ASSIGNMENT 4 - GROUP 14

Done by Henrik Haukaas & Thomas Østli.

1 CONTENT

1.1 WHO HAS DONE WHAT?

1.1.1 Henrik

Made the introduction, wrote about system 1 and our proposed system. Modelled most of the report. Did all the coding/implementation for assignment 4 as well as the testing and experiments. Wrote the conclusion. Also, did 5.1, 5.3 and 5.4

1.1.2 Thomas

Wrote about System 2 and did 5.2. Also made the image showing the system process in 6.

1.1.3 Markus

Markus did not contribute to this report or coding.

2 AI SYSTEMS FOR SMART HOMES

For this assignment, we are going to discuss and research about existing smart home concepts and get a larger understanding of how these can be improved by using Artificial Intelligence. Then we are going to look at two systems already in use which can improve smart homes and lastly try to combine these into a proposed system.

2.1 SMART HOMES

Smart homes are a building concept that is getting more developed and widespread. Most smart home solutions today are based around having different components integrated together to create a "network of things". Most components use sensors to add functionality and automation to the smart home. These systems might seem smart, but all in all it only uses a set of rules to dictate what happens. One of the biggest problems with today's smart home components is different manufacturers use different types of communication protocols and therefore it is almost a necessity to have components from the same manufacturer or else the complexity and cost will increase with each additional system dependency.

Google Home is one of the most advanced independent Smart home hubs available. It is supported with Google's newest speech recognition system and is fully integrated with other Google services. Anything you can google; Google Home can do. For this assignment, we will focus on some of Google's systems as these are one of the most advanced currently available. Google has the capacity and a framework to be able to test algorithms efficiently, adjust and can select the best ones. What makes Google's systems great is the amount of data they have access to. To be able to make use of this data, most of their systems and designs uses neural networks. Every time a user uses one of their systems, it will use that data to become better.

3 CURRENT SYSTEMS IN USE

3.1 SYSTEM 1 - GOOGLE SPEECH RECOGNITION

3.1.1 What is speech recognition?

Speech recognition is the ability for a machine to recognize speech, and in most cases, make it into text which can be more easily used by a machine. Speech recognition is a field which has seen a lot of improvements in the last decades. Speech recognition started as a system where speech signals had to be split into little units and then arranged into words. This seemed good in theory but it in practice this was inefficient and added unnecessary complexity. The next step in speech recognition came with the Hidden Markov method. Adopted in the 80s, used probability and brute force search to find the correct words. For the next 20-30 years, many other algorithms had been tried but none did perform as well as HMM. It was not before recent years and with the adoption of artificial neural networks (deep learning (LSTM), recurrent neural networks, among others) this field started to see improvements and now improvements are being made every week.

3.1.2 Classification problem

Speech recognition is a typical classification problem that predicts word sequences from speech waveforms. It is a many to many problem since many words can start with the same letter or pronunciation. Recurrent artificial networks is therefore used for the most advanced speech recognition systems today. Unlike Feed-Forward neural networks, RNN creates an internal memory which, with training, can be able to process arbitrary sequences of inputs. The internal memory in RNNs allows information to persist. This information influences predictions of words as each prediction should also predict the likelihood of what the next letter should be. For instance, if the speech recognizer has recognized “Hello Wo” so far, it is very likely that the last word should be “World” and not some random collection of letters. A RNN will make the system have more accurate predictions going forward.

“Humans don’t start their thinking from scratch every second. As you read this text, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.”

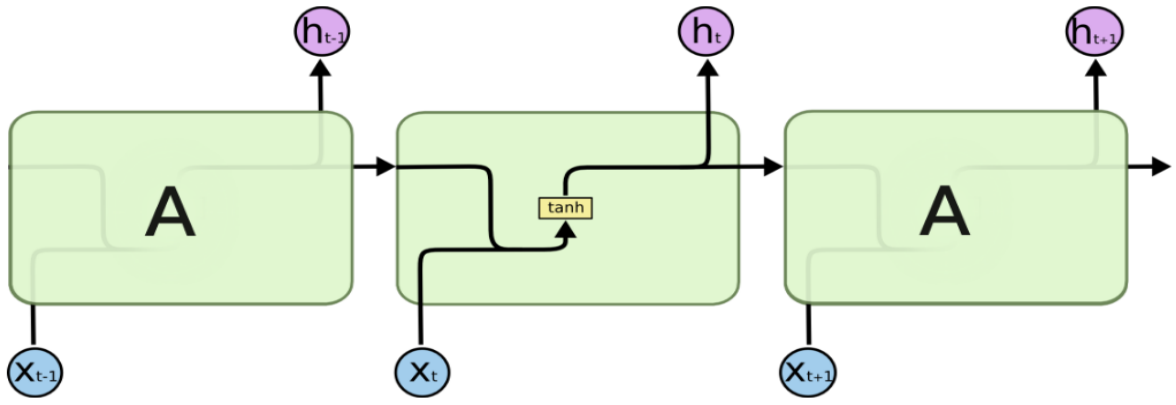


Figure 1 - The repeating module in a standard RNN contains a single layer. RNNs have a simple structure, such as a single activation layer (sigmoid (from 0 to 1) or tanh function (from -1 to 1).

3.1.3 Long Short-Term Memory Recurrent Neural Network

The biggest concern with RNN is long term dependencies. For instance, when the system is going to predict the following sentence; “I grew up in Norway”, it will struggle to predict the last word as it can have many different outcomes depending on who is asking. LSTM RNN addresses this issue, and

this is the algorithm which Google's speech recognizer uses. There is more algorithms in use here, for instance background noise reduction algorithms and natural language processing, but for this system LSTM RNN is the focus.

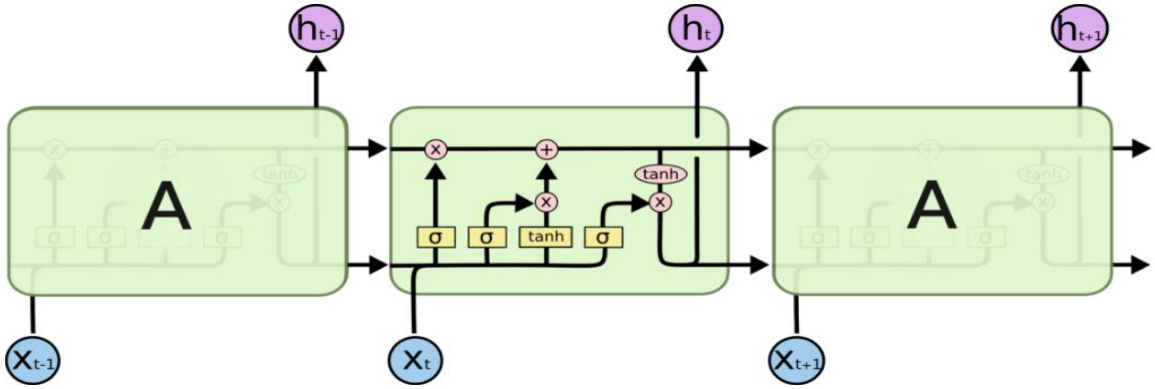


Figure 2 -LSTM RNN

LSTM RNN have four neural network layers and all these interact in a special way. The key to the LSTM is the cell state which is the horizontal line in the figure above. This will be the output, but is determined by activation layers. Below this line is "gates" to the NN layers. These layers are determined by activation functions, outputting a value from (0 to 1 for sigmoid or from -1 to 1 for tanh), and will decide how much information each component should let through.

The first layer in the LSTM work as the "forget" layer and decides what information should be thrown away from the input. The layer is determined by a sigmoid function (from 0 to 1), where a 0 would mean to toss all the information and 1 would be to keep it all. The next two layers decide what information should be kept in the cell state. The sigmoid layer will decide which values should be updated and the tanh layer will create new candidate values which could be added to the state. Next, the algorithm updates the cell state with what should be forgotten and what should be added. The last layer decides what should be output. The output is based on the current cell state, but will be a more filtered output.

3.1.4 Google speech recognizer

Google's speech recognizer improves over time as it is powered by machine learning. It supports 80 languages, returns text results in real time and provides advanced signal processing and noise cancellation. Through new algorithms like the LSTM ANN, Google's speech recognizer has seen large improvement the last years. One of the concerns with speech recognition is that the success rate needs to be very high for it to be widely used. Try to use Google Now and test for yourself!

3.2 System 2 - SyntaxNet/Google's Parsey McParseface

SyntaxNet is an open-source neural network framework that acts as a foundation for Natural Language Understanding(NLU) systems, developed by Google. SyntaxNet is implemented in TensorFlow, which is an open source library developed by Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research.

SyntaxNet is a syntactic parser, a sentence is given as input and the parser then finds the syntactic relationship between the words in the sentence. It does this using a part-of-speech (POS) tag that describes every word's syntactic function in the sentence. The parser uses these relationships to define the meaning of the sentence.

This is done partly by finding a syntactic head based on these relationships, where the head is the word with dependencies only. The words are given tags based on their relationship with the head.

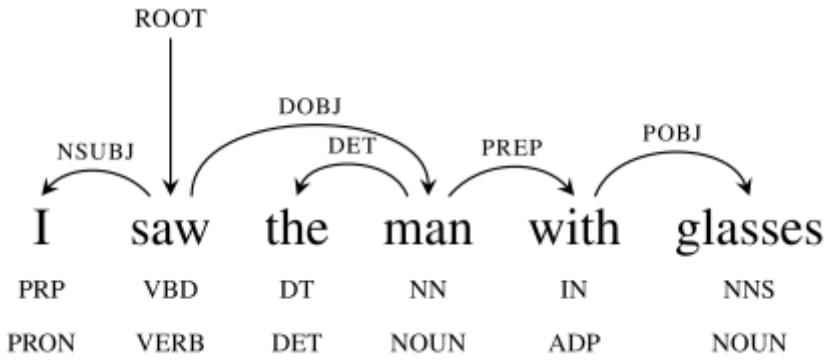


Figure 3 – POS

Below each word is a fine tag (1st row) and a course tag (2nd row), where course tags are basic grammatical categories such as verbs, nouns, pronouns etc. While fine tags make further specifications for example NN is a singular noun whereas NNS is a plural noun.
To assign tags correctly the parser must get an understanding of the entire sentence and context. Often just considering a small portion of the sentence can be enough, for example after "the" there is usually a noun or an adjective, and not verbs.

Sentences are taken in as input and then processed left-to-right. For every word features are extracted and then used in a feed-forward neural network classifier, it then predicts a probability distribution over POS tags. As it makes decisions in left-to-right order, it uses the decisions it has made before, as features in future decisions, learning that if a word was a noun before it is likely to be a noun again.

In practice, it works by having the input in a buffer, and step by step taking one word onto a stack at a time, as a step is taken the parser has 3 choices:

1. Shift: Retrieving another word from the buffer, placing it at the top of the stack.
2. Left-arc: Popping the top two words from the stack, attaching the second word to the first, and then creating and arc to the left and then pushing the first word back on the stack.
3. Right-arc: Same as Left-arc except it creates and arc to the right and pushes the second word back to the stack.

Best described by this animation:
https://github.com/tensorflow/models/blob/master/syntaxnet/looping-parser.gif

For left and right actions a relationship label is assigned. At each step many solutions are possible due to ambiguity. Ambiguity is what makes syntactic parsing so hard for computers, for humans the meaning of a sentence comes quickly but a sentence with 20-30 words can have hundreds or thousands of different syntactic structures.

Due to the amount of possible choices beam search is used to aid in better decision making. The neural network scores different decisions, scores are given based on how plausible they are. Beam search is used as a tool to help the parser make a better decision, it keeps multiple partial hypotheses in memory and only discards them when the neural network produces an option with a higher plausibility score.

While SyntaxNet provides the foundation for creating a syntactic parser, Parsey McParseface is already prepared by Google. Parsey McParseface gives state-of-the-art results, using a model that is runnable on normal desktop computers. It processes about 600 words per second.

Model	News	Web	Questions
Martins et al. (2013)	93.10	88.23	94.21
Zhang and McDonald (2014)	93.32	88.65	93.37
Weiss et al. (2015)	93.91	89.29	94.17
Andor et al. (2016)*	94.44	90.17	95.40
Parsey McParseface	94.15	89.08	94.77

Figure 4 - Table showing the accuracy of its parsing along with a few other examples

Andor et al. (2016)* is a SyntaxNet model with a larger beam and neural network.

4 OUR PROPOSED SYSTEM

For our proposed system, we would like to combine speech recognition, a natural language process-parser and an intelligent writer to make a smart home system which should be able to learn and complete most generalized smart home tasks without being restricted to a set functionality or rules, and thereby becoming a general smart home AI. The point of this system is to make it not only able to recognize but also understand. This is something that is currently not available and could greatly increase the demand and interest surrounding Smart Homes, among many other fields. This system is an extension of system 1 and 2 combined, as well as the inclusion of an intelligent writer.

4.1 OVERALL ARCHITECTURE

The first AI component of the system is a LSTM RNN speech recognizer which takes sound input from users and interprets this into text. The next component, a FFNN parser will then use this text to determine what the user wants. Depending on what the user requested, the system will execute this task. If necessary, the system should then output the result. The output should, if necessary, be done with a trained intelligent writer which would allow for more unique and personal answers from the system.

Tasks which could be done with this system:

- House related tasks. For instance, changing temperature in a room or turning on lights.
- Tasks which could be done on the internet. Retrieve information through Web queries to answer a specific question.
- Establishing connections to and use different APIs. E.g play music or watch movies.

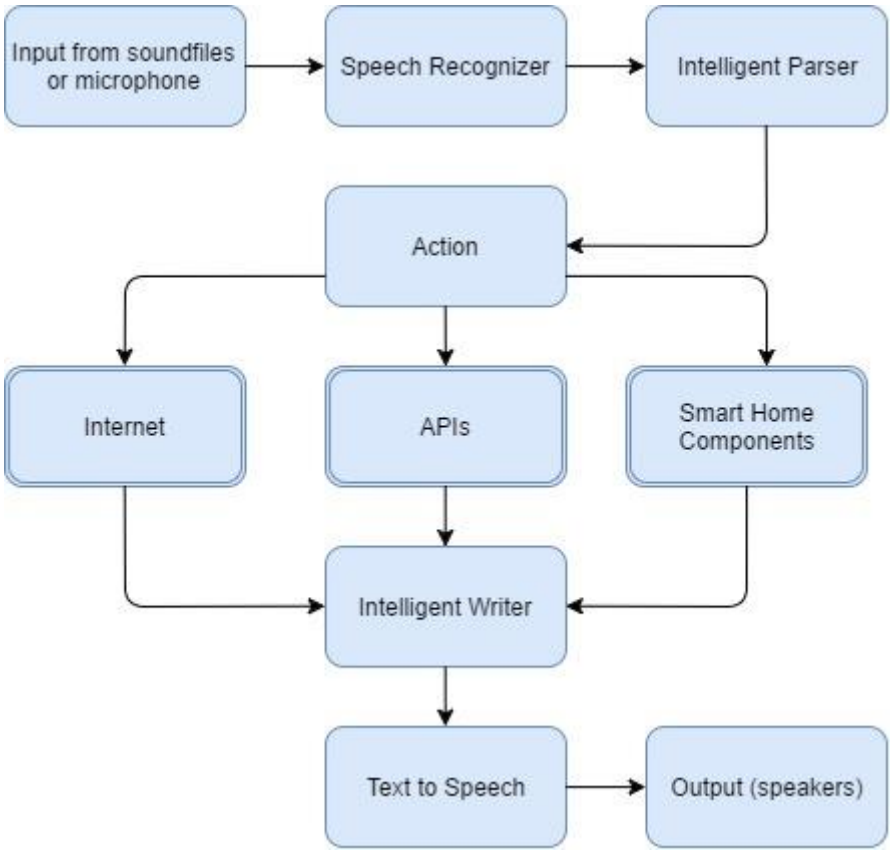


Figure 5 - System architecture

4.2 COMPONENT ARCHITECTURE

4.2.1 Speech recognizer and parser

The speech recognizer and parser of this system will mostly work in the same was as Google's current systems does. Except the parser needs to be able to comprehend what the user wants and based on that determine what action should be made.

4.2.2 Actions

The parser differentiates each request and determine if the request is smart home, internet or API related.

4.2.3 Intelligent writer and text to speech

The intelligent writer is a component which, based on what the user requested and what actions has been done by the system, should output a fitting result. It could be a more sophisticated chat-bot, based on neural network algorithms, trained on a large dataset of written text, able to comprehend and output at fitting result. The text to speech (TTS) is based on a NLP and a Digital Signal Processing component (DSP). The NLP takes the text generated from the intelligent writer and analyses it. This process will mainly work in the same way as the speech recognizer and parser system. The text is analyzed to establish a prosody, a way of speaking the text. The DSP system will then search through a database to find fitting speech units that fits the text best and puts them together and then output the result.

4.2.4 Restrictions and difficulties

One of the largest difficulties of this system is the possibility for accessing external APIs on demand and establishing connection with existing and new smart home components, e.g sensors and larger interconnected modules. Depending on what software the system is trying to establish connection to, the APIs could be very different and therefore hard to access. A solution for this is to have a more specific API collection (like Google Home) where only some APIs are supported (mostly Google's own cloud based API). Another solution is for the system to only access Web APIs and software based on the same architecture, e.g. programming language and data structure. For instance, if all the supported software is based on JavaScript and has the same data structure, it should be feasible for the system to generalize and access these systems, if the system is authorized. Another option would be to use a open source web based solution, which does not require authorization.
For the system to be able to fully control the smart home, the components should also have a generalized architecture, e.g. communicating on the same protocol (Z-Wave components). If not the system will not be able to access the components as it must also have a new set of receivers and transmitters.

For a system like this to be fully implemented, a lot of precautions needs to be made. Security breaches and misinterpreted commands can cause a lot of problems and unwanted results. The system also needs very large datasets and external dependencies to function properly.

5 ALGORITHMIC APPROACHES FOR THE PROPOSED SYSTEM

5.1 SMART HOME PARSER

Since our proposed system is quite large and not fitting of a small project, one of our tasks was to decide which part of the system we would like to further design and implement. After some discussion, we decided that we wanted to make a Smart Home parser which takes text as input and then based on that text, resolves what it should do. This is the Intelligent parser → Action part of figure 5 in 4.1. The parser should do this by retrieving the intent and object of any sentence. As a simplification and to make the workload more manageable we will take use of Google's Syntaxnet and the system will only output what it should do but not execute it.

5.2 A MORE TECHNICAL DESCRIPTION OF PARSEY MCPARSEFACE

Parsey's role in the system is to interpret the sentences input into the system. Parsey is, as stated earlier, an incremental transition-based parser. Descriptions used below are from the paper https://arxiv.org/abs/1603.06042 (2.1)

The FNN in Parsey pushes words on the stack and then decides if it should add another, pop either left or right and assigning POS tags. The way the transition system works is by defining:

- A set of states' S(x).
- A special start state "s" ∈ S(x).
- A set of allowed decisions A (s, x) for all s ∈ S(x).
- A transition function t (s, d, x) returning a new state "s" for any decision d ∈ A (s, x).

The function ρ (s, d, x; θ) is used to compute the score of "d" in state "s" for input "x". The model's parameters are contained in θ. In the paper describing this they simplified it to:

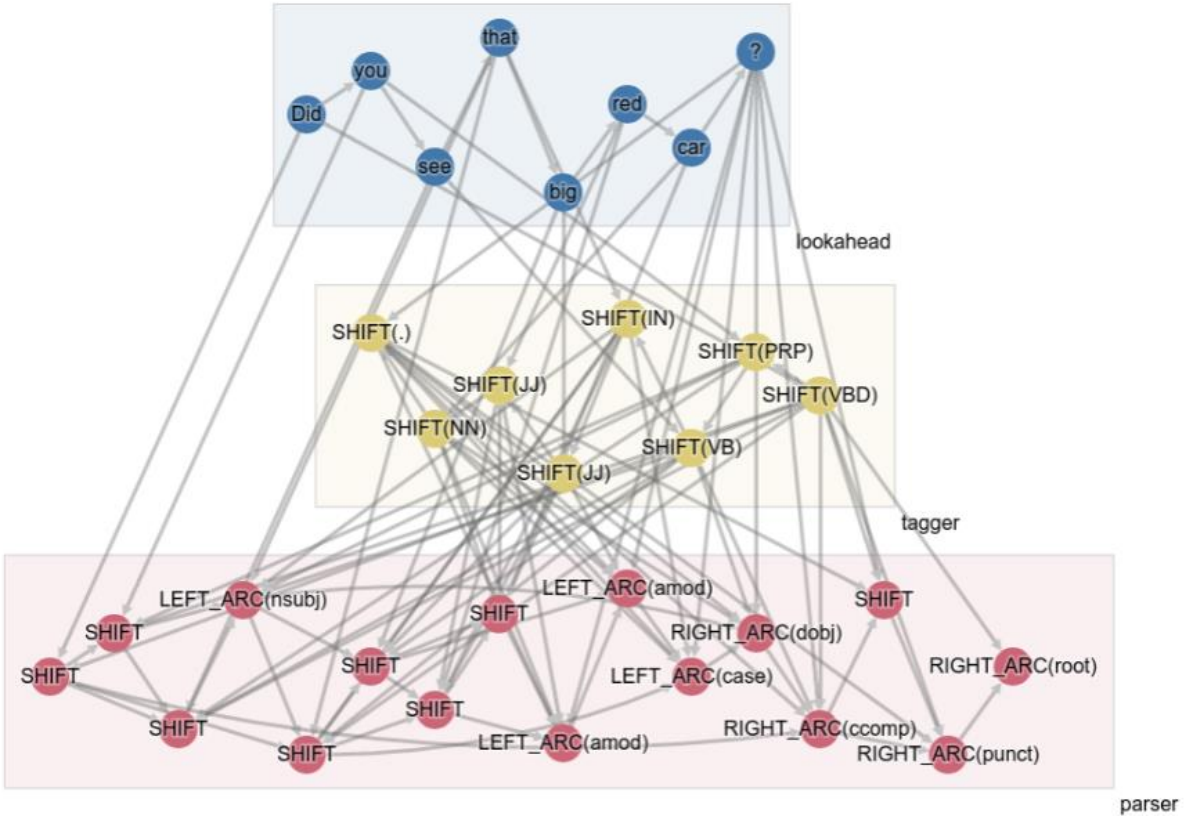
S, A(s), t (s, d), and ρ (s, d; θ)

The paper states that the scoring for the parser via function ρ (s, d; θ) can be defined in several ways. They defined it via a feedforward neural network as:

ρ (s, d; θ) = ϕ (s, θ (l)) · θ (d).

Here θ(l) are the parameters of the neural network. θ(d) are the final layer parameters for decision d.

ϕ (s; θ (l)) represents "s" state. Computed by the neural network under parameters θ (l). It processes the input "x" under these conditions and creates the parse. The output created by parsey is a string object.



The picture above further illustrates how the ANN parses and tags text, with the sentence to be parsed on top, the increments and decisions are displayed below. The graph above was created by Dynamic Recurrent Acyclic Graphical Neural Networks, shortened to DRAGGN. It is built as a python library with the intent to extend and simplify SyntaxNet.

5.2.1 Beam Search

Beam search is used as an alternative to having recurrence. Using beam search helps the parser to maintain multiple hypotheses despite not using more advanced neural networks like LSTM who can be more processor heavy. Using beam search enables the use of global normalization when coupled with conditional random field (CRF) as opposed to using maximum-entropy Markov model (MEMM), the developers chose to do this to avoid the label bias issues that occur with local normalization when using MEMM.

The mission of beam search is to find the approximate argmax (function value is at its maximum) and counter the interference problem produced by having large amounts of data.

5.3 Hyperparameters, Gradient Decent and Back Propagation

Hyperparameters are turning knobs of any NN and heavily affects accuracy and speed of the network. These could be learning rate, batch size, number of iterations (epochs), what kind of activation function and number of neurons and layers. Which hyperparameters are the best for the given NN might differ from another NN and therefore, the process of finding the best ones can be tedious and time consuming. It is possible to automate this process however. For instance, with Caffe Spearmint which uses Bayesian optimization. Neural networks are more sensitive to hyperparameters than many other machine learning algorithms. Therefore, fine tuning of hyperparameters is very important to achieve the best possible results. Parsey McParseface is an already trained system from Google. Hence, by using this in our system the importance of hyperparameters and optimization of these are not as important, as good hyperparameters are already in place. Though, we mean it is important to acknowledge understanding of hyperparameters and optimizing tools such as gradient decent and Back Propagation.

5.3.1 Gradient decent

Gradient decent is an optimization technique used to optimize or improve the accuracy of predictions.

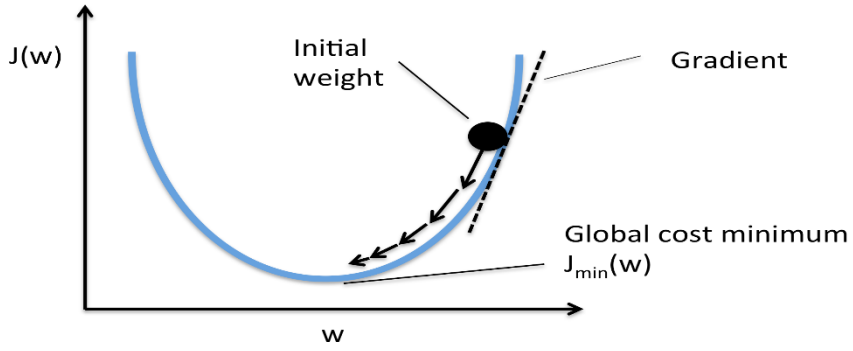


Figure 6 – Horisotnal axis → Weight value
Vertical axis → Error value

Neurons in a neural network relates to weights and the goal of gradient decent in a NN is to find the best weights with the least amount of error and therefore finding the best output for any given input.

5.3.2 Back propagation

Back propagation is simply put a recursive backtracking (hence back propagation) through the NN from the output to the input, visiting all weights and neurons in correlation to each other and then updating their weights with an optimizing algorithm, such as gradient decent. It does this by using an error value generated from the output and its difference from the desired output. Then the error value of each neuron in the hidden layers, going backwards, layer by layer.

5.4 System output

For the parser to be able to give a valid output and be as accurate as possible, it is required that this system is trained on more Smart Home related terms instead of just a generalization of text. This makes the system better at understanding commands which the user wants the Smart Home to execute. If Smart Home words and terms are not included in the training data, the system could miss out on important information as it will not be able to comprehend the full meaning of a sentence.

The system also needs to be able to inspect sentences for additional or missing information, not only the intent and object. Some commands can be interpreted wrong if this information is let out from the evaluation.

Here are some examples of inputs and outputs:

Input	Remark	Output
Turn on lights in the living room	Here the intent is “turn on” and the object is lights. Here a location is given and therefore the system needs to figure out that it only should turn on the lights in this area.	Turn on lights in the living room
Turn on lights	Here the intent is turn on and the object is lights. Here the system needs to figure out that no location is giving and that it should turn on all lights in the area.	Turn on lights in the current area
Lights	The system should evaluate where the input comes from and if the lights are off it should be turned on and vice versa.	Turn on or off lights
Give me the weather forecast for tomorrow	Here the intent is weather and the object, or the time in this case, is tomorrow. Here no location is given and therefore the system should set the location as the current geography of where the system is installed.	Finds the weather forecast for tomorrow through supported weather API

As we can see from the examples, simple commands might require a lot of additional logic. The complexity of the system will increase proportionally with how many external components the system should support and by how thorough the API support should be. The inclusion of an intelligent parser like the Smart Home parser however will reduce the input complexity by a large margin as many sentences can be treated in the same way, since the intent of a sentence is what matters in the end.

Pseudocode for when a weather forecast is requested:

```
Intent: weather
Location: Gjøvik
Time: Tomorrow

Finds actions for given intent (weather in this case)
        getForecast(Location, Time) {
                foreCast = Makes API call with location and time
                Return foreCast
        }
Output: “Sample text based on intent” (**) + foreCast
```

(**) Sample text could be generated through the intelligent writer in the full system

6 IMPLEMENTATION

To be able to implement the system, we first had to compile and install the most important dependency, Syntaxnet. For Syntaxnet to able to compile correctly, many other libraries had to be installed. We also had to use Google's own compiling tool, Bazel. It proved difficult to install Syntaxnet with Bazel, as it is based around Ubuntu's GCC compiler which makes Windows an unstable and highly experimental platform to run on. We therefore had to use a virtual machine with Ubuntu to take the installation process further. Even on Ubuntu with all dependencies installed and configured, the compiling was a tedious process. As the compiling had to be done on a virtual machine, available ram and CPU was reduced and the support for new library versions was lacking. This lead to a slow compiling process and lots of crashes. One compilation could last up to two hours and still end with a failed attempt. Lots of different configuration had to be tried before it finally worked, after trying for a few days.

When Syntaxnet was compiled and installed, further implementation went a lot smoother. Firstly, the output from the parser had to be analyzed and we had to figure out how to use it. Syntaxnet comes with an already trained parser, Parsey McParseface, so we also made a sub-process call to access this. We then added methods to be able to filter out the most important information from a parsed sentence and thereby retrieve the desired results.

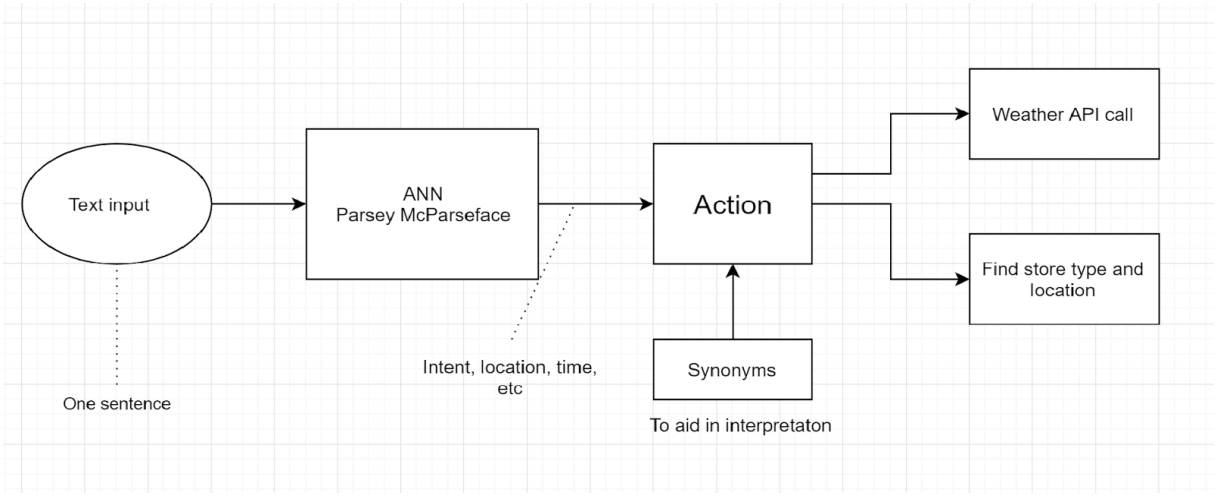
The output from the parser follows the Part of Speech (POS) tagging we have discussed earlier. The most important for this implementation was:

JJ	Adjective
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
VB	Verb, base form
xcomp	open clausal complement
pobj	object of a preposition
dobj	direct object/dependent object

The output from the parser was passed as an ascii tree, like this:

```
Input: find me a restaurant in Gjøvik
Parse:
find VB ROOT
 +-- restaurant NN xcomp
     +-- me PRP nsubj
     +-- a DT det
     +-- in IN prep
         +-- Gjøvik NN pobj
```

The tagging that was given in the output, made it easy to find words which gave the desired information. For instance, words containing the intent, location and time. To be able to filter the correct intents of a sentence, we used Natural Language Toolkit (NLTK) corpus (collection of text) and their python Wordnet library to able the find an intent synonym of the root word. For instance, sentences with a word like “find” would instead give the synonym “detect”, which is more generalizable and therefore is better to use as keyword for further use. Then, based on the intent of the sentence, the system would either call a weather API or a Google Place API. If the intent was weather information, the system would make a API call to OpenWeatherMap, which we used a python wrapper called Pyowm to be able to use in python. If the intent was to detect something, for instance find a restaurant, bar or a store, the system would call the Google Place API. We used a python wrapper called python-google-places to be able to access this API with python. If the parser found words containing information about time or location, these would also be used to provide the correct results. The results would then be output. Here is an illustration of the process:

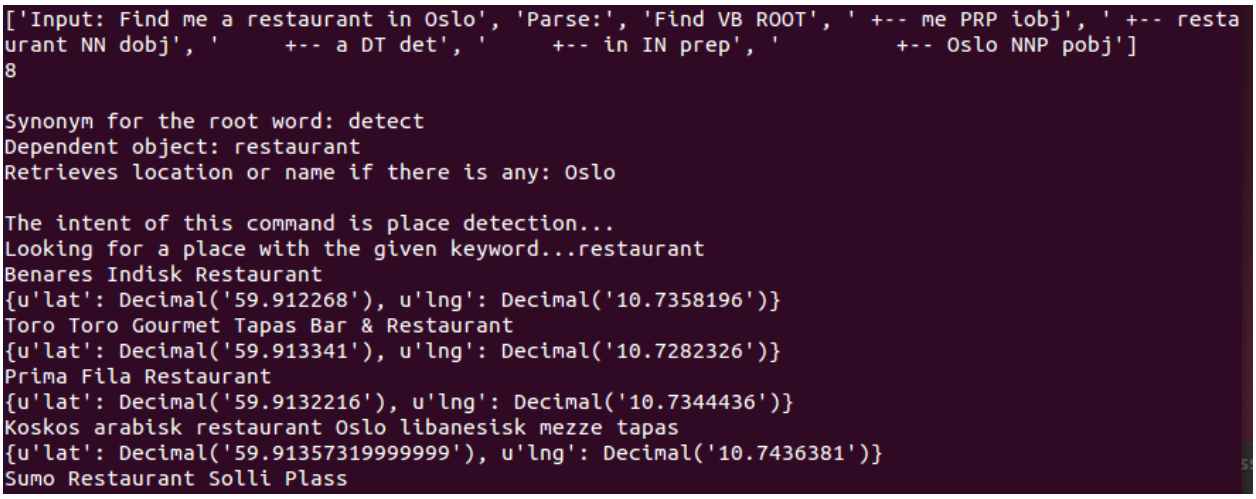


If we were to work more on this, we would clean up the output and add more API support, as well as support for smart home commands, like in the proposed system.

7 EXPERIMENTS AND RESULTS

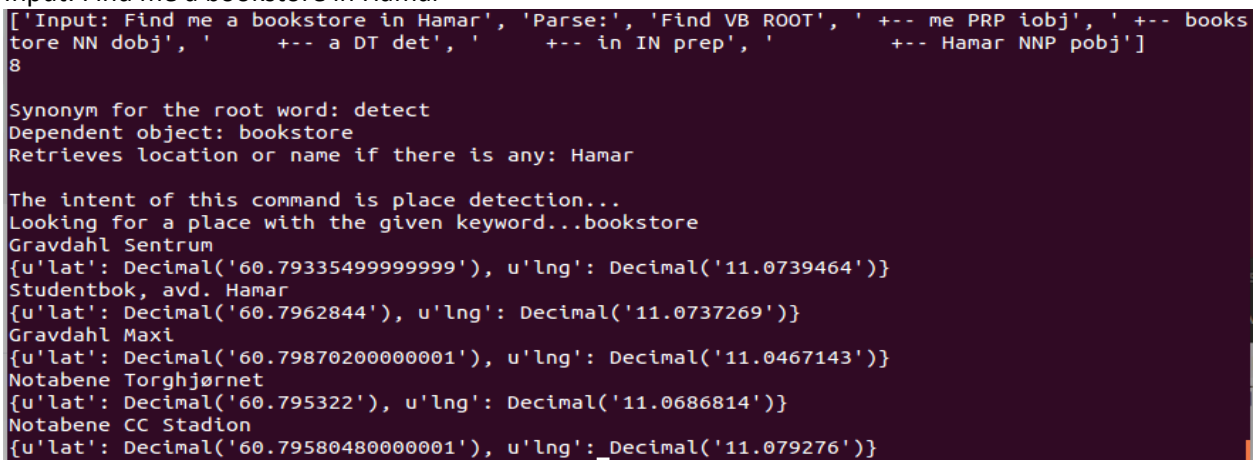
These experiments will mostly show what this system is capable of. It will not focus on time/capacity as the system is ran on a virtual Ubuntu operative system with a low amount of resources. The goal of these experiments is to figure out if the system helps the user by returning the correct results for the inputs. The following images shows the output after the parser has completed tagging the words of the input, and the results from the additional code we implemented.

Input: Find me a restaurant in Oslo:



Outputs the words and tagging. The verb root word (find), noun dependent object (restaurant) and the proper noun object of preposition (Oslo) is the most important to get the intent of the sentence. These words will then be analyzed further to get the desirable output. Here we can see that the intent of the sentence is to detect restaurants in Oslo. The system will then use a python Google Place API to retrieve information about restaurants in Oslo. On the image, we can see some of the restaurants it found, with name and location in coordinates.

Input: Find me a bookstore in Hamar



This input is like the first one, but with another keyword.

Input: Find me a pub in Bergen

```
['Input: Find me a pub in Bergen', 'Parse:', 'Find VB ROOT', ' +-- me PRP iobj', ' +-- pub NN dob
j', '     +-- a DT det', '     +-- in IN prep', '         +-- Bergen NNP pobj']
8

Synonym for the root word: detect
Dependent object: pub
Retrieves location or name if there is any: Bergen

The intent of this command is place detection...
Looking for a place with the given keyword...pub
Finnegans
{u'lat': Decimal('60.3926745'), u'lng': Decimal('5.3207057')}
Skipperstuen
{u'lat': Decimal('60.3912703'), u'lng': Decimal('5.322262699999999')}
Felix
{u'lat': Decimal('60.39475940000001'), u'lng': Decimal('5.3259289')}
Fotballpuben
{u'lat': Decimal('60.3902936'), u'lng': Decimal('5.3209103')}
Garage
{u'lat': Decimal('60.3894323'), u'lng': Decimal('5.323845599999999')}
Ricks AS
```

Input: What is the weather in Bergen?

```
['Input: what is the weather in Bergen', 'Parse:', 'weather NN ROOT', ' +-- what WP nsubj', ' +--
 is VBZ cop', ' +-- the DT det', ' +-- in IN prep', '     +-- Bergen NNP pobj']
8

Synonym for the root word: weather
Retrieves location or name if there is any: Bergen

The intent of this command is weather information...
Accessing weather-api for given location: Bergen for today
<pyowm.webapi25.weather.Weather - reference time=2017-05-06 18:46:54+00, status=Clear>
{u'speed': 1.16, u'deg': 269.502}
47
{'temp_max': 13.62, 'temp_kf': None, 'temp': 13.62, 'temp_min': 13.62}
```

This input requests weather information and is therefore different from the others. The system will then use a weather API to retrieve the information. Since time is not specified it will use the current time. We can see that the weather status is clear, the wind is 1.16 meters per second, the humidity is 47 % and the temperature is 13 degree celsius.

Input: What is the weather in Oslo tomorrow?

```
['Input: what is the weather in Oslo tomorrow', 'Parse:', 'weather NN ROOT', ' +-- what WP nsubj
, ' +-- is VBZ cop', ' +-- the DT det', ' +-- in IN prep', ' |   +-- Oslo NNP pobj', ' +-- tomor
ow NN tmod']
9

Synonym for the root word: weather
Retrieves location or name if there is any: Oslo

The intent of this command is weather information...
Accessing weather-api for given location: Oslo for tomorrow
<pyowm.webapi25.weather.Weather - reference time=2017-05-07 11:00:00+00, status=Rain>
{'speed': 2.7}
59
{u'min': 3.57, u'max': 11.78, u'eve': 9.16, u'morn': 5.13, u'night': 3.57, u'day': 10.77}
```

Another weather information request, but here time is specified (tomorrow). It looks like it will rain!

As we can see all inputs give valid outputs! With just a few lines of code and support for only two API we can retrieve a lot of useful information.

8 CONCLUSION

In this project, we have implemented a system which take use one of the most advance parsers currently available. Sadly, we were not able to implement our full proposed system as this would

have taken too much time. Nevertheless, we are happy with what we managed to implement. The system we implemented is just a preview of what you can do with a parser and it is powerful. Compared to a system which manually filter out keywords, locations, timestamps and intents of a sentence, this will require substantial less code and work. The trained parser can recognize patterns and generalize sentences, therefore it doesn't matter if you try to find weather information in Norway or USA, since both are supported without specifying rules to handle the requests. A system like this with a trained parser can give desirable outputs that is not possible to achieve with a system based around a set of hard coded rules, as it is impossible to predict every input. We are sure that systems like this will be a lot more common for the time to come.

9 TERMINOLOGY

AI - Artificial Intelligence
HMM - Hidden Markov Model. A statistical Markov Model
RNN - Recurrent Neural Network
LSTM - Long Short-Term Memory. Deep learning RNN
FFNN - Feed Forward Neural Network
API - Application Programming Interface
NLP - Natural Language Processing
DSP - Digital Signal Processing
Chat Bot - AI which responds to human text input
Smart homes - Modern houses which simplifies daily living.
POS - Part-of-Speech
Z-Wave – Open source communication protocol

10 REFERENCES

Back Propagation. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Backpropagation

Conditional random field. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Conditional_random_field

Deep learning. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Deep_learning

Deep learning for Acoustic Modeling in Parametric speech Generation. (n.d.). Retrieved from ieeexplore.ieee.org: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7078992

English speech dataset with different dialects. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/TIMIT

Feed forward neural network. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Feedforward_neural_network

Globally Normalized Transition-Based Neural Networks. (n.d.). Retrieved from arxiv.org: https://arxiv.org/pdf/1603.06042.pdf

Google. (n.d.). *Parsey Universal*. Retrieved from Github: https://github.com/tensorflow/models/blob/master/syntaxnet/g3doc/universal.md

Google speech API Documentation. (n.d.). Retrieved from cloud.google.com: https://cloud.google.com/speech/docs/

Hidden Markov Model. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Hidden_Markov_model

Introduction to conditional random fields. (n.d.). Retrieved from Blog.echen: http://blog.echen.me/2012/01/03/introduction-to-conditional-random-fields/

Long short term memory recurrent neural network (LSTM RNN). (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Long_short-term_memory

Markov Model. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Markov_model

Multi agent system. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Multi-agent_system

Neural network behind google voice. (n.d.). Retrieved from Google research blog: https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html

Neural networks. (n.d.). Retrieved from Standford Education: http://ufldl.stanford.edu/wiki/index.php/Neural_Networks

New algorithms used for speech recognition. (n.d.). Retrieved from Google research blog: https://research.googleblog.com/2015/09/google-voice-search-faster-and-more.html

Nivre, J. (n.d.). *Algorithms for Deterministic Incremental Dependency Parsing*. Retrieved from MIT: http://www.mitpressjournals.org/doi/pdf/10.1162/coli.07-056-R1-07-027

Parsing. (n.d.). Retrieved from demo.clab.cs.cmu: http://demo.clab.cs.cmu.edu/fa2015-11711/images/b/b1/TbparsingSmallCorrection.pdf

Publication of Google's speech recognition. (n.d.). Retrieved from Google research: https://research.google.com/pubs/SpeechProcessing.html

Recurrent neural networks. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Recurrent_neural_network

Short introduction to Text to speech. (n.d.). Retrieved from TCTS Lab: http://tcts.fpms.ac.be/synthesis/introtts_old.html

Sigmoid function. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Sigmoid_function

Speech recognition and deep learning (Google). (n.d.). Retrieved from Google research blog: https://research.googleblog.com/2012/08/speech-recognition-and-deep-learning.html

Statistitical inference. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Statistical_inference

Syntaxnet. (n.d.). Retrieved from Google research blog: https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

Syntaxnet. (n.d.). Retrieved from Github: https://github.com/tensorflow/models/tree/master/syntaxnet

Understanding LSTM. (n.d.). Retrieved from Github: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

University, S. (n.d.). *nlp.standford*. Retrieved from https://nlp.stanford.edu/software/dependencies_manual.pdf

Unsupervised context learning for Speech recognition. (n.d.). Retrieved from Research.Google.com: https://research.google.com/pubs/pub45759.html

What is text to speech and how does it work. (n.d.). Retrieved from blog.neospeech.com: http://blog.neospeech.com/2015/07/14/what-is-text-to-speech-and-how-does-it-work/

Wit.ai dev. (n.d.). *Wit.ai*. Retrieved from Wit.ai: https://wit.ai/