Deep Learning Network Management

Modern AI for modern systems

Ben Folland, Thomas Gregory, Arthur Hayhoe, James Readman & Joshua Smart

Exeter Mathematics School

`{ benfolland, thomasgregory, arthurhayhoe, jamesreadman, joshuasmart }@exe-coll.ac.uk`

EXETER MATHS SCHOOL
FREEDOM FOR THINKERS

Introduction

The world today is so much more complex than it was even a few decades ago, and with this, the systems that we use become more complicated and harder to manage. Today it is simply impossible for humans to work fast enough to keep up with all the information that needs to be processed.

Normally, a computer would deal with processes too fast for a human, but computers are not perfect. A computer program does what it is told and nothing else. This creates an issue when we deal with problems with more ambiguous answers. In order to get around this, a computer would have to think more like a human, to learn and adapt.

1 Fundamental concepts of Digital Twins

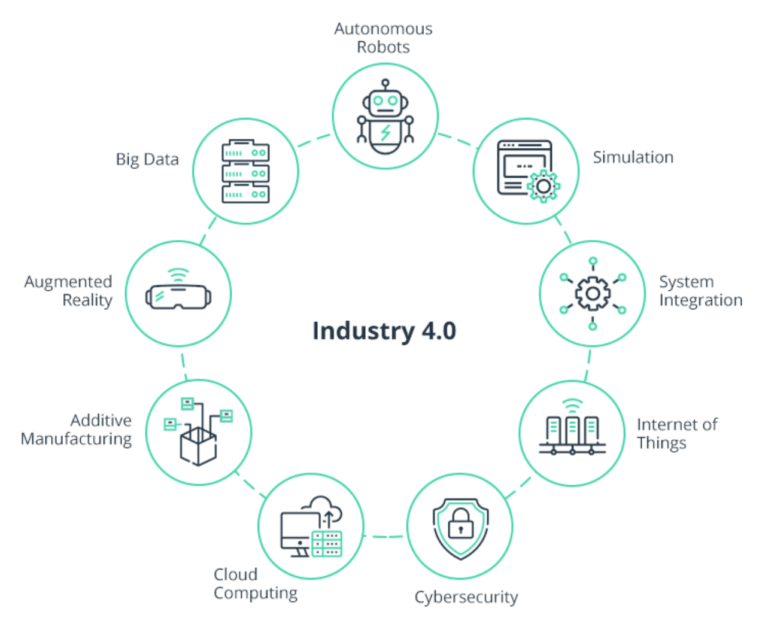

Figure 1: uses of a digital twin

A digital twin is a digital copy of a real world system. As technology improves, the systems that we implement get more complex. This comes with an issue, the more complex a system is, the more difficult it is to predict every outcome. Questions like "what would happen if X breaks" end up having so many possibilities that it becomes impossible for a human to think of all of them; In order to fix this issue, the concept of digital twin was designed. The idea is to have 2 versions of a system running at the same time, one being the real world system, and the other, a simulated copy running alongside the real system.

The digital twin takes in data gathered by the real system and runs it through many scenarios based off of what may happen to the real model (something breaking, upgrades to the system etc.). This allows for a developer to test scenarios for a product or system without spending money, time or resources changing the existing system.

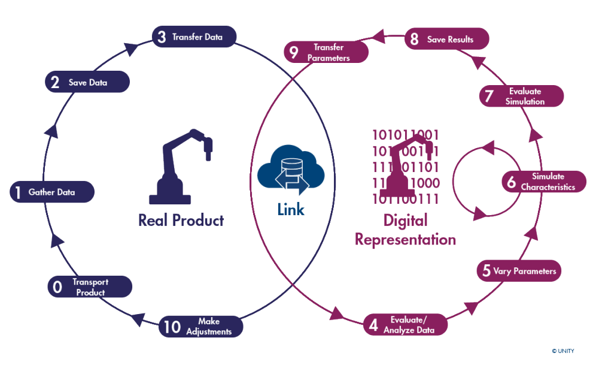

Figure 2: the standard model for a digital twin

To decide what to do with the data a twin may use a **Deep Neural Network.**

2 Deep Neural Networks

Leveraging the structure and function of the human brain, deep neural networks consist of nodes and connections between them, information is propagated from the input nodes, through a series of **layers**, **weights** and **biases** to the output nodes which are interpreted to form a *"decision"*.

A deep neural network is an artificial neural network with multiple layers between the input and output layers (hidden layers). The deep neural network finds the correct mathematical manipulation of **weights** and **biases** to turn the input into the output, whether it be a linear relationship or a non-linear relationship. Different subsets of deep neural networks include **Convolutional**, **Recurrent** and **Graph**. These types affect the structure of deep neural network and are specialised for different use cases. These different types will be summarised in the following chapters.

Standard deep neural networks can be represented by graphs, visualising the input and output nodes, along with the weighted connections between them:

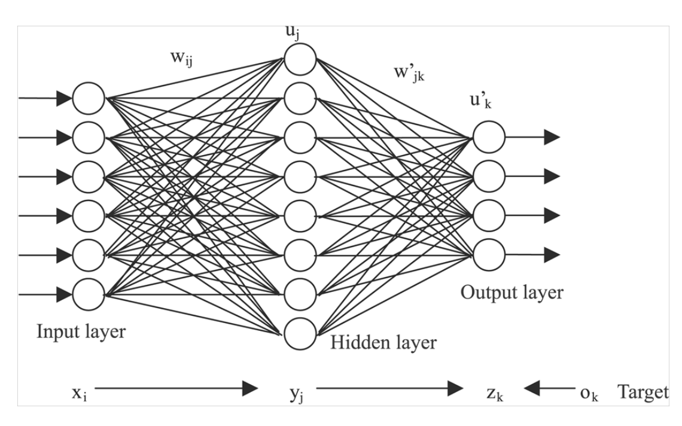

Figure 3: a representation of a neural network

2.1 Convolutional Neural Networks

Convolutional neural networks are a subset of deep neural networks commonly applied to tasks that requires identifying/analyzing images. They usually take an image as an input, and assign different weights and biases to various aspects and objects in the image to help differentiate from one and another.

The computer sees an input image as an array of pixels. For example an image could be represented as a 6 x 6 x 3 matrix, 6 pixels wide, 6 pixels high and 3 RGB color values.

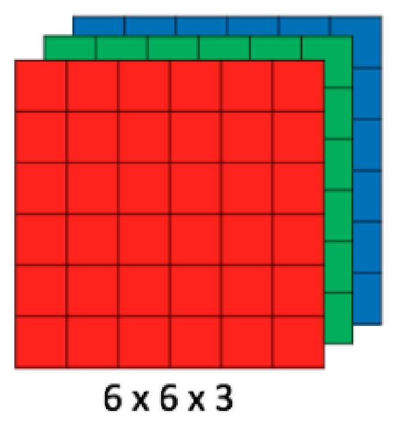

Figure 4: a pixel matrix

A trained neural network will then pass the values of the pixels through a series of convolutional layers, pooling, and more 'hidden' layers before applying a Softmax function which assigns a probability of the input being a specific object/image. The largest probability will be used to decide what the object is.

The network will then use data from previous cycles and see if the image provided has similarities with what it's trying to recognise.

2.2 Recurrent Neural Networks

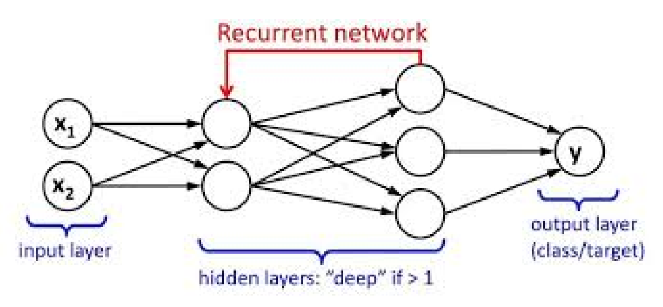

Figure 5: a diagram of a recurrent neural network

A recurrent network is another subset of Deep learning algorithms, commonly used to identify written words and different handwriting styles instead of images. Unlike the other kinds of neural network, this type is not deterministic based on the input data, the order in which the data is processed is now another factor that the network uses. For instance, in recognising speech patterns it is useful to understand how the inputs change over time.

2.3 Graph Neural Networks

Before understanding a graph neural network, we must first understand what is meant by graph. In it's simplest form a graph is a data structure that is described by it's set of vertices(nodes) and edges.

$$G = (V, E)$$

An edge can be directed or un-directed depending on weather or not it is necessary in the model; directed edges have a source and a destination, whereas un-directed edges simply connect 2 nodes.

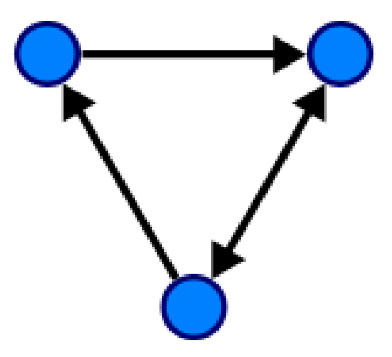

Figure 6: a directed graph

When a GNN operates on this graph structure, each node is given a "label", the network's job is to predict the labels of nodes given only a set of starting conditions. To define a label of a node a GNN takes in a weighted combination of all the connections in the system, 'passing' information through the graph. This process is repeated an arbitrary number of times and labels are calculated based on the information that was passed into the node.

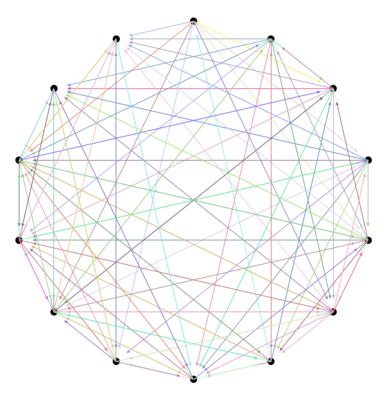

Figure 7: A representation of a much larger network

We were tasked with using Keras (a python GNN) to make a digital twin of a network.

3 Network optimisation

3.1 Building the AI

For our network we utilised a standard Deep Neural Network model, produced by the Python library Keras[2]. The required input data should be formatted as a one dimensional array, the pseudo-code algorithm below shows how the network data was converted into this:

```
Algorithm 1 Reformat network data to 1D array
Require: R, t
  out ← empty list
  index ← 0
  for row in R do
    for element in row do
      out[index] ← element
      index ← index + 1
    end for
  end for
  out[index] ← t
```

With R as the routing scheme, a 2D adjacency matrix representing the connections across the graph, and t as the traffic intensity.

This data was then fed into the network, allowing it to *"learn"* the properties of graphs.

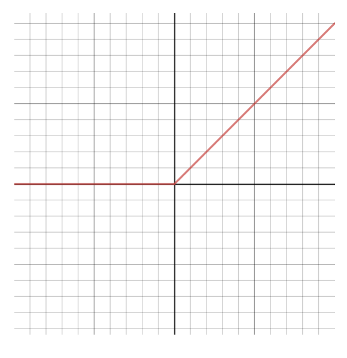

Figure 8: ReLU activation function

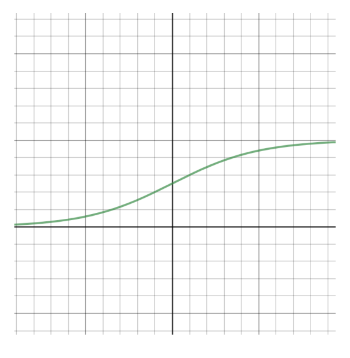

Figure 9: Sigmoid activation function

The sample data (of 598 entries) was split into 2 sets, training and testing. After the training process, through the Keras API, the model was evaluated on the testing data it had never seen before. Many of the choices for the model were made arbitrarily, simply making an educated guess and seeing what works. Using this model we can quickly and efficiently produce delay statistics for a network to a large degree of accuracy.

3.2 Optimisation algorithm

Below is a pseudo-code representation of the optimisation algorithm built in python:

```
Algorithm 1 Network optimisation algorithm
Require: R, t

  base_value ← NN(R,t)
  delta_score_values ← empty array
  for i ← 0 to length(R) do
    R[i] ← R[i] + 1
    delta_score_values[i] ← base_value − NN(R,t)
    R[i] ← R[i] − 1
  end for

  min_record, min_record_index ← 0, 0
  max_record, max_record_index ← 100, 0

  for i ← 0 to length(delta_score_values) do

    if delta_score_values[i] < min_record then
      min_record ← delta_score_values[i]
      min_record_index ← i
    end if

    if delta_score_values[i] > max_record then
      max_record ← delta_score_values[i]
      max_record_index ← i
    end if

  end for

  return min_record_index, max_record_index
```

With R as the 1D representation of the network data, from the algorithm in section 3.1. And t as the traffic intensity. This algorithm is used to tell which changes to the network topology will result in the highest and lowest changes in 'score' (an abstract measure of success defined as the sum of the delays of all node pairs in the network for this model)

4 Results

For this project we built 3 different network models, each with slightly different configurations.

4.1 Model Version 1

Input nodes	197
Hidden layer nodes	1000
Output nodes	196
Major activation function	Sigmoid
Loss mode	Binary Crossentropy
Accuracy on training data	49.75%
Accuracy on testing data	29.74%

This was our first attempt at building and training a network, from this we realised that the sigmoid activation function and Binary Crossentropy loss was detrimental and thus produced a low accuracy.

4.2 Model Version 2

Input nodes	197
Hidden layer nodes	500
Output nodes	196
Major activation function	ReLU
Loss mode	Mean Squared Error
Accuracy on training data	100%
Accuracy on testing data	100%

These results were much better than the first attempt, with results that rounded up to 100% accuracy. However there was a problem with this model, it was far too slow, in an attempt to improve accuracy we drastically increased the number of hidden layers in the network, this made it a lot more computationally intensive, so much so that it was no longer useful.

4.3 Model Version 3

Input nodes	197
Hidden layer nodes	250
Output nodes	196
Major activation function	ReLU
Loss mode	Mean Squared Error
Accuracy on training data	100%
Accuracy on testing data	94.94%

This final model, exemplifying all the successful properties from the last two, produces very accurate results at a dramatic reduction in computation from model 2, we reduced the number of nodes in each hidden layer, realising that the number of hidden layers was more significant than the nodes within them for our application.

5 Conclusions

With dramatic advances in AI technology in the last decade, new applications are being discovered and pursued every day, our research showing just one. Our delve into network management has highlighted the many benefits of Deep Learning.

Faster, simpler and more efficient; our network delay models show incredible accuracy across both testing and training data. With the rise of new 5G networks this kind of digital twin and AI simulation will be vital in testing and maintenance.

References

[1] Github repository for all the code used in this project. github.com/EMS-EMC-DeepLearningNetworkManagement /DeepLearningNetworkManagement.

[2] Keras. The python deep learning library used for this project.

[3] Fig 4. "miro.medium.com/max/347 /1*CBY94wikMUCZMB4-Xxs-pw.png".

[4] Fig 5. "deepai.org/machine-learning-glossary-and-terms/recurrent-neural-network".

[5] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Arparicio. The data sets provided for us for use in our digital twin. "github.com/knowledgedefinednetworking/Unveiling-the-potential-of-GNN-for-network-modeling-and-optimization-in-SDN".

[6] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Arparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn, 2019.

[7] Fig 3. "extremetech.com/wp-content/uploads/2015/07/NeuralNetwork.png".

[8] Fig 2. "unity.at/en/services/digital-twin/".

[9] Fig 1. "intellias.com/digital-twin-technology-a-guide-for-2019/".

[10] Dr Yulei Wu. Brief. The brief provided to us by Dr. Yulie Wu (University of Exeter).