Low Resource End-to-End Spoken Language Understanding with Capsule Networks

Jakob Poncelet*, Vincent Renkens and Hugo Van hamme

KU Leuven – Department Electrical Engineering ESAT-PSI
Kasteelpark Arenberg 10, Bus 2441, B-3001 Leuven Belgium

ARTICLE INFO

Keywords:

Spoken Language Understanding
End-to-End
Intent Recognition
Capsule Networks
Multitask Learning

ABSTRACT

Designing a Spoken Language Understanding (SLU) system for command-and-control applications is challenging. Both Automatic Speech Recognition and Natural Language Understanding are language and application dependent to a great extent. Even with a lot of design effort, users often still have to know what to say to the system for it to do what they want. We propose to use an end-to-end SLU system that maps speech directly to semantics and that can be trained by the user through demonstrations. The user can teach the system a new command by uttering the command and subsequently demonstrating its meaning through an alternative interface. The system will learn the mapping from the spoken command to the task. The dependency on the user also allows different languages and non-standard or impaired speech as valid inputs. Teaching the system requires effort from the user, so it is crucial that the system learns quickly. In this paper we propose to use capsule networks for this task, which are believed to be data efficient. We discuss two architectures for using capsule networks. We analyse their performance and compare them with two baseline systems, one based on Non-negative Matrix Factorisation (NMF) which has been successful for this task and one encoder-decoder approach. We show that in most cases the capsule network performs better than the baseline systems. Furthermore, we demonstrate the versatility of the architecture by inferring speaker identity and the user's word choice through multitask learning.

1. Introduction

In this paper we will discuss a low-resource end-to-end Spoken Language Understanding (SLU) system for command-and-control applications. The system can recognise spoken commands and forward a semantic representation of the task to be performed to some agent. An example of such an application is giving spoken commands to a robot. The command *"fetch a cup of tea for Jeff"* could be mapped to `bring(tea, Jeff)`. A task is represented as a function call where the function is the type of task, in this case bringing something, and the arguments represent the specifics of the task. This function call can then be represented as a set of *labels*, one for the function and one for each of the arguments. The task representation should be unambiguous and easy to interpret by the agent. The goal of the SLU system is to map the spoken command to the set of labels representing the task.

We will address design possibilities for an SLU system where the user can teach the system a new command by giving the command and subsequently demonstrating the command through some alternative interface, e.g. by clicking on the right task specification on a screen. The system learns the link between the spoken command and the demonstrated task. The user can choose how to specify a command and the task will be demonstrated in the environment and context of the agent. We opt for an end-to-end model that learns this mapping from speech to intent directly without intermediate steps, unlike traditional SLU systems. Because the user has to teach the system commands by uttering

*Corresponding author
jakob.poncelet@esat.kuleuven.be (J. Poncelet); vincent.renkens@esat.kuleuven.be (V. Renkens); hugo.vanhamme@esat.kuleuven.be (H. Van hamme)
ORCID(s): 0000-0002-5157-7577 (J. Poncelet); 0000-0002-2419-4642 (V. Renkens); 0000-0003-1331-5186 (H. Van hamme)

them, the model is automatically adapted to the speech, language and preferred wordings of the user. However, for ease of use the system should learn commands quickly without requiring many examples.

The primary goal of this manuscript is to evaluate several design options for a capsule-based end-to-end SLU (E2ESLU) system, to compare it to existing models on various datasets, to highlight its potential in performing multiple tasks and to gain more insight into the working and interpretability of the network.

1.1. Design of SLU systems

Traditional SLU

Traditionally an SLU system is implemented in a multi-step fashion [30, 4, 1]. First an Automatic Speech Recognition (ASR) component is used to find a textual transcription for the spoken utterance. The resulting text is processed by a Natural Language Understanding (NLU) module to figure out the intent of the user. This methodology has several issues.

For good speech recognition performance the ASR component is usually trained on lots of data in a given language [11, 31]. The resulting system only works for the language it is trained on, so a different ASR system has to be trained for every language supported by the system. This can be challenging for under resourced languages or dialects. For people with deviating speech, due to e.g. strong dialect, advanced age or a speech impairment, the performance of such an ASR system degrades dramatically [3].

Because the ASR and NLU systems are trained separately, the NLU system is not trained to be able to handle errors introduced by the ASR system. Usually the NLU system is trained with clean text data. Conversely, the ASR system is trained to minimise the number of errors, but cannot focus on words that are important for the NLU system [23]. Training the systems separately is sub-optimal and better performance could be obtained by training them jointly.

The task of the NLU system is to translate the textual transcription to a task to be performed in the environment of the agent. We will discuss three reasons why designing an NLU system for command-and-control applications is challenging.

1. **Task specification**: There are many ways the user can specify a task. There are two types of NLU systems: those based on hand crafted rules and those based on machine learning. The rule based systems, like systems using key phrases or context free grammars, are difficult to design. The designer has to predict all the ways a user will specify a task. This is often infeasible in practice, making these systems incomplete. The user will have to know what to say, i.e. adapt to the system, which is not a very natural way of communicating. The machine learning systems, based on e.g. Conditional Random Fields [19] or Recurrent Neural Networks [7], need lots of data to train in order to generalise well. Acquiring this data can be expensive and time consuming.
2. **User context**: For many command-and-control applications the context of the agent may differ from one user to another. In the example of fetching a cup of tea, the agent needs to know where the tea is in the house, which one of several containers with tea the user intended, who and where Jeff is, etc. From a design perspective it is very hard to predict what the agent will have to do in the context of each specific user.
3. **Application dependence**: The NLU system is application and language dependent. Whenever a new command-and-control application is launched or support for a new language is added, the NLU system has to be redesigned or trained on data collected in the domain of the application. This makes it difficult and expensive to launch such a new application.

End-to-end SLU

The goal of our proposed system is twofold. We want to make the design process for a command-and-control application easier and we want to eliminate the need for the user to adapt to the system. We propose to have the user teach the system the commands and corresponding tasks. We use an end-to-end SLU (E2ESLU) system that learns the meaning of the commands through a limited amount of demonstrations by the user. It is end-to-end, meaning that there is no textual representation anywhere in the system, but speech is mapped directly to semantics. Both the speech and language components are integrated into one model which is trained using the commands and demonstrations provided

Table 1
Examples of spoken commands from the `Fluent Speech Commands` dataset and how they are represented as tasks. Only these task representations are available as targets in the E2ESLU system, which learns to map the speech waveform or speech features directly to the intent of the command.

Spoken sentence	Intent representation
Turn the temperature in the bedroom down.	`<changewhere what="heat" how="decrease" where="bedroom"/>`
Lights off.	`<switch action="deactivate" thing="lights"/>`
That's too loud.	`<change what="volume" how="decrease"/>`
I need to practice my German, switch the language.	`<changelang language="German"/>`

by the user. The user can choose how to specify a command and the task will be demonstrated in the environment and context of the agent. The link between natural language and a task in the environment of the agent is learned. Table 1 shows example mappings of how the intent of a command is presented to the system.

E2ESLU does not contain language dependent components: the model learns the speech of the user in any language through giving examples, so universal language support is built in by design, as well as speaker-specific phrasings. Speech impairments are also not a fundamental issue as long as the speech is sufficiently consistent and discriminative. The application dependence is limited to specifying the semantic representation of the task, which allows for easy integration in multiple domains. Because of its end-to-end nature, propagation of recognition errors due to mismatched objective functions is less of an issue, compared to pipeline SLU systems.

The major drawback of E2ESLU is that it requires effort from the user to teach the system. This can somewhat be reduced by pre-training the model on a general corpus, but to solve the issues above user input is required. For a user friendly experience it is crucial that the amount of examples required to train the system is minimal. Most of the advances in speech processing and machine learning in general are related to deep learning [11, 2, 23]. Deep learning based approaches perform very well given that a lot of training data is available. Training such a system using only user data is infeasible, hence other methods and architectures have to be explored.

An additional challenge for E2ESLU is that no transcriptions of the commands are available for training and the input utterances are unsegmented, i.e. it is unknown where words start or end. Unlike in typical ASR training paradigms, the correct order of the words or even the order in which the labels are referred to is not available to the system. It is hence challenging for such a system to learn any grammatical structure.

Broadly speaking, E2ESLU systems have still not superseded the performance of well-trained multi-step SLU systems. While achieving high accuracies without large amounts of training data is hard in general E2E systems, with additional techniques like pre-training [14], artificial data generation [5] or transfer learning [27], a comparable performance to state-of-the-art ASR-NLU pipeline systems can be achieved with E2ESLU [23, 10].

1.2. Approaches for E2ESLU

Our previous and current approach for E2ESLU are based on compositional semantics in command-and-control [24]: the meaning of a complex expression (e.g. a sentence) is determined by the meaning of the parts and the way they are combined. Parts of a spoken sentence will be mapped onto labels (as defined earlier), i.e. semantic function calls and their arguments. These parts will re-occur in other commands where the parts can be combined to designate new meaning. For example, the command *"fetch me a cup of tea"* could be mapped to `bring(tea, user)` where the function `bring(., .)` and the argument `tea` are learned from other demonstrations. This avoids that all combinations of function and arguments have to be demonstrated, allowing the E2ESLU system to generalise and learn from fewer examples.

In the past we have proposed a method based on Non-negative Matrix Factorisation (NMF) for this task [8, 17, 20] and applied this method in systems for home automation, robotics and more. Compositionality at the sentence level is translated into mathematics as a (non-negative) linear combination of non-negative high-dimensional vectors representing sentence parts or words. This method achieves good performance for tasks with a limited vocabulary and simple commands [9]. NMF however has some limitations. It is a shallow, linear model, so it has a limited learning

capacity. It is also a bag-of-words model, so it can only infer meaning from which words occur, but not from word order. In simple command-and-control tasks this might be sufficient, but in more complex environments with more concepts to learn and more complex commands this approach falls short.

To alleviate the restrictions of NMF we have recently explored a method based on capsule networks [21]. A capsule network [22, 12] is a deep learning based approach that is specifically designed to learn part-whole relationships. Just like NMF a capsule network assumes compositionality, but in its whole it is a more powerful non-linear model where the order of the words can be encoded using recurrent connections.

A capsule network is composed of capsules which are activated by patterns in the input. A capsule is encoded as a vector with an orientation that reflects the pattern's properties and a length that reflects its activation. Capsules in a lower layer represent parts and contribute to a capsule in the next layer representing the whole. The extent to which a part contributes to the whole is determined dynamically by an algorithm called routing-by-agreement, a process that is reminiscent of attention [2]. Every 'part' capsule in the lower layer votes on the orientation of the 'whole' capsules in the higher layer using a linear transformation. If a group of capsules in the lower layer agrees on the orientation of a capsule in the higher layer, this capsule will be activated with its orientation as agreed upon by the capsules in the lower layer. Parts must however distribute their votes over wholes (via a softmax, see Eq. 2 below) and ideally contribute to just one whole.

The part-whole relationships, which are the voting transformations, are simple, so the model can potentially learn them with limited amounts of training data. Furthermore, the variability in the input can be represented by variation in the orientation of the capsules and does not have to be baked into the model parameters. This way more variability can be covered by a model with less parameters. A recurrent neural network can additionally encode temporal effects in the input features and pass this information to the capsules. Finally, capsule networks have shown to be able to exploit the semantic hierarchy present in slot filling and intent detection from natural text [32]. These are the reasons why capsule networks will be compared to the NMF method for the proposed system. A good SLU performance requires fast learning with little training data, but the asymptotic accuracy when more training data is available should also be high. In this paper we will further examine the model proposed in [21] and compare it with a capsule network that leans closer to the model proposed in [22].

To investigate the benefit of using capsule layers in a neural network, we will compare the capsule-based models to an encoder-decoder (ED) architecture for E2ESLU, proposed in [23]. The input features of this model are encoded by the same encoder, but transformed to a fixed-size output vector of label probabilities by max-pooling and a fully connected layer, instead of a capsule network.

An alternative deep neural network approach to E2ESLU has been proposed in [14]. The model consists of a stack of three modules: a phoneme, a word and an intent module. The first two modules can be pre-trained on additional data to predict phonemes and words respectively, as sequence-to-sequence classifiers. Afterwards the entire model is trained end-to-end on the supervised SLU task. The intent module comprises a recurrent layer and a global max-pooling layer. The performance of all models will be compared to this architecture on a benchmark.

1.3. Contributions

In this paper we will explore the extensive possibilities and potential of capsule networks in E2ESLU and make a performance comparison of capsule-based systems to several existing state-of-the-art approaches on various datasets.

First, in terms of network design, we will translate the work on image recognition with capsule networks in [22] to the E2ESLU problem in two architectures, one of which we presented earlier in [21]. A recurrent neural network encoder captures temporal aspects in the spoken utterance. This timing information can be encoded in the capsules in different ways, hence we differentiate between rate-coded and place-coded capsules and analyse the effects of both methods. The two proposed capsule-based models are compared to the NMF and ED model on several datasets of different sizes and domains.

Second, the original work in [22] proposed regularisation of the capsule network with a reconstruction loss to improve accuracy and interpretability. In our proposed SLU model, every output capsule corresponds to a specific label and the presence of a label in the input command is determined from the length of the output capsule's vector. Regularising the

orientation of the capsule vectors can enforce the capsule network to reflect additional properties about the utterance in its output dimensions. We will experiment with similar reconstruction loss regularisations and evaluate this technique from the perspective of intent recognition accuracy and data requirements.

Reconstruction of the input speech as a regularisation would also offer the advantages an autoencoder has to offer. In particular, part of the training data could be unlabeled. In the E2ESLU use case, this means we can speak relevant commands without giving the actual demonstration, which saves time and effort. On top of that, we could trace the contribution of each activated capsule to the input, and hence infer the order in which the command's arguments were spoken, which is important in many languages. This would be an alternative to the method that will be described in Section 6. As a final example, tracing a capsule's activation back to the input with sufficient temporal resolution would allow us to segment out instances of the user's word choice for its corresponding label. A dialogue system built on the E2ESLU system would then be able to speak back in the user's own wording.

However, reconstructing the input from the capsule content is less straightforward for a dynamic speech signal than for a still image of a handwritten digit. There is a lot of variation in the extracted features, because of different speakers, different acoustic environments, etc. Therefore in this work we will not apply a general reconstruction loss. Instead, we will study what information the capsules are able to contain (how far we can stretch the capacity of the capsule network) and whether leveraging this additional knowledge into the system actually helps the performance. On the one hand, this will be a good indicator for the feasibility of a complete reconstruction of the input signal from the capsules and its possible effect on the performance. On the other hand, it will demonstrate the versatility and capacity of the architecture fulfilling multiple tasks at once, regardless of whether the performance improves or not.

To this end, we extend the capsule networks with regularisations to employ multitask learning. We will show that regularisation can steer the capsule content towards an interpretable content, such as speaker identity. The extended model will jointly learn three tasks: recognising the intent labels, identifying the speaker and reconstructing the spoken words. In an SLU system, the identity of the speaker is useful information to learn [26]. It can encourage the capsules to differentiate the encoding of the vector orientation between speakers and allow some kind of adaptation to the speakers by learning characteristics about them, which is helpful for decoding following utterances from the same speaker. Note, however, that *no claim* is made that this would be the most accurate approach for speaker identification. In addition, we will provide transcripts of the utterances to the system together with the speech, and train the model to reconstruct which specific words were spoken in the utterance. Reconstruction of the uttered words is more closely related to the exact waveform of the speech and acts as an intermediate step towards the reconstruction of the speech input. Moreover, simultaneously learning words and intent can have a synergistic effect, if a recognised word gives a clue about the intent [32]. From the transcripts of the training examples, a lookup table is built containing all spoken words. These are the only words the model will be able to predict. The transcripts of the commands are provided as an unordered list of words: only a binary vector, indicating which words from the lookup table are present in the entire utterance, is supplied (without order or timing information). So the system will actually reconstruct the word occurrences. Evidently those transcripts are usually not present in practice, hence if providing this additional information to the system does not turn out to help the performance, or if reconstructing the words would already be too difficult, it is not very likely that complete reconstruction would be feasible. At the same time, if the capsules manage to learn the words, this would be a good indication and a nice proxy towards really reconstructing the speech signal or its spectrogram.

With the extensions outlined above, we will demonstrate the ability to integrate additional tasks into the capsule network model. This is a great benefit for designers of systems who would like to include extra possibilities. For example, adding speaker identification to the model allows one device to be shared by multiple users and using this information to the benefit of the user: the agent now knows e.g. to whom to bring the cup of tea, without the user having to specify it every time. However, we should stress that the goal is not to build a speaker identification system, since more dedicated models undoubtedly would outperform the command-learning capsule network. We only show that the capsules can contain other information and how additional tasks like speaker identification can come "for free" on the side, which opens possibilities for a designer.

Third and finally, to gain further insight into data propagation through the capsule network, we will perform an alignment analysis across the network. Additionally, this will show the system's potential to infer the order in which labels were spoken in the input utterance; information that was never provided during training. The intent of the

spoken command is presented to the system as a binary target vector, representing the task as a function call with arguments as shown in Table 1, without any timing information or chronological order. For example, the spoken commands to a robot *"slowly drive forward"* and *"drive forward slowly"* are presented as the same intent, namely `drive(direction=forward, throttle=low)`. Hence if label order could be inferred, the capsules would be able to extract meaning from the order of the words, in contrast to the NMF approach. Learning the order of the words or labels can also be promising for extracting grammatical structure. Similarly, preserving the order of predicted labels has already been shown to be feasible in NLU systems that extract intent from a textual input [6, 15].

The rest of the paper is structured as follows. In Section 2 we will explain capsule networks in more detail and propose two architectures for applying them to the task of intent recognition. The regularised versions of these models that allow speaker identification and word reconstruction will also be covered. In Section 3 we will discuss the baseline systems to which the capsule models are compared, the datasets that are used in the experiments and the experimental methodology. In Section 4 we will present the experimental results of this comparison among the basic models in the slot-filling task. Section 5 will cover the effect of adding regularisations to the capsule network and will present some performance results of the regularised capsule network that focuses on multiple tasks at the same time. In Section 6 we will mathematically derive and perform an alignment analysis to determine the order of labels in each utterance by measuring capsule contributions. Finally, we will conclude the work in Section 7 and give some suggestions on future work in Section 8.

2. Model

In this section we will first explain how basic capsule networks work in more detail. Then we will discuss two possible implementations of a capsule network for SLU. The first implementation will be called the Place-Coded Capsule Network (PCCN), an architecture closest to the network proposed in [22]. The second network was proposed in [21] and will be called the Rate-Coded Capsule Network (RCCN). Finally we will explain how these models can be extended with regularisations to incorporate multiple tasks and feed additional information into the model, focusing on word reconstruction and speaker identification.

2.1. Capsule Networks

A capsule network constructs bigger wholes from smaller parts using linear relationships. The first step for such a network is to find the smallest parts that can be combined to find bigger wholes. These smallest parts are called the primary capsules. The difference between the PCCN and the RCCN is how the timing of the primary capsules is coded. Place-coded capsules encode the timing of the capsule as its location in the primary capsule tensor $\boldsymbol{P}$. Rate-coded capsules encode the timing in their orientation. For place-coded capsules the same capsule can be activated multiple times, just at different times. This is not possible for rate-coded capsules, because a primary capsule is activated at most once for the complete utterance.

At the output there is one capsule per output label. The probability that the output label is present in the input is modeled by the norm of its capsule activation vector. The primary capsules and the output capsules are connected by several capsule layers. How each capsule in a lower layer contributes to the capsules in the higher layer is determined by the routing-by-agreement algorithm proposed in [22].

The capsules in the lower layer vote on the orientation of the capsules in the higher layer. The vote $\boldsymbol{v}_{ij}$ from lower capsule i for higher capsule j is computed using a linear mapping:

$$\boldsymbol{v}_{ij} = \boldsymbol{W}_{ij}\boldsymbol{c}_i \tag{1}$$

$\boldsymbol{c}_i$ is the vector of activations of the i^{th} lower capsule and $\boldsymbol{W}_{ij}$ is the voting transformation of the i^{th} lower capsule for the j^{th} higher capsule. The output capsules are activated if a group of lower capsules agree on its orientation. This

agreement is found by iteratively updating the routing weights between the capsules. Given the routing logits, the routing weights are computed using the softmax function:

$$\boldsymbol{w}_i = \text{softmax}(\boldsymbol{b}_i) \tag{2}$$

$\boldsymbol{w}_i$ and $\boldsymbol{b}_i$ are the routing weights and logits coming from the i^{th} lower capsule. This means that every lower capsule i is distributed over the higher capsules following the routing weights $\boldsymbol{w}_i$. Using the routing weights, the activation vectors of the higher capsules can be found:

$$\boldsymbol{o}_j = \sigma(\sum_i w_{ij}\boldsymbol{v}_{ij}) \tag{3}$$

$\boldsymbol{o}_j$ is the activation vector of the j^{th} higher capsule, which is found using an average of the votes of lower capsules, weighted by the routing weights. The norm of the activation vector represents the probability that the capsule is activated. To make sure it lies between 0 and 1 a squashing function $\sigma(\cdot)$ is used:

$$\sigma(\boldsymbol{x}) = \frac{||\boldsymbol{x}||^2}{||\boldsymbol{x}||^2 + 1}\frac{\boldsymbol{x}}{||\boldsymbol{x}||} \tag{4}$$

The routing logits can then be updated by how strongly the votes of the lower capsules agree with the current output capsule activations. The agreement is computed using a simple dot-product, which is added to the routing logits:

$$b_{ij} \leftarrow b_{ij} + \boldsymbol{v}_{ij} \cdot \boldsymbol{o}_j \tag{5}$$

The weight between the lower capsule i and the higher capsule j will be increased if its vote $\boldsymbol{v}_{ij}$ agrees with the current consensus for the orientation of the higher capsule $\boldsymbol{o}_j$. The updated routing logits can now be used to compute new higher capsules etc. The routing-by-agreement algorithm is summarised in Algorithm 1, where N is the number of iterations. The initial set of routing logits $\boldsymbol{B}^{(1)}$ is a parameter from the model that can be learned.

Define variable $\boldsymbol{B}^{(1)}$;
for $n = 1{:}N$ **do**
  For all lower capsules i: $\boldsymbol{w}_i = \text{softmax}(\boldsymbol{b}_i^{(n)})$;
  For all higher capsules j: $\boldsymbol{o}_j = \sigma(\sum_i w_{ij}\boldsymbol{v}_{ij})$;
  For all logits $b_{ij}^{(n)}$ in $\boldsymbol{B}^{(n)}$: $b_{ij}^{(n+1)} = b_{ij}^{(n)} + \boldsymbol{v}_{ij} \cdot \boldsymbol{o}_j$;
end

Algorithm 1: Dynamic routing algorithm

In the last layer of the capsule network there is one output capsule for every label. The norm of (the activation vector of) an output capsule yields the corresponding label probability. The network is then trained using the max-margin loss, as used in [22], where l_j is the output probability of label j and t_j is its reference target.

$$L = \sum_j t_j \max(0, 0.9 - l_j) + (1 - t_j)\max(0, l_j - 0.1) \tag{6}$$

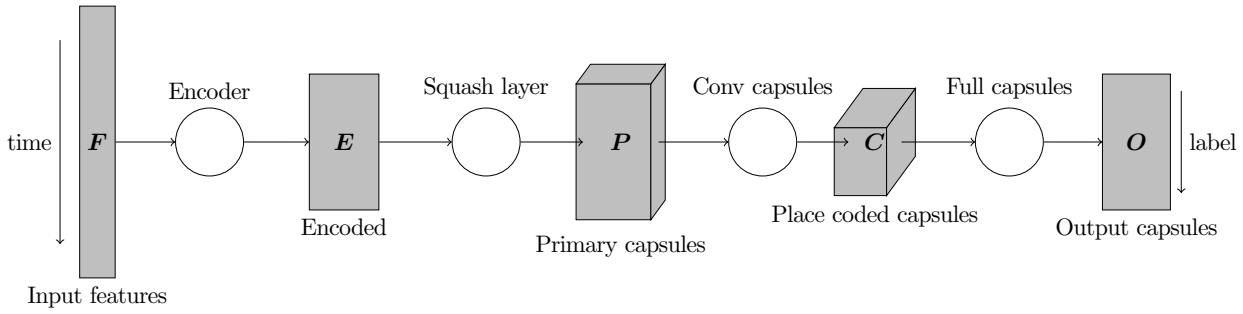

Figure 1: A schematic of the place-coded capsule network.

2.2. Place-Coded Capsule Network

A schematic of the PCCN can be found in Figure 1. Mapping the input features $\boldsymbol{F}$ to the primary capsules $\boldsymbol{P}$ is highly non-linear. This mapping is implemented using a Bidirectional GRU Encoder to encode the features. In between the layers of the GRU the activations are subsampled: only every other timestep is kept. The primary capsules $\boldsymbol{P}$ are found using a linear layer on the encoded features, followed by a squashing function. Note that there is a set of primary capsules for each timestep. The timing of the primary capsules is coded in where the primary capsules are located in the primary capsule tensor $\boldsymbol{P}$, which is why we call them place-coded.

The primary capsules are followed by a 1-dimensional convolutional capsule layer. In a convolutional capsule layer the voting is not only based on the activation of the lower capsule at the corresponding timestep, but also on the activation of that capsule in the neighboring timesteps. This allows the model to stitch together multiple timesteps to create longer units.

We expect that the place-coded capsules will model small acoustic units like phones. They are combined into words for the output labels by routing them to the output capsules $\boldsymbol{O}$. All place-coded caspules in all timesteps vote for all output capsules, hence we call this a fully connected capsule layer or full capsules in the schematic. Because the number of timesteps varies across examples, the voting transformations are shared between timesteps. The votes for the output capsules are computed as:

$$\boldsymbol{v}_{tij} = \boldsymbol{W}_{ij}\boldsymbol{c}_{ti} \tag{7}$$

$\boldsymbol{c}_{ti}$ is the i^{th} capsule in timestep t and $\boldsymbol{v}_{tij}$ is its vote for the j^{th} output capsule. $\boldsymbol{W}_{ij}$ is the voting transformation matrix.

2.3. Rate-Coded Capsule Network

A schematic of the RCCN can be found in Figure 2. In the RCCN the primary capsules are rate-coded. The input features $\boldsymbol{F}$ are first encoded with the same encoder as used in the PCCN. Every timestep is assigned to a primary capsule using the distributor. The distributor distributes each timestep among the primary capsules and is implemented using a softmax layer. The number of outputs in this layer is the same as the number of primary capsules. The outputs of the distributor $\boldsymbol{\delta}$ determine how each timestep is connected to each primary capsule. Furthermore, an attention layer is used to put weights $\boldsymbol{\alpha}$ on each timestep, making the network able to filter out irrelevant parts of the input. The attention is implemented using a sigmoid layer with a single output. For ever primary capsule, weighted averages $\boldsymbol{Q}$ of the encoded features $\boldsymbol{E}$ are taken using the attention and distributor to compute the weights of the average:

$$\boldsymbol{q}_p = \sum_t \alpha_t \delta_{tp} \boldsymbol{e}_t \tag{8}$$

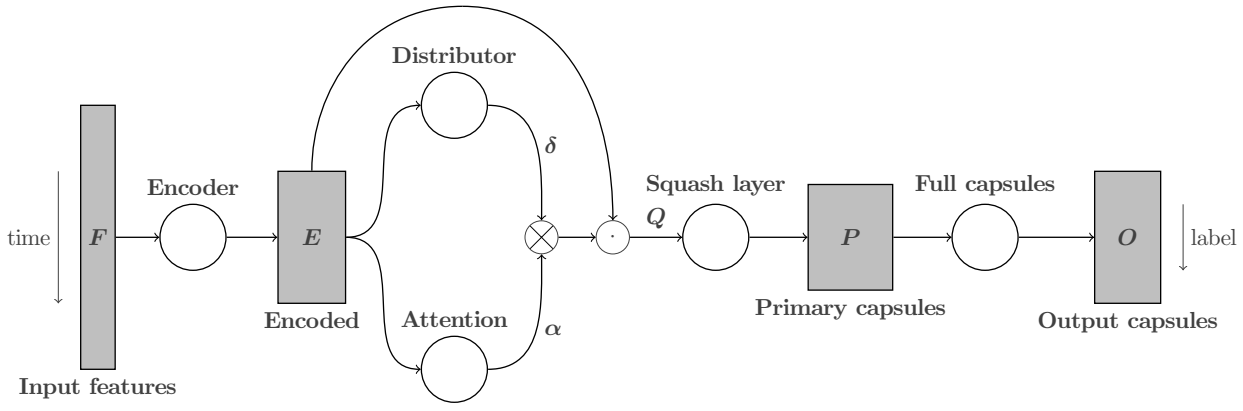

Figure 2: A schematic of the rate-coded capsule network.

$\boldsymbol{q}_p$ is the weighted average for primary capsule p and $\boldsymbol{e}_t$ contains the encoded features at timestep t. The weighted averages $\boldsymbol{Q}$ are then converted to the primary capsules $\boldsymbol{P}$ using a squashing layer. As opposed to the PCCN, the RCCN only has one set of primary capsules, the time dimension is no longer present. The timing of the primary capsules is coded in its orientation, which is why we call them rate-coded. The distributor attention mechanism in RCCN can attend to long units presumably like words or sub-words, while PCCN operates more locally through convolutions. Place-coded capsules will likely model e.g. phones or other sub-word units that can occur multiple times in an utterance. Finally, the primary capsules are routed to the output capsules $\boldsymbol{O}$ using a fully connected capsule layer.

2.4. Regularised Capsule Network

Based on the length of the activation vectors of the PCCN and RCCN output capsules, the semantic frame is filled and the intent is recognised. When trained with the loss function in Eq. 6, any variation in the data may be propagated to the orientation of the output capsules or the network may learn to ignore it (e.g. in the encoder). In order to force the network to learn useful representations in the output capsules, we propose to regularise the loss function. The aim of *reconstruction* as a regularisation, is to not only explicitly make use of the length of the output vectors (as in Eq. 6), but also encourage the output capsules to encode information in the orientation of the activation vectors. In our case, this consists of learning the identity of the speaker and reconstructing which words were spoken (trained on unordered transcripts of the utterances). Accordingly the terms *regularised* and *multitask* refer to the same thing.

We start with a definition of the average capsule $\boldsymbol{z}$ for each utterance, with N the number of output capsules, based on the output capsule vectors $\boldsymbol{v}$ as defined in Eq. 1.

$$\boldsymbol{z} = \frac{\sum_{i=1}^{N} \boldsymbol{v}_i}{\sum_{i=1}^{N} \|\boldsymbol{v}_i\|} \tag{9}$$

The average capsule is thus a column vector of dimension equal to the dimension of the output capsules. Since the average capsules "summarise" for every dimension the information that is encoded in it, they are a useful representation to derive properties of the utterance from by applying an extra linear mapping to get logits for every speaker and every word.

A single softmax layer maps the average capsules to speaker probabilities. The weight matrix of this layer will be called the projection matrix $\boldsymbol{W}_s$ and has dimensions $(n \times M)$, with n the output dimension of the capsules and M the number of speakers in the dataset. In the testing phase, the model chooses the speaker with the highest probability.

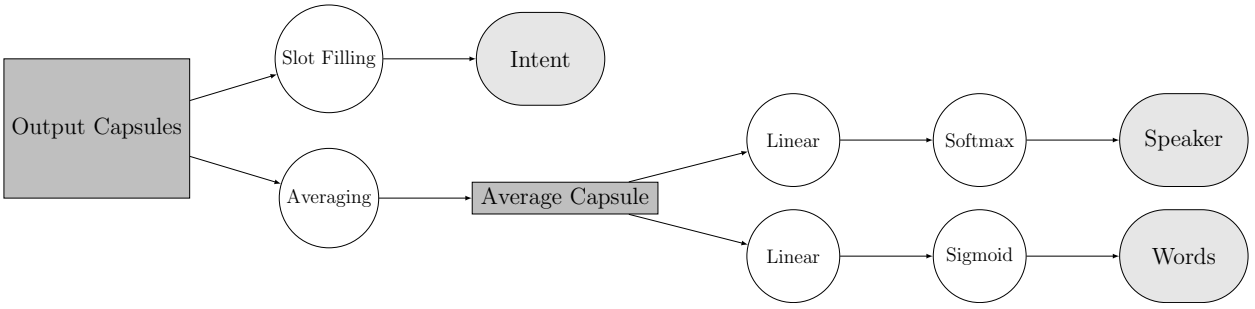

Figure 3: A schematic of the extension to the basic capsule network. The regularised capsule network consists of the basic capsule network combined with this extension.

Note that the number of speakers has to be chosen beforehand, but can easily be overestimated, i.e. additional empty slots can be reserved in case more speakers than anticipated will use the system. This allows adding speakers to the system without having to completely retrain the model.

Parallel to the speaker identification layer, a different single-layer neural network maps the average capsules to word logits. Like before, we define a new projection matrix $\boldsymbol{W_w}$ of dimensions $(n \times W)$, with n the output dimension of the capsules and W the vocabulary size of the dataset. Again this trainable matrix will map the average capsules to the words present in the sentences. However, since each utterance contains more than one word, the softmax layer is replaced by a sigmoid layer. Decoding in the testing phase requires a threshold above which the model decides if the word is present. Figure 3 shows a schematic overview of the extension to the basic capsule network. Combining the basic capsule network (Figure 2 for rate-coded capsules or Figure 1 for place-coded capsules) and the extension makes the regularised capsule network.

Notice that the model is not really learning to decode speech the way an ASR module would, but it will learn to use the capsule orientations to infer which words (e.g. among synonyms a user may employ) from a vocabulary are spoken in every utterance, without predicting their order. During training, the information provided is also a transcript as an unordered list of words.

The new information can be taught to the system by defining losses based on the predicted probabilities and the real speakers or words. Backpropagation of the loss will then adapt the trainable projection matrices and biases to make a better classification.

The speaker loss uses a simple cross-entropy loss function based on the target speaker (as a one-hot encoded vector $\boldsymbol{t}$) and the estimated probabilities P_i for every speaker i, and is summed over all M speakers.

$$L_s = -\sum_{i=1}^{M} t_i log(P_i) \tag{10}$$

For the word reconstruction loss, a cross-entropy loss function is used on the target words (binary vector $\boldsymbol{t}$) and the estimated word probabilities P_i for every word i, but with word specific factors weighing its two terms. The loss is summed over all words W.

$$L_w = -\sum_{j=1}^{W} \frac{a_j}{2} t_j log(P_j) + \frac{b_j}{2}(1 - t_j) log(1 - P_j) \tag{11}$$

These word specific weights a_j and b_j are defined in Eq. 12, where N is the total number of sentences in the training

set and Q_j is the number of sentences in the training set which contain the word j.

$$a_j = \frac{N}{Q_j} \qquad b_j = \frac{N}{N - Q_j} \tag{12}$$

These factors have been added to deal with infrequent words, much like other researchers have introduced weights in the loss for Noise Contrastive Estimation for dealing with sampling imbalance [16]. The first term of the loss is for when the word is present and the second term for when the word is absent. When a word rarely occurs, the model will suppress the probability P_j, because the second term is active in almost every sentence, and thus the loss will be small when the word is present. To amplify the loss if these words are present, the factors have been added. They also allow to choose a fixed threshold after the sigmoid in the decoding phase for deciding if a word has been said or not, instead of having to determine a word-specific threshold based on a held-out dataset.

Finally, both losses are added to the total loss (for now only consisting of the label loss as defined in Eq. 6), with a regularisation parameter for each term, to weigh their respective relevance and orders of magnitude. We define the regularisation parameters λ_s and λ_w as speaker weight and word weight respectively.

$$L_{tot} = L_l + \lambda_s L_s + \lambda_w L_w \tag{13}$$

3. Experimental Setup

3.1. Baselines

The model proposed in [17] uses NMF to find recurring patterns in the input data and link them to semantic labels. For every utterance a Histogram of Acoustic Co-occurrences (HAC) [28] is created, which is a single vector of counts of co-occurring acoustic events. The acoustic events are found by unsupervised fitting of a GMM to the input features, where every Gaussian represents an event. The HAC vectors are decomposed together with the semantic representations to get a dictionary of words linked to labels. A label can be linked to multiple words by introducing additional columns in the matrix representation. This dictionary is used to decompose other utterances to find which labels occur in the utterance. We will refer to this method as NMF.

In [23] an encoder-decoder (ED) neural net architecture was proposed for SLU. This model consists of the same encoder as used in the capsule networks above and a decoder. The decoder performs max-pooling on the encoded features to get a single vector for the entire utterance. This vector is converted to label probabilities with a single hidden layer network with sigmoidal outputs. The decoder has 1024 hidden units. We will refer to this network as ED.

In [14] the authors established benchmark accuracies of their model on the SLU dataset `Fluent Speech Commands`. The model consists of a stack of three different modules. The first module tries to predict phonemes, and gives an input to the second module, which tries to predict words. Both modules can be pre-trained on ASR data and contain multiple convolutional, recurrent and pooling layers. The word module is followed by an intent module, which consists of a recurrent layer followed by global max-pooling to get a fixed-size output vector of label logits. After pre-training the phoneme and word classifiers are discarded and the entire chain is trained end-to-end on the supervised SLU task. For a fair comparison we will use the accuracies of the model without pre-training in Section 5.4. Since the code for the model was not available at the time of writing, we only compare to the accuracies reported in their paper.

3.2. Datasets

The performance of the SLU systems will first be analysed on three datasets, each containing commands for a different application. All these datasets can be downloaded from `https://www.esat.kuleuven.be/psi/spraak/downloads`.

Table 2
Total number of utterances in the SLU datasets.

Dataset	# Utterances
GRABO	6007
PATCOR	2020
DOMOTICA-3	3012
Fluent Speech Commands	30043

The GRABO [20] dataset contains English and Dutch commands given to a robot. The robot can move in its environment, pick up objects and use a laser pointer. Typical commands given to the robot are "*move to position x*" or "*grab object y*". Output labels include positions in the world, the actions the robot can take, etc. There is a total of 33 output labels. Data was recorded from 11 speakers using 36 different commands with 15 repetitions each.

The PATCOR dataset [25] contains Dutch utterances from a vocally guided card game called Patience. The players can move cards or get new cards from the deck. Typical commands are "*Put card x on card y*" or "*New cards*". The output labels are the value and suit of the card being moved, the target position, etc. There are 38 output labels. Data was recorded from 8 speakers (speakers *pp5* and *pp11* have been excluded).

The DOMOTICA-3 dataset [17] is a follow-up of the DOMOTICA-2 dataset [25] and contains utterances from Dutch dysarthric speakers using voice commands in a home automation task. Typical commands are "*open door x*" or "*turn on light y*". The output labels include all the lights, doors and all the actions the system can take. In total there are 25 output labels. Data was recorded from 17 speakers with varying levels of dysarthria. Because speaking costs more effort for some speakers, the amount of data per speaker varies greatly.

Finally another dataset is introduced, called Fluent Speech Commands [14]. This dataset contains around 30000 English commands in a smart-home or virtual assistant setting from 97 speakers, which corresponds to 19 hours of speech. The dataset was specifically created to analyse the performance of SLU systems when the commands have many different phrasings (different wordings for each intent), as it would need to handle in a real practical system. There are a 248 phrasings that are mapped to 31 unique intents. For example, "*turn on the lights*", "*switch the lights on*" and "*lights on*" all correspond to the same intent. Another illustrative example is "*I need to practice my German, change the language*". Again for this dataset the amount of data per speaker varies, with some speakers having only very few recordings. The dataset is available online at https://www.fluent.ai/research/fluent-speech-commands/.

Table 2 summarises the number of utterances in the datasets. Apart from a comparison of the four basic models on all presented datasets, the later experiments will only focus on the GRABO and Fluent Speech Commands datasets.

3.3. Methodology

Our goal is to have a system that cannot only achieve a high accuracy, but also perform well with as few examples as possible to minimise user effort. To analyse this ability we measure learning curves that plot the performance in function of the number of training examples. To obtain robust results we use cross-validation. All data is first divided into blocks of about the same size and maximally similar semantic content. The blocks are composed by minimising the Jensen-Shannon divergence of the label frequency between blocks. An experiment is carried out by putting a random set of blocks in the training set and putting the rest in the test set. We create learning curves by putting an increasing number of blocks in the training set. For each number of training blocks, five experiments are carried out, each time with a different random training set. The resulting learning curve plots are slightly smoothed with a locally weighted smoothing method.

There will be two types of experiments in this paper. First there are speaker dependent experiments, where for every speaker a model is trained and a learning curve is created, i.e. by dividing all data from a single speaker into blocks and doing cross-validation. The resulting learning curve is then created as an average over all speakers. These experiments represent a setting where the system is used by one person only and visualise how many demonstrations a user has to give to obtain good performance on the set of intents he/she wants to cover. Because the Fluent Speech Commands

dataset has some speakers with very few recordings, the speaker dependent experiments only average over speakers with more than 100 utterances (this corresponds to about 80 percent of the speakers).

Secondly we carry out speaker independent experiments, where the data from all speakers is shuffled beforehand and the blocks are created from the entire dataset. Coping with multiple speakers is more challenging for the model because of acoustic speaker variability, but also because of differing lexical and grammatical choices to express the same intent. These experiments represent a setting where the system is shared by multiple people and allow to investigate the effect of speaker identification in the regularised model.

Finally the accuracy on the test set for the label classification task is measured with the f1-score. Hence we formulate the task as a detection problem: does the E2ESLU detect the user's intent to activate a label? For reconstruction of the words we use a similar metric to the f1-score, which we refer to as word-f1, where now the detected words take the place of the detected labels. For evaluation of the speaker identification capability, we calculate the percentage of correctly decoded speakers for the utterances in the test set.

3.4. Implementation

The encoder of all neural network models (ED, RCCN, PCCN) consists of a 2-layer bidirectional GRU with 256 units in each direction. The PCCN has 32 primary capsules with 16 neurons (i.e. each capsule has a vector of 16 dimensions), 16 place-coded capsules with 32 neurons and the output capsules have 16 neurons. The convolutional capsule layer has a kernel width of 9 timesteps and a stride of 2. The RCCN has 32 primary capsules with 64 neurons and the output capsules have 8 neurons. There is thus only one routing layer, and we use 3 routing iterations. The decoder network of the ED model consists of a max-pooling layer followed by a 1-layer feed-forward network with 1024 units. The input of the decoder is the output of the encoder.

The RCCN and ED models are trained using Adam optimisation [13] with the default TensorFlow parameters for 50 epochs with a batch size of 16 utterances. For the PCCN model, the Adam algorithm often needed a more conservative learning rate and ϵ for convergence. For the neural network models we use 40-dimensional MEL filter bank features + energy and for the NMF model we use 13 MFCC features. Delta and double delta features were included for all models.

The GMM in the NMF model has 100 components and diagonal covariance. The maximum number of iterations for the acoustic step is 100. For the HAC computation, delays 2-5-9-20 are used and 3 posteriors are kept. The fraction of extra words that is added to the dictionary to model non-topical or garbage words is 0.2. The number of training iterations for the NMF step is 100 and the number of decoding iterations is 30.

The considered neural network models were implemented using Tensorflow and Python and the code is available on Github: `https://github.com/vrenkens/assist`. The implementation for the regularised capsule networks is available at `https://github.com/JakobPoncelet/assist_reg`.

4. SLU Performance Without Multitask Learning

4.1. Speaker Dependent

We start with a performance comparison of the four SLU models: RCCN, PCCN, NMF and ED. The models are tested in a speaker dependent setting on the different datasets when the learning objective is only the max-margin label loss (see Eq. 6). The accuracy of the SLU systems is measured using the label f1-score and is plotted in Figure 4 as a function of the total number of training examples for all intents.

For the GRABO dataset, NMF is the best performing system when very little training data is available (about one example for every label). The PCCN and the RCCN perform equally well. The 'knee'-point in the learning curve occurs around 180 training examples, which amounts to about five examples per label. The 'knee' in the plots can be explained by the fact that easy and similar commands are learned very quickly, which is visible from the steep initial slope of the

learning curves. Some utterances might not be very consistent with the rest and the model needs more examples to correctly classify the intent, hence the curves saturate. Because for GRABO the sentences are quite fixed and the specified commands are not very difficult, the networks learn very fast and the accuracy quickly goes to almost 100 percent. Since this is the case for all systems, this indicates that this task is probably too easy, so conclusions about performance comparison should be made carefully.

The results on the DOMOTICA-3 dataset are similar to the GRABO dataset, with NMF leading at the very start, but being outperformed by RCCN with more data. Since we are dealing with dysarthric speech, the overall accuracy is lower. Here we notice that among the capsule networks, the PCCN has more trouble with the task.

The performance of all SLU systems on the PATCOR dataset is much worse than on the other two. This is because PATCOR is a more challenging dataset. Users were very free to choose their own wordings and often used several ways of naming the cards. The state of the game was also available to the user, so they often underspecified their commands, because the unspecified labels were obvious from the state of the game (for example the colour of specific cards that are already on the table or screen). This was however not available to the SLU systems, which obviously causes errors. The differences between the SLU systems is also most clear on the PATCOR dataset. RCCN significantly outperforms the other systems. In a card game the order of words in a command is important, which can be encoded in the recurrent encoder networks of RCCN, PCCN and ED networks. Since NMF follows a bag-of-words approach, its underperformance is not surprising.

Finally on the most challenging and realistic Fluent Speech Commands dataset, the RCCN is at all times the best performing system. While NMF has a relatively good accuracy when very few examples are available, the performance with more data becomes inferior. The PCCN and ED model have a similar performance. Again the RCCN performs better than the PCCN.

4.2. Speaker Independent

In this section we compare the models in a speaker independent setting on GRABO and the Fluent Speech Commands dataset. A speaker independent setting means that we are training a model with shuffled data from multiple speakers. As the results will show, this has a negative effect on the learning speed, because acoustic variation between speakers needs to be learnt as well. Furthermore different speakers can choose their own preferred phrasings, as in the Fluent Speech Commands dataset, which challenges the system even more. The results of the experiments on both datasets are shown in Figure 5.

First of all, the results on the GRABO dataset demonstrate the power of neural network approaches, and especially the capsule networks, with the RCCN being the best performing network, followed by the PCCN. The NMF model has great difficulties with extracting the acoustic patterns and mapping them to the correct semantic pattern when information from different speakers is introduced. The capsule networks can better incorporate the extra input variability than the NMF models. In order to reach an accuracy that is close to perfect, a little over 1000 examples are required. With 11 speakers and 33 unique intents, this corresponds to about three demonstrations per label for every speaker.

The Fluent Speech Commands dataset is a much more challenging, but also much larger dataset than GRABO, with around 100 speakers. The results are comparable to the previous experiment. NMF cannot handle the large variety in data and saturates at an f1-score below 80%. Both capsule networks and the ED model show a good performance, and again the curve of the RCCN starts higher.

4.3. Conclusion of the Base SLU Experiments

We can conclude that for the speaker dependent experiments in almost all cases the RCCN is the best performing system. The PCCN never outperforms the RCCN and often performs significantly worse, indicating that the RCCN is more appropriate for this task. For the more simple tasks NMF works well and is probably sufficient in some scenarios. However when the tasks get more challenging like e.g. PATCOR and Fluent Speech Commands, the capsule network is the better approach. The performance of the ED model is mediocre for some datasets when working speaker dependent and always inferior to the RCCN.

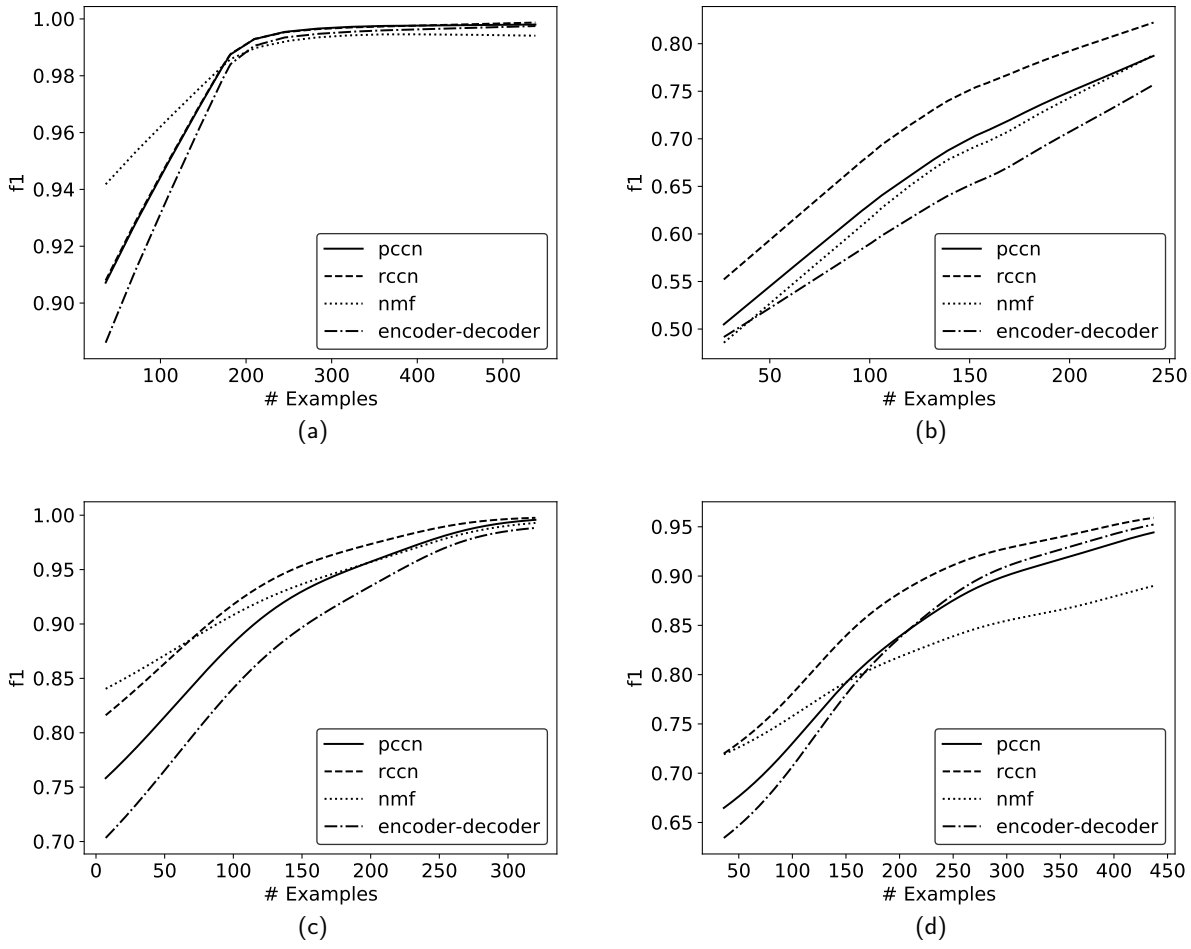

Figure 4: The label recognition f1-score of all considered SLU systems plotted in function of the number of example utterances in the training set for `GRABO` (a), `PATCOR` (b), `DOMOTICA-3` (c) and `Fluent Speech Commands` (d), in a speaker dependent setting.

The speaker independent experiments have confirmed these observations. Especially NMF has difficulties to cope with the speaker variation across the data. The ED can better deal with larger amounts of data and has less issues than NMF. The rate-coded capsule network is again the best performing system, possibly because it is expected to model longer units in its capsules than the place-coded capsule network. The improvement of adding dynamically routed capsules to the network instead of using a regular encoder-decoder architecture is verified on all experiments.

5. Analysis and Performance of the Regularised Capsule Network

The previous section focused on the performance of the SLU systems. The RCCN architecture turned out to be the best performing capsule network. In this section we will look at regularised versions of the RCCN and PCCN architectures. The goal of these regularisations is to extract more information from the capsules than just performing intent recognition, as well as possibly improving the performance by jointly learning multiple useful tasks. The regularised model therefore introduces speaker identification into the capsule properties, as well as word reconstruction from unordered transcripts.

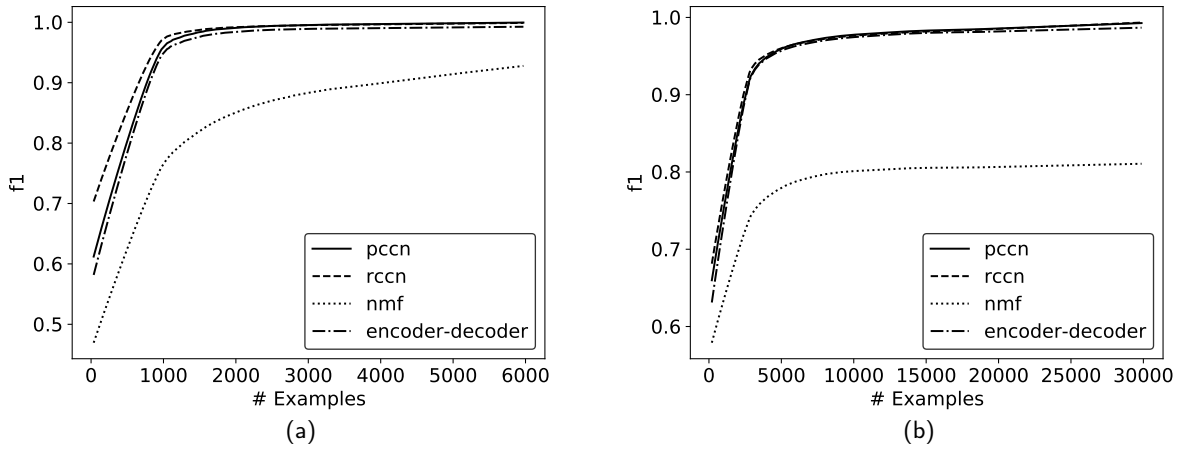

Figure 5: The label recognition f1-score of all SLU systems in function of the number of example utterances in the training set for `GRABO` (a) and `Fluent Speech Commands` (b), in a speaker independent setting.

In the experiments in this section we will look at the results of the regularised network in speaker dependent and speaker independent settings, and we will refer to the regularised models as RCCN-multi and PCCN-multi (because we're learning multiple tasks), and compare them to the basic capsule models. The dimension of the output capsules will be set to 8. The regularisation weights λ_s and λ_w are chosen as 1 and 10 respectively, to balance both regularisation loss terms in magnitude to the label recognition loss. The higher we choose these weights, the better the decoding capability of words and speakers, but there will be a penalty in intent recognition performance (cfr. Figure 2 in [18]). The chosen values are based on this trade-off. The decoding threshold for the detected words is set to 0.8, which we observed to be slightly better and more robust than a threshold of 0.5 when a lot of training examples are available to the system.

We will first analyse the effect of the dimension of the output vectors and look at the effect of the added regularisations. Then the SLU performance of the regularised models will be compared to the basic capsule networks, and we will examine their performance on speaker and word recognition. Finally we will make a comparison of all models in terms of SLU performance on a benchmark and in terms of complexity of the neural networks.

5.1. Dimension Analysis

An important parameter is the dimension of the activation vectors of the output capsules. These activation vectors are mapped to detected labels. First of all we examine what dimension for the output capsules the basic capsule network needs to classify labels. The network might be overparameterised. Lowering the dimension of the output vectors is an easy way to reduce the number of parameters. Figure 6 shows a comparison of the performance on the `GRABO` dataset of basic RCCN models (without regularisation) that have a different dimension for the output capsules, in a speaker dependent setting.

When the output dimension is lowered from 8 to 2, there is no drop in performance. All the necessary information needed to learn the task labels can be presented in two dimensions. There is hardly any structure present and it seems that the length or norm of the capsule vectors is much more important than their orientation.

As the results in Figure 6 confirm, the label classifying task does not explicitly require the capsules to use the orientation of the activation vectors and it is probably just easier and faster for the capsules to work only with the length of the vectors. Looking at the loss function, this makes sense because classification is entirely based on the vector norm. This leaves the capsules free to do as they like and the task is apparently easy enough to store all required information in two dimensions.

In a similar way as [22] applied a reconstruction loss regularisation term to the capsule network to encourage the

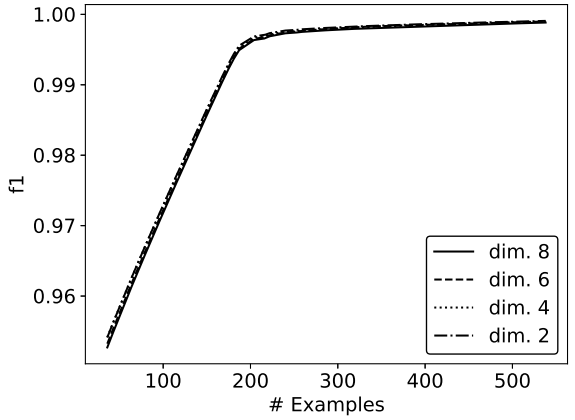

Figure 6: Label recognition f1-score comparison of RCCN models with different number of dimensions in the output capsule vectors on GRABO, in a speaker dependent setting.

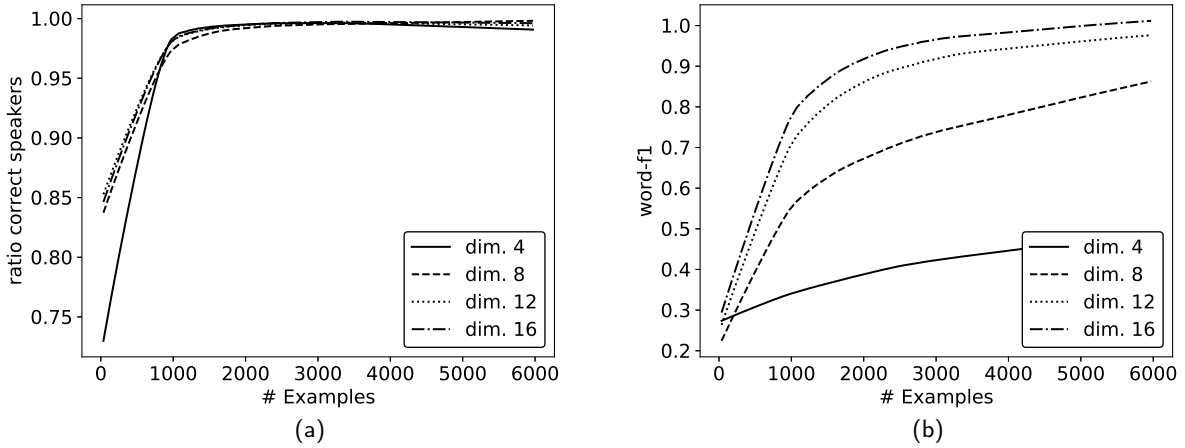

Figure 7: Comparison of RCCN-multi models with different output capsule vector dimension on GRABO in a speaker independent setting, with (a) showing the percentage of correctly decoded speakers and (b) the word-f1 score, in function of the number of utterances seen during training.

capsules to encode digit properties into its dimensions, we demand with the regularised model that the output capsules contain information about speaker identity and the words spoken in the utterance. Therefore we will analyse if the regularised model effectively makes use of a higher dimensionality.

Figure 7 shows the speaker and word detection performance for different output capsule dimensions on the GRABO dataset, using the regularised RCCN-multi model. The experiments were done in a speaker independent setting, since speaker identification otherwise doesn't make much sense. It is clear that the capsules require a higher dimensional space to encode all the necessary information to reliably detect the spoken words and speakers. The effect is bigger with the word reconstruction because there are more words than speakers in the database, so this is a more difficult task to learn. On top of that, it's harder to differentiate between all capsules after averaging when the dimension is low. We can conclude that the additional tasks have encouraged the output capsules to give more importance to the orientation of the activation vectors and the capsules are able to learn additional information.

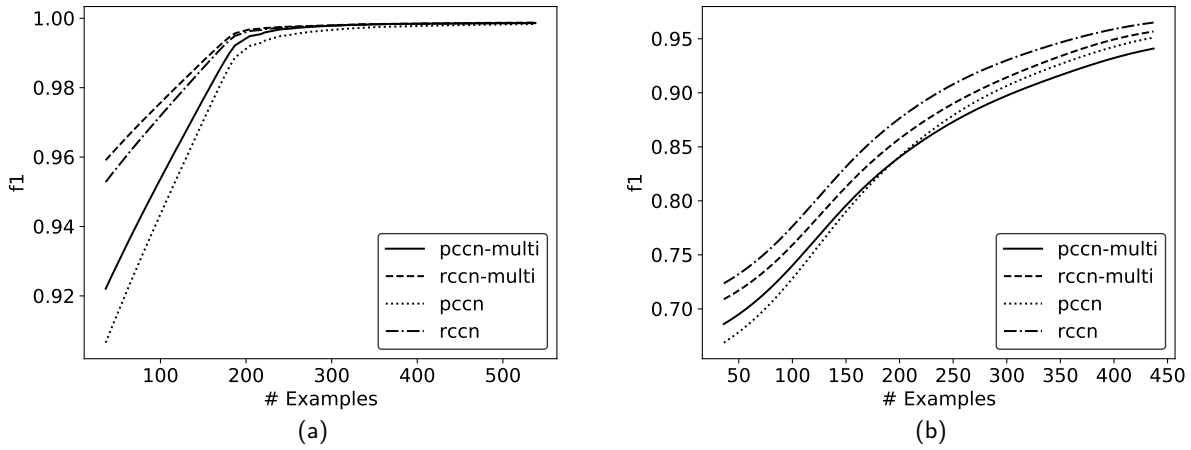

Figure 8: The label recognition f1-score of the regularised and basic capsule networks in function of the number of example utterances in the training set for `GRABO` (a) and `Fluent Speech Commands` (b), in a speaker dependent setting.

5.2. Speaker Dependent Experiments

The results in Figure 8 show that for the `GRABO` dataset in a speaker dependent setting, the multitask learning presumably helps the organisation of the network, resulting in a better f1-score in the intent label detection task. This is true across the board for RCCN and PCCN, and in particular when few training samples are available. However for the `Fluent Speech Commands` dataset, this is only the case for PCCN. The RCCN performs best, followed by RCCN-multi. It is therefore hard to draw strong conclusions.

An explanation might be that the sentences in `GRABO` are very structured, specific about the task and consistent in its words, whereas for `Fluent Speech Commands` lengthy phrasings and variations can occur. This observation is also visible from the word reconstruction performance shown in Figure 9, where the prediction accuracies for `GRABO` are higher. Note that increasing the output dimension from 8 to for example 16 might also significantly boost these word-f1 curves upwards.

The results lead us to believe that the reconstruction of the words can be helpful, but remember that with the (unordered) transcripts we're giving more information to the system than in the original intent recognition task. And even when it is not helpful, it doesn't hurt the performance much either. The fact that the model is successfully able to extract this additional information from the output capsules without having a detrimental effect on the performance is a positive indicator for its versatility to encode even richer information, opening pathways towards reconstructing the full spectrogram.

5.3. Speaker Independent Experiments

Figure 10 shows the results of the speaker independent experiments. For both `GRABO` and the `Fluent Speech Commands` dataset, learning the additional tasks of speaker and word recognition seems to have a small negative impact on the learning speed. Especially at the start of the curve, the f1-score is lower for the regularised capsule network models compared to their basic models. The RCCN is again the best performing architecture. The intent classification rather quickly reaches near-perfect performance. The word reconstruction on the other hand is more difficult when data from multiple speakers is fed to the system, as Figure 12 shows.

Figure 11 reveals that speaker recognition on `GRABO` quickly reaches a good performance. The regularised PCCN and RCCN models perform comparably in this aspect. On the `Fluent Speech Commands` dataset, we observe an advantage in asymptotic performance for RCCN, which is in line with the intent recognition results.

In these experiments, the f1 intent recognition performance did not seem to benefit from learning to classify the speak-

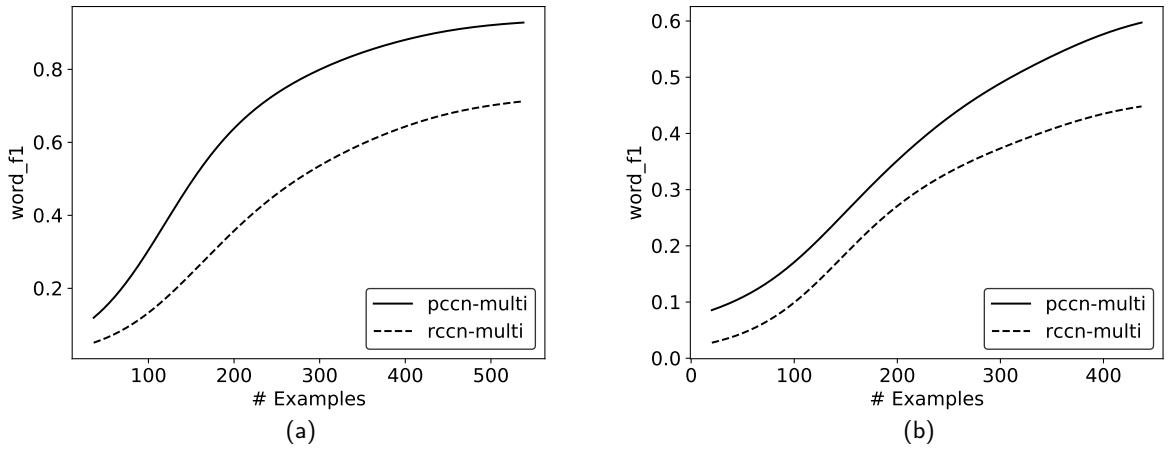

Figure 9: The word recognition f1-score for `GRABO` (a) and `Fluent Speech Commands` (b), in a speaker dependent setting, with an output dimension of 8.

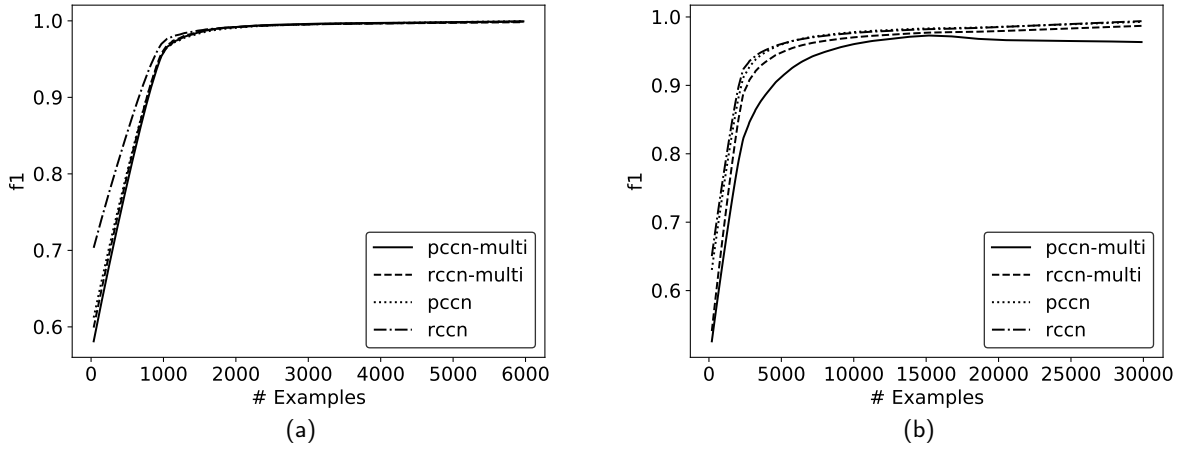

Figure 10: The label recognition f1-score of the regularised and basic capsule networks in function of the number of example utterances in the training set for `GRABO` (a) and `Fluent Speech Commands` (b), in a speaker independent setting.

ers and spoken words. One explanation is that the intent labelling task in itself is easier than the additional tasks, and learning other tasks leads to a trade-off, since we're adding terms to the loss function. However, apart from the nice theoretical possibilities, we refer to [18] in which the regularised model only contained speaker recognition, and this improved the asymptotical performance on the `Fluent Speech Commands` dataset. In our case deriving the spoken words from the output capsules seemed to have been a too large constraint, especially with an output dimension of only 8.

5.4. Benchmarking on Fluent Speech Commands

In Table 3 we compare all the models on a benchmark on the `Fluent Speech Commands` dataset. The models are compared to the model from [14] without pre-training, so that we can use the same data and train-test split. Every model is evaluated on the test set, first after training on only a part of the dataset (randomly extracted 10%, 2313 utterances), and second after training on the full dataset reserved for training (i.e. 23133 utterances). For consistency we adopt the accuracy metric as defined in [14], where the intent is only classified as correct if the prediction of all

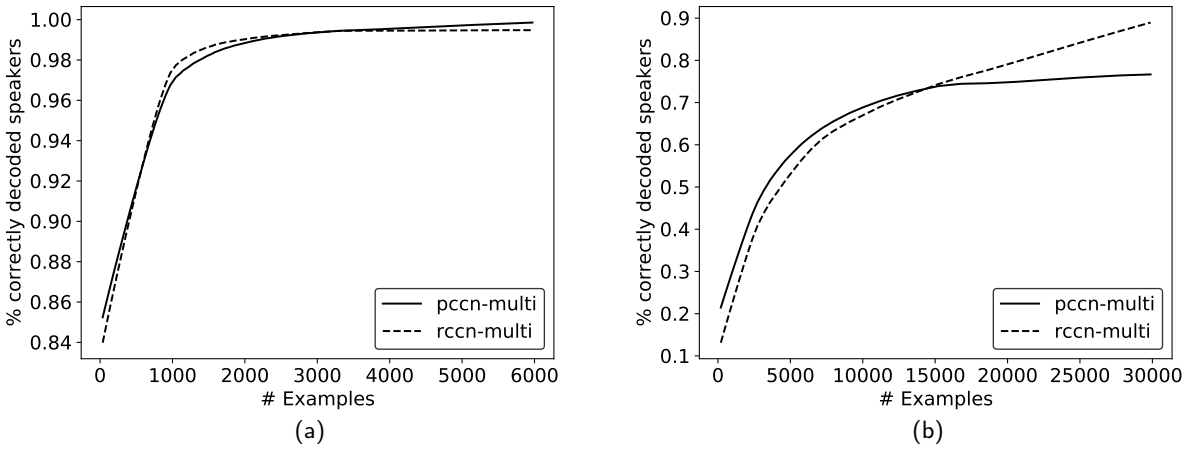

Figure 11: The speaker recognition accuracy for GRABO (a) and Fluent Speech Commands (b), in a speaker independent setting, with an output dimension of 8.

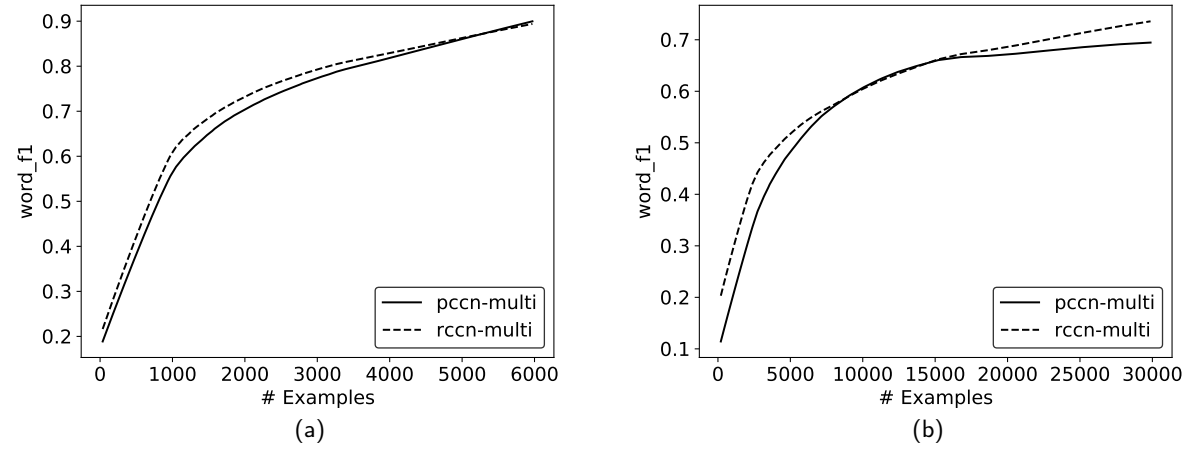

Figure 12: The word recognition f1-score for GRABO (a) and Fluent Speech Commands (b), in a speaker independent setting.

slots exactly corresponds to all slots in the true intent.

For NMF, the model order can be augmented by including multiple high-dimensional vectors per label instead of one. The idea is explained in Section 3.1. In the table this is referred to as NMF with 8 columns. The architecture of the model used in [14], which we will call *Fluent model*, is also explained in Section 3.1.

5.5. Number of Parameters

Finally we will give an overview of the number of parameters for the basic and regularised capsule networks, compared to the encoder-decoder model. This is required to get a view of what we're actually comparing and the complexity of the neural networks. The numbers in Table 4 come from experiments on the GRABO database. The decoder of the ED model consists of a max-pooling layer and a fully connected layer, so the number of parameters in the decoder is limited. The vast majority of the parameters comes from the encoder in the models and all could benefit from a more efficient encoder. We observe that the number of parameters in the capsule network is low, since the total amount in

Table 3
Benchmark accuracies on the `Fluent Speech Commands` dataset (speaker independent).

Model	Partial dataset	Full dataset
NMF (1 column per label)	60.09%	62.02%
NMF (8 columns per label)	68.95%	76.99%
ED	91.23%	92.89%
RCCN	**94.03%**	**98.92%**
PCCN	**92.19%**	**99.16%**
RCCN-multi	87.59%	97.76%
PCCN-multi	79.23%	97.38%
Fluent model without pre-training	88.90%	96.60%

Table 4
Comparison of the number of parameters among the SLU systems on the `GRABO` dataset.

Model	# Parameters
ED	2,324,001
RCCN	2,356,289
RCCN-multi	2,357,477
PCCN	4,522,512
PCCN-multi	4,523,700

the RCCN is only slightly higher than in the ED model. The capsule network is lightweight and there is an accuracy improvement over the ED model (as in Table 3). This confirms that using capsule layers to get a fixed-size output vector instead of standard feedforward layers can be beneficial in neural networks for SLU.

5.6. Conclusion of the Regularised SLU Experiments

The regularised model is able to identify the speaker of the command and the uttered words. The activation vectors encode this information in their orientation, which is confirmed by the improvement of using higher dimensional vectors. However, in general the intent recognition performance of the regularised models is not better than the basic capsule network architectures. The additional tasks are not easy to learn and lead to a trade-off in the loss function.

The RCCN is in all cases the best performing model and the capsule networks score the highest on the benchmark for the `Fluent Speech Commands` dataset, which represents a realistic command-and-control setting. The learning speed and classification accuracy are high enough for practical implementations.

6. Alignment Analysis

The supervision information for the SLU systems is a bag of labels. The order in which the labels are mentioned in the input utterance is not available to the system. We want to investigate whether the capsule networks are able to figure out where in the input utterance a label is specified. We therefore look at how each timestep contributes to each output label, or in other words how the output labels are aligned to the input sequence.

Because the part-whole relationships in the capsule network are linear, we can track the contributions of capsules to capsules deeper in the network. We will present how the contributions can be tracked for a single capsule layer and then expand it to multiple layers.

A capsule layer consists of voting, dynamic routing and a squashing function. The higher capsules can be written as:

$$\boldsymbol{c}_j^{l+1} = s_j^{l+1} \boldsymbol{x}_j^{l+1} \tag{14}$$

$\boldsymbol{c}_j^{l+1}$ is the j^{th} capsule at layer $l+1$. s_j^{l+1} is the factor by which it is squashed and is equal to:

$$s_j^{l+1} = \frac{||\boldsymbol{x}_j^{l+1}||^2}{1 + ||\boldsymbol{x}_j^{l+1}||^2} \frac{1}{||\boldsymbol{x}_j^{l+1}||} \tag{15}$$

$\boldsymbol{x}_j^{l+1}$ is the raw capsule before squashing. It is a linear combination of the votes from the lower capsule:

$$\boldsymbol{x}_j^{l+1} = \sum_i w_{ij}^{l+1} \boldsymbol{v}_{ij}^{l+1} \tag{16}$$

w_{ij}^{l+1} results from the dynamic routing and is the routing weight from the i^{th} capsule in layer l to the j^{th} capsule in layer $l+1$. $\boldsymbol{v}_{ij}^{l+1}$ is the corresponding vote and is a linear mapping of the i^{th} capsule in layer l:

$$\boldsymbol{v}_{ij}^{l+1} = \boldsymbol{M}_{ij}^{l+1} \boldsymbol{c}_i^l \tag{17}$$

$\boldsymbol{M}_{ij}^{l+1}$ is the linear mapping from capsule i to capsule j. Putting it all together the higher capsule can be computed as:

$$\boldsymbol{c}_j^{l+1} = \sum_i s_j^{l+1} w_{ij}^{l+1} \boldsymbol{M}_{ij}^{l+1} \boldsymbol{c}_i^l \tag{18}$$

and we can define the transformation to compute the contribution of the i^{th} capsule in layer l to the j^{th} capsule in layer $l+1$ as:

$$\boldsymbol{T}_{ij}^{l,l+1} = s_j^{l+1} w_{ij}^{l+1} \boldsymbol{M}_{ij}^{l+1} \tag{19}$$

To compute the contributions over multiple layers we can just combine these transformations:

$$\boldsymbol{T}_{kj}^{l-1,l+1} = \sum_i \boldsymbol{T}_{ki}^{l-1,l} \boldsymbol{T}_{ij}^{l,l+1} \tag{20}$$

Using the transformation matrix we can decompose a capsule deep into the network as contributions from capsules earlier in the network:

$$\boldsymbol{c}_j = \sum_i \boldsymbol{p}_{ij} \tag{21}$$

$\boldsymbol{p}_{ij}$ is the contribution of capsule i to capsule j. We can compute the (squared) contribution to the probability that the capsule is present by computing the squared norm:

$$||\boldsymbol{c}_j||^2 = \boldsymbol{c}_j \cdot \boldsymbol{c}_j = \sum_i \boldsymbol{p}_{ij} \cdot \boldsymbol{c}_j \tag{22}$$

$\boldsymbol{p}_{ij} \cdot \boldsymbol{c}_j$ is how much capsule i contributed to activating capsule j. We compute the alignments by computing how much each primary capsule contributed to activating the output capsule. For place-coded capsule networks, primary capsules have a *place*, i.e. a time index, so plotting the contribution of index i to capsule j (Eq. 22) yields the alignment plot. For the RCCN we additionally use Eq. 8 to compute how each timestep contributes to the primary capsules.

The alignments for a command from three datasets - `GRABO`, `DOMOTICA-3` and `PATCOR` - for both the RCCN (top) and PCCN (bottom) are shown in Figure 13. The alignments for the PCCN and the RCCN look very similar. The number of timesteps for the PCCN is half of those in the RCCN due to the stride in the convolutional capsule layer. From the alignment for the `GRABO` and `DOMOTICA-3` datasets it is clear that the output capsules are activated in the same order as their corresponding labels are mentioned in the input utterance. The picture for the `PATCOR` dataset is however not as clear. The labels for the card values are activated together. This is probably caused by the fact that a card is almost always moved to a card with a value of exactly one higher. If 'six' is mentioned in the beginning as the value of the card to be moved, the value of the target card is almost certainly going to be 'seven'. Hence the mentioning of 'six' contributes to the label corresponding to 'seven' and vice-versa. There are however exceptions to this rule and we observed that the system often made errors in this case.

7. Conclusion

In this paper we discussed design challenges in a Spoken Language Understanding system for command-and-control applications. In a traditional multi-step SLU system, the ASR is language dependent and works poorly for people with speech impairment. The NLU has to cope with changing contexts, varying task specification and is application dependent. We proposed to have the user teach the spoken commands to an end-to-end SLU system to make this design process easier and make the interaction more natural.

We presented two types of capsule networks for this task, because they are believed to be data efficient. In the Place-Coded Capsule Network, the location of the primary capsules encodes time and a subsequent convolutional capsule layer is used. The Rate-Coded Capsule Network has rate-coded primary capsules and uses attention and a distributor. Both models are tested and compared to NMF, which is a technique previously used for this task, and an encoder-decoder approach which was proposed earlier for end-to-end SLU. Except for a few exceptions the RCCN proved to be the best performing system in terms of intent classification, and the efficiency is promising for practical applications. Even though it is never specified to the system in which order the labels are mentioned in the input utterance, the capsule networks were able to figure out the correct order, which is quite important for certain tasks.

Apart from a performance comparison of the proposed capsule networks, we also investigated what information can be extracted from the output capsules. We extended the model by mapping activation vectors of the output capsules to probabilities of the identity of the speaker and the words that were spoken. By jointly training on these tasks, the output capsules are forced to encode additional properties about the input speech into the orientation of their activation vectors. These regularisations in some cases had a small negative impact on the learning speed of the models, but open doors for designers and show promising theoretical properties of the capsules. The ability to reconstruct the words spoken in a command (from transcripts) leads us to expect that a reconstruction of the speech signal from the output capsule content is feasible. As argued in the introduction, reconstruction as a regularisation would improve the robustness and performance of the capsule network for all kinds of tasks.

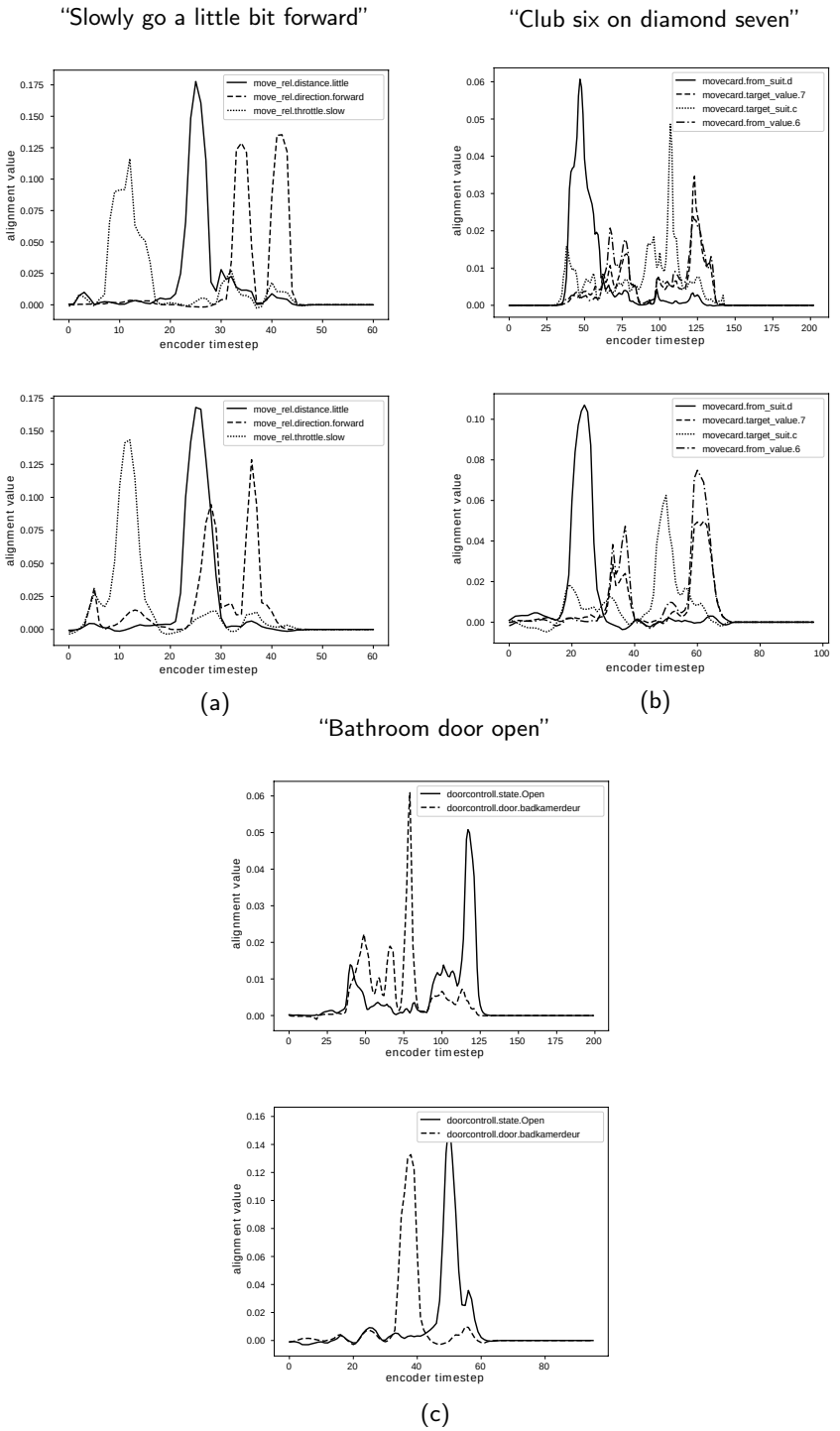

Figure 13: Alignments from datasets GRABO (a), PATCOR (b) and DOMOTICA-3 (c). The top alignments come from the RCCN and the bottom ones from the PCCN. The transcriptions are translated to English, but the concept order is retained, and are written above the figure. The figures show the contribution of each timestep to the squared norm of the output capsules, as in Eq. 22.

8. Future Work

The proposed SLU system relies on user interaction for training, hence the model should learn as fast as possible, and the accuracy should be high, both when little or a lot of data is available to the system. When looking at the number of parameters to train, over 80 percent of the parameters in the RCCN come from the encoder part of the network and not from the capsule layers. Pre-training the encoder on more generic tasks might help in some applications, but for the use case of disordered voices this is not easily achieved. As pointed out earlier, the routing-by-agreement shows similarities with transformers using self-attention [29], but so far we have not succeeded in obtaining better learning curves with them.

The capsule layers represent part-whole relationships between entities in capsules from different layers. Adding layers to the network could lead to an internal representation of grammar or sentences, and this information might be extracted in the same way from the capsules. Still, more research is required, and in particular more challenging databases are needed to assess these properties.

We have shown how the output capsules can be forced to propagate relevant variation in the speech utterances into the orientation of their activation vectors. A single layer network suffices for the speaker and word prediction tasks, however an additional hidden layer might improve their capacity. In [22], Hinton et al. explored a reconstruction of handwritten digit images from the output capsules after masking the inactive capsules, which led to an increased robustness and performance. In our case the ultimate goal would be to perform a complete reconstruction of the speech signal or its spectrogram. A difficulty to overcome w.r.t. the image processing problem of [22] is the variable length input data. Also, averaging over output capsules might be too detrimental to reconstruct something as detailed as a spectrogram, hence it might be needed to explore other constructs that are similar to the proposed average capsule, like stacking the output capsules. Due to the promising results in this paper, where both speaker identity and transcript words are successfully reconstructed from the output capsules, and the alignment analysis has shown that the order of labels can be induced by the capsules, we believe that a complete reconstruction would be possible and should be investigated. An intermediate step would be to reconstruct the envelope of the energy curve. From that point on, the spectrogram is not far away.

Acknowledgements

This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme and was funded by PhD grant 151014 of the Research Foundation Flanders (FWO).

References

[1] Bastianelli, E., Castellucci, G., Croce, D., Basili, R., Nardi, D., 2017. Structured learning for spoken language understanding in human-robot interaction. The International Journal of Robotics Research 36, 660–683. doi:10.1177/0278364917691112.

[2] Chorowski, J.K., Bahdanau, D., Serdyuk, D., Cho, K., Bengio, Y., 2015. Attention-based models for speech recognition, in: Advances in neural information processing systems, pp. 577–585.

[3] Christensen, H., Cunningham, S., Fox, C., Green, P., Hain, T., 2012. A comparative study of adaptive, automatic recognition of disordered speech, in: Thirteenth Annual Conference of the International Speech Communication Association.

[4] De Mori, R., Bechet, F., Hakkani-Tur, D., McTear, M., Riccardi, G., Tur, G., 2008. Spoken language understanding. IEEE Signal Processing Magazine 25.

[5] Desot, T., Portet, F., Vacher, M., 2019. Towards end-to-end spoken intent recognition in smart home, in: Proceedings of the 10th International Conference on Speech Technology and Human-Computer Dialogue (SpeD).

[6] Desot, T., Raimondo, S., Mishakova, A., Portet, F., Vacher, M., 2018. Towards a French smart-home voice command corpus: Design and nlu experiments, in: 21st International Conference on Text, Speech and Dialogue (TSD), Springer. pp. 509–517. doi:10.1007/978-3-030-00794-2_55.

[7] Dinarelli, M., Tellier, I., 2016. Improving recurrent neural networks for sequence labelling. arXiv:1606.02555 .

[8] Gemmeke, J., Ons, B., Tessema, N.M., Van hamme, H., Van de Loo, J., De Pauw, G., Daelemans, W., Huyghe, J., Derboven, J., Vuegen, L., Van Den Broeck, B., et al., 2013. Self-taught assistive vocal interfaces: An overview of the ALADIN project, in: Proceedings Interspeech 2013, ISCA. pp. 2038–2043.

[9] Gemmeke, J.F., Sehgal, S., Cunningham, S., Van hamme, H., 2014. Dysarthric vocal interfaces with minimal training data, in: Spoken Language Technology Workshop (SLT), IEEE. pp. 248–253.

[10] Haghani, P., Narayanan, A., Bacchiani, M., Chuang, G., Gaur, N., Moreno, P., Prabhavalkar, R., Qu, Z., Waters, A., 2018. From audio to semantics: Approaches to end-to-end spoken language understanding. arXiv:1809.09190.

[11] Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al., 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine 29, 82–97.

[12] Hinton, G., Sabour, S., Frosst, N., 2018. Matrix capsules with EM routing. URL: https://openreview.net/pdf?id=HJWLfGWRb.

[13] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv:1412.6980 .

[14] Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V.S., Bengio, Y., 2019. Speech model pre-training for end-to-end spoken language understanding, in: Interspeech. arXiv:1904.03670.

[15] Mishakova, A., Portet, F., Desot, T., Vacher, M., 2019. Learning natural language understanding systems from unaligned labels for voice command in smart homes, in: The 1st International Workshop on Pervasive Computing and Spoken Dialogue Systems Technology (PerDial).

[16] Mnih, A., Kavukcuoglu, K., 2013. Learning word embeddings efficiently with noise-contrastive estimation, in: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, p. 2265–2273.

[17] Ons, B., Gemmeke, J.F., Van hamme, H., 2014. The self-taught vocal interface. EURASIP Journal on Audio, Speech, and Music Processing 2014, 43.

[18] Poncelet, J., Van hamme, H., 2020. Multitask learning with capsule networks for speech-to-intent applications, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8494–8498. doi:10.1109/ICASSP40776.2020.9053832.

[19] Raymond, C., Riccardi, G., 2007. Generative and discriminative algorithms for spoken language understanding, in: Eighth Annual Conference of the International Speech Communication Association.

[20] Renkens, V., Janssens, S., Ons, B., Gemmeke, J.F., Van hamme, H., 2014. Acquisition of ordinal words using weakly supervised nmf, in: Spoken Language Technology Workshop (SLT), IEEE. pp. 30–35.

[21] Renkens, V., Van hamme, H., 2018. Capsule networks for low resource spoken language understanding arXiv:1805.02922.

[22] Sabour, S., Frosst, N., Hinton, G.E., 2017. Dynamic routing between capsules, in: Advances in Neural Information Processing Systems, pp. 3859–3869.

[23] Serdyuk, D., Wang, Y., Fuegen, C., Kumar, A., Liu, B., Bengio, Y., 2018. Towards end-to-end spoken language understanding. arXiv preprint arXiv:1802.08395 .

[24] Sugita, Y., Tani, J., 2003. A holistic approach to compositional semantics: A connectionist model and robot experiments., in: Advances in Neural Information Processing Systems.

[25] Tessema, N.M., Ons, B., van de Loo, J., Gemmeke, J., De Pauw, G., Daelemans, W., Van hamme, H., 2013. Metadata for corpora patcor and domotica-2 .

[26] Tomashenko, N., Caubrière, A., Estève, Y., 2019a. Investigating adaptation and transfer learning for end-to-end spoken language understanding from speech, in: Interspeech. doi:10.21437/Interspeech.2019-2158.

[27] Tomashenko, N., Caubrière, A., Estève, Y., Laurent, A., Morin, E., 2019b. Recent advances in end-to-end spoken language understanding, in: Statistical Language and Speech Processing (SLSP). doi:10.1007/978-3-030-31372-2_4.

[28] Van hamme, H., 2008. Hac-models: A novel approach to continuous speech recognition, in: Ninth Annual Conference of the International Speech Communication Association.

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I., 2017. Attention is all you need, in: Advances in Neural Information Processing Systems 30, pp. 5998–6008.

[30] Wang, Y.Y., Deng, L., Acero, A., 2005. Spoken language understanding. IEEE Signal Processing Magazine 22, 16–31.

[31] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., Zweig, G., 2016. Achieving human parity in conversational speech recognition. CoRR arXiv:1610.05256.

[32] Zhang, C., Li, Y., Du, N., Fan, W., Yu, P., 2019. Joint slot filling and intent detection via capsule neural networks, Association for Computational Linguistics (ACL). pp. 5259–5267. doi:10.18653/v1/P19-1519.