Khosravi and Bina, 2010).

One of our main goals in this work is to exploit the the logical structure of relational data expressed in PGM syntax to solve a classical machine learning question for relational data - document label prediction from a citation network - and offer an in depth examination of the effect of the modeling design decisions made in implementing the machine learning models. To be sure, this is not the first work to leverage deep learning over knowledge graphs, however the bulk of previous research has focused on producing efficient embeddings to simplify their manipulation while preserving their logical structure; see Wang, Mao, Wang, and Guo, 2017 for a thorough survey of knowledge graph embedding techniques. In contrast to the previous focus on knowledge graph embedding, this work is less focused on the results of the knowl-

edge graph embedding and more focused on directly consuming the property and relational structure of the knowledge graph as input to the classification models.

In addition to the problem of document label prediction, we focus our efforts on machine learning in a less traditional graph data domain - vehicle traffic flow prediction where the traffic network is defined graphically. Again, this is not the first application of Neural Networks to the problem of traffic flow prediction. Ma et al. in Ma, Zhang, and Ihler, 2020 present a thorough examination of numerous novel modeling architectures for traffic prediction on various traffic data sources, including PeMS data. However, the question of utilizing the graphical structure of the traffic network in the predictive models is less thoroughly treated in the literature.

For both problems - document labeling and traffic flow forecasting - we provide several novel neural network architectures which directly exploit the raw relational and property structure of the graph data for predictive power. In the case of document label prediction we also utilize a popular SRL model - Probabilistic Soft Logic - to compare our novel NN results against.

The rest of this document is organized as follows: in Chapter 2 we discuss Probabilistic Soft Logic (PSL), a PGM designed specifically to exploit the structure of highly relational data; in Chapter 3 we briefly present background material on neural networks and discuss several neural network architectures relevant to this work; in Chapter 4 we discuss the problem of Document Label Classification and thoroughly examine the application of PSL and several novel neural network architectures to the problem, and in Chapter 5 we discuss the problem of vehicle traffic forecasting and examine several neural network architectures applied to this particular time series prediction problem, and finally in chapter 6 we discuss conclusion and future work.

Chapter 2

Probabilistic Soft Logic

Probabilistic Soft Logic (PSL) is a declarative syntax for specifying Hinge-Loss Markov Random Fields (HL-MRF), a tractable and expressive PGM that describes classes of probability density functions over a joint *continuous* domain $[0, 1]^n$. Thus, to understand the syntax and semantics of PSL it is first necessary to develop a passing understanding of the HL-MRF, and since the HL-MRF is a generalization of the discrete Markov Random Field (MRF) - a PGM describing probability density functions over the joint *discrete* domain $\{0, 1\}^n$ - we first present a brief introduction to the MRF and secondly describe its generalization to the HL-MRF.

2.1 Markov Random Fields

For an excellent and thorough discussion of the Markov Random Field (MRF) one may consult Domingos et al., 2008. Here we will provide an intuitive description of the MRF model; see appenendix A for a more thorough and rigorous discussion of the model. Regardless, a MRF is a PGM based on a non-directed graph, G, and a set of potential functions, Φ. A graph $G = (V, E)$ is defined by a set of vertices, V, and edges, E, where $E \subseteq V \times V$. A clique, $C \subseteq V$ of G is a completely connected subset of the vertices of G; in other words $\forall i, j(x_i \in C \wedge x_j \in C \rightarrow (x_i, x_j) \in E)$. If G is the graph associated with some MRF modeling the joint probability distribution for a set of random variables $X = \{X_1, \cdots, X_n\}$, then

dependencies among some subset of the random variables $X_i \subseteq X$ is expressed as a clique in G. Furthermore, the dependencies associated with the clique are modeled by a potential function $\phi_i : X \rightarrow \mathbb{R}^+$ which shoud capture the relevant information about the underlying probability distrubition of X; in other words, for each pair of assignments, x_1, x_2, to the random variables in X, if x_1 is less likely than x_2 then $\phi_i(x_1) < \phi_i(x_2)$.

The probability density function of the MRF is

$$P(x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{2.1}$$

where $x_{\{k\}}$ is the state of the k'th clique in the graph according to x. Since the potential functions are unbounded, non-negative real values, Z is the partitioning function

$$Z = \sum_{x \in X} \prod_k \phi_k(x_{\{k\}})$$

which confines the probability density function to a range of $[0, 1]$. A more convenient (and common) expression of the above probability density function is the log-linear model

$$P(x) \propto \exp(\sum_j w_j f_j(x)) = \exp(w^T f(x)) \tag{2.2}$$

where the potential functions in (A.1) are replaced with an exponentially weighted sum of features of the state of X.

One of the original intentions behind the MRF model was to produce what Richardson et al. in Richardson and Domingos, 2006 referred to as the Markov Logic Network (MLN). Consider a first order knowledge base - as described in Hayes-Roth, Waterman, and Lenat, 1983 - consisting of a set of disjunctive logical clauses $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$. Let $\mathcal{I}_j^-$ and $\mathcal{I}_j^+$ be (respectively) the sets of the indices of the negated and non-negated literals in clause $\mathcal{C}_j$. The KB may be embedded in an MRF by assigning a potential function, f_j, to each clause, $\mathcal{C}_j$, where $f_j(x) = 1$ whenever $\mathcal{C}_j$ is satisfied by x and 0 otherwise.

The corresponding weight, w_j, for the potential f_j in the MRF should express a degree of confidence for $\mathcal{C}_j$. In other words, the weight w_j should be proportional to the probability that the clause $\mathcal{C}_j$ is satisfied for any variable assignment x. Thus, a MRF defines a joint probability distribution function over a set of random variables related to one another by a set of binary valued logical predicate functions.

2.2 Hinge-Loss Markov Random Fields

The HL-MRF is a generalization of the above MLN; here we take a short preamble to describe the generalization of the MLN to the HL-MRF, including the conept of the hinge-loss objective functions as well as a final generality to improve the model's flexibility.

2.2.1 The Unified Objective Function and Arbitrary Linear Constraints

See equation A.3 in appendix A for a description of the objective function for training a MRF. HL-MRFs are a generalization of the MRF according to the equivalences discovered in Bach, Broecheler, Huang, and Getoor, 2017 with the unified objective function

$$\arg\max_{y \in [0,1]^n} \sum_{C_j \in \mathcal{C}} w_j \min\{\sum_{i \in \mathcal{I}^+} y_i + \sum_{i \in \mathcal{I}^+} (1 - y_i), 1\} \tag{2.3}$$

Notice that the discrete random variables from the objective function (A.3) are now replaced with continuous random variables in objective function (2.3).

Regarding the unweighted terms of the summation in objective function (2.3), we say that the term is satisfied when it achieves its maximum value of 1, and we refer to its distance to satisfaction as

$$1 - \sum_{i \in \mathcal{I}^+} y_i + \sum_{i \in \mathcal{I}^+} (1 - y_i)$$

We can refactor the objective function (2.3) to minimize the distance to satisfaction of each of the terms of the summation:

$$\arg\min_{y\in[0,1]^n} \sum_{C_j\in\mathcal{C}} w_j \max\{1 - \sum_{i\in\mathcal{I}^+} y_i + \sum_{i\in\mathcal{I}^+}(1-y_i), 0\} \tag{2.4}$$

and in so doing we introduce the concept of the relaxed linear constraint:

$$1 - \sum_{i\in\mathcal{I}^+} y_i + \sum_{i\in\mathcal{I}^+}(1-y_i) = \mathcal{L}_j(y) \leq 0$$

It is a trivial generalization that linear constraint $\mathcal{L}_j(y)$ may be any linear function over the domain of y; in other words, the requirement that the linear constraints be defined over logical clauses is no longer necessary, and instead they can be defined to capture any arbitrary understanding of the domain of y. The weight w_j is an indication of how important it is to satisfy the linear constraint $\mathcal{L}_j(y)$: a higher weight means satisfying $\mathcal{L}_j(y)$ is more important, a lower weight less important, and an infinite weight indicates a hard (instead of a relaxed) linear constraint. Furthermore, the relaxed linear constraints $\mathcal{L}_j(y) \leq 0$ and $-\mathcal{L}_j(y) \leq 0$ may be combined to express a relaxed equality constraint $\mathcal{L}_j(y) = 0$.

2.2.2 Hinge-loss and Squared Hinge-loss Functions

Hinge-loss objective functions were first introduced in Gentile and Warmuth, 1998, and the objective function (2.4) re-written in terms of the linear constraint functions

$$\arg\min_{y\in[0,1]^n} \sum_j w_j \max\{\mathcal{L}_j(y), 0\} \tag{2.5}$$

makes explicit the hinge-loss functions at its heart: $\max\{\mathcal{L}_j(y), 0\}$. While the hinge-loss function allows for reasoning over arbitrary linear constraints over the domain of y, it can be less than ideal for modeling in certain circumstances, particularly in domains which require a smooth trade off between conflicting constraints. In these cases, a squared hinge-loss function may be more appropriate : $(\max\{\mathcal{L}_j(y), 0\})^2$.

Thus, HL-MRFs are defined using any combination of either regular or squared hinge-loss functions for all of the linear constraints.

2.2.3 The Rigorous Model Definition

Let $\Phi = (\phi_1, \cdots, \phi_m)$ be a vector of generalized hinge-loss functions defined over the continuous domain $[0,1]^n$ with

$$\phi_j(y) = (\max\{\mathcal{L}_j(y), 0\})^{p_j}$$

where $p_j \in \{1, 2\}$. Then with a vector of potential function weights $\vec{w} = (w_1, \cdots, w_m)$, the Hinge-Loss Markov Random Field is defined by the probability density function

$$P(y) = \frac{1}{Z_y} \exp\left(-\sum_j w_j \phi_j(y)\right) \tag{2.6}$$

with the familiar partition function

$$Z_y = \int_{[0,1]^n} \exp\left(-\sum_j w_j \phi_j(y)\right) \partial y$$

2.3 Probabilistic Soft Logic

PSL is a declarative syntax for defining classes of HL-MRFs, while a PSL program is a set of rules which are templates for defining hinge-loss potentials and hard linear constraints. Once the rules of the program are defined, they are grounded out over a base of ground atoms to induce a HL-MRF. PSL syntax is defined over the familiar first order logical syntax: constants define the elements in a Universe, variables may be substituted with constants, a set of terms which is the union of the set of constants and variables, n-ary predicates which are functions $p : T^n \rightarrow \{True, False\}$, atoms which are an application of n many terms to an n-ary predicate, and literals which are either an atom or the negation of an atom. A ground atom is a specific type of atom which has nothing but constants as its arguments, and each valid ground

atom will be associated with a random variable in the induced HL-MRF. Constants are defined according to a schema (i.e. - each constant belongs to a type), and predicates are defined similarly, thus limiting the allowable atoms to be grounded out to only those atoms that match the schema. For instance, given the sets of constants

```
Students={Mary, Joe, Sue, Hao}
Professors={Dan, Jin, Jane}
```

the 2-ary Predicate $Advises/2$ can be confined to the schema

```
Professors X Students
```

where `Advises(x,y)` says that Professor "x" advises Student "y".

2.3.1 Logical Rules

The first type of PSL rule is a logical rule, which is a disjunction of literals, i.e. - a disjunctive clause. A disjunctive clause is written with the usual Boolean operators - conjunction (&), disjunction (|) and negation (!) - and any valid disjunctive clause is a valid PSL logical rule. A logical rule may either be weighted, unweighted, defined as a hard constraint, or defined with a squared hinge-loss potential in mind. For instance

```
1 : Advises(X,Y) & Department(X,Z) -> Major(Y,Z) ^2
```

defines a rule that says a student's major is administered by the department of her advisor. The rule is annotated with a weight of 1 and the atoms that will ground out from this rule will be associated with squared hinge-loss potentials.

Recall what we said earlier: PSL rules are templates for producing hinge-loss potentials and linear constraints that will form a HL-MRF, and each ground atom resulting from grounding out the PSL rules is associated with a random variable in the HL-MRF. Consider a grounded predicate with random variables corresponding to the the set of negated and un-negated atoms in the grounded predicate $\mathcal{I}^{-}$ and

$\mathcal{I}^+$, respectively. The corresponding linear constraint induced by this rule in the PSL should be familiar:

$$1 - \sum_{y \in \mathcal{I}^+} y - \sum_{y \in \mathcal{I}^-} (1 - y) \leq 0$$

If the logical rule which produced the linear constraint is weighted, then a hinge-loss potential created from the constraint is added to the HL-MRF. Furthermore, if it is annotated with a squared exponent, the potential will be squared hinge-loss potential. If the logical rule is NOT weighted, then a hard linear constraint is added to the HL-MRF instead.

2.3.2 Arithmetic Rules

In addition to the logical rules, PSL provides arithmetic rules which allow for expressing a greater variety of relations. At its most basic, an arithmetic rule can relate linear combinations of atoms with equalities or inequalities. For instance, the rules

```
Liberal(P) + Conservative(P) = 1
Liberal(P) + Conservative(P) <= 1
```

express the idea that a person must be either politically liberal or politically conservative and the idea that a person may be politically liberal or politically conservative but not both, respectively.

There is additional syntax to make arithmetic rules more flexible and useful. An arithmetic rule may be defined using summation atoms, an atom which takes as its arguments either terms or sum variables, where a sum variable is a variable prepended with a "+" sign. The summation atom represents the summation of the ground atoms that can be obtained by grounding the free variables and summing over the all of the possible values for the sum variables. For instance, the rules

```
Friends(X,+P) <= 10
Label(X,+L) = 1
```

say that no person can have more than 10 friends and that the labels for any item must sum to 1, respectively.

In addition to sum variables, PSL includes filter clauses which allow us to filter the constants which can be substituted into the sum variables. For instance, the following arithmetic rule with a filter clause

```
Advises(X,+Y) >= 1
{ Y : Undergraduate(Y) }
```

says that every professor must advise at least one undergraduate student.

Arithmetic rules can also contain coefficients on their atoms, which can be either hard-coded constants or specified using the carnality function or other built in coefficient functions. For instance, the rules

```
Advises(X,+Y) <= .5 Teaches(X,+Z)
1/|Y| Friends(X,+Y) = Friendliness(X)
Friends(X,+Y) + WorksWith(X,+Z) <  @Min[|Y|,|Z|]
```

say that a professor can only advise at most half as many students as she teaches, that a person's friendliness is defined as the percentage of people they are friends with in a social network, and that the number of people that a person is friends with plus the number of people that a person works with must be less than the smaller size of the two potential groups. The first rule uses a simple hard-coded coefficient, the second rule takes advantage of the cardinality function, while the third uses the built in `@Min` coefficient function to programmatically derive the coefficient.

To ground out an arithmetic rule to find the linear constraints it induces, the variables (not sum variables) in the rule are individually grounded out to find the set of ground atoms from the rule, and the summation atoms are substituted for the appropriate summations over ground atoms (potentially filtered by the filter clauses). If the rule is an unweighted (in)equality then the resulting ground atoms are manipulated to the form $c(y) <= 0$ or $c(y) = 0$ - respectively - and the hard linear constraint is added to the HL-MRF. If the arithmetic rule is weighted, then the resulting ground atoms are manipulated to the form $\mathcal{L}(y) < 0$ and the resulting hinge-loss potentials added to the HL-MRF depend on the specifics of the arithmetic rule. In the case that the arithmetic rule is an inequality without an exponent indicator,

then the hinge-loss potentials added to the HL-MRF take the form

$$\max\{\mathcal{L}(y), 0\}$$

If, on the other hand, the rule is an equality without an exponent indicator, then pairs of potentials are added for each grounding:

$$\max\{\mathcal{L}(y), 0\} \max\{-\mathcal{L}(y), 0\}$$

In all cases, the hinge-loss potentials are squared if the arithmetic rule includes an exponent indicator.

2.3.3 MAP Inference

Maximum a posteriori (MAP) inference is the assignment of values to the free random variables in the HL-MRF given observations (i.e. - values) for some subset of the random variables in the model. MAP inference is the most important inference problem in HL-MRFs, for two reasons:

1. It is the mechanism by which predictions are made from the model.
2. It is often used as a sub-routine in other learning algorithms for HL-MRFs, including weight learning.

MAP inference in HL-MRFs enjoys a substantial advantage over MAP inference in regular MRFs in that the question is one of convex optimization in the former as opposed to combinatorial optimization in the latter. However, exact MAP inference in HL-MRFs is nonetheless still intractable, since MAP inference over a MRF is generally NP-hard and HL-MRFs are a generalization of the MRF approach by the equivalences that are proved in Bach et al., 2017. However, context optimization can provide approximate results for MAP inference in HL-MRFs in time polynomial in the number of variables, potentials, and constraints in the HL-MRF.

Another of the advantages HL-MRFs have over other logic based PGMs is the inclusion of hard linear constraints in model definition. By defining the hard linear constraints as a limit on the viable ground

atoms of the model, the number of viable interpretations of the logical rules can be drastically limited. Lazy MAP inference algorithms have been developed to take advantage of the restricted viable space to explore a limited materialization of the joint probability distribution.

Finally, the sparse dependency structure of the HL-MRF means that lifted MAP inference algorithms have been developed as well. Lifted inference discovers common substructures in the input data and performs inference individually on the substructures to avoid redundant computation on the entire data structure. Lifted MAP inference for HL-MRFs was proposed in Srinivasan, Babaki, Farnadi, and Getoor, 2019 and showed substantial improvements in processing time over non-lifted MAP inference (more than 2.5x speed up.)

Generally speaking the combination of convex optimization, hard linear constraints, and lifted inference algorithms means that most PSL programs are tractable for even very large data sets. However, one of the main design decisions to consider to ensure their tractability includes defining well thought out hard constraint rules. Without these constraints the size of the viable space of the probability distribution can potentially become cost prohibitive. Also, forecasting directly over a time series itself with HL-MRFs is not particularly promising; the huge number of time dependencies that need to be specified and reasoned over make this a cost-prohibitive exercise. Instead, the HL-MRF should be used as part of a stacked or hybrid modeling effort.

Chapter 3

Neural Networks

The artificial neural network, more commonly referred to simply as a neural network, is a machine learning model originally intended to mimic the structure and behavior of the human brain. The basic unit of the model is the artificial neuron or node, n, comprised of a vector of weights, $\vec{w}_n = [w_{n0}, \ldots, w_{ni}]$, a bias, b_n and an activation function, f_n, which transform a linear combination of some input vector, $\vec{v} = [v_0, \ldots, v_i]$ to a scalar output value, o_n, according to the following equation:

$$o_n(\vec{v}) = f_n\left(\vec{v} \cdot \vec{w}_n + b_n\right))$$

See figure 3.1 for a diagram of an artificial neuron.

Let Θ_n denote the parameters of the node n, i.e. - the weight vector and bias value. Given a vector of input vectors, i.e. - a matrix, $V = [v_1, \ldots, v_j]$ which we want to use to predict a vector of target values, $\vec{t} = [t_1, \ldots, t_j]$, then a loss function, J_{Θ_n}, can be defined to describe the error of the predictions relative to the target values. While strictly speaking optimizing (i.e. - training) the parameters of Θ is a non-convex optimization problem whenever the underlying neural network contains at least one hidden layer, nonetheless Jiang showed in H. Jiang, 2019 that non-convex optimization in the case of training neural network model parameters behaves essentially as a convex optimization problem and thus gradient based optimization techniques may be applied efficiently to train the model parameters.

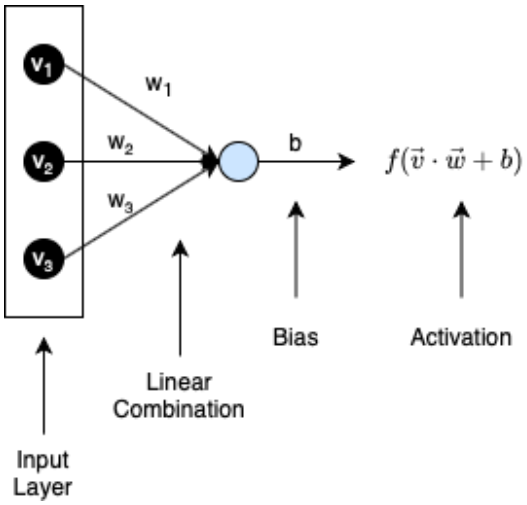


Figure 3.1: An Artificial Neuron

Groups of nodes can be organized into layers which are connected to one another in various configurations - or architectures - depending on the application the model is designed for. The input to such a model is referred to as the input layer, and the output as the output layer, while any other layers are referred to as the hidden layers. The same convex optimization techniques are used to train such a layered model, however such models are only efficiently trainable recently since the advent of the backpropogation algorithm. See figure 3.2 for an example of an artificial neural network with an input layer that accepts vectors of length 3, an input layer with 4 nodes, and an output layer with 7 nodes. The output of 3.2 could be a model of a multi-dimensional target vector of length 7 or even a multi-categorical classification problem with 7 categories, among others. Such a neural net is often referred to as a densely connected net, since each node in the hidden layer computes an activated output of a linear combination of all of the input vector values.

Neural networks can be composed of multiple layers with varying sizes and activation functions to achieve a truly dizzying variety of architectures. The following is a brief examination of some of the different types of neural network designs to date.

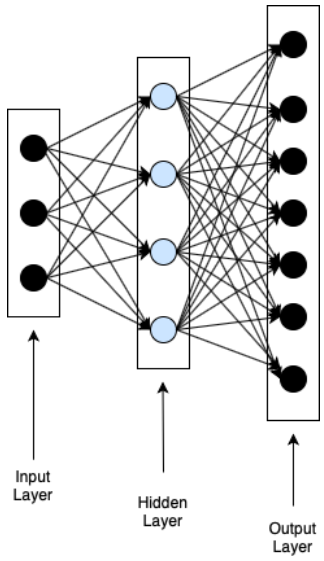


Figure 3.2: A Single Layer Neural Network

3.1 Deep Neural Networks

A deep neural network is an artificial neural network with multiple hidden layers between the input layer and the output layer, and it is chiefly defined by two important features : the depth of the network is the number of hidden layers, while the width of the network is the number of nodes in the hidden layers. Figure 3.3 presents a densely connected deep neural network with a depth of 3 and a width of 4. It is important to note that the width of the layers need not be consistent among all layers; the architecture referred to here was chosen merely for the convenience of its presentation.

3.2 Recurrent Neural Networks

The neural networks described above are often referred to as feed forward neural networks to distinguish them from a popular variant called recurrent neural networks. Recurrent neural networks are used in predictive models where the ordered sequence of the input and output is as important as the values themselves, for instance in time series modeling and sequence prediction tasks such as natural language

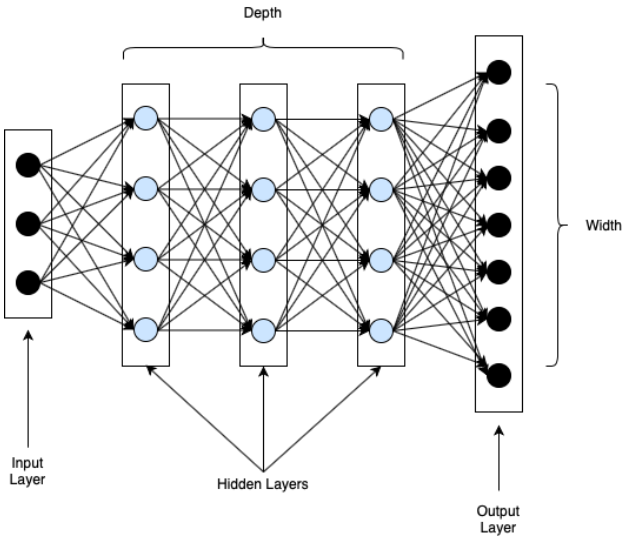


Figure 3.3: A Densely Connected Deep Neural Network

modeling. In a recurrent neural network, the output of the model is not solely based on the recombined and activated outputs of the nodes in the network, but also upon the network's previous outputs for the previous input in the sequence. A basic diagram of an artificial recurrent neuron is presented in 3.4 where t_i is the value of the input sequence at time i and $h(t_i)$ is the prediction (i.e. - hypothesis) of the model for the target value at index i. In the diagram the node has been "unrolled" through time to more clearly present the mechanism of the model. In other words, while there are seemingly several nodes in the diagram, in reality there is just a single node, and we are presenting its input and output at various points in the sequence. In its most naive implementation this implies an infinite look back window, which results in what is known as the vanishing / exploding gradient problem, where the gradient of the error function with respect to the weights of the model are so small as to prevent effective training with gradient based optimization. To rectify the problems associated with the infinite look back, several variations on the vanilla recurrent node have been explored. The following is a brief examination of two of those variants: the long short term memory unit and the gated recurrent unit.

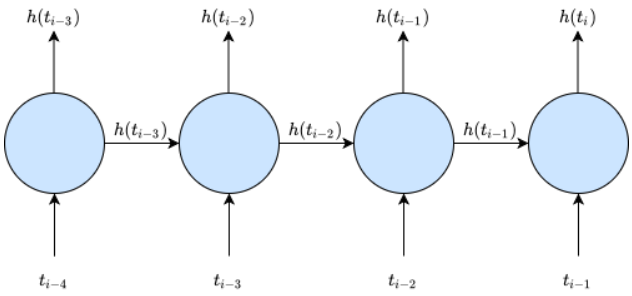


Figure 3.4: A Recurrent Neural Network Node

3.2.1 Long Sort Term Memory Networks

One popular variant of the RNN is the LSTM network, the basic unit of which is the LSTM memory cell which preserves an internal cell state by selectively forgetting old information out of the state and selectively incorporating new information into the state. The cell state is applied to the new data to produce a hypothesis. The cell utilizes a number of "gates" to selectively forget old information and incorporate new information into the model, as well as to update the cell state and produce the current hypothesis. The gates include the forget gate, the input gate, and the output gate.

According to Chung, Gulcehre, Cho, and Bengio, 2014, the output of the forget and input gates - the gates which allow the LSTM cell to forget irrelevant information from the past and incorporate relevant information from the present - is a linear combination of the weighted current input, the weighted previous hypothesis, and the weighted current cell state:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + V_f C_{t-1}) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + V_i C_{t-1}) \end{aligned} \tag{3.1}$$

The current data and the previous hypothesis are also used to create a set of candidate new cell state values:

$$\tilde{C}_t = tanh(W_C x_t + U_C h_{t-1})$$

The cell state is then updated using the sigmoid masks from the input and forget gates as well as the candidate cell states:

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

A linear combination of the current cell state, the previous hypothesis, and the current input can now be sent through the output gate:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_0 C_t)$$

Finally, the the result of the output gate is multiplied with the result of pushing the current state through a tanh to generate the current hypothesis:

$$h_t = o_t tanh(C_t)$$

3.2.2 Gated Recurrent Unit

The Gated Recurrent Unit is a popular variant on the LSTM. Again, according to Chung et al., 2014, to make a prediction from a GRU, first the current data and the previous hypothesis are combined and passed through a reset gate reset, similar to the update gate of the LSTM:

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

A candidate activation is then computed from a linear combination of the current input and a point wise multiplication of the reset gate with the previous hypothesis :

$$\tilde{h}_t = tanh(Wx_t + (r_t * h_{t-1}))$$

Next, a linear combination of the current input and the previous hypothesis are passed through an update gate to determine how much of the current input data should contribute to the new hypothesis:

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

Finally, the current hypothesis is created as a linear interpolation with the update, the previous hypothesis, and the current candidate hypothesis:

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

The LSTM and the GRU both share the feature that their internal state is additive, meaning that relevant features discovered - even very early in the training process - will subsist across vast time periods without suffering from an exploding / vanishing gradient from being repeatedly passed through multiplicative non-linearities. The GRU differs from the LSTM principally in that the state of the GRU does not require a dedicated state field separate from the current input and previous hypothesis fields, as well as in the fact that the entire output of the GRU is exposed in the hypothesis while the output of the LSTM is filtered by an output sigmoid gate first.

3.3 Temporal Convolutional Neural Networks

The Convolutional Neural Network (CNN) was originally developed for work on image recognition, and CNNs have since been applied to time series forecasting as well. Given an image described as a 2-dimensional array of floating point pixel values, the number of parameters required to process such data

using a densely connected, feed forward neural network is prohibitively large. The CNN overcomes this by applying a (set of) fixed convolutional kernel(s) (aka - filter(s)) to the image data instead. In the two dimensional case, the filter(s) is a two dimensional matrix of size $n_f \times m_f$. Depending on whether padding is added to the image, the convolution kernel applied to the image yields an intermediate representation of the original image of the same size or smaller with some relevant features exposed and/or accentuated. If the original image is f and our kernel is h then the intermediate representation of the image, G, at any pixel m, n is

$$G[m, n] = (f * h)[m, n] = \sum_j \sum_k h[j, k] f[m - j, n - k]$$

where $*$ is the convolutional operation.

Typically multiple kernels are passed over the data, each of which is independently optimized and each of which may separately tease out a distinctly important feature of the data.

In addition to the convolutional layers, CNNs also often include pooling layers. In convolutional passes, the convolutional kernel will be applied to most of the data points either $n_f * m_f$ many times in the 2D case or just n_f many times in the 1D case. Pooling, on the other hand, splits the data into separate, non-overlapping chunks and applies a pooling operation to each chunk individually to produce a new value. Common pooling functions include mean, max, and min pooling.

From the 2D CNN definition we can easilly define the Temporal Convolutional Neural Network (TCN). The TCN is the result of passing a 1D convolutional filter over input data with at least one dimension. In the case of the 1-D convolutional filter we have

$$G[m] = (i * k)[m] = \sum_j h[j] f[m - j]$$

3.4 Dilated TCN

The Dilated TCN (DTCN) is based on the time dilations presented first presented by van den Oord et al., 2016 in the WaveNet model. In a dilated convolutional network with multiple layers, the convolutional

results for applying a convolutional kernel to the data up to time point t_k may be fed not just forward to the next layer in the model but also forward to nodes in the next layer that will receive data from dilations run on time points t_l where $k < l$. The use of dilations allows the model to train and predict with a receptive field that increases in size orders of magnitude at a rapid pace. By using time dilations at increasing powers of two, the WaveNet can increase the receptive field by 1024x with just 10 layers.

For a normal convolutional neural network, the activation at layer $l+1$ for time t is

$$\begin{aligned} h_t^{l+1} &= A((W * h)(l,t)) \\ (W * h)(l,t) &= \sum_{\tau=0}^{k} W(l,\tau) h_{t-\tau}^{l} \end{aligned} \tag{3.2}$$

where A is an activation function, and $W(l,\tau)$ is a filter weight. By expanding the network with dilated convolutions, we find the following activation for time t at layer $l+1$:

$$(W * h)(l,t,d_l) = \sum_{\tau=0}^{\lfloor k/d_l \rfloor} W(l,\tau) h_{t-d_l\tau}^{l}$$

Hence, dilated convolutions provide a trade off between reducing focus in the near term and increasing the receptive field in the long term. Figure 3.5 provides a graphical diagram of the forward flow of data through a stack of DTCN layers with a kernel size of 3 and dilation rate increasing by a factor of two at each layer level : 0,1,2,4.

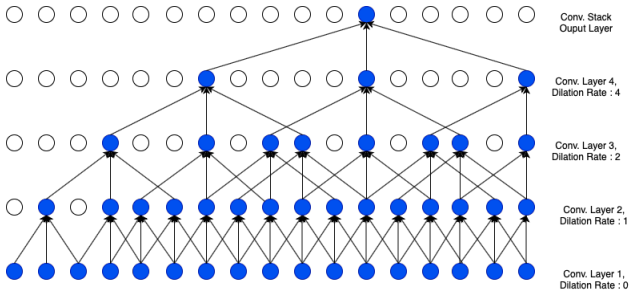


Figure 3.5: A Stack of DTCN Layers with Increasing Rates of Dilation

Chapter 4

Document Label Prediction

Consider a labeled property graph with vertices $\mathcal{V}$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Consider a property, P, where for any vertex in the graph, v_i, the value of the property associated with the vertex is $v_i[P]$, and the set of all values attributable to a vertex in the graph is $V[P]$. Given a set of vertices, $V_{evidence}$ for which the values of the property P is observed, the goal of label prediction is to find a predictive model which accurately predicts the unknown value of the property P for the vertices in the graph which are unobserved, i.e. $V/V_{evidence}$. Given a set of evidence documents the labels for which are all known, the document label classification task is to predict the document labels of a set of unlabeled documents from their properties as well as their edge connections to the documents in the evidence set.

Previous efforts have reported success in document labeling by utilizing the document citation network structure, a la Lu and Getoor in Lu and Getoor, 2003 and Sen, et al. in Sen et al., 2008 who used the network structure to achieve FScore measures of between .6 and .8 on the document labeling task. Our goal is to either match or exceed those results using models which are simpler in their implementation. To that end we have developed several models and compared their performance against one another; chiefly, we want to compare models without any awareness of the graph structure of the data to those models that leverage the graph structure. Two general types of models with numerous different configurations are compared against one another - and the results are presented in Appendix B. Below we discuss both the data used in the experiments as well as the models used.

4.1 Data

We are using three different document data sets in this work: the Citeseer, Cora, and PubMed dataset, all three of which may be downloaded at "LINQS Datasets", 2021. The Citeseer dataset consists of 3,312 publications classified into one of six classes as well as a citation network with 4,732 edges. In addition to the edges and node labels, each document is also described by a one-hot encoded vocabulary vector from a vocabulary of 3,703 words. The Cora dataset contains 2,708 documents, 5429 edges, and a vocabulary of 1,433 words. Finally, the PubMed dataset consists of 19,717 documents, 44,388 edges, and a vocabulary of 500 words. For the PubMed dataset, the vocabulary vector for each document is a TF-IDF weighted vector instead of a one-hot encoding.

4.2 PSL Models

The following PSL rules were used to create the model for document classification:

```
Cites(X,Y) & Label(Y,Z) -> Label(X,Z)
Cites(X,Y) & Label(X,Z) -> Label(Y,Z)
1/|X| Label(+X,Y)  = Label(Z,Y)
{ X : Cites(Z,X) }
1/|X| Label(+X,Y) = Label(Z,Y)
{ X : Cites(X,Z} }
Contains(X,W) & Contains(Y,W) & Label(Y,Z) -> Label(X,Z)
Label(X,+Y) = 1
```

The first two rules capture our intuition that a document which cites (or is cited by) another document is likely to share the same label with it. The second two arithmetic rules exploit the connectedness of nodes. Together the rules express the intuition that for any node/label combination the degree to which we believe that the node has the label is equal to the proportion of the node's neighbors that have the label. The next rule relates documents and the words the documents contain to their labels; it says that if two documents use the same word then they probably have the same label. The final rule enforces the normalization that each document has just a single label.

We tested models formed of various combinations of the above rules as well as with squared and non-squared hinge-losses to try to find the smallest possible effective rule set for our problem. The various table configurations are listed in table B.1, which can be used to identify the results presented in the results tables in the conclusion section.

4.3 Neural Network Models

One of the major questions we are interested in answering in this work is to assess the effectiveness of Knowledge Graph Neural Networks on the document classification task, and to answer that question three novel neural network layers were implemented and assessed. Each of the three novel layers was used as the first hidden layer of a neural network model with a series of densely connected layers appended behind it. Thus, the models are in essence Deep Knowledge Graph Neural Networks.

4.3.1 Simple Edge Layer

The first novel layer implemented is referred to as the Simple Edge Layer. This layer takes no account of the *labels* of the evidence nodes whatsoever - just the edge connections to the evidence nodes. It consists of u many weight variables for each possible connection to one of the evidence nodes, meaning that if there are n_e many evidence nodes then the layer consists of a matrix of trainable network weights of shape

$n_e \times u$. The input into the layer is a vector $x_i \in \{0,1\}^{n_e}$ describing the edge connections from vector v_i of the target vertices to the evidence vertices; $x_i, j = 1$ if $(e_i, e_j) \in E$ otherwise and otherwise $x_i, j = 0$.

4.3.2 Label Sum Layer

The second novel layer implemented is referred to as the Label Sum Layer. This layer transforms the edge set for a target node into the total number of nodes from each category that the edge is connected to. It consists of u many weight variables for each label category, meaning that if there are n_c many categories then the layer consists of a matrix of trainable network weights of shape $n_c \times u$. The input into this layer is the same vector described as input into the above layer.

4.3.3 Flat Label Layer

The last novel layer implemented is the Flat Label Layer. This layer uses a separate set of trainable variables for each vertex in each label group of the evidence variables. Thus if there are n_e many evidence vertices and l many labels, then this layer consists of u many trainable variables for each label/vertex combination, which is a training matrix of shape $n_e * l \times u$. The intuition behind this layer is that it is capturing more information than either of the above layers since the vertex labels are explicitly encoded in the training variables. The input into this layer is the same vector described as the input into the above layers.

4.3.4 Neural Network Architectures

Both the simple edge layer and label sum layers were used as the first layer of a neural network architecture backed by a densely connected deep neural network, denoted the Simple Edge Model (SEM) and Label Sum Model (LSM) respectively. Four additional general architectures were also tested, each of which used (at a minimum) the one-hot encoded vocabulary vector for the target document as an input to the model; three of the four were designed to also accept the edge vector for the target document as a second input. The three double-input architectures were built using the three novel layers described above, resulting in the Simple Edge Layer Vocabulary Model (SELVM), the Label Sum Layer Vocabulary Model (LSLVM)

and the Flat Label Layer Vocabulary Model (FLLVM). The final model included in the experiments was a control model which relied solely on the document vocabulary vector to predict the document labels.

4.4 Model Training and Assessment

Each of the above models was trained using a 10-fold cross validation scheme. Each dataset was split into 10 separate training/testing folds, and the models were independently trained on each of the 10 folds. For a binary classification task, the precision, recall, and FScore are measures of the accuracy of a model's predictions based on the true/false positives and true/false negatives predicted by the model. If TP is the number of positive samples that were accurately predicted by the model as positive, FP is the number of negative samples that were erroneously predicted as positive by the model, and FN is the number of positive samples erroneously predicted as negative by the model, the precision, recall, and FScore of the model are defined as

$$precision = \frac{TP}{TP + FP}, recall = \frac{TP}{TP + FN}, FScore = \frac{2 * (precision * recall)}{precision + recall}$$

Now, consider a testing/training fold with n many samples with L many distinct labels in the fold where n_l is the number of samples in the fold with label l. For each label l if we calculate the precision, recall and FScore metrics for the fold as p_l,r_l, and F_l respectively, then the weighted average precision ($p_{weighted}$), recall ($r_{weighted}$) and FScore ($F_{weighted}$) for the fold are defined as :

$$p_{weighted} = \frac{\sum_l n_l * p_l}{\sum_l n_l}, r_{weighted} = \frac{\sum_l n_l * r_l}{\sum_l n_l}, F_{weighted} = \frac{\sum_l n_l * F_l}{\sum_l n_l}$$

Each model was trained on each fold and the weighted precision, recall and FScore saved and annotated with the model configuration. Later we present our conclusions on the collected results.

Chapter 5

Traffic Forecasting

Traffic flow forecasting is a well-studied problem in the computer science literature, due mainly to the positive benefits accrued from more accurate traffic flow predictions; more accurate vehicle traffic flow predictions allow for more efficient management of traffic systems; predicting congestion more accurately allows for better congestion mitigation resulting in less lost productivity as well as less traveler frustration. Due to its high degree of utility, research into vehicle traffic forecasting is constantly evolving, as evidenced by several studies published in just the past 2 years, i.e. - Wei et al., 2019, Mallick, Balaprakash, Rask, and Macfarlane, 2019, and Boukerche and Wang, 2020 to name just a few.

One of the main questions we are interested in answering in this work is the effect of varying neural network model architectures on traffic flow prediction accuracy. We have trained several models, outlined below, with a variety of hyperparameters to assess the effect of model depth and width - among other parameters - on traffic flow forecasting accuracy. In addition, we are interesting in assessing the potential to improve the accuracy of neural network forecasting results by incorporating knowledge graph structure in the modeling process. Jiang and Luo present a comprehensive survey of traffic forecasting with graph neural networks in W. Jiang and Luo, 2021. The following is a brief description of the data used in this work and the models trained to solve the problem of multi-step traffic flow prediction.

5.1 Data

The data used in this work comes from the California Department of Transportation's (Caltrans) Performance Metric System (PeMS) and is freely available on the web ("Caltrans PeMS Data Warehouse", 2021). The data was collected between January 1, 2017 and December 31, 2018 in Caltrans District 4 - locations in and around the San Francisco Bay area. Among other sources, Caltrans collects vehicle traffic data on California roadways by inductive loop sensors (ILS) deployed widely throughout the California road system. The ILS record the flow, average occupancy, and average speed data for vehicles flowing past the ILS at a 5 minute resolution. In this work we have aggregated the data to a 15 minute resolution to yield a smoother dataset for predicting.

In addition to the ILS vehicle traffic data, PeMS also makes available the station metadata for the ILS through the data warehouse, including the station's unique station ID, the freeway the ILS is recording on, the freeway direction that the ILS is measuring traffic on, and the postmile at which the station is positioned. For instance, station 135 is located at on Hwy 1S at absolute post mile 78; absolute postmiles on a California roadway begin at 0 at the northern (western) most point of the roadway in California and increases by one for each mile driven south (east) on the road for N/S oriented roadways (E/W respectively). The meta-data for the ILS allows us to build a knowledge graph about the road system including properties such as direction, roadway, & postmile and edges indicating that one station is upstream or downstream from another, that two stations are on the same road, etc.

5.2 Neural Network Forecasting Models

Several neural network models each with numerous model architectures were trained in order to assess the varying model hyperparameter values on forecasting accuracy, including two recurrent neural network models (LSTM and GRU), TCN, DTCN, as well as two novel graph models: a Graph CNN and a Graph LSTM. The results of the neural network models applied to the vehicle traffic time series data are reported in Appendix C. For both the recurrent and convolutional networks, we examine the effect of varying the

depth and width of the model on the forecasting accuracy, and for the convolutional networks we also explore the effect of varying numbers of convolutional filters, kernel width, and temporal dilation as well. For the convolutional networks, the model configurations varied in depth (5,15, or 20 layers deep), width (16,32, or 64 nodes wide), kernel size (4, 8 , or 12 steps wide), number of convolutional filters (16, 32 ,or 64 filters per layer) and finally on whether or not to include time dilation in the convolutional filters. In the case of the recurrent networks, the depth and width of the models were varied between 5,10, & 20 and 16, 32, nd 64 respectively. Each of the graph neural networks consisted of a deep convolutional back end with an initial hidden layer handling the graph data transformations; specifically the LSTM and GRU graph neural networks were backed by the most performant LSTM and GRU architectures from the comparative architectural experiments - 15 layers deep, 16 units wide, 32 filters per layer with a dilated kernel of width 4 in the case of the Graph CNN; and 10 layers deep with 32 node per layer for the Graph LSTM. The action of the graph data transformation hidden layer transforms - for each sensor (i.e. - graph node) at each time step - the traffic flow measurements in the network according to the adjacency matrix and property values of the node in question, for instance by concatenating the flow measurement for the node with the flow measurements of its adjacent nodes as well as the nodes sharing its properties, i.e. - freeway and direction.

In addition to the neural network models, the classic statistical time series analysis AutoRegressive (AR) model was also fit to the data to provide a statistical benchmark for the neural network models. The `statsmodels.tsa.ar_model` module of the popular `statsmodels` python package exposes a number of methods for training and predicting with an AR model, including the `ar_select_order` method which given a set of training data finds the model order which minimizes the AIC of the resulting AR model. The models were trained and evaluated using a rolling validation model, a la Bergmeir and Benitez, 2012 : for each set of sensor data, first we used the `ar_select_order` function to train an "optimal" AR model on one month's worth of training data, then we used the trained model to forecast the traffic flow at the sensor 12 time steps out into the future, and finally we moved the training window up 12 time steps to re-fit and predict the model on a new month's worth of flow data.

Finally, a baseline historical average model was used to compare against the results of the neural network models; if $\vec{y} = (y_0, \cdots, y_n)$ is the recorded (i.e. - true) flow measurements for some ILS, then the historical average model forecasts flow from time t at time steps $t + i$ for $i \in [1, 12]$ as

$$\hat{y}_{t+i} = \frac{\sum_{j=1}^{4} y_{t-j*672}}{4}$$

i.e. - the average of the recorded flow measurements at this time of day on this day of the week for the past four weeks (4 time stamps per hour, 24 hours per day, 7 days per week implies $4 * 24 * 7 = 672$ time stamps per week).

5.3 Model Training and Assessment

Each model was trained according to a 10-fold cross validation scheme. 80% of the training fold was used to train the model, while 20% was held back as validation data. The models were trained as long as the validation loss continued to decrease, with a patience of 20 epochs without an improvement in validation loss before terminating training epochs. The model weights from the training epoch with the best validation loss were then re-loaded on the model, and this best-fitting model was used to predict the traffic flow values out to 12 steps from the training data. Tensorflow/Keras was the software used to efficiently train the neural network models. The MAPE metric was used to calculate the the accuracy of the model independently for each time step prediction. If the size of each testing fold is n, the true flow values for the n training samples at time step are $y \in \mathbb{R}^n$, and our predictions for time step i is $\hat{y} \in \mathbb{R}^n$, then the MAPE for the fold at time step i is defined as

$$MAPE_i = \sum_{j}^{n} \frac{|y_j - \hat{y}_j|}{y_j}$$

We then save the MAPE for each model at each time step annotated with their model features, i.e. - the depth / width of the dense blocks of the model, and the size of the kernel, number of convolutional filters and time dilation indicator if and when this is reasonable.

The training data includes 70,000 timestamps of flow values recorded at each of the approximately 3,000 different ILS in the traffic network; the sheer size of the training data precluded training each of the modeling configurations on the full dataset. Thus, to assess the effect of the various model hyperparameters on forecasting accuracy, a random sample of 300 of the 3,000 sensors was selected as the training data set. Once the most effective model was discovered (as measured by lowest average MAPE across all time steps) the "best" model configurations were then trained on the full set of training data and the results compared against one another. The results and conclusions from those results are presented in later chapters, including our analysis of the model MAPE values for each model configuration.

Chapter 6

Conclusions

In this chapter we discuss the conclusions which can be drawn from the results of the experiments run on both the document label prediction task as well as vehicle traffic forecasting. The raw numerical results of the document label prediction experiments may be found in B while the raw results of the vehicle traffic forecasting model can be found in Appendix C. Below we analyze the results and draw the appropriate conclusions.

6.1 Document Label Prediction

We will analyze the PSL model results and neural network model results separately, as their results are wildly different.

6.1.1 PSL Model Conclusions

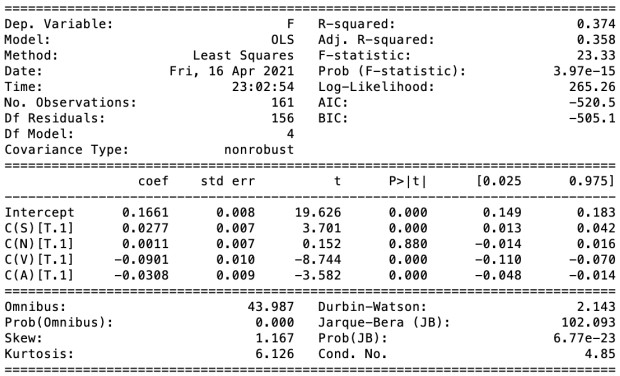
In assessing the PSL models for the document label prediction task, we would like to not only answer - affirmatively or negatively - whether PSL models are effective and efficient models for document label prediction, but we also identified four separate modeling decisions whose affect on modeling accuracy we are interested in assessing: including the use of squared hinge loss functions vs regular hinge loss function (d1) and the inclusion of each of the non citation rules : the normalizing rule (d2) - the rule which

normalizes the sum of the predicate $Label(X, L)$ for each document X over all labels L to 1, the vocab feature rule (d3), and the arithmetic rules (d4). Table 6.1 shows the average accuracy metrics for the PSL models grouped by the above modeling decisions.

Table 6.1: PSL Model Accuracies Aggregated by Design Decision

d1	d2	d3	d4	prec	rec	fscore
0	0	0	0	0.214	0.267	0.192
			1	0.085	0.288	0.131
		1	0	0.028	0.158	0.047
	1	0	0	0.212	0.164	0.174
			1	0.167	0.131	0.137
		1	0	0.041	0.195	0.067
1	0	0	0	0.210	0.266	0.185
			1	0.188	0.276	0.158
		1	0	0.173	0.161	0.127
	1	0	0	0.206	0.158	0.167
			1	0.194	0.260	0.172
		1	0	0.186	0.162	0.128

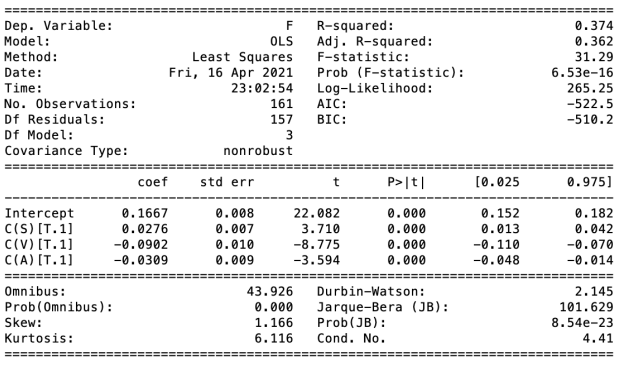
In general, we found that the difference in the mean FScore of the PSL models stratified according to the four modeling decisions was not equal. For each training/testing fold for each modeling configuration, let F denote the FScore of evaluating the model on the testing data, S be an indicator variable of whether squared hinge-loss functions were used, N an indicator variable for whether the normalizing rule was used, V an indicator variable for whether the vocab rules were used, and A an indicator variable for whether the arithmetic rule was used. The linear model $F \approx S + N + V + A$ is an ANOVA model assessing whether any of the above design decisions has a statistically significant effect on the accuracy of a PSL model in our experiments. In this case, the null hypothesis is that none of the design decisions have a statistically significant effect on the FScore of the PSL models - i.e. - the mean accuracy for a model with any of the design decisions is the same as the mean accuracy for a model with none of them. The p-value for this model is <.01, meaning that there is more than a 99% chance that at least one of the design decisions has a statistically significant effect on the modeling accuracy. See figure 6.1 for a detailed view of the ANOVA analysis.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      F   R-squared:                       0.374
Model:                            OLS   Adj. R-squared:                  0.358
Method:                 Least Squares   F-statistic:                     23.33
Date:                Fri, 16 Apr 2021   Prob (F-statistic):           3.97e-15
Time:                        23:02:54   Log-Likelihood:                 265.26
No. Observations:                 161   AIC:                            -520.5
Df Residuals:                     156   BIC:                            -505.1
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1661      0.008     19.626      0.000       0.149       0.183
C(S)[T.1]      0.0277      0.007      3.701      0.000       0.013       0.042
C(N)[T.1]      0.0011      0.007      0.152      0.880      -0.014       0.016
C(V)[T.1]     -0.0901      0.010     -8.744      0.000      -0.110      -0.070
C(A)[T.1]     -0.0308      0.009     -3.582      0.000      -0.048      -0.014
==============================================================================
Omnibus:                       43.987   Durbin-Watson:                   2.143
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              102.093
Skew:                           1.167   Prob(JB):                     6.77e-23
Kurtosis:                       6.126   Cond. No.                         4.85
==============================================================================
```

Figure 6.1: ANOVA Analysis of the 4 PSL Design Modeling Decisions

Since not all of the coefficients in the above ANOVA model have significant p-values - note that the p-value for the normalization variable is .88 - a forward selection process revealed the model with the most predictive power for determining FScore from a subset of the 4 features described above: the model which includes indicators for the vocabulary rule, the arithmetic rules and the squared hinge loss functions. See figure 6.2 for a detailed view of this ANOVA analysis as well. In particular, the ANOVA model tells us that including either the vocabulary rule or the arithmetic rule has a deleterious effect on label prediction accuracy of somewhere around 9% in the case of the former and about 3% in the case of the latter, and that including squared hinge loss functions improved the model accuracy by around 2.8%. This is completely in line with the results of the "full" ANOVA model.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      F   R-squared:                       0.374
Model:                            OLS   Adj. R-squared:                  0.362
Method:                 Least Squares   F-statistic:                     31.29
Date:                Fri, 16 Apr 2021   Prob (F-statistic):           6.53e-16
Time:                        23:02:54   Log-Likelihood:                 265.25
No. Observations:                 161   AIC:                            -522.5
Df Residuals:                     157   BIC:                            -510.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1667      0.008     22.082      0.000       0.152       0.182
C(S)[T.1]      0.0276      0.007      3.710      0.000       0.013       0.042
C(V)[T.1]     -0.0902      0.010     -8.775      0.000      -0.110      -0.070
C(A)[T.1]     -0.0309      0.009     -3.594      0.000      -0.048      -0.014
==============================================================================
Omnibus:                       43.926   Durbin-Watson:                   2.145
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              101.629
Skew:                           1.166   Prob(JB):                     8.54e-23
Kurtosis:                       6.116   Cond. No.                         4.41
==============================================================================
```

Figure 6.2: Forward Selected Model for FScore by Design Decisions

Digging further in we see that the average FScore of a model which includes the vocabulary rules is around 8.9% while a model that does not include the vocabulary rules is over 16%. While a 16% Fscore is not exactly an astounding accuracy, nonetheless the vocabulary rule is by far the most difficult rule to build a model with as the number of words in the vocabulary is huge and grounding out any rule whose variables range over words in the vocabulary is going to take a cost prohibitive amount of time. Interestingly, grouping the data by arithmetic rule indicator does not reveal a similar agreement with the ANOVA model, however; the average FScore for models including the arithmetic rule is actually slightly higher than the average FScore of the models that do not contain the arithmetic rule. This could be indicative of a Type-I error in the forward selected model. Grouping the data by squared hinge-loss indicator indicates an approximately 13% FScore for models with regular hinge-losses vs. around 16% for models which utilize the squared hinge losses, a general agreement with the ANOVA model.

6.1.2 NN Model Conclusions

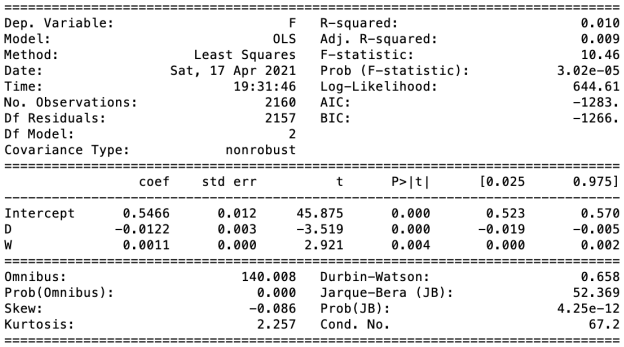
The neural network models built for document label prediction were far more performant than the PSL models, and they also took far less time to train. One of the questions we are most interested in answering is the effect - positive or negative - of using the graph edge information in the training scheme for the neural network models trained to answer the document label prediction problem. The Simple Vocabulary Model (SVM) provides an adequate benchmark for helping us answer this question and comparing the results of the various knowledge graph models against, since it is based on defining each document by its one-hot encoded vocabulary vector - a simple and common approach to document embedding which does not take into account any graph structure. If we consider the average precision of the most performant architecture (depth and width) of the SVM, we see that the most performant of the novel knowledge graph neural network architectures - the Label Sum Layer Vocabulary Model (LSLVM) - outperformed the SVM's most performant architecture not only at its own most performant architecture but in several less than optimally performing architectures as well. This answers affirmatively that leveraging the structure of the knowledge graph in training neural networks for the document label prediction task can

have a net beneficial effect on the performance of the model. Table 6.2 contains a convenient side by side comparison of the two models.

Table 6.2: Comparison of LSLVM Avg. FScore with SVM Best Fscore

depth	width	avg. LSLVM	Peak SVM
1	8	0.624	0.625
1	16	0.635	0.625
1	32	0.652	0.625
2	8	0.627	0.625
2	16	0.591	0.625
2	32	0.663	0.625
3	8	0.569	0.625
3	16	0.592	0.625
3	32	0.605	0.625
4	8	0.604	0.625
4	16	0.539	0.625
4	32	0.589	0.625

In addition to positively affirming the benefit of utilizing the document citation graph structure in training the neural network models for label prediction, we are also interested in assessing the impact of utilizing various depths and widths of model architecture in our experiments. If for each training/testing fold for each model configuration (depth/width) we let F be the FScore obtained by evaluating the model on the testing data, D the depth of the model and W the width, then the model $F \sim D + W$ should give us an indication of whether or not the depth and width of the neural network has a statistically significant effect on the accuracy of the model in label prediction. Evaluating the above linear model tells us that the depth and width of the neural network trained for document label prediction both have a significant effect on the accuracy of the model; see the linear model summary details in 6.3 for details of the effect. Specifically, each extra layer in a neural network document label prediction model retards the model's accuracy by approximately 1.22%, while each extra node added to the width of the model increases the model accuracy by around 0.1%. While this effect may seem completely insignificant at first blush, one must remember that the moderately wide neural networks trained in this work contained dense blocks which were 32 units wide, which would ultimately point to a 3.6% increase in accuracy over a single node layer.

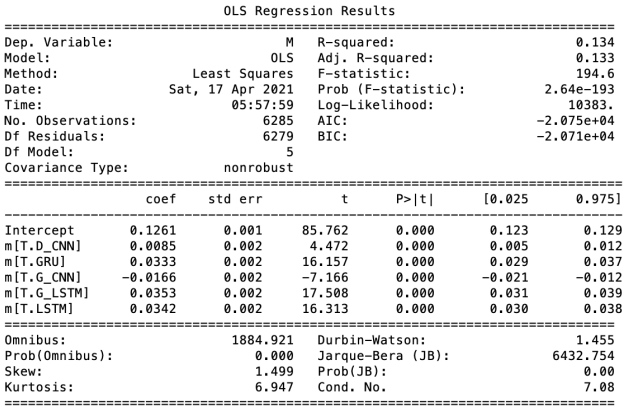


```
==============================================================================
Dep. Variable:                      F   R-squared:                       0.010
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     10.46
Date:                Sat, 17 Apr 2021   Prob (F-statistic):           3.02e-05
Time:                        19:31:46   Log-Likelihood:                644.61
No. Observations:                2160   AIC:                            -1283.
Df Residuals:                    2157   BIC:                            -1266.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5466      0.012     45.875      0.000       0.523       0.570
D             -0.0122      0.003     -3.519      0.000      -0.019      -0.005
W              0.0011      0.000      2.921      0.004       0.000       0.002
==============================================================================
Omnibus:                      140.008   Durbin-Watson:                   0.658
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               52.369
Skew:                          -0.086   Prob(JB):                     4.25e-12
Kurtosis:                       2.257   Cond. No.                         67.2
==============================================================================
```

Figure 6.3: ANOVA Model For Depth and Width of Deep NN Document Label Prediction Models

6.2 Traffic Flow Forecasting

One important question we wanted to explore in this work was the relative effectiveness of various neural network architectures in vehicle traffic flow forecasting. To that effort, we used the MAPE of the various model architectures trained on the testing/training splits to fit an ANOVA model assessing the difference in mean MAPE between the various model types, i.e. - Dilated TCN (DTCN), GRU, LSTM, Graph CNN (GCNN), Graph LSTM (GLSTM), vanilla LSTM and TCN. Figure 6.4 presents the results of the ANOVA test for forecasting traffic flow at one time step into the future; the ANOVA model indicates the relative accuracy of the various top level neural network styles, where the TCN is the base model type not represented explicitly in the ANOVA model. According to the ANOVA model the GRU is clearly the most accurate model of the bunch at the first time step, while in order of decreasing accuracy are the TCN, DTCN, GRU, LSTM, and the Graph LSTM. Since the p-values for the ANOVA model coefficients are all <.001 we can be quite sure that this is an appropriate description of the relative effectiveness of the neural network models in forecasting out to one time step. The trend is quite stable until four time steps of forecasting, at which point the p-value of the coefficients of the ANOVA model become less stable/reliable.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      M   R-squared:                       0.134
Model:                            OLS   Adj. R-squared:                  0.133
Method:                 Least Squares   F-statistic:                     194.6
Date:                Sat, 17 Apr 2021   Prob (F-statistic):          2.64e-193
Time:                        05:57:59   Log-Likelihood:                 10383.
No. Observations:                6285   AIC:                        -2.075e+04
Df Residuals:                    6279   BIC:                        -2.071e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1261      0.001     85.762      0.000       0.123       0.129
m[T.D_CNN]     0.0085      0.002      4.472      0.000       0.005       0.012
m[T.GRU]       0.0333      0.002     16.157      0.000       0.029       0.037
m[T.G_CNN]    -0.0166      0.002     -7.166      0.000      -0.021      -0.012
m[T.G_LSTM]    0.0353      0.002     17.508      0.000       0.031       0.039
m[T.LSTM]      0.0342      0.002     16.313      0.000       0.030       0.038
==============================================================================
Omnibus:                     1884.921   Durbin-Watson:                   1.455
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             6432.754
Skew:                           1.499   Prob(JB):                         0.00
Kurtosis:                       6.947   Cond. No.                         7.08
==============================================================================
```

Figure 6.4: Comparison of MAPE Values for the Various NN Model Forecasts

Figure 6.5 provides a comparison of MAPE of forecasts at all 12 time steps for all of the NN model types, as well as the historical average and AR(p) statistical baseline models. Clearly, the neural network models outperform the statistical baseline models by a wide margin.

For that reason, Figure 6.6 provides a comparison of the forecasts at all 12 time steps for the NN model types without including the statistical baseline models for a more illuminating comparison. See figure 6.7 to compare the performance of just the three CNN models; specifically notice the inflection point at time step 5 where the DTCN begins to outperform the TCN. There is a similarly inflection point in 6.8 regarding the performance of the GLSTM vs. the GRU and LSTM models.

Generally speaking the Graph Neural Network Models are much more stable in their forecast error at future time points than the vanilla RNN, TCN, and even DTCN models.

In addition to testing the relative effectiveness of the top level neural network architectures, we were interested in assessing the impact of wider/deeper architectures on traffic flow forecasting as well as the effect of increasing the number of filters, the size of the filter, and number of convolutional layers in the convolutional model, and of course the effect of dilation on the accuracy of the 1D CNN.

Figures 6.9 and 6.10 provide evidence that at a single time step prediction, a deeper model is deleterious to prediction accuracy while a wider model is typically more accurate: each new layer in depth increases

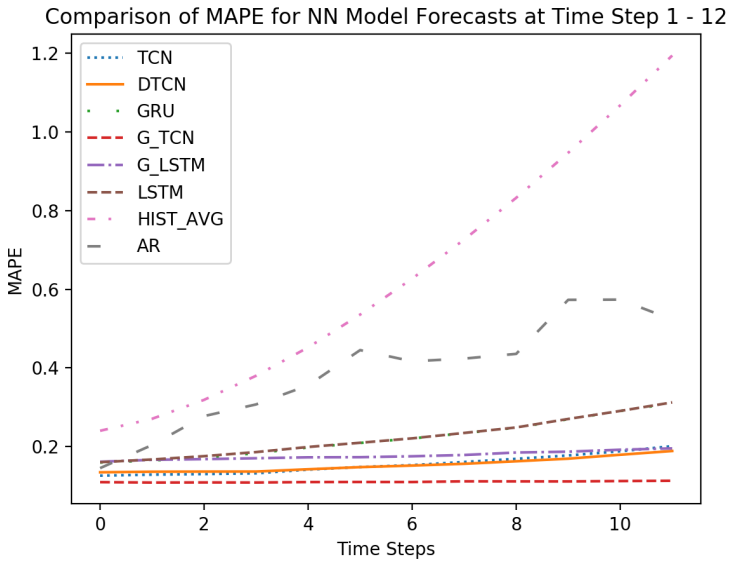


Figure 6.5: Comparison of MAPE Values for the Various NN Model Forecasts w/ Baseline

MAPE by approximately 1.65% while each unit in width decreases the MAPE by approximately 0.11%. Those results are consistent until 10 time steps out at which point there is no longer any evidence that either depth or width have any effect on modeling accuracy.

There is ample evidence in the case of the DTCN networks to suggest that time dilation achieves a lower MAPE in forecasting at all time steps, all other things being equal. Figure 6.11 is the ANOVA model suggesting the effectiveness of time dilation in improving traffic flow forecasting at a one time-step forecast, and those results are typical of the ANOVA results for all future time steps.

Similar to the dilation, there is also evidence that a larger kernel on a convolutional neural network achieves a lower MAPE in forecasting; in other words a more accurate forecaster. Figure 6.12 suggests that on average each extra time step in the kernel size results in an average MAPE approximately .53% lower. Interestingly, these results are not valid at time steps 1-5 but are valid again from time steps 6-12. This could be indicative of a type I error or it may be a legitimate result. Finally, figure 6.13 indicates that

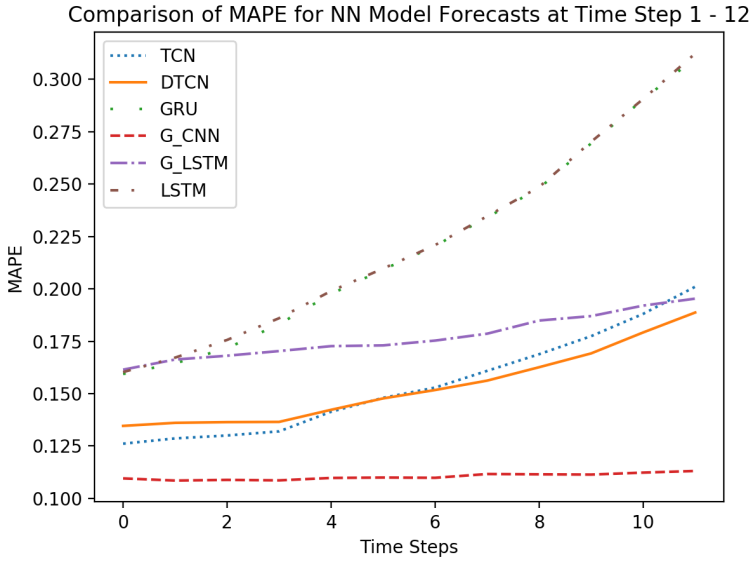


Figure 6.6: Comparison of MAPE Values for the Various NN Model Forecasts w/o Baseline

each additional filter added to a convolutional layer results in an increase in MAPE of between 1.5 and 3% depending on the time step of forecast.

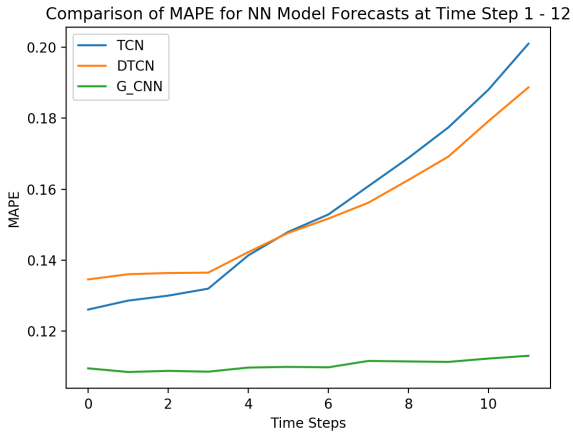


Figure 6.7: Comparison of MAPE Values for CNN Model Forecasts

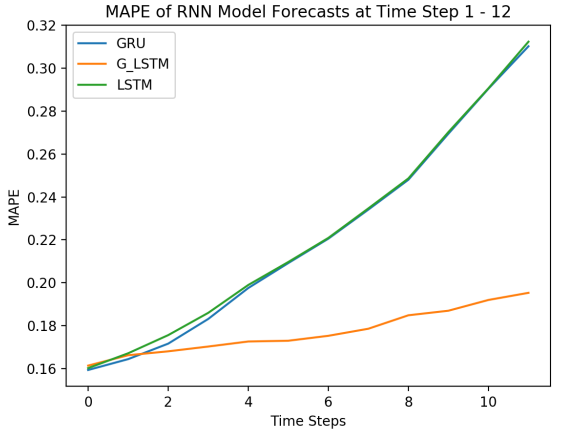
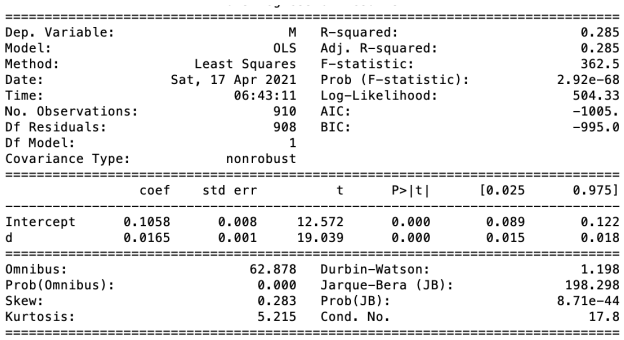


Figure 6.8: Comparison of MAPE Values for RNN Model Forecasts

Dep. Variable:	M		R-squared:		0.010	
Model:	OLS		Adj. R-squared:		0.009	
Method:	Least Squares		F-statistic:		9.276	
Date:	Sat, 17 Apr 2021		Prob (F-statistic):		0.00239	
Time:	06:43:11		Log-Likelihood:		356.13	
No. Observations:	910		AIC:		-708.3	
Df Residuals:	908		BIC:		-698.6	
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.2788	0.014	20.070	0.000	0.252	0.306
w	-0.0011	0.000	-3.046	0.002	-0.002	-0.000
Omnibus:	234.709		Durbin-Watson:		0.956	
Prob(Omnibus):	0.000		Jarque-Bera (JB):		519.836	
Skew:	1.413		Prob(JB):		1.32e-113	
Kurtosis:	5.393		Cond. No.		97.1	

Figure 6.9: ANOVA Model for Width



Dep. Variable:			M	R-squared:			0.285
Model:			OLS	Adj. R-squared:			0.285
Method:		Least Squares		F-statistic:			362.5
Date:		Sat, 17 Apr 2021		Prob (F-statistic):			2.92e-68
Time:		06:43:11		Log-Likelihood:			504.33
No. Observations:		910		AIC:			-1005.
Df Residuals:		908		BIC:			-995.0
Df Model:		1					
Covariance Type:		nonrobust					
	coef	std err	t	P>|t|	[0.025	0.975]	
Intercept	0.1058	0.008	12.572	0.000	0.089	0.122	
d	0.0165	0.001	19.039	0.000	0.015	0.018	
Omnibus:			62.878	Durbin-Watson:			1.198
Prob(Omnibus):			0.000	Jarque-Bera (JB):			198.298
Skew:			0.283	Prob(JB):			8.71e-44
Kurtosis:			5.215	Cond. No.			17.8

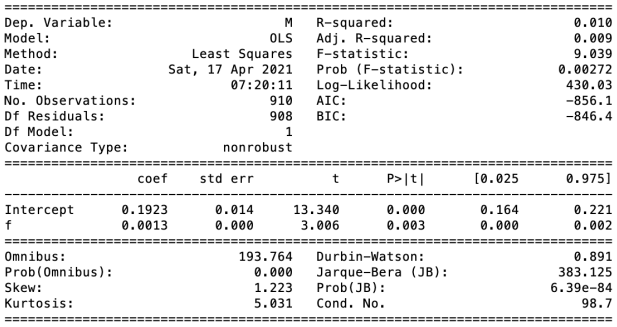
Figure 6.10: ANOVA Model for Depth

Dep. Variable:			M	R-squared:			0.016
Model:			OLS	Adj. R-squared:			0.016
Method:		Least Squares		F-statistic:			109.6
Date:		Sat, 17 Apr 2021		Prob (F-statistic):			1.88e-25
Time:		06:31:53		Log-Likelihood:			-11697.
No. Observations:		6793		AIC:			2.340e+04
Df Residuals:		6791		BIC:			2.341e+04
Df Model:		1					
Covariance Type:		nonrobust					
	coef	std err	t	P>|t|	[0.025	0.975]	
Intercept	0.8574	0.019	45.981	0.000	0.821	0.894	
C(d)[T.1]	-0.4127	0.039	-10.469	0.000	-0.490	-0.335	
Omnibus:			10223.283	Durbin-Watson:			1.816
Prob(Omnibus):			0.000	Jarque-Bera (JB):			4653794.775
Skew:			9.314	Prob(JB):			0.00
Kurtosis:			129.866	Cond. No.			2.54

Figure 6.11: ANOVA Model for Time Dilation

Dep. Variable:			M	R-squared:			0.006
Model:			OLS	Adj. R-squared:			0.005
Method:		Least Squares		F-statistic:			5.915
Date:		Sat, 17 Apr 2021		Prob (F-statistic):			0.0152
Time:		07:14:42		Log-Likelihood:			354.45
No. Observations:		910		AIC:			-704.9
Df Residuals:		908		BIC:			-695.3
Df Model:		1					
Covariance Type:		nonrobust					
	coef	std err	t	P>|t|	[0.025	0.975]	
Intercept	0.2671	0.012	21.414	0.000	0.243	0.292	
k	-0.0053	0.002	-2.432	0.015	-0.010	-0.001	
Omnibus:			220.689	Durbin-Watson:			0.924
Prob(Omnibus):			0.000	Jarque-Bera (JB):			467.475
Skew:			1.351	Prob(JB):			3.08e-102
Kurtosis:			5.242	Cond. No.			13.3

Figure 6.12: ANOVA Model for Convolutional Kernel Size



```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      M   R-squared:                       0.010
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     9.039
Date:                Sat, 17 Apr 2021   Prob (F-statistic):            0.00272
Time:                        07:20:11   Log-Likelihood:                 430.03
No. Observations:                 910   AIC:                            -856.1
Df Residuals:                     908   BIC:                            -846.4
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1923      0.014     13.340      0.000       0.164       0.221
f              0.0013      0.000      3.006      0.003       0.000       0.002
==============================================================================
Omnibus:                      193.764   Durbin-Watson:                   0.891
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              383.125
Skew:                           1.223   Prob(JB):                     6.39e-84
Kurtosis:                       5.031   Cond. No.                         98.7
==============================================================================
```

Figure 6.13: ANOVA Model for Filters

Appendix A

Markov Random Fields

An MRF is a PGM of the joint distribution of a set of random variables $X = (X_1, \cdots, X_n)$ according to an undirected graph, $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$ where $V = \{v_1, \cdots, v_n\}$ has one vertex corresponding to each variable in X. Given a graph $G = (V, E)$ a clique $C \subseteq E$ is a completely connected subset of nodes in G, i.e. - $\forall j, k(x_i \in C \wedge x_j \in C \rightarrow (x_i, x_j) \in E)$. In addition to the nodes and edges in the graph G, the MRF associated with the graph also has a set of potential functions, $\{\phi_1, \cdots, \phi_m\}$ associated with each clique C_i of the graph where $\phi_i : X \rightarrow \mathbb{R}^+$. The potential function ϕ_i *should* capture some relevant information about the underlying probability distribution associated with the MRF, i.e. - for all $x, y \in X$, if x is less likely than y, then $\phi(x) < \phi(y)$. The probability density function of the MRF is

$$P(x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{A.1}$$

where $x_{\{k\}}$ is the state of the k'th clique in the graph according to x. Since the potential functions are unbounded, non-negative real values, Z is the partitioning function

$$Z = \sum_{x \in X} \prod_k \phi_k(x_{\{k\}})$$

which confines the probability density function to a range of $[0, 1]$. A more convenient (and common) expression of the above probability density function is the log-linear model

$$P(x) \propto \exp(\sum_j w_j f_j(x)) = \exp(w^T f(x)) \tag{A.2}$$

where the potential functions in (A.1) are replaced with an exponentially weighted sum of features of the state of X. While the features in (A.2) may be any real-valued function of the state of X, they are often assumed to be binary valued functions $f_j(x) \in \{0, 1\}$.

While the most pedantic reading of (A.2) suggests that there should be one feature for each clique in the graph, we are free to define a much smaller number of features to yield a more tractable probability density function. Moving towards our definition of the HL-MRF, the logical rules of a first order knowledge base (KB) provide a means of specifying a compact set of features to define a Markov Random Field.

Consider a KB which consisting of a set of disjunctive logical clauses $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$. Let $\mathcal{I}_j^-$ and $\mathcal{I}_j^+$ be (respectively) the sets of the indices of the negated and non-negated literals in clause $\mathcal{C}_j$. The KB may be embedded in an MRF by assigning a potential function, f_j, to each clause, $\mathcal{C}_j$, where $f_j(x) = 1$ whenever $\mathcal{C}_j$ is satisfied by x and 0 otherwise. The corresponding weight, w_j, for the potential f_j in the MRF should express a degree of confidence for $\mathcal{C}_j$. In other words, the weight w_j should be proportional to the probability that the clause $\mathcal{C}_j$ is satisfied for any variable assignment x.

Given a MRF defined by a KB according to the above schema, MAP inference over the MRF is the integer linear program (ILP) equation

$$\begin{aligned} \arg\max_{x \in \{0,1\}^n} P(x) &= \arg\max_{x \in \{0,1\}^n} w^T f(x)) \\ &= \arg\max_{x \in \{0,1\}^n} \sum_{C_j \in \mathcal{C}} w_j \min\{\sum_{i \in \mathcal{I}_j^+} x_i + \sum_{i \in \mathcal{I}_j^-} (1 - x_i), 1\} \end{aligned} \tag{A.3}$$

Maximization of objective function (A.3) can be achieved through either mapping the ILP to an instance of the MAX-SAT problem and approximating the optimal solution through convex optimization or alternatively by applying local consistency relaxation to the MRF. However, the authors of Bach et al., 2017 showed that not only are these two approaches equivalent in that they provide identical solutions to the same unified optimization problem, but they also show that approximate optimization of the unified objective function is equivalent to exact MAP inference when the KB is interpreted through Lukasiewicz logic, a logic for reasoning and inference over vague or "fuzzy" logic as well as reasoning over other continuous domains wherein logical atoms are assigned a value in the continuous interval $[0, 1]$ instead of the classical binary $\{0, 1\}$ values. The equivalence allows for reasoning over both discrete and continuous domains with a single modeling paradigm, namely HL-MRF.

Appendix B

Document Classification Model Results Tables

The results for the various document classification models - both the PSL models as well as NN models - are presented here. Each PSL model was built with a combination of the four rules presented in table B.1, and the models were defined using both squared and first-order hinge-loss functions. When a rule configuration includes a squared hinge-loss function for some rule, then the key for that rule will be appended with an "x". Hence, the configuration "1,2,3,4" denotes a PSL model which includes all 4 PSL rules, and all 4 rules are defined as first-order hinge-loss functions. On the other hand, the configuration "1,3x,4" only includes 3 PSL rules (rule 2 is missing) and rule 3 is configured with a squared hinge-loss function. The accuracy metrics reported are the categorically weighted average precision, recall and FScores. All combinations of rules and hinge-loss exponents from the experiments are included in one of the tables below.

The results for the NN models are presented by the base model configuration : the Flat Label Layer Vocab Model (FLLVM), the Label Sum Layer Vocab Model (LSLVM), the Simple Edge Layer Vocab Model (SELVM), the Simple Edge Model (SEM), the Simple Label Sum Model (SLSM), and the Simple Vocabulary Model (SVM). The results for each base model are more specifically reported according to

the dense architectures backing them, and those architectures are defined by their reported widths and depths.

Table B.1: PSL Rule Key Legend

PSL Rule	Rule Key Number
Cites(X,Y) && Label(Y,Z) => Label(X,Z)	1
Cites(X,Y) && Label(X,Z) => Label(Y,Z)	1
1/|X| Label(+X,Y) = Label(Z,Y)	2
1/|X| Label(+X,Y) = Label(Z,Y)	2
Contains(X,W) && Contains(Y,W) && Label(X,L) => Label(Y,L)	3
Label(X,+Y) = 1	4

Table B.2: Accuracy Metrics of Various PSL Configurations, Part 1

PSL Rules Included (By Key)	Avg Precision	Avg Recall	Avg F1
3x	.145	.139	0.110
1x,3x	.123	.088	.077
1,3,4	.056	.192	.086
1x,3x,4	0.156	0.110	0.0971
3x,4	0.117	0.106	0.0723
3	0.024	0.134	0.040
3,4	0.023	0.136	0.039
1x	0.123	0.139	0.121
1x,4	0.160	0.197	0.146
1,4	0.164	0.1215	0.132
1	0.1645	0.2	0.1505
1,2,4	0.141	0.110	0.114
1,2	0.075	0.257	0.115
2x	0.161	0.254	0.133
2x,4	0.180	0.227	0.157
1x,2x,4	0.165	0.239	0.148
1x,2x	0.179	0.244	0.153
2,4	0.161	0.126	0.133
2	0.079	0.264	0.121

Table B.3: Accuracy Metrics of Various FLLVM Configurations

depth	width	precision	recall	fscore
1	8	0.649	0.609	0.606
1	16	0.676	0.615	0.613
1	32	0.684	0.609	0.611
2	8	0.614	0.562	0.548
2	16	0.666	0.560	0.547
2	32	0.700	0.606	0.603
3	8	0.630	0.562	0.543
3	16	0.669	0.572	0.559
3	32	0.711	0.594	0.590
4	8	0.648	0.586	0.571
4	16	0.692	0.579	0.565
4	32	0.707	0.590	0.579

Table B.4: Accuracy Metrics of Various LSLVM Configurations

depth	width	precision	recall	fscore
1	8	0.686	0.630	0.624
1	16	0.700	0.641	0.635
1	32	0.717	0.651	0.652
2	8	0.685	0.632	0.627
2	16	0.706	0.601	0.591
2	32	0.724	0.665	0.663
3	8	0.662	0.591	0.569
3	16	0.698	0.605	0.592
3	32	0.710	0.614	0.605
4	8	0.658	0.615	0.604
4	16	0.683	0.561	0.539
4	32	0.709	0.602	0.589

Table B.5: Accuracy Metrics of Various SELVM Configurations

depth	width	precision	recall	fscore
1	8	0.660	0.605	0.600
1	16	0.671	0.605	0.603
1	32	0.674	0.611	0.607
2	8	0.645	0.590	0.582
2	16	0.671	0.586	0.578
2	32	0.695	0.615	0.613
3	8	0.642	0.591	0.576
3	16	0.668	0.578	0.564
3	32	0.698	0.589	0.581
4	8	0.621	0.576	0.561
4	16	0.662	0.561	0.549
4	32	0.705	0.590	0.582

Table B.6: Accuracy Metrics of Various SVM Configurations

depth	width	precision	recall	fscore
1	8	0.658	0.592	0.584
1	16	0.665	0.624	0.618
1	32	0.701	0.622	0.623
2	8	0.629	0.571	0.557
2	16	0.672	0.587	0.577
2	32	0.695	0.630	0.625
3	8	0.645	0.599	0.585
3	16	0.695	0.568	0.554
3	32	0.706	0.619	0.611
4	8	0.589	0.526	0.493
4	16	0.674	0.572	0.557
4	32	0.688	0.573	0.559

Table B.7: Accuracy Metrics of Various SEM Configurations

depth	width	precision	recall	fscore
1	8	0.535	0.465	0.435
1	16	0.528	0.461	0.435
1	32	0.563	0.469	0.451
2	8	0.539	0.466	0.440
2	16	0.540	0.474	0.447
2	32	0.553	0.472	0.452
3	8	0.518	0.451	0.423
3	16	0.554	0.474	0.449
3	32	0.561	0.480	0.454
4	8	0.508	0.441	0.410
4	16	0.540	0.466	0.439
4	32	0.545	0.463	0.428

Table B.8: Accuracy Metrics of Various SLSM Configurations

depth	width	precision	recall	fscore
1	8	0.527	0.447	0.423
1	16	0.527	0.466	0.437
1	32	0.545	0.475	0.448
2	8	0.512	0.443	0.414
2	16	0.527	0.463	0.438
2	32	0.543	0.470	0.445
3	8	0.541	0.442	0.416
3	16	0.536	0.463	0.437
3	32	0.559	0.480	0.453
4	8	0.550	0.464	0.437
4	16	0.537	0.466	0.435
4	32	0.542	0.469	0.440

Appendix C

Time Series NN Forecasting Model Results

The results for the various neural network time series forecasting model approaches are presented here. Each time series model was fit to the 15 minute resolution data, and each model was used to predict traffic flow from 1 to 12 time steps into the future, which for the 15 minute resolution data is out to 2 hours. The accuracy of the models is reported as the mean average percentage error (MAPE) for each forecasting step in the future. A variety of model architectures were trained for each general neural network type: Gated Recurrent Unit (GRU), Long Short Term Memory (LSTM), Temporal Convolutional Neural Network (TCN) - both dilated and not. While the only configurable parameters for the LSTM and GRU MODELS are the depth and width of the models, the CNN models also include a kernel size as well as number of filters.

Table C.1: Comparison of MAPE for Various TCN Traffic Forecasters, Part 1

depth	width	kernel	filters	dilated	step0	step1	step2	step3	step4	step5
5	16	4	32	0	0.188	0.198	0.225	0.262	0.315	0.341
				1	0.202	0.204	0.225	0.260	0.305	0.341
	32	4	16	0	0.194	0.201	0.226	0.261	0.306	0.331
				1	0.207	0.212	0.233	0.269	0.314	0.350
			32	0	0.196	0.202	0.225	0.260	0.304	0.330
				1	0.200	0.206	0.227	0.264	0.310	0.346
			64	0	0.213	0.219	0.239	0.273	0.325	0.348
				1	0.209	0.211	0.229	0.264	0.311	0.347
		8	32	0	0.215	0.222	0.242	0.274	0.317	0.341
				1	0.210	0.211	0.228	0.262	0.309	0.344
		12	32	0	0.211	0.217	0.237	0.269	0.315	0.334
				1	0.206	0.210	0.230	0.267	0.312	0.347
	64	4	16	0	0.097	0.100	0.123	0.101	0.124	0.142
				1	0.107	0.086	0.088	0.106	0.121	0.148
			32	0	0.195	0.205	0.230	0.263	0.307	0.331
				1	0.199	0.204	0.224	0.260	0.306	0.340
15	16	4	32	0	0.097	0.100	0.113	0.102	0.130	0.143
				1	0.089	0.090	0.093	0.110	0.126	0.156
	32	4	32	0	0.495	0.443	0.385	0.363	0.385	0.364
				1	0.520	0.464	0.401	0.381	0.392	0.380
20	32	4	32	0	0.473	0.426	0.373	0.356	0.387	0.369
				1	0.509	0.453	0.396	0.377	0.392	0.384

Table C.2: Comparison of MAPE for Various TCN Traffic Forecasters, Part 2

depth	width	kernel	filters	dilated	step6	step7	step8	step9	step10	step11
5	16	4	32	0	0.358	0.395	0.412	0.437	0.455	0.492
				1	0.359	0.389	0.406	0.440	0.441	0.473
	32	4	16	0	0.348	0.388	0.410	0.437	0.453	0.487
				1	0.367	0.396	0.414	0.447	0.449	0.481
			32	0	0.346	0.385	0.407	0.433	0.447	0.482
				1	0.364	0.396	0.414	0.450	0.452	0.485
			64	0	0.363	0.402	0.425	0.452	0.467	0.505
				1	0.366	0.401	0.421	0.459	0.463	0.498
		8	32	0	0.354	0.394	0.416	0.443	0.456	0.493
				1	0.362	0.398	0.420	0.458	0.459	0.493
		12	32	0	0.348	0.385	0.406	0.433	0.448	0.482
				1	0.363	0.396	0.415	0.449	0.449	0.482
	64	4	16	0	0.153	0.179	0.203	0.229	0.228	0.251
				1	0.159	0.176	0.191	0.178	0.200	0.235
			32	0	0.346	0.386	0.409	0.435	0.450	0.484
				1	0.359	0.393	0.412	0.449	0.453	0.487
15	16	4	32	0	0.146	0.172	0.203	0.226	0.211	0.242
				1	0.157	0.163	0.171	0.164	0.173	0.199
	32	4	32	0	0.350	0.383	0.405	0.439	0.462	0.509
				1	0.370	0.400	0.423	0.469	0.479	0.524
20	32	4	32	0	0.362	0.400	0.424	0.458	0.484	0.536
				1	0.374	0.401	0.423	0.465	0.476	0.517

Table C.3: Comparison of MAPE for Various RNN Traffic Forecasters, Part 1

RNN type	width	depth	step0	step1	step2	step3	step4	step5
GRU	16	10	0.173	0.159	0.146	0.178	0.223	0.233
	32	5	0.164	0.158	0.140	0.171	0.209	0.223
		10	0.127	0.135	0.129	0.147	0.163	0.192
		20	0.145	0.147	0.150	0.168	0.190	0.209
	64	10	0.131	0.142	0.139	0.160	0.192	0.207
LSTM	16	10	0.161	0.147	0.131	0.142	0.162	0.183
	32	5	0.152	0.137	0.121	0.135	0.155	0.171
		10	0.132	0.136	0.127	0.140	0.152	0.166
		20	0.142	0.151	0.149	0.178	0.197	0.198
	64	10	0.136	0.138	0.143	0.176	0.192	0.181

Table C.4: Comparison of MAPE for Various RNN Traffic Forecasters, Part 2

RNN type	width	depth	step6	step7	step8	step9	step10	step11
GRU	16	10	0.231	0.255	0.241	0.262	0.308	0.348
	32	5	0.218	0.243	0.232	0.250	0.293	0.315
		10	0.194	0.228	0.216	0.242	0.282	0.323
		20	0.218	0.245	0.236	0.262	0.302	0.341
	64	10	0.204	0.236	0.228	0.255	0.295	0.324
LSTM	16	10	0.184	0.220	0.207	0.213	0.256	0.320
	32	5	0.176	0.207	0.198	0.195	0.232	0.301
		10	0.175	0.212	0.209	0.216	0.261	0.319
		20	0.210	0.236	0.247	0.249	0.294	0.351
	64	10	0.191	0.217	0.215	0.235	0.261	0.327

Table C.5: NN Traffic Forecasting MAPE, Part 1

model	step0	step1	step2	step3	step4	step5
TCN	0.151	0.157	0.160	0.160	0.170	0.178
DTCN	0.170	0.174	0.175	0.174	0.179	0.186
GRU	0.150	0.156	0.164	0.175	0.188	0.200
GCNN	0.177	0.178	0.181	0.183	0.187	0.190
GLSTM	0.161	0.166	0.169	0.171	0.174	0.175
LSTM	0.160	0.169	0.179	0.190	0.203	0.215
HistAvg	0.240	0.270	0.318	0.380	0.452	0.535
AR	0.317	0.434	0.647	0.902	1.172	1.296

Table C.6: NN Traffic Forecasting MAPE, Part 2

model	step6	step7	step8	step9	step10	step11
TCN	0.184	0.191	0.196	0.205	0.215	0.226
DTCN	0.191	0.194	0.198	0.204	0.213	0.220
GRU	0.213	0.225	0.237	0.254	0.270	0.288
GCNN	0.192	0.197	0.199	0.202	0.206	0.210
GLSTM	0.178	0.182	0.189	0.192	0.198	0.202
LSTM	0.226	0.240	0.253	0.269	0.285	0.303
HistAvg	0.627	0.726	0.832	0.947	1.067	1.194
AR	1.444	1.604	1.789	1.825	1.866	1.913

Bibliography

Apache Jena. (2021). https://jena.apache.org. Accessed: 2021-01-01.

Bach, S. H., Broecheler, M., Huang, B., & Getoor, L. (2017). Hinge-loss markov random fields and probabilistic soft logic. *The Journal of Machine Learning Research*, *18*(1), 3846–3912.

Bergmeir, C., & Benitez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, *191*, 192–213.

Boukerche, A., & Wang, J. (2020). Machine learning-based traffic prediction models for intelligent transportation systems. *Computer Networks*, *181*, 107530.

Bruckner, D., Rosen, J., & Sparks, E. (2013). Deepviz: Visualizing convolutional neural networks for image classification. *URL http://vis. berkeley. edu/courses/cs294-10-fa13/wiki/images/f/fd/DeepVizPaper. pdf. DIGITS*.

Caltrans PeMS Data Warehouse. (2021). pems.dot.ca.gov. Accessed: 2021-01-01.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Domingos, P., Kok, S., Lowd, D., Poon, H., Richardson, M., & Singla, P. (2008). Markov logic. In *Probabilistic inductive logic programming* (pp. 92–117). Springer.

Gentile, C., & Warmuth, M. K. (1998). Linear hinge loss and average margin. *Advances in neural information processing systems*, *11*, 225–231.

Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. (1983). Building expert system.

Jiang, H. (2019). Why learning of large-scale neural networks behaves like convex optimization. *CoRR*, *abs/1903.02140*. arXiv: 1903.02140. Retrieved from http://arxiv.org/abs/1903.02140

Jiang, W., & Luo, J. (2021). Graph neural network for traffic forecasting: A survey. *arXiv preprint arXiv:2101.11174*.

Khosravi, H., & Bina, B. (2010). A survey on statistical relational learning. In *Canadian conference on artificial intelligence* (pp. 256–268). Springer.

LINQS Datasets. (2021). https://linqs.soe.ucsc.edu/data. Accessed: 2021-01-01.

Lu, Q., & Getoor, L. (2003). Link-based classification. In *International conference on machine learning (icml)*, Washington, DC, USA: HP.

Ma, Y., Zhang, Z., & Ihler, A. (2020). Multi-lane short-term traffic forecasting with convolutional lstm network. *IEEE Access*, *8*, 34629–34643.

Mallick, T., Balaprakash, P., Rask, E., & Macfarlane, J. (2019). Graph-partitioningbased diffusion convolution recurrent neural network for large-scale traffic forecasting. *arXiv preprint arXiv:1909.11197*.

Neo4J. (2021). Accessed: 2021-01-01.

Oracle Graph Database. (2021). https://www.oracle.com/database/graph/. Accessed: 2021-01-01.

Resource Description Framework - RDF. (2021). https://www.w3.org/2001/sw/wiki/RDF. Accessed: 2021-01-01.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine learning*, *62*(1-2), 107–136.

Sen, P., Namata, G. M., Bilgic, M., Getoor, L., Gallagher, B., & Eliassi-Rad, T. (2008). Collective classification in network data. *29*(3), 93–106.

Srinivasan, S., Babaki, B., Farnadi, G., & Getoor, L. (2019). Lifted hinge-loss markov random fields. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 7975–7983).

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., … Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv: 1609.03499 [cs.SD]

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*(12), 2724–2743.

Wei, L., Yu, Z., Jin, Z., Xie, L., Huang, J., Cai, D., … Hua, X.-S. (2019). Dual graph for traffic forecasting. *IEEE Access*.