Media Engineering and Technology Faculty
German University in Cairo

Bachelor Thesis Title

Bachelor Thesis

Author:	Maggie Ezzat Gamil Gaid
Supervisors:	Sup 1
	Sup 2
	Sup 3
Submission Date:	XX July, 20XX

Media Engineering and Technology Faculty
German University in Cairo

Bachelor Thesis Title

Bachelor Thesis

Author:	Maggie Ezzat Gamil Gaid
Supervisors:	Sup 1
	Sup 2
	Sup 3
Submission Date:	XX July, 20XX

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Maggie Ezzat Gamil Gaid
XX July, 20XX

Acknowledgments

Text

Abstract

Abstact

Contents

Chapter 1

Introduction

Natural Language Processing (NLP) and contextual analysis techniques have matured greatly over the last decade, with applications rising in many industries. The evolution of transport network operators to more automation requires an AI-enabled control center room. In this project, we aim to enhance the capability of our system, enabling it to understand human speech from vehicle driver input and trigger dispatcher actions automatically. As such, an AI dispatcher agent needs to understand a message/call from a driver in order to trigger the necessary action.

A second valuable application of such NLP engine, is increased automation inside the control center. As such, an artificially-intelligent (AI) dispatcher agent needs to understand a message/call from a driver in order to trigger the necessary action.

Chapter 2

Literature Review

2.1 Natural Language Processing

Natural Languages refer to languages written and spoken by human beings. They are English, German, French, etc. Humans can easily learn and understand such languages. On the other hand, computers have difficulty understanding these languages because of the ambiguity problem, as computers understand structured and unambiguous programming languages.

Natural Language Processing (NLP) is a field which aims at enhancing the human-computer interaction by giving computers the ability to understand natural languages. It encompasses two sub-fields: Natural Language Generation (NLG) and Natural Language Understanding (NLU). NLG targets making computers generate human-like sentences. NLU focuses on semantics extraction or building a comprehension of the intent. For quite a long time, NLP researches have been striving to build ASR systems and language models for narrowing down the gap between humans and machines.

In this Literature Review, two sub-domains are explored: Speech Recognition and Text Analytics, but at the beginning a quick review of the widely popular Artificial Neural Networks is elaborated in the next section.

2.2 Artificial Neural Networks

Artificial Neural Network (ANN)s are models of computation modeled after the biological neural networks that constitute the human brain. In early researches of Neural Networks, biological resemblance was emphasized [13] [14] [7], however, nowadays it is obvious that ANNs have little in common compared to biological neural networks. That is due to the biological emphasis being abandoned in favor of achieving satisfying computational results.

The basic building block of ANNs are "neurons", commonly called nodes. The set of nodes are connected to each other with weighted edges, with the weights of the edges resembling the strength of the "synapses" between the neurons as suggested by the original biological model. The neurons are represented diagrammatically by drawing them as circles, and the weighted edges as arrows connecting them. Each node has an activation function associated with it, which takes as input a weighted sum of its input nodes (Figure 2.1).

The most popular activation functions are the sigmoid function $\sigma(z) = \frac{1}{1+e^{-}z}$, the tanh function $\phi(z) = \frac{e^z - e - z}{e^z + e^{-}z}$, and the Re-Lu function $l(z) = max(0, z)$. The activation function at the output nodes is considered to be task-specific. However, the most popular ones are the softmax function $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ for $i = 1, .., K$ and $\mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K$, the sigmoid function, or simple linear functions.

There exist many variations of ANNs, the simplest form of them are those whose edges do not form any cycles. These are called Feed Forward Neural Networks.

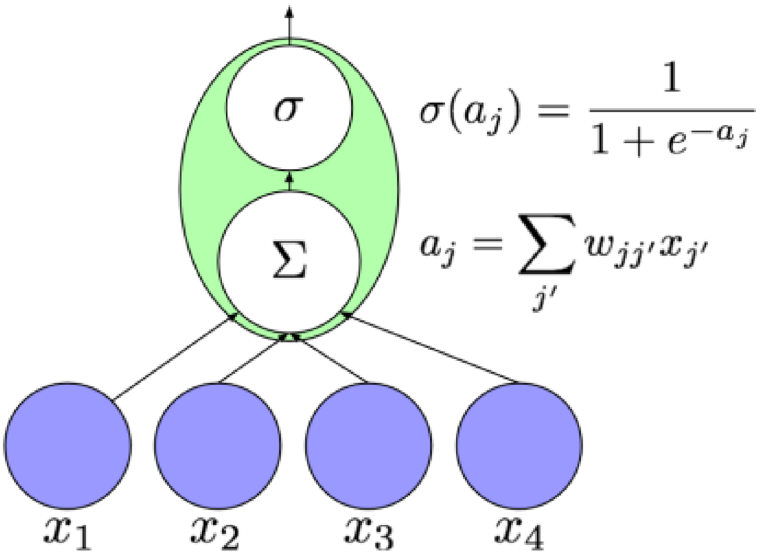
σ
$\sigma(a_j) = \frac{1}{1+e^{-a_j}}$
Σ
$a_j = \sum_{j'} w_{jj'} x_{j'}$
x_1
x_2
x_3
x_4

Figure 2.1: A neuron represented as a circle and the weighted edges as arrows. The activation function is a function of the sum of the weighted edges. [?]

2.2.1 Feed Forward Neural Networks

The most popular form of Feed Forward Neural Network (FNN)s is the Multi Layer Perceptrons (MLP)s [18] [21] [4]. With the absence of cycles, the nodes are arranged into layers, as seen in Figure 2.2, where the layers are classified as an input layer, one or many hidden layers, and an output layer.

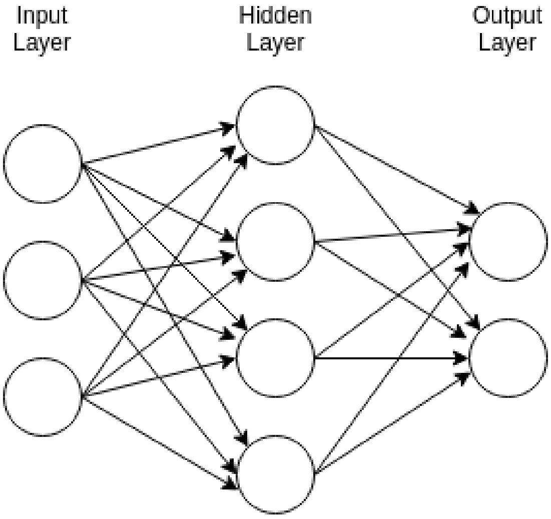

Figure 2.2: A simple Feed Forward Neural Network consisting of an input layer, one hidden layer, and an output layer

The input to FNNs is applied to the input layer. Values of nodes in a given layer, are successively calculated using the values of nodes in the lower layer, until the output is generated at the highest layer: the output layer. This is known as the “Forward pass”. Neural Networks learn by looking at input examples without being explicitly programmed any hard-coded rules about the required task. The learning process is achieved by continuously modifying the weights to minimize an error represented by a loss function $L(\widehat{y}, y)$, which measures the distance between the output y (predicted value of y) and the actual value of y (ground-truth).

The algorithm for training neural networks is back-propagation [18]. Back-propagation uses the chain rule to calculate the derivative of the loss function $L(\widehat{y}, y)$ with respect to each parameter in the network. The parameters (weights) are then adjusted in the direction of less error by an optimization algorithm called gradient descent. This is known as the "Backward Pass"

Sequence Models and the Problem with FNNs

A distinctive feature of the FNNs is the "independence assumption". That is the presented examples (data points) are assumed to be independent of each other, rendering the FNNs unable to correctly represent input or output sequences with dependencies either in time or space. Examples are words forming sentences, letters forming words, frames of video, snippets of audio clips, DNA sequences, etc. FNNs knows no concept of context when analyzing the given examples, they simply are unable to capture dependencies. With the context being a crucial element when analyzing sequences, a simple solution that addresses that matter is the "time-window" solution, i.e. collect the data from either side of the current input into a window. The fact that the range of useful context (either on the left or the right) vary widely from sequence to sequence and in most cases is unknown makes this approach not very efficient. For example, a model trained using a finite-length context window of length n could never be trained to answer the simple question, "what was the data point seen $n + 1$ time steps ago?"

Another problem with FNNs is that they treat inputs and outputs as "fixed-length vectors". Some representations, such as sentences can not be represented in such way. Some solutions such as "padding" assume a maximum-length for the inputs and/or outputs. Such approach is not a general one. Thus, it was required to extend these powerful and successful models to better suit the sequential nature of some data, and that is where the Recurrent Neural Networks came into picture.

2.2.2 Recurrent Neural Networks

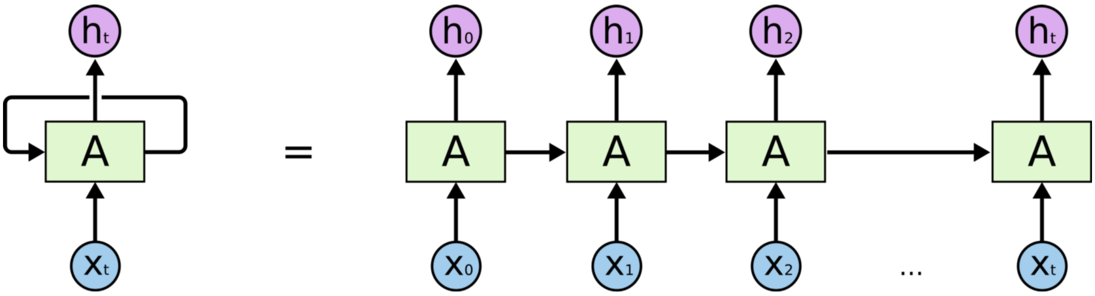

Figure 2.3: A Recurrent Neural Network

ANNs containing cycles are referred to as recursive, or recurrent neural networks. Recurrent Neural Network (RNN)s are models which have the ability to pass information learnt across past data points, while processing sequential data points one by one. Thus they can model inputs and outputs which are correlated either in time or space. They are considered to be neural networks possessing memory.

The RNNs have the following architecture: each hidden layer - commonly referred to as "hidden state" - has two sources of inputs, which are the present data point, and information from the hidden state of the past data point (Figure 2.3). This is how contextual information is propagated across the hidden states of the sequential data points. Equation 2.1 explains how each hidden state nodes values are calculated. Each hidden state $\mathbf{h_t}$ is a function of the present data point $\mathbf{x}$ multiplied by some weight matrix W^x and the previous hidden state $\mathbf{h_{t-1}}$ multiplied by some weight matrix W^h and some bias term $\mathbf{b_h}$. The weight matrices are used to determine how much importance is given to both the present data point and the past hidden state. The output $\widehat{\mathbf{y}}_\mathbf{t}$ at time step t is given by equation 2.2, where $\widehat{\mathbf{y_t}}$ is obtained by applying the softmax function to the hidden state $\mathbf{h_t}$ multiplied by some weight matrix W^y and adding to it some bias term $\mathbf{b_y}$. Similar to the vanilla ANNs, the weights are continuously adjusted to minimize a cost function. This is done using an algorithm called Backpropagation Through Time (BPTT) [22]

$$\mathbf{h_t} = \phi(W^x\mathbf{x} + W^h\mathbf{h_{t-1}} + \mathbf{b_h}) \tag{2.1}$$

$$\widehat{\mathbf{y_t}} = softmax(W^y\mathbf{h_t} + \mathbf{b_y}) \tag{2.2}$$

Bidirectional RNNs

A slight variation of the RNNs is the Bidirectional Recurrent Neural Network (BRNN)s [19]. The BRNNs have a slight different architecture which allows it to take into consideration information not only from the present and the past input but also from the future input. Note that by past, present, and future, we not only refer to temporal sequences, but also sequences which have a strong emphasis of the order, but bear no explicit notion of time; this is actually the case with natural languages. In BRNNs each hidden layer is duplicated into two layers (Figure 2.4), one layer takes as inputs the present data point and information from the past data point. This is referred to as the "forward direction". The other layer takes as input the present data point and information from the future data point. This is referred to as the "backward direction". The BRNNs are fully described by equations 2.3, 2.4 and 2.5, where $\mathbf{h_t^{<f>}}$ represents the forward direction of the hidden layers and $\mathbf{h_t^{<b>}}$ represents the backward direction of the hidden layers. The predicted value $\widehat{\mathbf{y_t}}$ is now a function of both the forward and the backward direction. Considering information from both sides of the sequence instead of the left side only adds much power to the network as the context is understood much better. Consider the following example: "She said, 'Teddy bears are on sale.'", "She said, 'Teddy Roosevelt was an amazing president'" On these two examples, considering "Teddy" as the current data point, the left sequence is the same, however, the right sequence is crucial in understanding the context. Thus for an application like named-entity recognition, using BRNNs adds much gain. One drawback about BRNNs is that the entire sequence is needed before any predictions can be made, therefore for real-time systems it is a bit slow as it introduces some delay.

$$\mathbf{h_t^{<f>}} = \phi(W^{xf}\mathbf{x} + W^{hf}\ \mathbf{h_{t-1}^{<f>}} + \mathbf{b_{hf}}) \tag{2.3}$$

$$\mathbf{h_t^{<b>}} = \phi(W^{xb}\mathbf{x} + W^{hb}\ \mathbf{h_{t+1}^{<b>}} + \mathbf{b_{hb}}) \tag{2.4}$$

$$\widehat{\mathbf{y_t}} = softmax(W^{yf}\ \mathbf{h_t^{<f>}} + W^{yb}\ \mathbf{h_t^{<b>}} + \mathbf{b_y}) \tag{2.5}$$

Problems with RNNs

TODO: vanishing and exploding gradients [10] [11] [3]

Long Short-Term Memory

Long Short-Term Memory (LSTM) [12] is a variation of vanilla RNNs which was designed as a solution to the vanishing gradients problem. The main idea was to replace the ordinary node with a "memory cell". The cell uses "gates" in order to make decisions about which information to keep and which to discard. It has three gates: output, input, and forget gate, which are analogous to read,

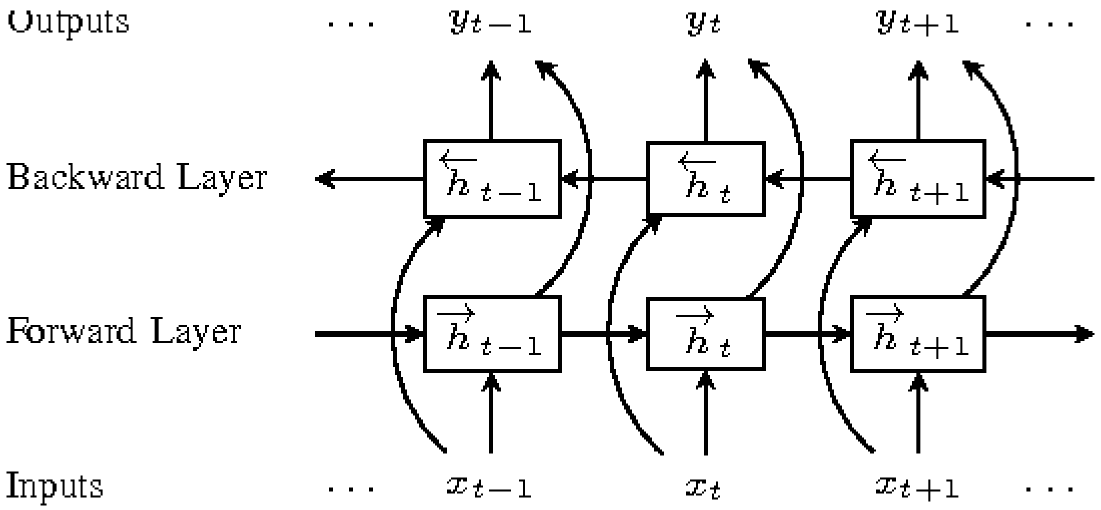

Figure 2.4: A Bidirectional Recurrent Neural Network

write, and reset operations for the memory cell. The gates uses sigmoid as the activation function, where the values of the gates ranges from 0 to 1.

TODO 1: continue LSTM

$$\tilde{\mathbf{c_t}} = tanh(W^{ch}\, \mathbf{h_{t-1}} + W^{cx}\, \mathbf{x_t} + \mathbf{b_c}) \tag{2.6}$$

$$\mathbf{f_t} = \sigma(W^{fh}\, \mathbf{h_{t-1}} + W^{fx}\, \mathbf{x_t} + \mathbf{b_f}) \tag{2.7}$$

$$\mathbf{i_t} = \sigma(W^{ih}\, \mathbf{h_{t-1}} + W^{ix}\, \mathbf{x_t} + \mathbf{b_i}) \tag{2.8}$$

$$\mathbf{o_t} = \sigma(W^{oh}\, \mathbf{h_{t-1}} + W^{ox}\, \mathbf{x_t} + \mathbf{b_o}) \tag{2.9}$$

$$\mathbf{c_t} = \mathbf{i_t}\ \tilde{\mathbf{c_t}}\ +\ \mathbf{f_t}\ \mathbf{c_{t-1}} \tag{2.10}$$

$$\mathbf{h_t} = tanh(\mathbf{c_t})\ \mathbf{o_t} \tag{2.11}$$

Problems with LSTM

TODO 2: Problems with LSTMS

2.2.3 Convolution Neural Networks

TODO 3: CNNs

Now that we have examined neural networks, we move forward to discussing the speech recognition problem in the next section.

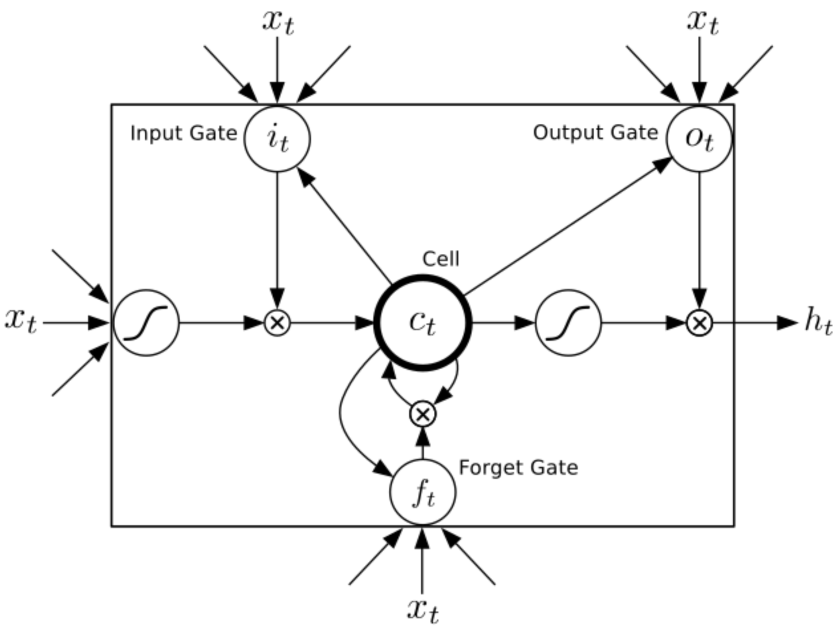

Figure 2.5: An LSTM Cell

2.3 Speech Recognition

The speech recognition problem is defined as follows: given an audio waveform, the task is to find the closest possible transcription to what an accurate human would generate upon listening to that audio. This problem dates back to 1960's where the former Soviet Union were among the pioneers to work on it, however, the basic Hidden Markov Model (HMM) speech recognition systems dates back to mid 1980's. In this section we investigate the mechanics of the ASR systems based on HMMs, moving to the so called "hybrid models" and eventually the growingly popular "End-to-End Systems".

2.3.1 HMM-Based ASR Systems

We begin by demonstrating the main components of an ASR system. As depicted in figure FIGURE, an audio waveform is passed to a "feature extraction" module, which outputs a sequence of acoustic vectors, $\mathbf{Y} = y_1, y_2, ..., y_T$. The audio fragment corresponds to a sequence of words $W = w_1, w_2, ..., w_n$, and it is the objective of the ASR system is to find the most probable word sequence W given a previously unknown audio signal $\mathbf{Y}$. More formally, the target is to find $W = arg\, max_W \; P(W|\mathbf{Y})$. Using Bayes' Rule, this probability can be broken down into two probabilities as shown in equation 2.12.

$$W = arg\, max_W \; P(W|\mathbf{Y}) = arg\, max_W \; \frac{P(W)\; P(\mathbf{Y}|W)}{P(\mathbf{Y})} \tag{2.12}$$

This indicates that we need to find $P(W)$ and $P(\mathbf{Y}|W)$ which maximizes 2.12, in order to find the most probable word sequence W. There are two main components in the ASR system, the "language model" is used to compute $P(W)$, which is the probability of observing the word sequence W independent of the audio signal. The second component, which is the "acoustic model" computes $P(\mathbf{Y}|W)$ which is the probability of the sequence of acoustic vectors Y, given a word sequence W.

Figure FIGURE illustrates the flow of the ASR mechanics. Firstly, $P(W)$ is computed by the language model, then the word sequence W is passed to a "pronouncing dictionary", which breaks the words into “phones”. Phones are distinct units of sounds, and they are specific to every language. For example, English has 44 phones, despite having 26 letters only. That means there 44 different sounds in English. The audio signal is passed to a feature extraction module which outputs numerical features representing the speech signal. For every phone, a statistical HMM model is built, and these models are concatenated together in order to represent the whole sequence of phones making up the utterance using a single model. Then the probability of this model generating the sequence of acoustic vectors Y, given the word sequence W is calculated *i.e.* $P(\mathbf{Y}|W)$. For every possible word sequence W, this process can be repeated until we get the most probable word sequence, however, this is obviously impractical and more efficient methods are used. The process of searching for the most probable word sequence is referred to as “decoding”.

We begin by shedding light on the feature extraction module.

Feature Extraction

The premier step in any ASR system is to extract features. That is to turn the raw speech waveforms into a sequence of numerical vectors. The issue, however, lies in the fact that audio signals are constantly changing. For us to be able to deal with them, we make the assumption that for sufficiently small period of time the signals are stationary. Therefore we sample the signal into 25ms frames, with a step of about 10ms so that the frames are overlapping. Then we apply some operations on each frame to extract features. The most widely used feature extraction method is Mel-Frequency Cepstral Coefficents (MFCC), which we explain in brief.

For each frame, the Fast Forier Transform (FFT) is calculated, in order to move from the time domain to the spectral domain. Human ears cannot distinguish between two closely spaced frequencies, specially for higher frequencies; to mimic this effect, we need to know how much energy exists in different frequency regions. This is done by applying special filters called Mel filterbanks. The first filter is the narrowest, and indicates how much energy exists around 0 Hertz. Then the filters become wider as the frequencies get higher because we become less concerned about variations. Afterward, we take the log of the filterbank energies. This is also done to mimic the human ear as we do not hear loudness on a linear scale. Then we compute the Discrete Cosine Transform (DCT) of the log filterbank energies. This is due to the fact that the filterbanks are overlapping, so we compute DCT to reduce the correlation between filter bank amplitudes. The resulting DCT coefficients are referred to as MFCC coefficients. Only 12 coefficients are kept and the rest are dropped, this is proven to improve the ASR performance. These 12 coefficients, together with the normalized energy, they form the feature vector.

Acoustic Model

The acoustic model provides a mechanism for computing the probability of a sequence of acoustic vectors Y given a word sequence w. To be able to do this, each word is broken into its constituent phones using the pronouncing dictionary, and we model each phone as a HMM. HMMs are statistical models that are used to predict a sequence of unknown random variables - hence the name “hidden” - from a set of observed variables. Here the unknown variable is the phones sequence, and the observed variables are the sequence of acoustic vectors. HMMs **FST!** (**FST!**) which are based on the “Markov Assumption” which states that, in order to predict the future (next state), all we need to know is the present (current state) only, neglecting the past (previous states). *i.e.* $P(x_i|x_1, x_2, ..., x_{i-1}) = P(x_i|x_{i-1})$. Each phone model has an entry state and an exit state which are used to connect different phone models together forming words, and words to be connected together forming sentences. The states are connected by arrows which represent the probabilities of moving from one state to another. Thus, a_{ij} represents the probability of moving from state i to state j. Each time t, the HMM changes state moving from state i to j with probability a_{ij}, and generating an acoustic vector $\mathbf{y_t}$ with probability $b_j(\mathbf{y_t})$. HMMs are also based on the "Output Independence Assumption", which states that the probability of an output observation o_i depends solely on the state x_i that generated that observation, not on any other state or any other output observation. From that, the HMM is fully described by:

1. $X = x_1, x_2, ...x_n$ A set of n **states**.
2. $A = a_{11}, .., a_{ij}, .., a_{nn}$ A **transition probability matrix A**, where a_{ij} is the probability of moving from state i to state j
3. $O = o_1, o_2, ..., o_T$ A sequence of **T observations**
4. $B = b_i(o_t)$ A sequence of **output probabilities**, where $b_i(o_t)$ is the probability of state i generating observation o_t
5. $\pi = \pi_1, \pi_2, ..., \pi_n$ An **initial probability distribution** over the states, where π_i is the probability that the HMM will start in state i. [15]

In [16], Rabiner shows that the HMMs are charecterized by three major problems:

1. **Computing Likelihood**: For a HMM model $M = (A, B)$ and output sequence O, compute the probability $P(O|M)$
2. **Decoding**: For an output sequence O and a HMM $M = (A, B)$, find the best hidden states sequence X
3. **Learning**: For an output sequence O and a hidden state sequence X, determine the parameters A and B [15]

Language Model

The sole objective of the language model is to compute the probability of a word w_k given the previous words $W_1^{k-1} = w_1, w_2, ..., w_{k-1}$. "N-grams", are one of the earliest and simplest, but still efficient methods used for language modeling. As implied by the name, n-grams make the assumption that the probability of a certain word w_k, depends only on the previous $n-1$ words. More formally, $P(w_k|W_1^{k-1}) = P(W_{k-n+1}^{k-1})$. N-grams are pretty simple to compute; using frequency counts from textual data and storing them in look-up tables simply gets us the probability distribution. They also capture dependencies and semantics and hence are highly suitable for languages like English due to the fact that word order matters a lot and nearest neighbors contribute largely to the contextual meaning of a word. For more compound languages like German, higher order grams would be a more convenient choice.

2.3.2 Hybrid Systems

TODO 6 : Hybrid Systems PROBLEMS WITH HYBRID MODELS THE NEED TO SEGMENT THE DATA

Although RNNs seemed to be the best suit for sequence models, their use in speech recognition has been limited to hybrid models which do not make use of the full capabilities of the RNNs

2.3.3 End-to-End Systems

In 2006, Graves *et al.* introduced a novel way of using RNNs for labeling unsegmented sequence data which they called Connectionist Temporal Classification (CTC) [8]. We shall demonstrate their paper in this section. The basic idea is to model the RNN outputs as a conditional probability distribution over all possible sequences of labels given a certain input sequence. With that, an objective function is defined that tries to maximize the probability of the correct label sequence. The neural network can then be trained using standard BPTT.

Let L be a finite alphabet of labels and L^* is the set of all sequences over the alphabet L. Likewise, let S be a set of training examples which comprises pairs of sequences $(\mathbf{x}, \mathbf{z})$, with $\mathbf{x} = (x_1, x_2, ..., x_T)$ being an input sequence of real-valued feature vectors, and $\mathbf{z} = (z_1, z_2, ..., z_U) \in L^*$ is a target sequence of labels which is at most the same length as the input sequence x. *i.e.* $U \leq T$. Our goal is to train a classifier h that uses the training examples set S to classify formerly unseen input sequences in a manner that minimizes our "label error rate". Label error rate is defined as the minimum number of insertions, deletions and substitutions required to change the predicted word into the ground-truth word.

The RNN output layer is a softmax layer, consisting of $|L|+1$ units. The first L units correspond to the probabilities of the L labels of our alphabet, the extra unit correspond to the probability of the output being no label, or "blank". This softmax layer corresponds to the probabilities of the entire possible permutations of labels, hence giving us the probabilities of all possible label sequences for a given input sequence.

For the purpose of investigating the matter in a more formal way, let us define alphabet $L^{'} = \{L \cup blank\}$, and let $L^{'}T$ be all the label sequences of $L^{'}$ of length T, we refer to elements of $L^{'}T$ as "paths" and denote them as π. Also let y_k^t be the probability of the output being label k at time step t. With this, the probability of a certain path, given the input sequence is given by equation 2.13

$$P(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t, \; \forall \pi \in L^{'T} \tag{2.13}$$

In equation 2.13, it is assumed that the network outputs at different time steps are independent. This is achieved by not allowing any feedback connection from the outputs to the network itself.

The next step is to remove all blanks and repeated labels from every path. This gives us a new set $L^{\leq T}$ which is the set of all possible label sequences having length less than or equal T defined over the alphabet L without the blank. We then can calculate the probability of a given label sequence l by simply summing the probabilities of all the paths producing that label sequence l. Note that after removing all blanks and repeated labels, many paths would generate the same label sequence. Hence, the probability of a certain label l is given by equation 2.14

$$P(l|x) = \sum P(\pi|x), \;\; \forall \; \pi \; generating \; l \tag{2.14}$$

The output of our classifier should be the label sequence with the highest probability $h(x) = arg\,max_{l \in L^{\leq T}} P(l|x)$.

There are two efficient mechanisms for finding the most probable label sequence:

1. **Best Path Decoding**
   Best Path Decoding is a greedy algorithm based on the assumption that the most probable path, corresponds to the most probable label sequence. The advantage of Best Path Decoding is that it is remarkably easy to compute; the most probable path is the concatenation of the most probable labels for every time step. The disadvantage is that does not ensure finding the most probable label sequence.

2. **Prefix Search Decoding**
   Prefix Search Decoding works by calculating the probabilities of successive extensions of prefixes of label sequences. Despite Prefix Search Decoding being slower, it is guaranteed to identify the most probable label sequence. The drawback here is that the number of prefixes that must be expanded grows exponentially with the length of the input sequence. To overcome this issue, the authors of the paper make use of an observation which is that the outputs of a CTC network form spikes of labels with strongly predicted blanks. Using this observation, a certain threshold is chosen, and points with blank probabilities higher than that threshold are chosen as boundary points, forming sections or regions. For every region, we calculate the most probable label sequence for each region separately, and then we concatenate the results to get the final label sequence.

2.4 Text Analytics

Text Analytics is the process of extracting information and semantics from written text. Some famous text-analytics tasks are text classification, sentiment analysis, document summarization, named-entity recognition and entity-relation extraction. Due to the sequential nature of text, RNNs were widely used for many text analytics tasks for quite a long time. This was the case because their sequential nature made them a good candidate for the task. A special architecture, named "Encoder-Decoder Architecture", which made use of RNNs when first introduced, has been marked broadly useful in modeling sequence-to-sequence models, where the input is a sequence, and the output is as well a sequence. An example is machine translation, where a model is trained to find an output sentence y which maximizes the conditional probability of y given an input sentence x. We discussed RNNs and their variations in section 2.2.2. In the following section, we take a look into the encoder-encoder architecture.

2.4.1 Encoder-Decoder Architecture

The popular encoder-decoder architecture which was first proposed by Cho *et al.* (2014a) [6] and Sutskever *et al.* (2014) [20]. This system consists of two **RNNs!** (**RNNs!**) which work together as an encoder-decoder pair. The encoder encodes a variable-length sequence (e.g. a sentence) into a fixed-length vector which we call summary vector **c**. The decoder then uses this fixed-length vector to generate a variable-length output sequence. The vector **c** has information from each data point in the input sequence.

The encoder is a RNN which has a hidden state updated at each time step according to equation 2.15, where f is a non-linear activation function; Sutskever *et al.* (2014) [20] uses LSTMs for this purpose while Cho *et al.* (2014a) [6] used a variation of LSTMs instead.

$$\mathbf{h_t} = f(x_t, \mathbf{h_{t-1}}) \tag{2.15}$$

The decoder is a RNN that is trained to generate the output sequence **y** one by one. It uses the hidden state $\mathbf{h_t}$ to calculate y_t. The hidden state $\mathbf{h_t}$ is calculated according to equation 2.16, where it uses the previously generated symbol y_{t-1}, the previous hidden state $\mathbf{h_{t-1}}$ and the summary vector **c** in predicting the target symbol. Similarly, the conditional probability of the target symbol is given by 2.17

$$\mathbf{h_t} = f(\mathbf{h_{t-1}}, y_{t-1}, \mathbf{c}) \tag{2.16}$$

$$P(y_t|y_1, y_2, .., y_{t-1}, \mathbf{c}) = g(\mathbf{h_t}, y_{t-1}, \mathbf{c}) \tag{2.17}$$

The two components are then trained to maximize the conditional probability of the output sequence $\mathbf{y} = (y_1, y_2, .., y_{T_y})$ given the input sequence $\mathbf{x} = (x_1, x_2, .., x_{T_x})$

The problem with this architecture is that the performance deteriorates as the length of the input sequences increases [5]. This is due to the difficulty of cramming all the necessary information into a fixed-length vector. In order to address this issue, the attention mechanism was introduced by Dzmitry *et al.* [2]

2.4.2 Attention Mechanism

Attention is the ability to focus on important details and discard unimportant or irrelevant information. Dzmitry *et al.* (2015) [2] proposes modifying the encoder-decoder architecture through arming the decoder with attention mechanism which allows it to attend to only parts in the input sentence which are most relevant to the target word in the output sequence. The characteristic feature of this approach is that it doesn't encode all the input sequence into a fixed-length vector as the basic encoder-decoder approach explained in section 2.4.1. Instead, it encodes the input sequence into a number of vectors, and chooses which of these vectors are relevant to the target word in order to make a prediction. This approach performs better on longer input sequences as it is no longer needed to suppress all the information given in the sentence in one fixed-length vector. We demonstrate the model proposed by Dzmitry *et al.* (2015) [2] starting by the encoder, then the decoder.

I. Encoder

No major modifications were performed on the encoder, however, a BRNN was used instead of a uni-directional one. This is done in order to gather left and right context as discussed in section 2.2.2. By concatenating the forward hidden state $\overrightarrow{h_j}$ and the backward hidden state $\overleftarrow{h_j}$ for each word x_j in the sequence, an annotation $h_j = [\overrightarrow{h_j}^T, \overleftarrow{h_j}^T]$ for that word is obtained. The decoder would use these annotations later in the attention layer as we will explain.

II. Decoder

The decoder is a RNN that also uses the hidden state $\mathbf{h_i}$ to calculate y_i. The hidden state $\mathbf{h_i}$ of the decoder is calculated according to equation 2.18, where it uses the previously generated symbol y_{i-1}, the previous hidden state $\mathbf{h_{i-1}}$ and a context vector $\mathbf{c_i}$ in predicting the target symbol. The conditional probability of the target symbol is given by 2.19. The major difference here from the basic encoder-decoder approach is that there is a distinct context vector $\mathbf{c_i}$ for each target word y_i. The context vector $\mathbf{c_i}$ is calculated as a weighted sum of the sequence of annotations produced by the encoder as seen in equation 2.20. The weights of the annotations are learned by the attention model as in equation 2.21 and 2.22

$$\mathbf{h_i} = f(\mathbf{h_{i-1}}, y_{i-1}, \mathbf{c_i}) \tag{2.18}$$

$$P(y_i|y_1, y_2, .., y_{i-1}, \mathbf{x}) = g(\mathbf{h_i}, y_{i-1}, \mathbf{c_i}) \tag{2.19}$$

$$\mathbf{c_i} = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.20}$$

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{2.21}$$

where

$$e_{ij} = a(h_{i-1}, s_i) \tag{2.22}$$

III. Additive Attention

The attention model helps solving long range dependencies efficiently because it can focus on what matter and disregard what doesn't matter.

In 2017, Google came up with a novel architecture using only the attention mechanism and eliminated the use of RNNs, this was called "The Transformer" and we explain the idea behind it in the next section.

2.4.3 The Transformer

I. Problem with Recurrence

As seen in equation 2.11, **RNNs!** calculate the hidden state as a function of the current data point and the previous hidden state. This sequential nature of calculation largely suits the sequential nature of natural languages, however, this introduces a major problem as it hinders parallelization among the training inputs. This issue is manifested when the input sequences are of longer lengths. The Transformer abandons recurrence in order to achieve more parallelization and efficiency in performance.

II. Encoder

The Transformer follows the prominent encoder-decoder architecture, however, introducing major modifications to both the encoder and the decoder. The encoder consists of N layers, each layer made up of two sub-layers. The first sub-layer is called "Self-Attention" or "Intra-Attention" sub-layer. This is one of the most influential changes proposed in the Transformer. What self-attention does is that it calculates the relevance of each word in the input sequence to every other word in the sequence. To demonstrate the gain we achieve with self-attention, consider the following examples which highlight the problem of linking the pronouns to their antecedents:

1. "The animal did not cross the street because *it* was too tired."

   When translated to French this becomes: "L'animal n'a pas traversé la rue car *il* était trés fatigué".

2. "The animal did not cross the street because *it* was too wide."

   In French, this is "L'animal n'a pas traversé la rue car *elle* était trés large."

A model implementing no self-attention mechanism has difficulty determining the right pronoun in French; whether "il" for males or "elle" for females, due to the fact that it did not learn any link between "it" and its antecedent when encoding the input sentence.

The second sub-layer is a simple fully-connected feed-forward neural network. A residual connection [9] is applied to each sub-layer, then layer normalization [1] is performed.

III. Decoder

The decoder is made up from N layers as well, with each layer composed of three sub-layers: a self-attention layer similar to the self-attention layer implemented in the encoder, however, modified so that each word can attend only to words earlier in the sequence and not to consequent words, a feed-forward network, and an additional layer which implements the encoder-decoder attention as explained in section 2.4.2. As in the encoder, residual connections are applied on each sub-layer followed by layer normalization.

IV. Scaled Dot-Product Attention

V. Positional Encodings

2.4.4 Bidirectional Encoder from Transformer (BERT)

Word Embeddings

Transfer Learning

Pre-Training

Feature Extraction

Fine Tuning

Bidirectional vs uni-directional

PreTraining Tasks

Chapter 3

Methodology

In this chapter, the methodology used to implement the system is illustrated. Firstly, an overview of the system is discussed, along with the datasets used. Secondly, an interpretation of the choice of architecture and decisions is elaborated. Eventually, the system implementation steps are demonstrated in detail.

3.1 System Overview and Datasets

As discussed earlier, the main purpose of this study is to implement a system able to issue dispatcher actions automatically when provided with audio signals from the vehicle driver, hence, introducing more automaton in the control-center.

Our system consists of two sub-systems, as shown in figure 3.1, the first sub-system is the ASR unit, which takes as input raw speech waveforms produced by the driver, and produces the information in a text form. Then the text is passed to a trained Text Classifier, which given the text information from the ASR, issues the corresponding appropriate dispatcher action.

TODO IN THE CHOICE OF ARCHITECTURE NLP PROBLEMS TO MAKE AN END TO END SYSTEM

3.1.1 ASR Datasets

End-to-End ASR systems require large amounts of transcribed audio data. For this purpose we make use of three open-source German datasets and clean them. We list the three datasets here with the cleaning operations performed for each.

I. Common Voice

Common Voice is the largest open source, multi-language dataset of voices available for use. It is managed by Mozilla and was collected by volunteers on-line who were either recording samples or validating other samples. Mozilla began work on this project in 2017 and contribution to the dataset continues up till now. For our ASR, the German subset of the dataset was selected. It incorporates 340 total hours, with 325 validated hours and 15 invalidated hours which we excluded. The dataset has 5007 speakers but as we discarded the invalidated utterances we end up with only 4823 speakers. All the utterances were in “mp3” format and sampled using a sampling rate of $44kHz$ so we converted them to “wav” format and we performed down-sampling to obtain sample rate of $16kHz$. We checked for any corrupted files but there were none.

II. M-AILABS Speech Dataset

M-AILABS Speech Dataset is an open-source multi-lingual dataset provided by Munich Artificial Intelligence Laboratories GmbH. Most of the data is based on LibriVox [1] and Project Gutenberg [2]. We make use of the German subset which is 237 hours 22 minutes with a total of 5 speakers. The data is available in "wav" format and sample rate of $16kHz$ so we perform no modifications. We also check for corrupted files but all of them were healthy.

III. German Speech Data [17]

This open-source corpus is provided by Technische Universität Darmstadt. It has 36 hours read by 180 speakers, and recorded using 5 different microphones simultaneously. They made use of the KisRecord [3] toolkit, which allows for recording with multiple microphones concurrently. Their target was distant speech recognition, thus a distance of one meter between speakers and microphones was chosen. The sentences which volunteers were provided to read were extracted randomly from three text resources: German Wikipedia, German section of the European Parliament transcriptions, short commands for command-and-control settings. Unfortunately, there were some corrupted files in the dataset, which we discarded.

TODO SPEAKER INDEPENDENT CLASSIFICATION

3.1.2 Text Classifier Datasets

I. German Wikipedia Dump

II. 10K German Articles

3.2 Automatic Speech Recognition Unit

3.3 Text Classifier Unit

[1] https://librivox.org

[2] https://www.gutenberg.org

[3] http://kisrecord.sourceforge.net

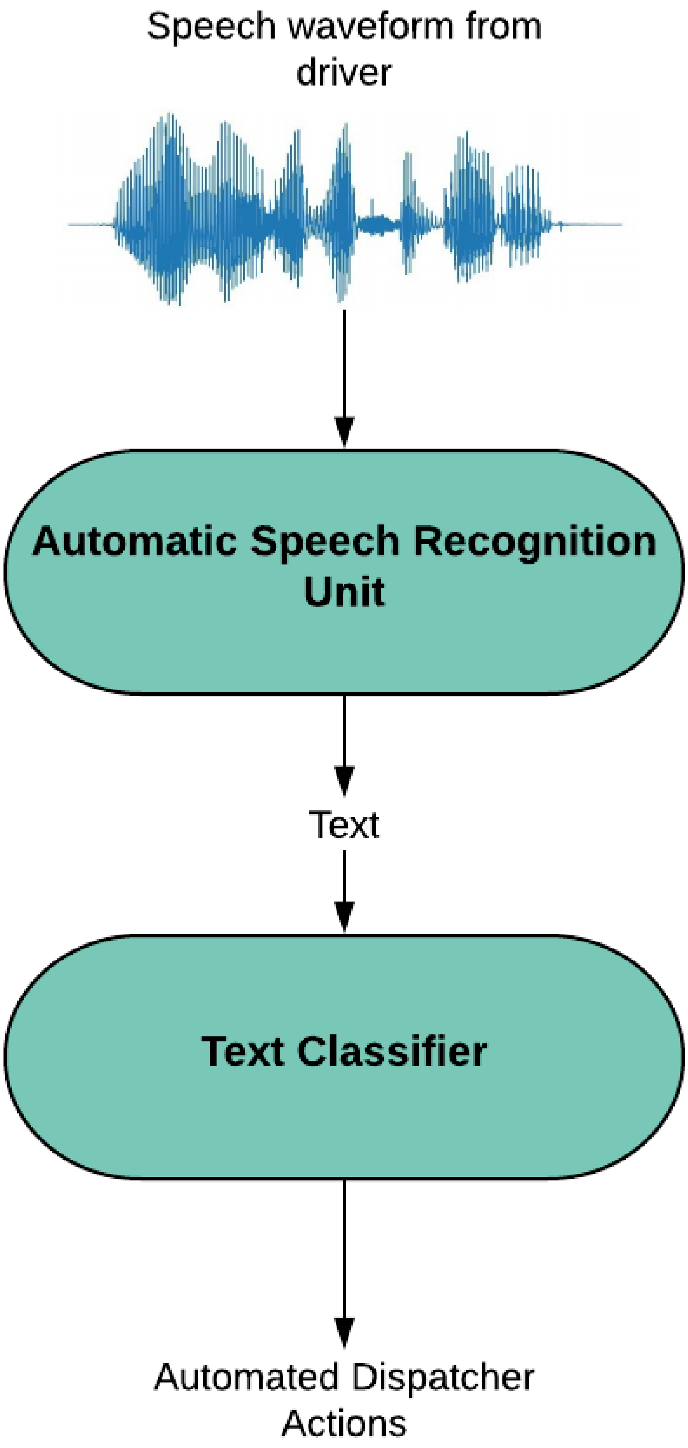
Speech waveform from driver
Automatic Speech Recognition Unit
Text
Text Classifier
Automated Dispatcher Actions

Figure 3.1: Automated Dispatcher Actions System comprising two sub-systems: Automatic Speech Recognition unit taking input as raw speech from the driver. The text output is fed into a trained Text Classifier which issues the proper action accordingly.

Chapter 4

Results

Results

Chapter 5

Conclusion

Conclusion

Chapter 6

Future Work

Text

Appendix

Appendix A

Lists

NLP	Natural Language Processing
NLG	Natural Language Generation
NLU	Natural Language Understanding
ANN	Artificial Neural Network
FNN	Feed Forward Neural Network
MLP	Multi Layer Perceptrons
RNN	Recurrent Neural Network
BPTT	Backpropagation Through Time
BRNN	Bidirectional Recurrent Neural Network
LSTM	Long Short-Term Memory
ASR	Automatic Speech Recognition
MFCC	Mel-Frequency Cepstral Coefficents
FFT	Fast Forier Transform
DCT	Discrete Cosine Transform
CTC	Connectionist Temporal Classification
HMM	Hidden Markov Model

List of Figures

Bibliography

[1] J Ba, J Kiros, and G Hinton. Layer normalization. arxiv. 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Christopher M Bishop et al. *Neural networks for pattern recognition.* Oxford university press, 1995.

[5] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[8] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[11] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[14] MI Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. Technical report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.

[15] Vlado Keselj. Speech and language processing daniel jurafsky and james h. martin, 2009.

[16] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[17] Stephan Radeck-Arneth, Benjamin Milde, Arvid Lange, Evandro Gouvêa, Stefan Radomski, Max Mühlhäuser, and Chris Biemann. Open source german distant speech recognition: Corpus and acoustic model. In *International Conference on Text, Speech, and Dialogue*, pages 480–488. Springer, 2015.

[18] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[19] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[20] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[21] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[22] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.