Karlsruhe Institute of Technology

Carnegie Mellon University

Creating Audio Level Dependency Parse Trees from Speech

Bachelor's Thesis

Theo Beffart

at the Interactive Systems Lab
Department of Informatics

Reviewer:	Prof. Dr. Alexander Waibel
Secondary Reviewer:	Prof. Dr. Tamim Asfour
Advisor at KIT:	Dr. Sebastian Stüker
Advisor at CMU:	Prof. Dr. Florian Metze

Bearbeitungszeit: 22. Februar 2018 – 22. Juni 2018

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 22. Juni 2018

Abstract

In this work we explore the feasibility of creating dependency parse trees over audio in an end-to-end fashion. These trees assert relations over segments of audio without transforming it into a textual representation. Because this occurs before any speech recognition would these trees could for example be used within a speech recognition system alongside the language model or in applications that do not need the actual transcript of the words that are said but rather their relations with each other. The model is constructed by adapting an existing parser that loosely resembles an encoder-decoder architecture. The presented models replace its encoder and show reasonable performance when word boundaries are given as an additional input. Some future possibilities are also discussed and preliminary experiments for combining it with an attention based sequence to sequence model that suffices without word boundaries are also presented and discussed. The ideas and concepts that are developed in this work are a proof of concept for this kind of speech representation and offer a foundation upon which to build more sophisticated solutions.

Zusammenfassung

In dieser Arbeit beschäftigen wir uns mit der Möglichkeit Dependency Parsebäume Ende-zu-Ende direkt über Audio zu extrahieren. Diese Bäume repräsentieren Relation zwischen Segmenten von Audio ohne dieses in eine textuelle Repräsentation zu überführen. Das Modell wird konstruiert indem eine existierende Parser-Architektur, die grob einem Encoder-Decoder Ansatz ähnelt, adaptiert wird. Die vorgestellten Modelle ersetzen dessen Encoder und zeigen akzeptable Ergebnisse solange die Wortgrenzen dem Modell als zusätzliche Eingabe vorliegen. Einige zukünftige Möglichkeiten werden ebenfalls angesprochen. Vorläufige Experimente zum vereinen der vorgestellten Modelle mit einem attention-based Sequence-to-Sequence Modell, welches ohne explizite Wortgrenzen auskommt, werden auch gezeigt. Die Ideen und Konzepte, die hier entwickelt werden sind ein Nachweis über die grundsätzliche Machbarkeit dieser Art der Sprachrepräsentation und bieten eine Grundlage auf deren Basis komplexere Lösungen erarbeitet werden können.

Acknowledgements

This work would not have been possible without the the interACT program and the support of the people at the interACT lab at Carnegie Mellon.
For creating the program, I would like to thank Alexander Waibel who in doing so made my experiences of the last months possible.
At CMU I want to thank my advisor Florian Metze for countless discussions with me about my work no matter how busy he was and for always providing new and helpful ideas.
I want to thank Ramon Sanabria who provided me with everything surrounding feature extraction and who, even though being always stressed and busy with his own work, always found the time to discuss the latest developments of my research.
A big "Thank you!" to Shruti Palaskar for her help with all things sequence to sequence and for always providing insightful discussions.
I also want to thank Sebastian Stüker, my advisor at KIT .
Lastly I want to thank Emanuel Jöbstl whose judging glare has kept me focused and productive throughout the months.

Contents

List of Figures

1. Introduction

With recent, ongoing rise in popularity of speech driven systems like home assistants or voice controls in cars and phones, the importance of methods to extract meaning from speech is greater than ever.
Parse trees are an established way to represent the syntactic structure of a sentence in a way that is accessible for usage by other applications. These applications can use the wealth of information those trees provide to extract higher level meaning from the sentence.
Dependency Parses are a category of such representations where the entire syntactic structure is expressed as relations between the words of the sentence . These are labeled to describe the nature of the relationship they stand for.
Parsing traditionally happens with text as input, not audio. To parse speech it is first transcribed using an speech recognizer and that transcript is then parsed. However, recently end-to-end approaches have become increasingly popular in other language processing problems. This motivates the question if such an approach is also applicable to the generation of parse trees from speech.

The goal of this thesis is to examine the feasibility of producing dependency parse trees directly from audio without creating an explicit representation of the words in the sentence such as text. These trees are created on the audio level, that is, they assert relations between segments of audio that correspond to word, not between the words as text. This is different from the "pipeline" approach which uses a transcription of speech because there the result contains the words as text.
Creating these trees *before* any speech recognition takes place changes the use cases for such a system. It could, for example, be used as a part of a speech recognition system that provides additional information alongside the language model about the expected syntactic structure of the sentence that is being recognized. Some application might not even need the actual content of a sentence and instead only require knowledge of how the different parts of it depend on each other. Such a system could work directly with the audio parse trees.
Independent of possible uses, this representation is a novel way to process speech in the context of parsing.

1.1 Outline

In chapter 2 relevant concepts that are used throughout this work are introduced.
Chapter 3 discusses related work in relation to the presented approaches.
Chapter 4 gives an introduction to dependency parsing and the approach to it that is used throughout this work. The implementation on which the approaches in this work are based upon is also presented in this chapter.
Chapter 5 presents the models developed in this work and the decision made in the process.
Chapter 6 describes implementation details, data and experiment results.
Chapter 7 offers a conclusion of the presented results and an outlook of the next steps that can be made from what was presented here.

2. Background

2.1 Neural Networks

During recent years neural networks have seen a surge in popularity in different machine learning tasks. For example, neural networks are able to produce state of the art results in Image Processing ([KrSH12]) and in multiple natural language processing tasks such as Machine Translation ([BaCB14]), speech recognition ([MiGM15]) and parsing ([BDGS17]).
In any Machine Learning task in the most general sense we seek a model to approximate some unknown function $f^*(x)$. A neural network can be understood as parametrized function $f(x, \theta)$[GoBC16]. The goal is to find parameters θ such that $f(x, \theta) \approx f^*(x)$ for all inputs x.
As f^* is unknown, we rely on optimizing the network with respect its behavior on some training data for which the desired output is known. This data contains samples $(x_i, y_i = f^*(x))$ of an input and its corresponding known output. This section will introduce the neural network architectures relevant to this work - the functions $f(x, \theta)$ - and how to optimize their parameters θ.

2.1.1 Multilayer Perceptron

Multilayer Perceptrons are powerful models that have been proven to be able to (under certain general assumptions) approximate any function (Universal approximation theorem[Horn91]). As the name suggests, multilayer perceptrons consist of layers of neurons that are stacked on top of each other. Each layer contains a fixed number of neurons. Originally inspired by their biological counterpart, such computing units were first described in [Rose58]. Each neuron processes the input vector by first calculating a weighted sum over it and then applying some activation function to this sum. Typically the units in one layer all share the same activation function. Therefore, instead of considering the neurons as independent units, in a fully connected Multilayer Perceptron each layer can be represented by a matrix-vector multiplication and subsequent application of the activation function. In a Multilayer Perceptron the input is passed through all layers and the final layer outputs the network output.

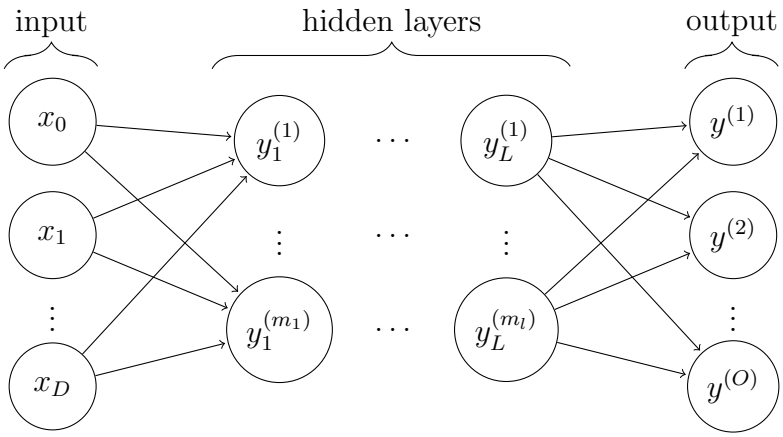


Figure 2.1: Multilayer Perceptron with L hidden layers of size m_l, input dimension D and output size O.

In other words the output of the network is the function concatenation of each layer's individual function f_k.

$$\begin{aligned} y(x, \theta) &= f_n \circ f_{n-1} \cdots \circ f_1(x) \\ y_k &= f_k(y_{k-1}) = \varphi(W_k y_{k-1} + b_k) \\ y_0 &= x \end{aligned}$$

The weight matrices W_k and biases b_k of each layer f_k are the parameters θ that are learned to make the network function approximate the desired output function. The activation function φ usually does not contain any learned parameters and is chosen during the design of the network topology.

2.1.2 Training with Stochastic Gradient Descent and Backpropagation

Gradient Descent

Gradient Descent is an optimization method to minimize a function $F(x)$, that is to find $x^* = \arg\min F(x)$. This is done by iteratively taking a step x_i according to the gradient $\nabla F(x_i)$ and a learning rate γ, so that

$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

In the context of machine learning and neural networks this function F is a loss or cost function associated with the network output. This function is chosen with the intuition that minimizing this loss also leads to $f(x, \theta)$ being a good approximation of $f^*(x)$. For each sample from the training data, the loss depends on the corresponding label and the network output. The output depends on the input and the parameters

θ. We use the gradient with respect to θ because we want to modify θ to minimize L given the training data.

$$L(\theta) = \sum_{i=1}^{n} L(y(x_i, \theta), y_i)$$
$$\nabla L(\theta) = \sum_{i=1}^{n} \nabla L(y(x_i, \theta), y_i)$$

After all samples have been seen the parameters are updated according to accumulated gradient of all samples. An alternative to this approach is Stochastic Gradient descent, the most widely used optimization procedure for neural networks. Traditional gradient descent calculates ∇L over the entire training data and then updates θ. Stochastic gradient descent approximates the gradient over the entire data using the gradient over a single sample or a very small subset of the training data. The update rule becomes

$$\theta_t = \theta_{t-1} - \gamma \nabla L(y(x_i, \theta_{t_1}), y_i)$$

This approximation adds noise to the gradient because the loss functions for each sample or minibatch differ slightly from each other. This noise necessitates that we reduce the learning rate over time, because as the θ approaches a minimum the gradient becomes smaller but the noise does not vanish. Reducing the rate over time ensures that the algorithm can converge[GoBC16]. [WiMa03] argues that this approach introduces some significant benefits. Stochastic Gradient descent tends to converge faster especially as the size of training set grows. By design Gradient Descent is unable to escape saddle points. There the gradient of L is zero and therefore the parameters θ will not update in future iterations. Because of the noise in the loss function SGD can eventually escape from such a point.

Backpropagation

Backpropagation is an algorithm to efficiently calculate the gradient $\nabla L(\theta)$ of the loss function with respect to the individual weights in θ of a neural network. For each weight vector w_{ki} in layer i the partial derivative of the loss function with respect to that weight needs to be computed. We recall that a multilayer neural network can be expressed as the concatenation of the layer functions f_i. Therefore the chain rule can be invoked to get each of those derivatives.

$$\frac{\delta L}{\delta w_{ki}} = \frac{\delta L}{\delta y} \frac{\delta y}{\delta w_{ik}}$$
$$\frac{\delta y}{\delta w_{ki}} = \frac{\delta f_n}{\delta y_{n-1}} \frac{\delta f_{n-1}}{\delta y_{n-2}} \cdots \frac{\delta f_{i+1}}{\delta y_i} \frac{\delta f_i}{\delta w_{ki}}$$

This can be thought of as the error signal flowing backwards through the network starting from the output layer. The partial derivatives $\frac{\delta f_i}{\delta y_{i-1}}$ need only be calculated once and can then be stored for reuse thus making this an efficient way of calculating the derivative with respect to each weight.

2.1.3 Activation Functions

There is a wide variety of choices for the activation function φ at the output of each neuron or layer. This section introduces some commonly used functions that are

present in the models used in this work. Activation functions that are used in the hidden layers share the property of being non-linear because a multilayer perceptron with linear activations can be expressed as a single linear transformation and would therefore lose its universal approximation capabilities. For gradient descent to work they also have to be differentiable as the gradient cannot be computed otherwise.

Sigmoidal Activations

Sigmoidal activations are differentiable approximations of the sign function

$$sgn(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

It is necessary to use an approximation here because the derivative of $sgn(x)$ is zero wherever it is differentiable. This makes it inappropriate to be used with gradient descent. However, the binary nature of the output is useful and intuitive[KaKw92]. Sigmoidal activation functions are smooth and differentiable during the transition from negative to positive arguments.

Logistic Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Modeling the behavior of a step function, the logistic sigmoid function was one of the earliest activation functions used in neural networks with backpropagation. A notable feature is the easy to calculate derivative that can be obtained with minimum extra effort.
However, it has fallen out of favor due to its performance in neural networks with a large number of layers. The cause for this is the saturation for values outside of the transition zone around zero, which makes training inefficient and hard[GlBe10]. It still has use if the desired output range is $(0, 1)$

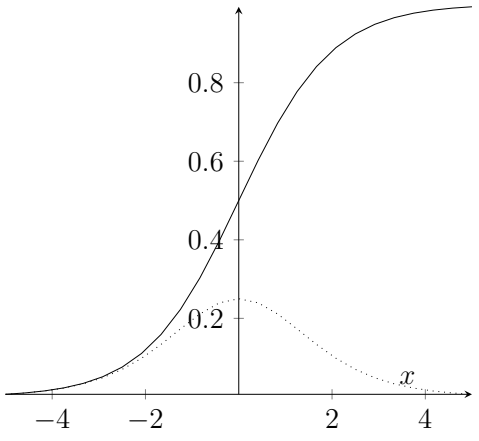


Figure 2.2: logistic activation and derivative

Tangens Hyberbolicus

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\tanh'(x) = 1 - \tanh^2(x)$$

The tanh function is essentially the sign function with a differentiable transition from -1 to 1. [KaKw92] argues that the tanh function is the best sigmoidal activation function for use in multilayer neural networks with backpropagation.

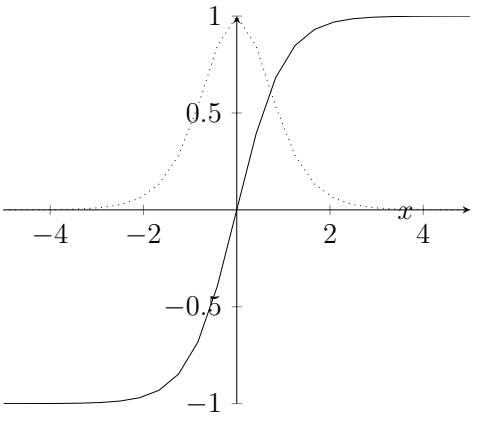


Figure 2.3: tanh activation and derivative

Rectified Linear Unit

$$ReLU(x) = \max\{0, x\}$$

The Rectified Linear Unit is currently considered the best choice as an activation function in the hidden layers of multilayer networks [LeBH15].ReLUs improve results compared to the above sigmoidal activations [MaHN13]. An additional benefit is that the ReLU and its derivative are trivially easy to compute.

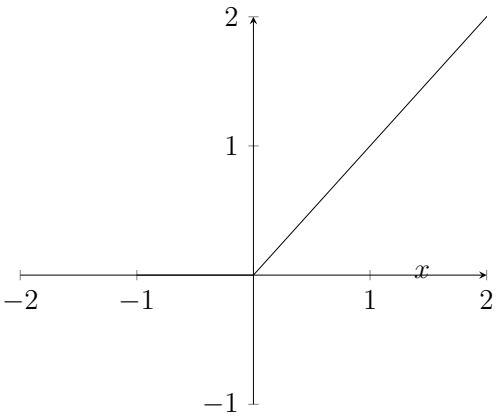


Figure 2.4: rectified linear activation

2.1.3.1 Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

The softmax activation function is often used in the last layer of a network that implements a classifier. In contrast to most other activation functions, each output depends on all other outputs. The softmax scales the activations to be between zero and one while preserving the relative order of each output. Another important property is that all outputs sum up to be equal to one, that is the softmax activation can be interpreted as a probability distribution.If softmax is used as an activation in a layer that projects its input to a an output size that is equal to the number of classes, then the softmax an be viewed as a probability distribution over those classes[Brid90].

2.1.4 Recurrent Neural Networks

Recurrent Neural Networks deal with inputs that are sequential in nature. This can be a sentence, a sequence of audio or data points taken at different times -

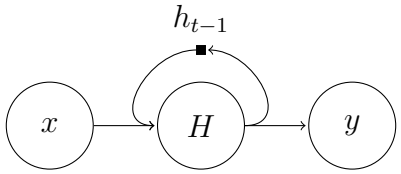


Figure 2.5: Simple RNN: hidden layers H get previous hidden output h_{t-1} as additional input alongside x to produce output y.

any kind of sequential input where each timestep is dependent on its predecessors. When processing a sequence, a multilayer perceptron would have to learn independent representations for every step of the input sequence. It would not be able to generalize across different positions in the input. Recurrent Networks share their parameters across every step of a variable length sequence[[GoBC16] p.367]. That is, at every time step the network uses the same learned weights to process the input. This reduces the amount of parameters that need to be learned and allows the network to recognize structures in the input, regardless of their position in the input sequence. To process its input as a sequence where each time step depends on past inputs there has to be a way for the current state to influence the future states. The input at t has to be connected to the future inputs, starting at $t+1$. This relation between current and future inputs is modeled as recurrent connections within the network. These connections link the output of a hidden layer to future outputs by feeding that output back to its input with a 1-time-step delay.

Consider a simple RNN with a single hidden layer, where the output of the network is used as the input of the next step. Let the input to that RNN be a sequence of single inputs $x^{(t)} = x^{(1)}, x^{(2)}, ..., x^{(T)}$. The hidden state $h^{(t)}$ (in this case also the output) at time step t is

$$h^{(t)} = \varphi(W_h h^{(t-1)} + W_i x^{(t)} + b)$$

where W_h and W_i are weight matrices of the recurrent connection and hidden layer respectively. This can be generalized to arbitrarily deep RNNs by stacking multiple hidden layers on top of each other. In such a scenario the network works similar to a deep multilayer perceptron with the addition that each layer also gets it previous state as input.
By learning W_h during training, the network learns how to treat its past states and how they should influence the current output. While in theory $h^{(T)}$ depends on all $x^{(t)}, t \leq T$ it has been shown that the short term dependencies dominate the long term and that learning these long term dependencies is computationally very expensive. This is due to the problem of "vanishing gradient", where due to the t-fold composition of the network's function the gradient becomes very small (or arbitrarily big). If it explodes the training diverges and if it vanishes the updates regarding the long term dependencies are very minor and overshadowed by the short term signals[Hoch98].

2.1.5 Long Short Term Memory

Long Short Term Memories, first introduced in [HoSc97a], are a specialized form of Recurrent Neural Network that are designed to deal with the vanishing gradient problem. In place of the simple recurrent connection, LSTMs introduce a more

complex structure to persist information across time steps.
The core idea is to store and manage the recurrent state of the network within a *memory block*. In this block the cell state vector c_t stores the state of the LSTM at time step t. The only true recurrent connection of the LSTM is a self loop on that cell vector. The activation function along this connection is the identity function $id(x) = x$ with $\frac{\delta}{\delta x} id(x) = 1$. This avoids the exploding and vanishing gradient problems according to [HoSc97a].
At each time step the cell state is updated with the input to the memory block and the previous cell state. The output h_t of the LSTM is based on the application of an activation function φ to c_t. [HoSc97a] shows that while a simple identity self connection may solve the vanishing gradient problem, it lacks representational power. To mitigate this they introduce additional units to control the flow of information within the memory block. These gating units output a value between 0 and 1 which is then multiplied with a signal, allowing for selective suppression of information flow. Each of those gates are implemented as a neural network with a logistic function as activation. Their activation depends on the current input of the block x_t and its last output h_{t-1}. [GeSS02] also connects the last cell state c_{t-1} to these gates, which improves performance.
There are three gating units in a LSTM memory block: The input gate i_t on the connection from the input to the cell state controls how much of the current input is considered, the forget gate f_t determines how much the previous cell state influences the next and the output gate o_t on the output signal of the block. For the LSTM models in this work the input and forget gates are coupled to always be complementary to each other. Formally defined the gating units are:

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \\
o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_{t-1} + b_o) \\
f_t &= 1 - i_t
\end{aligned}
$$

With these the output hidden state h_t and cell state c_t are defined as

$$
\begin{aligned}
c_t &= f_t \odot c_{t-1} + i_t \odot \varphi(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\
h_t &= o_t \odot \varphi(c_t)
\end{aligned}
$$

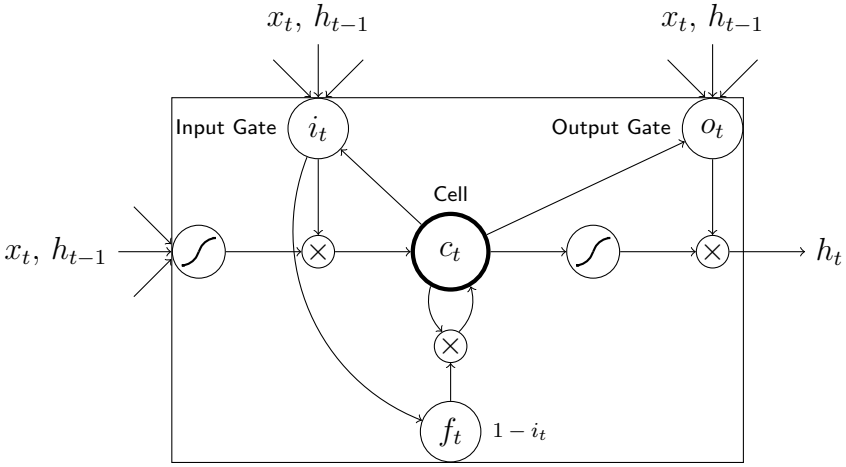


Figure 2.6: LSTM computation graph as defined above. Input- and Output-Gate control the information flow within the memory block. The Forget-Gate is coupled to the input gate.

W_{lm} are weight matrices that connect different parts of the LSTM.
An extension of the LSTM architecture are Bidirectional LSTMs. These consist of two LSTM networks where, as the name suggests, one processes the input sequence in order while the other network processes the input in reverse. The combined output of these models allows the model to consider past and future context [ScPa97].

2.1.6 Stack LSTM

Stack LSTM are a variation on LSTMs that is designed to provide a representation of a stack data structure. To do so, it has to break away from the strict sequentiality of inputs assumed by traditional LSTM and allow an input order that resembles the pushing and popping of elements to and from a stack. As the stack operation `pop()` removes the most recently added element, the Stack LSTM has to reflect this by removing that element's influence on future inputs and the current state. The main augmentation of the Stack LSTM compared to tradional LSTM that allows it to do this is the addition of a *stack pointer* $p_{\text{TOP}} = (c_{\text{TOP}}, h_{\text{TOP}})$[BDGS17]. This pointer stores a reference to the top of the stack LSTM: the hidden state and cell to be used as h_{t-1} and c_{t-1} at the next input.
When an element is `push`ed onto the stack, one time step is made in the LSTM using that element as input and the elements from the stack pointer as the previous state. After this operation the stack pointer is updated to point to the now most recent hidden state. To pop() an element from the stack the LSTM has to forget about its current top element. This is done by resetting the stack pointer to its predecessor. That way the popped element is not taken into consideration in the next push operation. Additionally the stack pointer provides easy access to a representation

of the entire stack content.

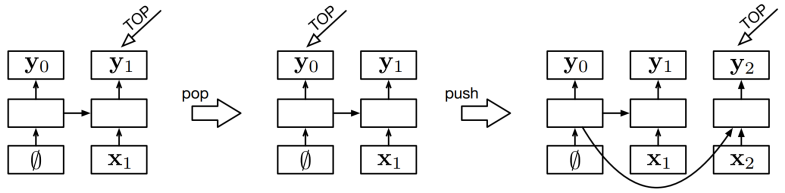


Figure 2.7: Shows how pop and push change the information flow in a stack LSTM to be nonsequential. Notice how y_2 is not influenced by y_1 because it was popped from the stack. Figure from [DBLM+15a]

2.2 Sequence to Sequence Models

Sequence to Sequence (Seq2Seq) models map an input sequence to an output sequence: their training goal is to maximize the probability $p(y_1 \dots y_{T\prime}|x_1 \dots x_t)t)$ of an output sequence given the input. While any Recurrent Neural Network does this due to the nature of their design, their usability is limited in cases where the sequences have different lengths or where there are complex dependencies between input and output that are non-monotonic. Sequence2Sequence models work by splitting the task into encoding and decoding. First the input sequence is encoded and a fixed size representation of the input is created. This encoding is used as the intial state of the decoder. Starting from this state, the decoder emits the output sequence. At each time step the previous emission is fed back into the decoder as the input for that step. Encoder and Decoder are usually implemented as LSTMs that either process the input sequence in the encoder or take the previous predictions as input sequence in the decoder[SuVL14a].

Attention

While traditional sequence to sequence is in theory capable of producing the output sequence just from the encoding vector alone, the model can be improved by allowing the decoder to look at the input while decoding. When attention is added to the model the next prediction depends on both the previous state and the attended input encoding. The latter is obtained by calculating a weighted sum over all encoder states, where the weights represent some measure of importance of the respective state for the next prediction. These weights are usually calculated as a learned function of encoder state, last decoder state and sometimes the previous attention vector [CBSC+15].

3. Related Work

3.1 Dependency Parsing

Dependency Parsing of text is a well examined problem for which multiple ways of approaching exist. The two shown here illustrate the wide ranges of angles of approach that can be chosen to solve this problem.

3.1.1 Non-projective Dependency Parsing using Spanning Tree Algorithms

[MPRH05] operates directly and explicitly on the structure we seek to find by interpreting dependency parsing as a graph problem. This is done by representing the sentence as a directed graph. Each word has an edge to each other word and a special root node is connected to all words. Each edge represents a possible dependency relation between the word nodes it connects and is assigned a weight based on some learned score of this relation. Finding the most likely parse tree is now equivalent to finding a spanning tree of maximum weight within the graph.
In contrast to transition based dependency parsing, non-projective trees (parse trees with crossing edges) can be handled without any further measures.

3.1.2 Transition Based Dependency Parsing using Stack LSTM

[DBLM+15a], which serves as the foundation for the models presented in this work, is an implementation of a transition based parsing scheme. The sentence is processed by manipulating its words on a stack and an input buffer. The dependency relations are asserted one by one while the parser consumes the words from the stack. At the end of the parsing process the parse tree is constructed by combining the found relations. There is a set of valid transition from one parse state to another that either combine elements from the stack or add move a word from the buffer to the stack. Parsing comes down to making the optimal transition given the parser state. Here a neural network makes that decision based on a state that is tracked by representing stack and buffer as stack LSTMs. An advantage of the transition based approach is the linear run time that results from every action consuming a

word from either buffer or stack and therefore limiting the amount of transitions that can be made. Breaking dependency parsing down to a classification problem or one of emitting a target label sequence makes this also nicely suited to be solved with a fully neural network based approach. This implementation will be examined in detail in the next chapter 4.4

3.2 Joint Modeling of Text and Acoustic-Prosodic Cues for Neural Parsing

[TTBG+17] is an encoder-decoder based parser that incorporated audio features into parsing. The encoder combines word embeddings generated from text with acoustic features that are extracted directly from audio using neural networks. The key difference between this model and the models presented in this work, is that here audio features are used alongside text to augment parsing performance on transcribed audio, whereas our goal is to create a system that forgoes any explicit representation of the input as text and only operates on the audio.
Introducing audio into the parser has been shown to improve performance, however using audio as the sole input to the parsing system is a novel and unexplored approach.

3.3 Towards End-to-End Spoken Language Understanding

[SWFK+18] describes the attempt to create a system for domain and intent classification of speech, that does not explicitly use a text representation. Traditionally in that kind of task the speech is first transcribed by an automatic speech recognition framework. That textual representation is then given to the next layer of that system and the problem is solved based on the text. In [SWFK+18] an encoding is directly produced from audio features. A bidirectional LSTM with multiple layers compute an encoding of audio features that are passed frame by frame. That output is then used by a hidden layer with softmax output to produce a classification result for the task at hand. They show that this works but lacks behind traditional systems in performance. This approach and problem bears similarity to this work. Both try to solve a problem where traditionally an intermediate representation of speech as text is required, without explicitly modeling such a representation.

4. Dependency Parsing

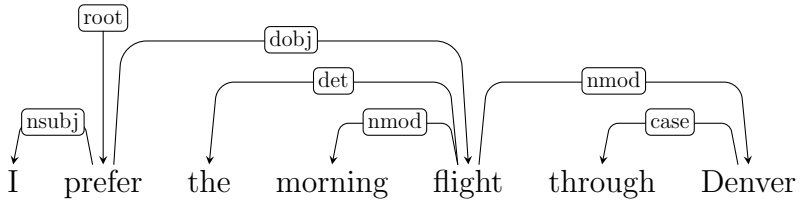


Figure 4.1: Dependency parse for "I prefer the morning flight through Denver."

In a dependency parse, the syntactic structure of a sentence is described only by means of the binary relations between its words. They provide a simple structure that is easier to learn and represent as other parse formalisms but still contains most of the necessary information[MPRH05].

4.1 Formalism

A dependency parse T of a sentence $W = \langle w_1, \ldots, w_n \rangle$ is a directed acyclic graph $P = (V, E)$ with the words of the sentence $V = \{w | w \in W\}$ as vertex set and its dependency relations $DepRel(W)$ captured within the edge set. In a typed dependency structure the arcs of this graph are additionally labeled with the type of relation they represent. The labels are drawn from a fixed set of grammatical relations $\mathscr{L}$, where $l(u \to v) \in \mathscr{L}$ is the label of the relation from u to v.

$$P = \{(u, v, l(u \to v)) \in V \times V \times \mathscr{L} \mid (u, v) \in DepRel(W)\}$$

The resulting graph is a tree usually rooted in the verb of the sentence.
4.1 illustrates how the tree structure reflects the binary nature of the grammatical relations. Following the notation from above, (*"prefer"* $\to$ *"I"*, NSUBJ) $\in DepRel(W)$ would hold for this example.

4.2 Stanford Dependencies

The dependency relations used in this work are the Stanford Dependencies put forth by [DMMa08]. One of the main design goals is to provide a intuitive and

easy to understand framework of dependency relations that still capture everything as binary relations. In an effort to keep it easier to understand relation types are modeled after traditional notions from grammar (such as subject, object etc.).

Relation	**Description**	**Example**: *head* → **dependent**
`nsubj`	nominal subject	**I** *went* home.
`dobj`	direct object	She *gave* me a **raise**.
`amod`	adjectival modifier	These are **harsh** *words*.
`conj`	conjunct	Bill is *big* and **honest**.
`neg`	negation	I *might* **not** attend.

Figure 4.2: Some examples of dependency relations described in [DMMa08]

4.2 shows some examples of relations from the Stanford Dependencies.
In addition to the *basic* dependencies where each word of the sentence is represented by a node in the parse tree, there are *collapsed* dependencies where words that represent a relation (such as prepositions) are omitted from the node set and instead used as a specifier on the binary relation itself.

Basic Dependency *Collapsed Dependency*

Figure 4.3: Preposition "in" is removed from the parse tree and instead becomes part of the relation label.

In this work the *basic* Stanford Dependencies are used because they are better suited to a parsing architecture that only operates on two words at a time.

4.3 Transition Based Dependency Parsing

The parsing architectures used throughout this work all implement a transition based parsing scheme which is loosely based on the shift-reduce parsing approach used by [AhUl72]. Such a transition based parser consists of an input buffer and a stack. During parsing, elements from the buffer are placed on the stack and manipulated there. A parser *configuration* consists of the stack, the buffer and the set of dependency relations discovered for that sentence.
Such a configuration is a triple (S, B, A) from the space of valid configurations $\mathscr{C}$, where

- $S = \langle s_1, \ldots, s_k \rangle$ is the stack where words are processed
- $B = \langle b_1, \ldots, b_n \rangle$ is the input buffer where the unprocessed words reside
- $A \in \{(u \to v, l) | u, v \in W, l \in \mathscr{L}\}$ is the set of relations discovered so far

Parsing can therefore be understood as finding a sequence of valid configuration transitions from a starting configuration to a terminal configuration. In the initial configuration C_0 the buffer contains the entire sentence and the stack only holds

the special ROOT symbol that marks the root of the dependency tree. The set of discovered relations is empty.

$$C_0 = (\langle \text{ROOT} \rangle, \langle w_1, \ldots, w_n \rangle, \emptyset)$$

A terminal configuration C_T is any configuration where stack and buffer are empty. The set of relations A now contains all dependency relations for the sentence.

$$C_T = (\langle\rangle, \langle\rangle, DepRel(W))$$

The following transition operators are used by the approaches in this work to transition form one valid state to another. This operator system is known as the arc-standard transition system (with a swap operation)[Nivr09].

LEFTARC$_l$ asserts the relation $s_1 \rightarrow s_2$ with label l

$$(\langle \sigma | s_2, s_1 \rangle, B, A) \rightsquigarrow (\langle \sigma | s_1 \rangle, B, \{(s_1 \rightarrow s_2, l)\} \cup A), \text{ if } s_2 \neq \text{ROOT}$$

RIGHTARC$_l$ asserts the relation $s_2 \rightarrow s_1$ with label l

$$(\langle \sigma | s_2, s_1 \rangle, B, A) \rightsquigarrow (\langle \sigma | s_2 \rangle, B, \{(s_2 \rightarrow s_1, l)\} \cup A)$$

SHIFT moves b_1 to the top of the stack

$$(\langle \sigma \rangle, \langle \beta | b_1 \rangle, A) \rightsquigarrow (\langle \sigma | b_1 \rangle, \langle \beta \rangle, A)$$

SWAP swaps the order of s_1 and s_2

$$(\langle \sigma | s_2, s_1 \rangle, \langle \beta \rangle, A) \rightsquigarrow (\langle \sigma | s_1 \rangle, \langle \beta | s_2 \rangle, A), \text{if ord}(s_2) < \text{ord}(s_1)$$

where $ord(w)$ is the position of w in the original sentence and σ, β describe the rest of the stack and buffer.

Based on the current configuration, an oracle makes the decision which transition to apply next. In an actual implementation of such a parser the oracle has to be approximated by some mechanism that makes the decision. For example, [BDGS17] train a neural network with the help of a predefined static oracle to predict the next action.
Such a reference oracle can be calculated using the following method if a reference parse tree is available for a given sentence (ie. obtainable from a treebank like [MaMS93])[Jura]:
Given configuration $c_t = (S, B, A)$ and reference parse tree $A_r = \text{DepRel}(W)$, the next action a_t is:

$$\begin{aligned}
\text{LEFTARC}_l \quad & \text{if } (s_1 \rightarrow s_2, l) \in A_r \\
\text{RIGHTARC}_l \quad & \text{if } (s_2 \rightarrow s_1, l) \in A_r \\
& \wedge \forall w \in W, l \in \mathscr{L} : (s_1 \rightarrow w, l) \in R_p \implies (s_1 \rightarrow w, l) \in A \\
\text{SHIFT} \quad & \text{else}
\end{aligned}$$

In other words, perform LEFTARC$_l$ whenever it asserts a correct relation. Only perform RIGHTARC$_l$ whenever it asserts a correct relation and all dependents of s_1 have already been assigned to it. If none of these conditions hold, SHIFT the next word from the buffer unto the stack.

With this procedure training data in the form of example transition sequences can be generated from a treebank of existing parse trees.

4.4 Implementation using Stack LSTMs

[BDGS17] ties together the above concepts to introduce a parsing architecture that produces competitive results. The resulting parser is an implementation of a transition based dependency parser. It utilizes a neural network to approximate the transition oracl that is trained to predict the next action given the current parser state. That state is computed using Stack LSTMs that offer continuous embeddings of the state of parser's core components. The input sentence is embedded using either learned word embeddings or a LSTM that processes each word character by character. Because this architecture serves as the basis for the explorations of this work we will discuss its components, the parser and the word embeddings, in detail.

4.4.1 Parser

The parser comprises of the Stack S, the Buffer B and a list of actions taken so far A. B and S are the stack and buffer in the sense of a parser configuration as defined above. The list of actions is very similar to the set of discovered relations as the latter can be derived from looking at the LEFTARC and RIGHTARC transitions in A. All of these components are implemented as Stack LSTMs to provide a representation of the parser state as well as actual stacks to manipulate the data. Fundamental to the architecture is a classifier that predicts the next appropriate action based on the current parser state. This parser state is obtained by combining the different state representations from the three Stack LSTMs through their top-pointer. The three state embeddings are then concatenated and fed through a Multilayer Perceptron with a ReLU activation to form an embedding of the entire parser state. This embedding contains information on all words currently on the stack and in the buffer, as well as the sequence of all actions taken so far. This state representation is

$$p_t = max\{0, W * [s_t; b_t; a_t]\}$$

where s_t, a_t, b_t are the top pointer of the stack LSTMs at time t.

Transition Prediction

A Multilayer Perceptron with a softmax activation is used to predict the most likely action out of all currently valid action label combinations, given the current parser state. This is possible because the set of actions is finite as the set of labels is finite. The size of the softmax layer is then $2|\mathscr{L}| + 2$ to account for SHIFT, SWAP and each RIGHTARC or LEFTARC combined with each label. Finding the next action is interpreted as a k-class classification problem . The probability for parser action a at time t is:

$$p(a_t|p_t) = \frac{e^{g_z^T + q_{qzi}}}{\sum_{\mathscr{A}(S,B)} e^{g_z^T + q_z}}$$

where $\mathscr{A}(S, B)$ is the set of valid transitions given the state of buffer and stack. p_t depends on all previous states and the actions taken so far, so the probability of a sequence of actions $\vec{a}$ given an input sentence w is then dependent on the probability of each action at each time step[BDGS17].

$$p(\vec{a}|w) = \prod_{t=1}^{|\vec{a}|} p(a_t|p_t)$$

The chosen action is executed by manipulating stack and buffer as described in 4.3. Additionally a learned embedding of the action is pushed onto the action history. The Stack LSTMs are updated alongside their non-neural counterparts to reflect the changes made to the parser state by action. This process is repeated until a terminal configuration is reached. Then the parse tree can be extracted from the action history.

Parse Tree Representation

During the parse process, the parse tree is built edge by edge. Every word on the stack or in the buffer represents a subtree of the complete parse tree that is rooted in that word. Words correspond to a tree with just a single node. A LEFTARC or RIGHTARC transition attaches the dependent d to the head h. d is removed from the stack but h now represents a subtree where d is one of its children. When h is attached in a future transition its head will represent a subtree that includes the one h was heading. The complete parse tree is built by iteratively attaching words to each other and thereby growing the subtrees they represent.

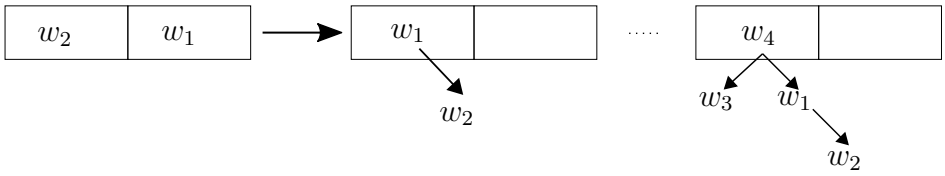


Figure 4.4: During each Arc transition the words on the stack are combined to form a tree that is a subtree of the total parse tree.

This means that as the parsing process proceeds, each element on the stack starts to represent more and more words and relations.[BDGS17] shows that it is beneficial to reflect this fact in the way the words are managed on the Stack LSTM. Instead of simply keeping the representation of the head, the parser calculates a new combined embedding of head, dependent and asserted relation. The resulting embedding contains information about the whole parse tree fragment it represents. This embedding then replaces the original embedding of the head on the stack. The composition embedding c of head h,dependent d and relation embedding r is

$$c = \tanh(W * [h; d; r] + b)$$

W is a weight matrix of appropriate dimension to project the embedding back into the same space as the original embedding of h.

4.4.2 Embedding

The stack LSTMs in the parser rely on meaningful word embeddings as a foundation for their ability to encode the parser state. These embeddings are computed before the parsing process starts and then fed to the Buffer Stack LSTM during set up for the initial configuration. Additionally they are kept as individual embeddings to be pushed onto the stack and then combined into tree representations as the parse process goes on. [BDGS17] discusses two different approaches to compute these embeddings.

Word Based

In the original implementation from [DBLM+15a] the embeddings are created at the word level. For each word from a fixed vocabulary an encoding is learned. Words from outside that vocabulary are encoded using a special Out-Of-Vocabulary encoding. The embeddings are combined from three different parts. The first part is a pretrained, preexisting embedding like the gloVe embeddings [PeSM14]. These are not optimized as part of the model training but they are already a meaningful representation of the word.
The second part is a learned embedding that is refined during training. For each word it is initialized randomly and then trained as part of parameters of the model. The pretrained word embeddings also provide a fall back embedding when the parser encounters a word it has not come across during training as these embeddings usually contain more words than the parse trees from the training set. Additionally the embedding is augmented with a POS Tag for the word if available.
The word embedding is a component wise rectifier, applied to an affine transformation of the concatenated word embeddings $\widetilde{w}$ and w and the POS tag t.

$$x = max\{0, V * [\widetilde{w}; w; t] + b\}$$

Character Based

In [BaDS15] the words are each processed letter by letter by a bidirectional LSTM. For each word, its embedding is the final hidden state of that LSTM after it has processed the sequence of characters for that word. The results of the forward and backward direction and optionally a POS tag are then combined. The embedding is again a rectifier applied to an affine transformation of the LSTM hidden states $\overrightarrow{w}$ and $\overleftarrow{w}$ and POS tag t.

$$x = max\{0, V * [\overrightarrow{w}; \overleftarrow{w}; t] + b\}$$

This approach has the upside that there are no words for which there is no embedding.A word that has not been seen during training can still be processed by the LSTM. The resulting embedding will be similar to that of a similar known word. The results from [BDGS17] show that this approach is especially preferable if there are no POS tags available. This is because a lot of the information that is in the POS tags is also deductible from the word ending. In the experiments in this work we assume that we do not have POS tags as input data.

4.4.3 Training

The final goal of the training is for the parser to be able to create accurate parse trees. The metrics for this accuracy are defined in the Data 6.1 section. However, these metrics are hard to express as a loss function to train the network on. Instead the model is optimized to make the correct transition at every step in the parsing process.
The core assumption behind this training objective is that optimizing the individual transitions will also optimize the resulting parse tree[BDGS17].
Given a configuration of stack, buffer and action history the parser has to determine which transition is the next in a sequence of transistions that lead to a good parse

tree. The parser is trained to maximize the probability of the correct parse action given the current state. This is done by using a Cross Entropy loss over the output of the softmax layer in the oracle. The training labels are a sequence of reference actions. These are generated from an existing parse tree using the algorithm described in 4.3.
To keep the model consistent with the reference sequence, the model always applies the reference action irrespective of the most likely prediction.

4.4.4 Interpretation as a Sequence to Sequence Model

This parser can also be looked at as a variation of a sequence to sequence model. It takes an input sequence of words, encodes them and then outputs a target sequence of parse actions. The encoder is the word embedding that processes the input words and puts them into the buffer. The initial state of the decoder is the state of the buffer LSTM after it has received all inputs. This corresponds to the initial configuration of the parser state. Starting from the initial configuration the parser outputs a sequence of parse actions. After each emission, it updates its internal state according to the action taken - similar to a decoder in a sequence to sequence model. The decoder additionally gets a new word embedding as an input whenever a SHIFT operation puts a new word onto the stack. This context dependent referencing of a specific part of the input encoding is functionally similar to what attention does in sequence to sequence models. One key difference however, is the requirement for distinct encodings of each individual words. This is different from encoding the entire input into the initial decoder state.

5. Design

Our goal is to create a parsing architecture that can extract parse trees directly from audio. Thus asserting semantic and syntactic relations between segments in the input audio in the form of a parse tree. The distinguishing characteristic is that these relations are available *before* any transcription or word recognition occurs. The trees are asserted on the audio and their nodes do not contain decoded words but rather segments of audio. Therefore these trees can be used on a lower level in the speech recognition process. Another example that shows promising results on another language understanding task is [SWFK+18]. The following approaches explore if something like that is also feasible here.

5.1 Parse Input Encoder

In 4.4.4 we observed that the model can be thought of as an encoder-decoder architecture. [DBLM+15a] and [BaDS15] show that this decoder can work with different kind of encoders, given that they provide a meaningful representation of the encoded word. Instead of reworking the entier architectuer, it seems reasonable to keep the elaborate, proven to work design of the decoder and only modify the encoder in a way that enables it to encode audio such that it suits the decoder.
Following this line of thought we replace the character based word embeddings from [BDGS17] with an audio encoder. The parser architecture remains unmodified and will be trained to decode the embeddings provided by the audio encoder.
Bidirectional LSTMs have proven to be good models for encoding audio in speech recognition. Both [GrJM13] and [MiGM15] use bidirectional LSTM to encode the incoming audio features. [SWFK+18] also uses a multi layer bidirectional LSTM to do the audio encoding before extracting higher level understanding from it. This motivates the use of a similar model for our parser architecture. We put a bidirectional LSTM in place of the character encoder. Because we implement a transition based parsing scheme, the encoder needs to produce a distinct embedding for each word to be placed on the Stack and Buffer during at encode time, that is, without any information about the parsing process. These embeddings are then combined into the initial buffer and also added to the stack whenever a SHIFT or SWAP operation is executed during decoding.

It is also crucial that during training the number of word embeddings produced is equal to the actual number of words in the training sample. This consistency is necessary as the target transition sequence we obtain from the treebank is only guaranteed to produce valid configurations if the initial configuration contains the right amount of words in the buffer.

For each word, the encoder LSTM processes the segment of audio that is associated with it. In theory this segment contains enough information to extract a meaningful embedding of the corresponding word. The segments for each word are provided externally by giving the model information about the word boundaries in the audio as an additional input.

5.1.1 Independent Embeddings

The first approach to the encoder is designed in analogy to the character LSTM from [BaDS15] by treating the audio frames of a word like the characters that make it up. The audio is segmented according to the provided word alignments and each segment is processed individually to form the embedding of the corresponding word. This is done by passing the audio frames in that segment through the bidirectional LSTM and using its last hidden state as an encoding of that word (Figure 5.1.1). Let e_k be the embedding of the k-th word, $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$ the input sequence of audio features and $\langle h_k \rangle = g(\langle x_k \rangle)$ the application of the encoder LSTM to some subsequence of the input. Let t_k^{start} and t_k^{end} be the time steps where word w_k starts and ends according to the word alignments. $s_k = \langle x_{t_k^{start}}, \ldots, x_{t_k^{end}} \rangle$ is then the sequence of audio features associated with that word. Then the encoding process can be illustrated as follows:

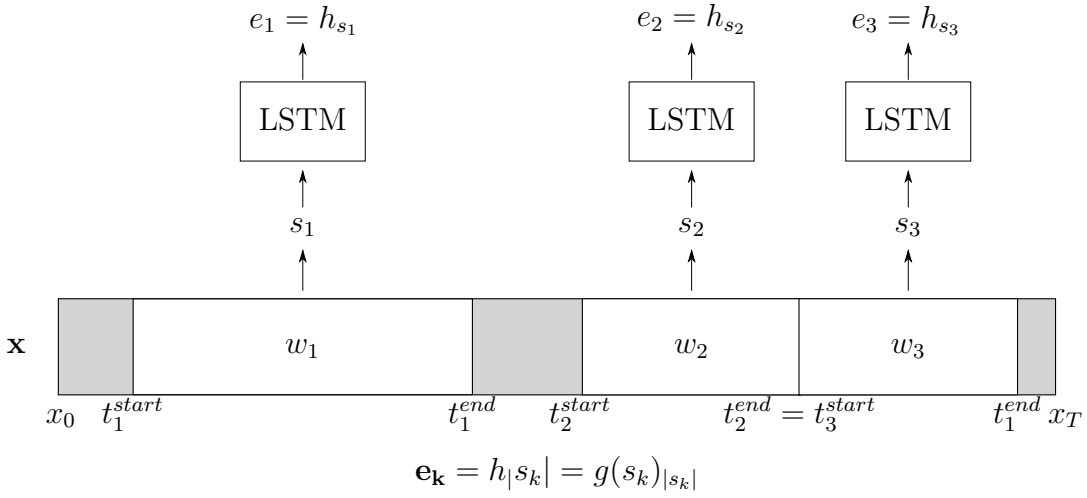


Figure 5.1: Embeddings with individual LSTM passes for each word producing an independent embedding for each word

5.1.2 Continuous Embeddings

A potential downside to the previous approach in spontaneous speech is that the model cannot take into account dependencies between the words at embedding level. To improve upon this, this approach modifies the way in which the audio segments

are processed. Instead of treating each word segment as an independent input sequence, we concatenate all segments defined by the word alignments. The resulting sequence of audio frames is passed through the encoder LSTM in a single pass. Each word will be embedded by picking the hidden state of this LSTM at the last time step of its corresponding word segment (Figure 5.2). This allows the model consider dependencies between the words during encoding time by viewing each word in its context. Following the notation from above this encoding approach can be illustrated like this:

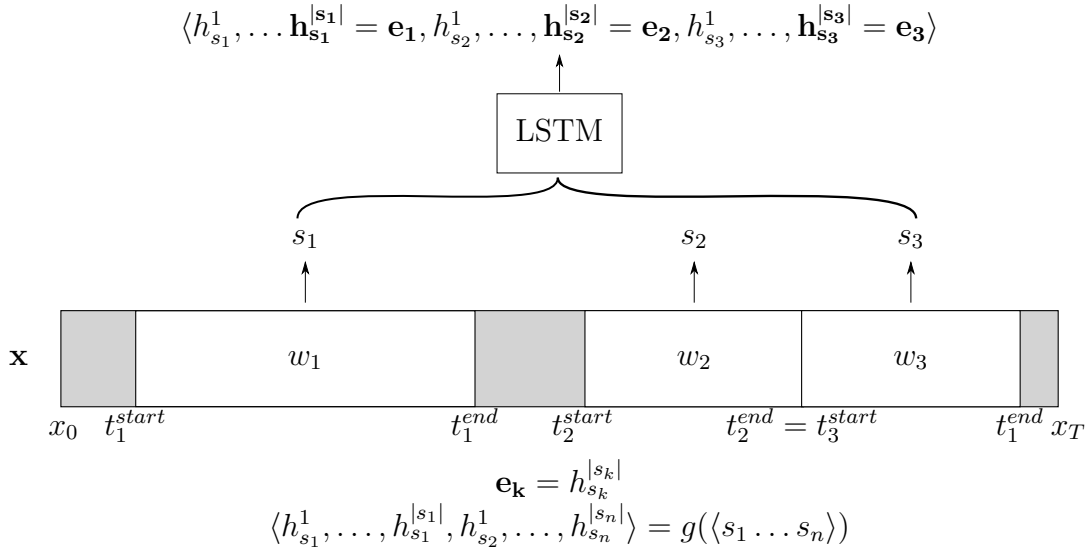


Figure 5.2: Embeddings with word segments concatenated into continuous sequence of audio and one LSTM pass per sentence producing context dependent audio encodings.

5.2 End to End Training

Because of the similarity in design between the presented encoders and those from [BDGS17] out model is compatible with the way that parser was trained. This end-to-end training is the most straightforward approach. The model is constructed by combining the discussed encoders with parse decoder from [BDGS17] and then trained to optimize the classification of the next action. Ideally the model learns to extract an encoding for each word from the audio while simultaneously also learning to make the transition decisions based on its state.

5.3 Transfer from Text Model

While training the model end to end promises a good solution, it also combines two hard problems: transforming the audio into an embedding and then parsing from this embedding. Encoding the audio is a lot harder than getting the embedding for a sequence of characters because of not only the much greater feature space but also the differences in pronunciation between each instance of a word. Therefore this joined optimization task might prove too complex.
Because of the large number of operations that take place in the decoder after the encoding is computed, the error signal from the parse decision has to be propagated

far back which might also cause issues with training.
In this approach we first find an independent solution for each of those problems and then combine them in a second step. This is motivated in particular by that fact that we already have a solution for one half of the problem. On text encodings the parser performs very well. If we split the problem into encoding and decoding we can reuse this existing solution for decoding. Instead of training the entire model end to end, we first train an audio encoder to produce encodings that match the embedding of the same word produced by a trained text encoder. Because the text model performs well, its encoder produces embeddings that are well suited to be used with the parse decoder. Replicating these embeddings gives us a good starting point for how the audio features should be represented. Once this stage of training converges, the trained encoder is combined with the original text parser and then trained end to end as described above.
This approach reduces the complexity of the problem and allows us to reuse the representations that the text model learned.
It also opens the model up to a greater amount of training data because the embedding target step can be done on data for which there are no parse trees available (for example the parts of Switchboard that are not in the Penn Tree Bank)

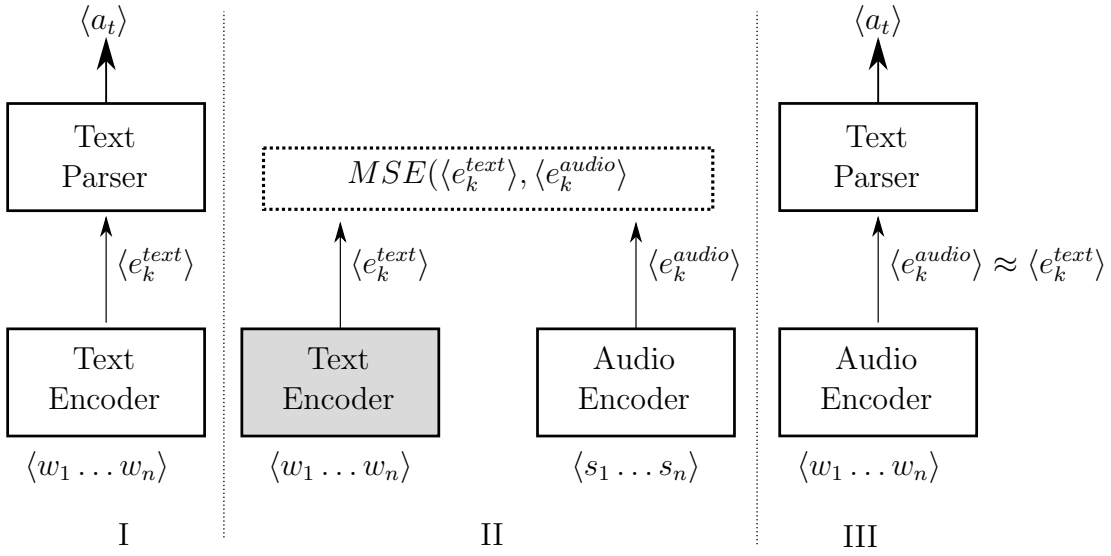


Figure 5.3: Training procedure for transferring text embeddings. First a text model is trained, then embeddings are transferred to an audio encoder and then that encoder is fine tuned in an end to end setting.

5.4 Using Encoder from a Sequence to Sequence Model

All of the approaches mentioned so far rely on explicitly defined, externally provided word alignments. Even when these are obtained through an automated process like generating forced alignments with an HMM audio model, such alignments are still external to the model.
Ideally we would like to avoid this limitation entirely and have a model that can suffice without external alignments or at least generate them within itself in an end to end fashion. Once again we draw on the similarities to sequence to sequence models. An attention based decoder in a seq2seq model produces its next output

based on its previous state but also on the attended input encoding. Qualitative analysis of the attention suggests that the attention vector tends to be highly focused around only a few time steps of the encoder and close to zero everywhere else. The focused frames align with where the corresponding word is in the audio. If this assumption holds, then the attended encoder state might contain some information about w_t. Such a model could even be trained with a multi-task training to use the same encoder, similar to the combination of CTC and Seq2Seq in [KiHW16]. As a first step to investigate the viability of this we use the attended encoder states of a trained seq2seq model as word embeddings and train the parse decoder to operate on these. Further advances in that direction might yield more promising results but are out of the scope of this work.

6. Experiments

6.1 Data

6.1.1 Penn Tree Bank

To train the aforementioned parsing model we need training data that is annotated with parse trees. The trees used in this work come from the Penn Tree Bank [MaMS93]. Because the Penn Tree Bank contains parse trees that follow a constituency parsing scheme, the dependency relations have to be extracted from those using [DMMMo06]. [DMMa08] notes that this process is not infallible the parse trees used as the training set might contain errors that were introduced by the conversion tool. [BDGS17] trained and evaluated their model on the WSJ part of this corpus which consists of a collection of articles from the Wall Street Journal. These are well written, grammatically sound sentences that contain complex syntactic structures. However, as we are interested in operating on speech WSJ is not applicable to the problem at hand because we need training data that contains both parse trees and the corresponding audio.

Switchboard

The switchboard corpus [GoHM92] consists of about 2,400 telephone conversations between two people each. The Penn Tree Bank includes a subset of the transcripts which was annotated with parse trees. This subset is used throughout this work.
All samples from this corpus are conversational, spontaneous speech. Participants are given a topic and are told to discuss it. Because of the unprepared nature of these conversations they include many filler words like “well”, “uhm” or “like”. Another common feature are sentences where the speaker is interrupted or corrects themselves. Some of the sentences are grammatically incorrect. The parse trees have corresponding annotations for those instances and most of these annotations can be mapped to dependency relations from the Stanford Dependencies which are then treated as any other dependency relation. However, manual inspection shows that not all of these annotation carry over well when run through the converter from [DMMMo06].

Because of the nature of conversational speech the Switchboard corpus contains many very short sentences. Most of the sentences with less than three words

6.1.2 Accuracy Metric

The accuracy of the generated parse trees compared to the reference is measured as *attachment score*. The *Labeled Attachment Score* (LAS) is the ratio of correctly predicted relations to the number of total relations in a parse tree. Whereas *Unlabeled Attachment Score* cares only about the correct words being in a relation, irrespective of the label.
Following the notation from above, let $A, R \in \{(u \to v, l) \mid u, v \in W, l \in \mathscr{L}\}$ be the sets of dependency relations found by the parser and the reference relations obtained from a tree bank respectively. The attachment scores are then defined as

$$\text{LAS} = \frac{|A \cap R|}{|R|}$$
$$\text{UAS} = \frac{|A \cap \{(u \to v, l) \mid \exists l \in \mathscr{L} : (u \to v, l) \in R\}|}{|R|}$$

6.1.3 Data Preparation

Word Alignments

As argued above, the parser requires an individual embedding for each word and therefore their location in the audio. We use externally created word alignments provided by [Pico]. Contractions like “it’s”, “that’s” or “don’t” cause issues because in the word alignments and transcriptions these are treated as one unit. However, in the dependency parse they are two distinct units, usually with a dependency relation between them. In the case of text input it is easy to simply split those words at the apostrophe. With the alignments on the other hand, it is hard to tell exactly which part of an alignment belongs to the base word and which to the contracted second word.
To work around this both are assigned the same frame window.

Resegmentation

Switchboard and the Penn Tree Bank are not immediately compatible for our task. They based on different versions of transcriptions of the original telephone conversations and are segmented differently. The mismatch in transcription is largely mitigated by the word alignments [Pico] which already clean up most of the differences.
Switchboard is segmented into utterances which comprise of an uninterrupted segment of speech, whereas data in the Penn Tree Bank is split at sentence boundaries for the parse trees to work. Utterances may cover exactly one, less than one or multiple sentences fully or partially, therefore there is no general assumption possible that would allow a naive mapping from one to another. For the experiments we resegment switchboard into utterances that exactly correspond to one sentence in the Penn Tree Bank each. This is done by matching the parse trees with the SWBD transcript and then using the word alignments to find the interval of audio

that corresponds to the words in the parse tree and then resegmenting accordingly at the word level.
Using this process 98.4% of the parse trees could be matched with the transcript. For the remaining sentences there were either discrepancies between the transcriptions or the alignments were ambiguous, where the latter only occurred twice. Additionally sentences with a length of less than three are discarded from the data because they usually only contain a one word answer or a filler word or both (e.g. “Well, okay.”, “Yes.”). Their parse trees do not contain much information and their inclusion would artificially increase the UAS because random guessing yields 50% on such a sentence.

6.2 Results on Text

To get an initial understanding of the parser and its performance on the data, we reproduced the results from [BDGS17] and also experimented with the parser on the Switchboard corpus. 6.1 shows the results of the model on Switchboard and Wallstreet Journal. Because of the difference in the size between the two datasets we also train a model on a subset of wall street journal that has roughly the same size as Switchboard. Additionally we train a model on a combined dataset that has all of Switchboard and all of the Wall Street Journal training data. We omit punctuation in all experiments.
Switchboard always refers to the Switchboard subset found in the Penn Tree Bank. We split it into training, development and validation set with parts 2,3 being the training set and 4 split evenly between development and validation. For WSJ we use the standard split used in [BDGS17]: Train 02-21, Dev 22 and Valid 23.

Data	**SWBD**		**SWBD + WSJ**		**WSJ**		**WSJ subset**	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Train	92.1	-	90.7	-	94.3	-	95.7	-
WSJ dev	69.5	61.9	85.4	82.3	90.3	87.8	88.60	-
WSJ valid	71.0	64.2	85.6	82.4	91.8	89.40	89.08	85.93
SWBD dev	87.1	83.5	85.2	83.1	73.2	64.7	70.1	61.8
SWBD valid	86.5	83.0	84.9	83.0	73.0	64.8	70.1	61.9

Figure 6.1: Results for the parser from [BDGS17] when trained on WSJ, SWBD, both or a subset of WSJ. Training scores taken from a subset of the training data of the same size as the development set.

These results illustrate the domain differences between the two datasets as neither model performs well on the opposite dataset. It is also of note that even though we expect Switchboard to be harder to parse, the difference in accuracy seems to be due to the amount of training data. The results on the Wall Street Journal subset are not significantly better than those on Switchboard. This might be because even though Switchboard has grammatical errors and incomplete sentences the structure of those sentences is not nearly as complex as those in WSJ.

6.2.1 Dataset Comparison

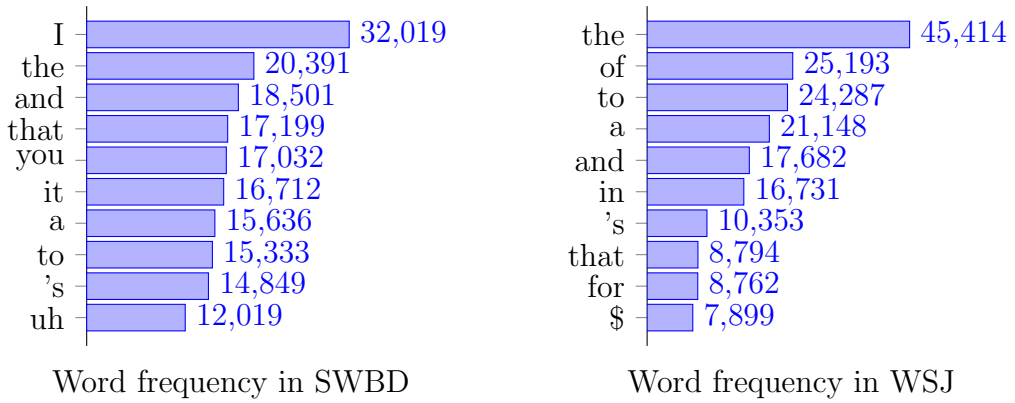


Figure 6.2: Relative frequency of 10 most frequent words in the SWBD and WSJ data used in this work

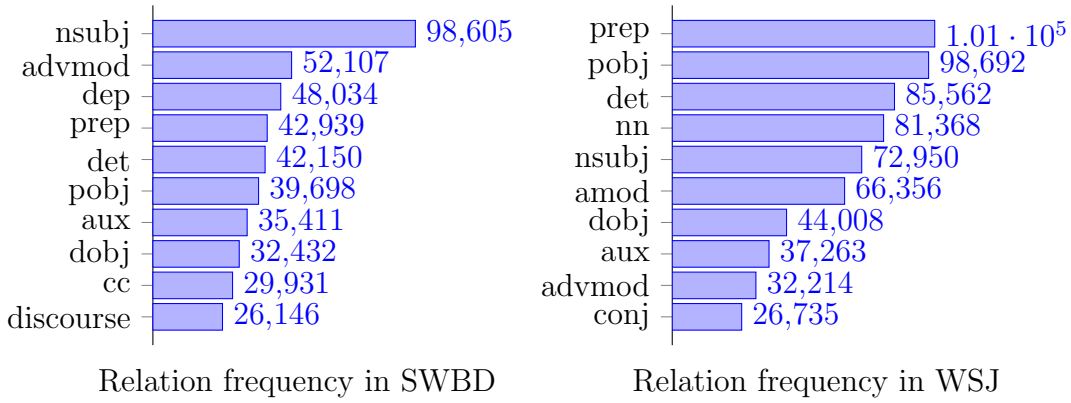


Figure 6.3: Relative frequency of 10 most frequent relations in the SWBD and WSJ data used in this work. The total number of relations are 82 and 79 respectively.

The difference in domain becomes evident in Figures 6.2 and 6.3. By far the most common word in Switchboard is "I" and the the most common relation is NSUBJ. This alludes to a simple overall sentence structure centered around sentences that follow the scheme "I [verb] ...". The many fill words cause DISCOURSE to be one of the most common relations. We also notice that the third most common relation is the fall back relation DEP. [DMMMo06] outputs this relation when no matching relation from the Stanford Dependencies can be found. This confirms that some of Switchboard's parse trees do not translate well into dependency parses.
The grammatical complexity of WSJ shows in the wide range of different relations that have a similar frequency.

6.3 Parser on ASR Output

To get a first overview of parse performance the output of an ASR system of good quality is fed into the parse and compared against the parse trees of the ground truth. We used [ZSMN+17] with greedy search CTC and bytepair encodings. Because a comparison only makes sense if output has the same length as the ground truth, we chose to only look at sentences without deletions and insertions. The resulting system is not an actual baseline but a reference for what kind of performance one can expect. Not only is it in no way optimized for this, it also performs a different task. The intriguing characteristic of the presented parsing approaches is that the parse tree is asserted between segments of audio *before* the transcript is available this is different from what is evaluated here.

Input	UAS	Las
Ground Truth	92.4	88.5
ASR output	68.0	58.1

Figure 6.4: Results of parsing the output of the ASR system through the trained text parser. Due to restrictions in which sentences could be used this was only calculated with around 100 sentences.

Even on the sentences that match the ground truth in length there is a significant drop in performance when compared to feeding the ground truth.

6.4 Model Implementation

All models are implemented using dynet [NDGM+17]. The implementation of the parsing logic is based on [BaDS15] with dimensions kept largely the same. The stack, buffer and action history are each represented by a 2 layer Stack LSTM with a hidden size of 80. The word embeddings on the stack have a dimensionality of 100. The input to the action history are 20 dimensional embeddings of the parse actions that are taken. The relation embedding used in the composition of the parse tree embeddings (4.4.1) also has 20 dimensions. The activation functions are identical to those described in 4.4.Tuning these parameters can increase the overall performance of the model, however it is outside the scope of this work and thus these parameters remain unchanged through all experiments.
Motivated by [MiGM15] in the choice of encoder size we present results for a 4 and 5 layered BiLSTM with a hidden size of 300 and 150 units per direction. The output of each direction is combined the same way the character LSTM output is processed in 4.4.
[SWFK+18] also experiments with variations of the bidirectional LSTM where instead of taking the last hidden state of the encoder, they chose the time step by doing max pooling over all encoder states. We also experiment with this modification.

6.5 End to End

Training the End to End approach is done as described in 4.4 and [BDGS17]. The model uses stochastic gradient descent to minimize the negative log likelihood of

the correct action at each step in the action sequence. The initial learning rate was $\gamma_0 = 0.1$ with decay of 0.1 per epoch, no momentum is used. Models that showed significantly inferior performance were not trained the full 15 epochs to save computing time.

Model	Train UAS	Dev UAS	Dev LAS	Valid UAS	**Valid LAS**
5x150 independent	66.7	63.2	52.4	62.2	**51.1**
4x300 independent	39.7	38.3	22.9	37.4	**22.3**
5x150 continuous	77.7	65.2	54.6	65.1	**54.2**

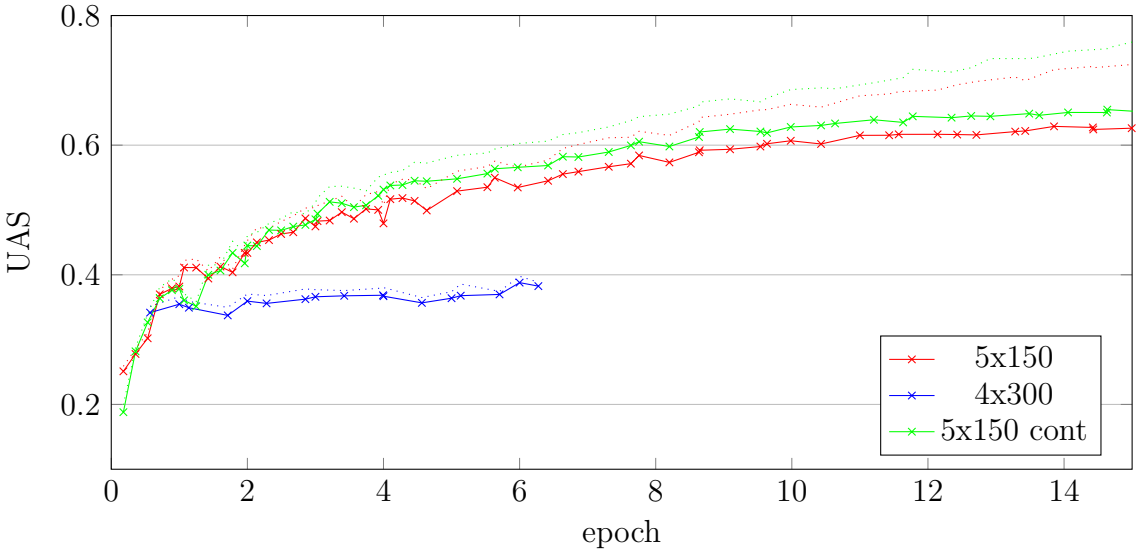


Figure 6.5: Results for the configurations with BiLSTMs with 5 and 4 layers and size 300. Development set UAS is shown, training set UAS is also plotted as dotted line for reference.

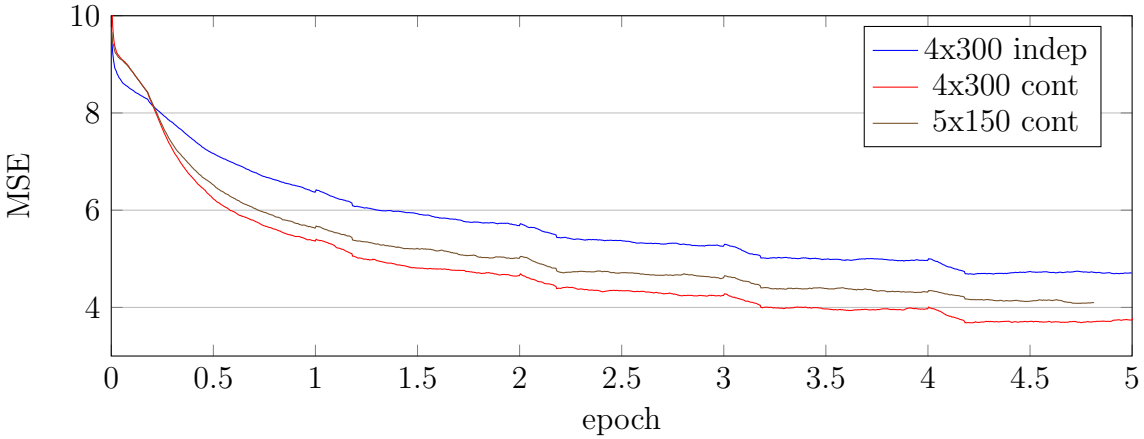
The two best scoring models show promising results. They reach a labeled attachment score of well above 50% which means that on average more than half of all relations in a sentence will be recognized entirely correct, including their label. Considering that the input to this is a sequence of audio frames instead of text, this result demonstrates a reasonable performance of the model. The concatenated approach improved overall performance thus showing that having the context information on encoder level is helpful. This is especially valuable considering that the computational effort is identical for both encoding approaches.
However, training convergence was also highly dependent on the parameters and some configurations that seemed reasonable didn't converge at all. 0

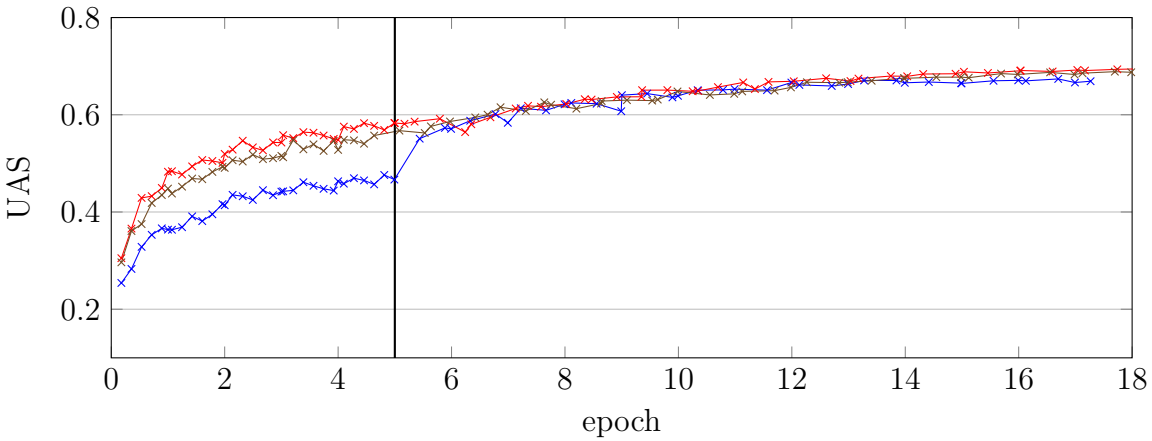
6.6 Transfer From Text

We transfer the embeddings from the text parser trained on SWBD. It achieves an LAS of 85.04 on Switchboard. For each sample we compute the word embeddings using that parser's encoder. During the stage II of this training we optimize the Mean Squared Error between the text embedding and the output of the audio encoder. We use stochastic gradient descent without momentum and a initial learning rate $\gamma_0 = 0.1$. We decay the learning rate by 0.1 per epoch.
After five epochs of training we initialize a new parser with the encoder that was

just trained, together with the parse decoder from which the target embeddings were created originally. In the next state this parser is trained end-to-end as described above with a starting learning rate $\gamma_0 = 0.075$.

Model	Embedding		Train	Dev		Valid	
	MSE	UAS	UAS	UAS	LAS	UAS	LAS
4x300 independent	4.7	50.0	82.0	67.4	57.3	66.5	**56.6**
4x300 continuous	3.5	57.1	83.1	69.7	60.19	68.4	**58.6**
5x150 continuous	4.1	60.0	82.1	69.1	59.1	68.7	**58.6**





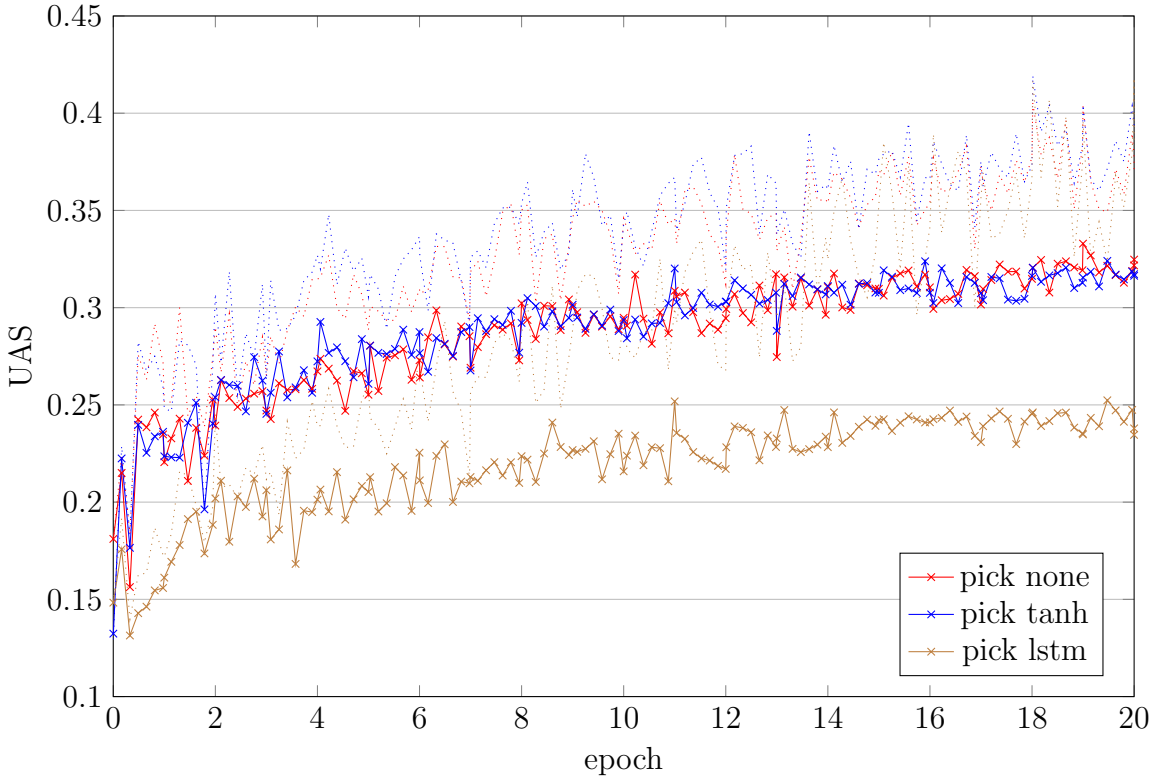
Transferring the embeddings from text showed an overall increase in training stability and performance. All examined configurations converged to a reasonable solution with roughly the same results. The two stage training approach improves results and provides a faster and more stable training routine.
All in all, these approaches yield an audio-to-parse model that shows the feasibility of the task. Further tuning of the parameters is expected to improve those results.

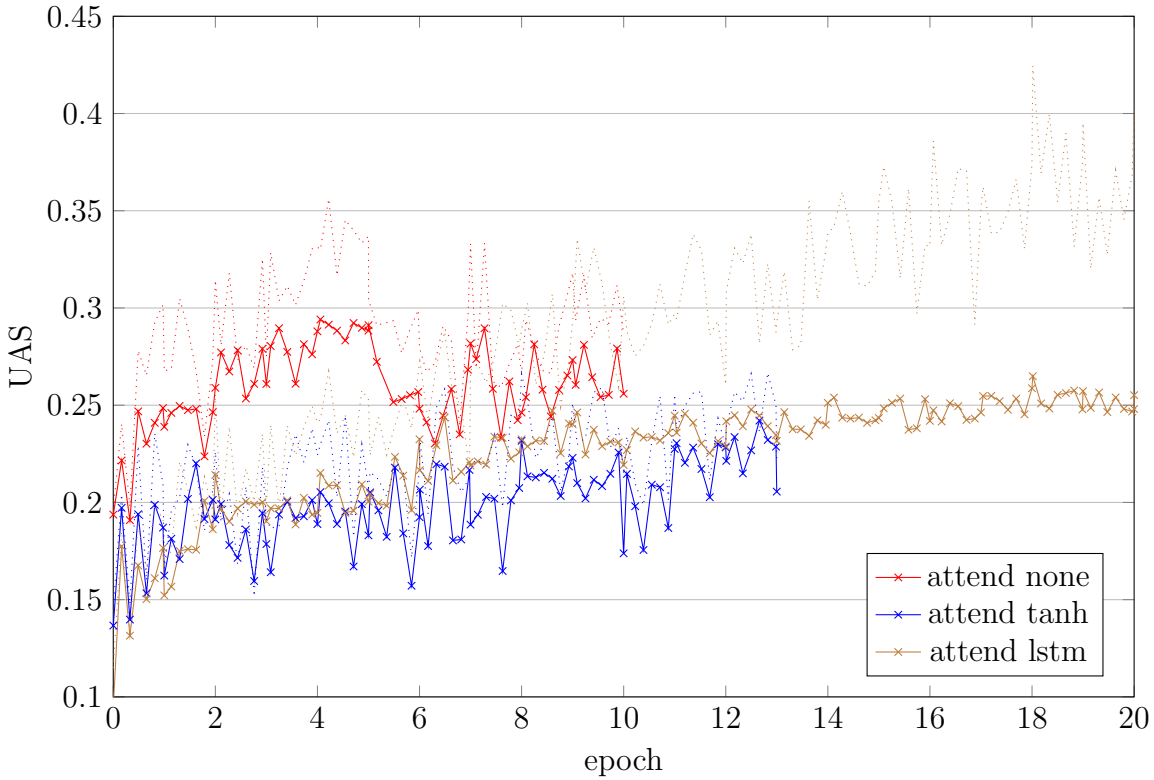
6.7 Feature vectors from seq2seq

The above experiments show promising results for a system that has externally provided word alignments. However, as stated before we want to overcome this limitation. In this experiment we train on features that are extracted from a trained sequence to sequence word model[WHKH+18]. The model is run in training mode with weight updates disabled. We save the encoder state for each utterance along

with the attention vector for each output the decoder makes in that utterance. From this we get the embedding of each word by applying the corresponding attention vector to the encoder state for the sentence and using the result as word embedding. Because the ground truth is fed as the previous decoder state after each prediction during training of the seq2seq model, the model output matches the training sample in word count and we get a consistent number of encodings.
We experiment with a number of ways to pass the encoding to the parse decoder after it is extracted from the dumped state of the seq2seq model. We project it down to the input size of the decoder LSTM and pass it on without applying a nonlinearity. In a second approach we pass the extracted features through Multilayer Perceptron with a tanh activation to allow the model to transform the feats to a representation more suited to the parsing task. Lastly, because in the original decoder these states were used in a sequential context, we put a small 2 layer BiLSTM between them and the parse decoder and take the output at each time step as the corresponding embedding. Because the attention peaks are noisier than anticipated we also experiment with picking the encoder state for which the attention vector has the highest peak.

Model		Train UAS	Valid UAS
pick	no transform	**40.35**	**33.2**
	1x320 tanh	37.15	32.4
	LSTM	38.37	24.5
attend	no transform	33	29.4
	1x320 tanh	24.9	24.2
	LSTM	42.45	26.4





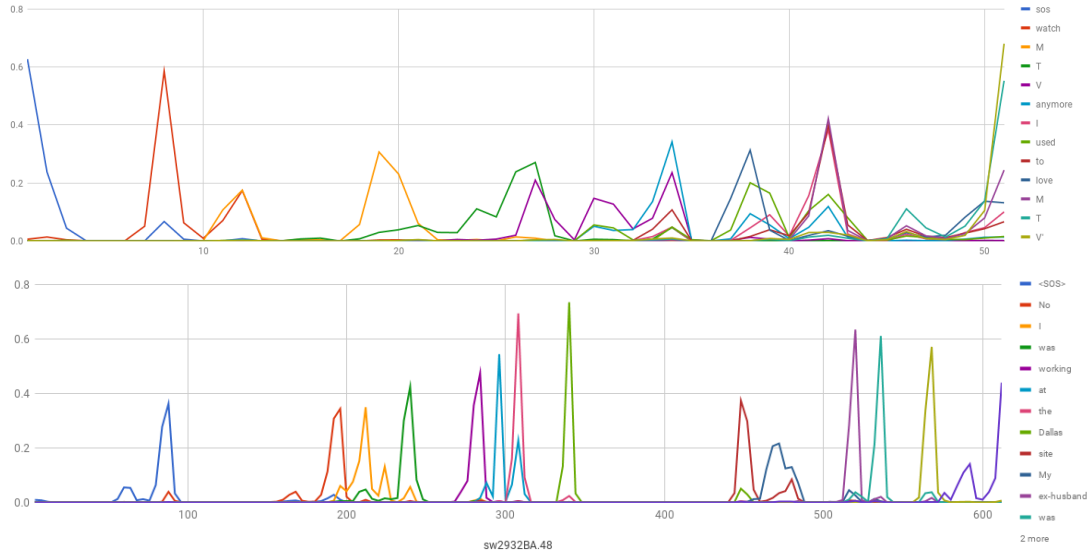


Figure 6.6: Examples of attention vectors from the dumped features. The first is an example for a very noisy attention where the peaks give very little indication about the location of the word in the input encoding. The second example shows a good example of the peakiness of attention.

Unfortunately none of the aforementioned approaches show good performance.UAS values of 30% are too low to show the feasibility of the task. The models show a significantly higher amount of overfitting compared to the previous approaches. However, other, more sophisticated methods for implementing this might yet yield good results. Some possible follow-ups to this are discussed later.

7. Conclusion and Future Work

We have shown that it is feasible to obtain a parse tree as the representation of semantic and syntactic relations between segments of audio. The presented models are a foundation for a more practical audio-to-parse model. Further refining of the approaches introduced in this work can hopefully remove some of the limitations the current model has. The biggest of those is that the good model currently relies on external word alignments. However, even with its limitations this model presents a novel approach to the extraction of parse information from audio.

7.1 Improvements to the Presented Models

Because the goal of this work was to produce a proof of feasibility there is a lot of room for improvements. The following points illustrate ways of improving the performance of the presented architectures.

Hyper Parameter Tuning

Because tuning of hyper parameters was intentionally left out of the scope of this work, the most obvious approach to improve the performance of the presented models is to investigate further into well suited hyper parameters. Along with the model sizes in both encoder and decoder, the transition between embedding and end to end training needs more exploration. Changing the point in the training at which to progress to the next stage and also the learning rate after the switch could bring performance improvements.

Better Input Data

The Switchboard corpus that was used in this work was not optimal. Because of its irregular structure the conversion to dependency parses did not go over as well as hoped (DEP being one of the most common relations). A corpus designed for dependency parses is better suited to train the parser on. Even with the manual alignments and the cleaned up transcriptions, there are utterances that do not map to parse trees and vice versa.

7.2 Next Steps

Even with optimized parameters and more data the models presented here are only a first step towards end to end audio parsing and improvement can be made by adapting the architecture to be better suited to the task. The most important goal is to rid the model of its reliance on external alignments.

Use Attention as Word Alignments

The naive use of a sequence to sequence encoder and attention to get an embedding for the words could not be brought to fruition. However, the peaks in the attention vectors have a strong relation with the position of words in the audio. These peaks could be used to get the hidden states that represent a word from the continuous encoding approach 5.2. We would still train an encoder specific to the parser but would process all frames of the utterance and then pick hidden states according to those peaks. This allows the parser to operate without external alignments. In contrast to manual alignments an end-to-end approach to training the attention within the model would then be possible.

Multi-Task Training with Sequence2Sequence

The next step to simply extracting the features from a given encoder is to train that encoder jointly on its ASR task and the parse output. In its current formulation the decoding would be a two step process: First the sequence-to-sequence decoder outputs its decoding along with attention at each state. The attention vectors are then used by the parser along with the encoder state to do the parsing. The encoder is now being optimized with both the parser and the word recognition task. Another interesting aspect is that during inference the parser always operates on a word sequence that is consistent with the hypothesis of the ASR model therefore this system would always provide a parse tree for its transcriptions.

Transforming to Sequence2Sequence

In 4.4.4 we argue for the similarity between the parser and a seq2seq model. The most significant difference is the need for individual word embeddings at encoding time. By restructuring the parser this requirement can be overcome. The reason why we need to precompute all embeddings is because the model is built to exactly implement an initial configuration of an transition based dependency parser, where all words are in the buffer at the beginning of the parse process. However, words are only really added to anywhere besides the buffer during decoding when a shift operation is executed and the next word is put into the stack. Instead of relying on a previously computed embedding the model could use its current state of buffer, action history and stack to select parts of the input that are relevant to the parser and and those to the stack. In short it could use its state to calculate an attention to select the next word. If we allow the model to predict the ROOT word as part of this process we can model the entire parsing process as a sequence to sequence problem with words as input and parse actions as target output. This would rid the model entirely of any restrictions regarding word alignment and is an interesting approach that should be investigated.

Bibliography

[AhUl72] A. V. Aho und J. D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 1972.

[AyVT16] Y. Aytar, C. Vondrick und A. Torralba. SoundNet: Learning Sound Representations from Unlabeled Video. *CoRR* Band abs/1610.09001, 2016.

[BaCB14] D. Bahdanau, K. Cho und Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BaDS15] M. Ballesteros, C. Dyer und N. A. Smith. Improved transition-based parsing by modeling characters instead of words with lstms. *arXiv preprint arXiv:1508.00657*, 2015.

[BDGS17] M. Ballesteros, C. Dyer, Y. Goldberg und N. A. Smith. Greedy transition-based dependency parsing with stack lstms. *Computational Linguistics* 43(2), 2017, S. 311–347.

[Brid90] J. S. Bridle. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In F. F. Soulie und J. Herault (Hrsg.), *Neurocomputing*, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg, S. 227–236.

[CBSC+15] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho und Y. Bengio. Attention-Based Models for Speech Recognition. *CoRR* Band abs/1506.07503, 2015.

[ChRu13] Y. Chauvin und D. E. Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology Press. 2013.

[DBLM+15a] C. Dyer, M. Ballesteros, W. Ling, A. Matthews und N. A. Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

[DBLM+15b] C. Dyer, M. Ballesteros, W. Ling, A. Matthews und N. A. Smith. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. *CoRR* Band abs/1505.08075, 2015.

[DMMa08] M.-C. De Marneffe und C. D. Manning. The Stanford typed dependencies representation. In *Coling 2008: proceedings of the workshop*

on cross-framework and cross-domain parser evaluation. Association for Computational Linguistics, 2008, S. 1–8.

[DMMMo06] M.-C. De Marneffe, B. MacCartney, C. D. Manning und andere. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, Band 6. Genoa Italy, 2006, S. 449–454.

[DuCa] R. Dunne und N. Campbelly. On The Pairing Of The Softmax Activation And Cross Entropy Penalty Functions And The Derivation Of The Softmax Activation Function.

[GeSS02] F. A. Gers, N. N. Schraudolph und J. Schmidhuber. Learning precise timing with LSTM recurrent networks. *Journal of machine learning research* 3(Aug), 2002, S. 115–143.

[GlBe10] X. Glorot und Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, S. 249–256.

[GoBC16] I. Goodfellow, Y. Bengio und A. Courville. *Deep Learning*. MIT Press. http://www.deeplearningbook.org, 2016.

[GoHM92] J. J. Godfrey, E. C. Holliman und J. McDaniel. SWITCHBOARD: Telephone Speech Corpus for Research and Development. In *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech and Signal Processing - Volume 1*, ICASSP'92, Washington, DC, USA, 1992. IEEE Computer Society, S. 517–520.

[GrFS05] A. Graves, S. Fernández und J. Schmidhuber. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. In W. Duch, J. Kacprzyk, E. Oja und S. Zadrożny (Hrsg.), *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg, S. 799–804.

[GrJM13] A. Graves, N. Jaitly und A.-r. Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, S. 273–278.

[GrSc05] A. Graves und J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18(5-6), 2005, S. 602–610.

[Hass95] M. H. Hassoun. *Fundamentals of artificial neural networks*. 1995.

[Hoch98] S. Hochreiter. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06(02), 1998, S. 107–116.

[Horn91] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4(2), 1991, S. 251 – 257.

[HoSc97a] S. Hochreiter und J. Schmidhuber. Long Short-term Memory. Band 9, 12 1997, S. 1735–80.

[HoSc97b] S. Hochreiter und J. Schmidhuber. Long Short-Term Memory. *Neural Comput.* 9(8), November 1997, S. 1735–1780.

[Jura] D. Jurafsky. *Speech and Language Processing.*

[KaKw92] B. L. Kalman und S. C. Kwasny. Why tanh: choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, Band 4, Jun 1992, S. 578–581 vol.4.

[KiHW16] S. Kim, T. Hori und S. Watanabe. Joint CTC-Attention based End-to-End Speech Recognition using Multi-task Learning. *CoRR* Band abs/1609.06773, 2016.

[KrSH12] A. Krizhevsky, I. Sutskever und G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, S. 1097–1105.

[LeBH15] Y. LeCun, Y. Bengio und G. Hinton. Deep learning. *nature* 521(7553), 2015, S. 436.

[MaHN13] A. L. Maas, A. Y. Hannun und A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Band 30, 2013, S. 3.

[MaMS93] M. P. Marcus, M. A. Marcinkiewicz und B. Santorini. Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguist.* 19(2), Juni 1993, S. 313–330.

[MiGM15] Y. Miao, M. Gowayyed und F. Metze. EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 2015, S. 167–174.

[MPRH05] R. McDonald, F. Pereira, K. Ribarov und J. Hajič. Non-projective Dependency Parsing Using Spanning Tree Algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics, S. 523–530.

[NDGM+17] G. Neubig, C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta und P. Yin. DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[Nivr09] J. Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*. Association for Computational Linguistics, 2009, S. 351–359.

[PeSM14] J. Pennington, R. Socher und C. Manning. Glove: Global vectors for word representation. 2014, S. 1532–1543.

[Pico] J. Picone. Switchboard word alignments. https://www.isip.piconepress.com/projects/switchboard/. Accessed: 2018-05-20.

[RiLi91] M. D. Richard und R. P. Lippmann. Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural computation* 3(4), 1991, S. 461–483.

[Rose58] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65(6), 1958, S. 386.

[RRKO+90] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley und B. W. Suter. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transactions on Neural Networks* 1(4), Dec 1990, S. 296–298.

[ScPa97] M. Schuster und K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11), Nov 1997, S. 2673–2681.

[SiZi14] K. Simonyan und A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[SuSN12] M. Sundermeyer, R. Schlüter und H. Ney. LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[SuVL14a] I. Sutskever, O. Vinyals und Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 2014, S. 3104–3112.

[SuVL14b] I. Sutskever, O. Vinyals und Q. V. Le. Sequence to Sequence Learning with Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence und K. Q. Weinberger (Hrsg.), *Advances in Neural Information Processing Systems 27*, S. 3104–3112. Curran Associates, Inc., 2014.

[SWFK+18] D. Serdyuk, Y. Wang, C. Fuegen, A. Kumar, B. Liu und Y. Bengio. Towards end-to-end spoken language understanding. *arXiv preprint arXiv:1802.08395*, 2018.

[TTBG+17] T. Tran, S. Toshniwal, M. Bansal, K. Gimpel, K. Livescu und M. Ostendorf. Joint Modeling of Text and Acoustic-Prosodic Cues for Neural Parsing. *CoRR* Band abs/1704.07287, 2017.

[WHKH+18] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala und T. Ochiai. ESPnet: End-to-End Speech Processing Toolkit. *CoRR* Band abs/1804.00015, 2018.

[WiMa03] D. R. Wilson und T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks* 16(10), 2003, S. 1429–1451.

[ZSMN+17] T. Zenkel, R. Sanabria, F. Metze, J. Niehues, M. Sperber, S. Stüker und A. Waibel. Comparison of decoding strategies for ctc acoustic models. *arXiv preprint arXiv:1708.04469*, 2017.