Karlsruhe Institute of Technology

Backchannel Prediction for Conversational Speech Using Recurrent Neural Networks

Bachelor's Thesis of

Robin

At the Department of Informatics
Institute for Anthropomatics and Robotics
Interactive Systems Labs

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	?
Advisor:	Markus Müller

Duration: ??. Monat 20?? – ??. Monat 20??

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, ??. ?????? 200?

Contents

1. Introduction

Motivation, Goals

2. Related Work

Collection:

- Watanabe and Yuuki (1989)
  - not found
- Okato et al. (1996)
  - Languages: Japanese
  - Truth type: utterances
  - Features: Pause
  - Method: HMM for pitch contour
  -
  - Evaluation Method:
  - Margin of Error: (-100ms, +300ms) from target utterance end (?)
- Ward (1996)
  - Japanese
- Noguchi and Den (1998)
  - Languages: Japanese
  - Features: Prosodic (pause, frequency F0)
- Ward and Tsukahara (2000)
  - English, Japanese
  - Features: Low Pitch Cue, Pause
  - Margin of error: (-500, 500)
- Cathcart, Carletta, and Klein (2003)
  - English
  - Features: trigrams, pauses

  - Corpus: HCRC Map Task Corpus
  - Eval Method: Precision, recall, F1

- Fujie, Fukushima, and Kobayashi (2004)
  - Japanese
  - Features: utterances, prosodic

- M. Takeuchi, Kitaoka, and Nakagawa (2004)
  - Japanese
  - features: porosodic
  - Method: decision tree, C4.5 learning algorithm
  - Corpus: SIG of Corpus-Based Research for Discourse and Dialogue, JSAI, 1999. "Constructing a spoken dialogue dorpus as sharable research resource"
  - Eval method: recall, precision

- N. Kitaoka et al. (2005)
  - Japanese
  - Pitch, pause,
  - Eval: precision, recall, F1

- Nishimura, Kitaoka, and Nakagawa (2007)
  - Japanese
  - Features: Speech recog,

- L.-P. Morency, Kok, and Gratch (2008)
  - English
  - Features: Eye gaze, low pitch, pause
  - HMM, CRF
  - Margin of error: happens during actual BC utterance

- De Kok and Heylen (2009)
  - English
  - Features: dialog, attention, head gestures, prosody (pitch, pause, etc)
  - Margin: peak in our probabilities (see Section 3) occurs during an actual end-of-speaker-turn.

- L.-P. Morency, de Kok, and Gratch (2010)
  - Dutch
  - Corpus: MultiLis corpus
  - Special: building consensus Fconsensus

- de Kok et al. (2010)
- Huang, Morency, and Gratch (2010)
  - Subjective, on live corpus
  - Fconsensus

- Ozkan and Morency (2010)

  - RAPPORT dataset

- Ozkan, Sagae, and Morency (2010)
- R. Poppe et al. (2010)
- de Kok, Heylen, and Morency (2013)
- D. Ozkan and Morency (2013)
- de Kok, Poppe, and Heylen (2014)
- Mueller et al. (2015)

3. Backchannel Prediction

A listener backchannel is generally defined as any kind of feedback a listener gives a speaker as an acknowledgment in a segment of conversation that is primarily one-way. They include but are not limited to nodding (Watanabe and Yuuki 1989), a shift in the gaze direction and short phrases. Backchannels are said to help build rapport, which is the feeling of comfortableness or being "in sync" with conversation partners (Huang, Morency, and Gratch 2011).

(-> motivation) This thesis concentrates on short phrasal backchannels consisting of a maximum of three words. We try to predict these for a given speaker audio channel in a causal way, using only past information.

This would allow the predictor to be used in an online environment, for example to make a conversation with an artificial assistant more natural.

3.1 BC Utterance selection

The definition of backchannels varies in literature. There are many different kinds of phrasal backchannels, they can be non-committal ("uh huh", "yeah"), positive/confirming ("oh how neat", "great"), negative/surprised ("you're kidding", "oh my god"), questioning ("oh are you", "is that right"), et cetera. To simplify the problem, we initially only try to predict the trigger times for any type of backchannel, ignoring the distinction between different kinds of positive or negative responses. Later we also try to distinguish between a limited set of categories.

3.2 Feature selection

The most commonly used audio features in related research are fast and slow voice pitch slopes and pauses of varying lengths. (Ward and Tsukahara 2000; Truong, Poppe, and Heylen 2010; L.-P. Morency, de Kok, and Gratch 2010). A neural network is able to learn advantageous feature representations on its own, so we simply feed it the absolute pitch and power (signal energy) values for a given time context, from which it is able to calculate the pitch slopes and pause triggers on its own by subtracting the neighboring values in the time context for each feature. The

power value used is the logarithm of the raw `adc2pow` value output by the Janus Recognition Toolkit (JRTk).

We also try to use other tonal features used for speech recognition in addition to and instead of pitch and power. The first feature is the fundamental frequency variation spectrum (FFV) (Laskowski, Heldner, and Edlund 2008), which is a representation of changes in the fundamental frequency over time, giving a more accurate view of the pitch progression than the single-dimensional pitch value which can be very noisy. This feature has seven dimensions in the default configuration given by Janus.

Other features we tried include the Mel-frequency cepstral coefficients (MFCC) with 20 dimensions [ref] and a set of bottleneck features trained on phoneme recognition using a feed forward neural network, which is used for speech recognition at the Interactive Systems Lab.

3.3 Training area selection

We generally assume to have two separate but synchronized audio tracks, one for the speaker and one for the listener, each with the corresponding transcriptions. We need to choose which areas of audio we use to train the network. We want to predict the backchannel without future information (also called *causally* or *online*), so we need to train the network to detect segments of audio from the speaker track that probably cause a backchannel in the listener track. The easiest method is to choose the beginning of the utterance in the transcript of the listener channel as an anchor t, and then use a fixed context range of width w before that as the audio range to train the network to predict a backchannel ($[t - w, t]$). The width can range from a few hundred milliseconds to multiple seconds. We feed all the extracted features for this time range into the network at once, from which it will predict if a backchannel at time t is appropriate. This approach is easy because it only requires searching for all backchannel utterance timestamps and then extracting the calculated range of audio. It may not be optimal though, because the delay between the last utterance of the speaker and the backchannel can vary significantly in the training data. This means it is not guaranteed that the training range will contain the actual trigger for the backchannel, which is assumed to be the last few words said by the speaker, and even if it does the last word will not be aligned within the context. This causes the need for the network to first learn to align it's input, making training harder and slower.

Another interesting anchor is the last few words before a backchannel. We could for example choose t as the center of the last speaker utterance before the backchannel, and then use $[t - 0.5s, t + 0.5s]$ as the training range. While experimenting this proved to be hard because without manual alignment it isn't clear what the last relevant utterance even is, and in many cases the relevant utterance ends after the backchannel happens, so we would need to be careful not to use any future data. The first approach seemed to work reasonably well, so we did not do any further experiments with the second approach.

We also need to choose areas to predict zero i.e. "no backchannel" (NBC). The number of training samples of this kind should be about the same amount as backchannel samples so the network is not biased towards one or the other. To create this balanced data set, we can choose the range a few seconds before each backchannel as

a negative sample. This gives us an exactly balanced data set, and the negative samples are intuitively meaningful, because in that area the listener explicitly decided not to give a backchannel response yet, so it is sensible to assume whatever the speaker is saying is not a trigger for backchannels.

The previous method trains the network on binary values, only $1 = 100\% =$ "Backchannel happens here" and $0 = 0\% =$ "Definitely no backchannel happens here. Another approach for choosing a training area would be to not choose two separate areas to predict binary 1 and 0 values, but to instead use a area around every actual backchannel trigger and train the network on a bell curve with the center being at the ground truth with the maximum value of 1, and lower values between 0 and 1 for later and earlier values. For example, if there is a backchannel at t=5,s and we identify t=4.5,s as the actual trigger time, we could train the network on output=1 for a context centered at 4.5,s, 0.5 for 200ms earlier and later and on $v \ll 1$ for more distant times. This has the same problem as described above in that we would need to know the exact time of the event that triggered the backchannel. Testing this approach by simply using a fixed offset before the onset of the backchannel utterance gave far worse results than the binary training approach, so we discarded the idea in favor of concentrating on finding the optimal context ranges for the binary approach.

3.4 Training and Neural Network Design

We begin with a simple feed forward network architecture. The input layer consists of all the chosen features over a fixed time context. With a time context of c ms and a feature dimension of f, this gives us a input dimension of $f \times \lfloor \frac{c}{10\,\mathrm{ms}} \rfloor$. One or more hidden layers with varying numbers of neurons follow. After every layer we apply an activation function (also called a nonlinearity) like tanh or ReLU. The output layer is $(n + 1)$–dimensional, where n is the number of backchannel categories we want to predict. This layer has softmax as an activation function, which maps a K-dimensional vector of arbitrary values to values that are in the range $(0, 1]$ and that add up to 1, which allows us to interpret them as class probabilities.

We then calculate categorical cross-entropy of the output values of the network and the ground truth from the training data set. This gives us the loss function as the function mapping from the network inputs to the cross-entropy output. We can now train the parameters of the network by deriving it individually for each of the neurons and descending the resulting gradient using the back-propagation algorithm (Rumelhart, Hinton, and Williams 1986).

In the simplest case (ignoring different kinds of backchannels) we train the network on the outputs $[1, 0]$ for backchannels and $[0, 1]$ for non-backchannels. When evaluating we can ignore the second dimension of this output, simply interpreting the first dimension as a probability. A visualization of this architecture can be seen in fig. 3.1. In the following sections we will concentrate on this architecture.

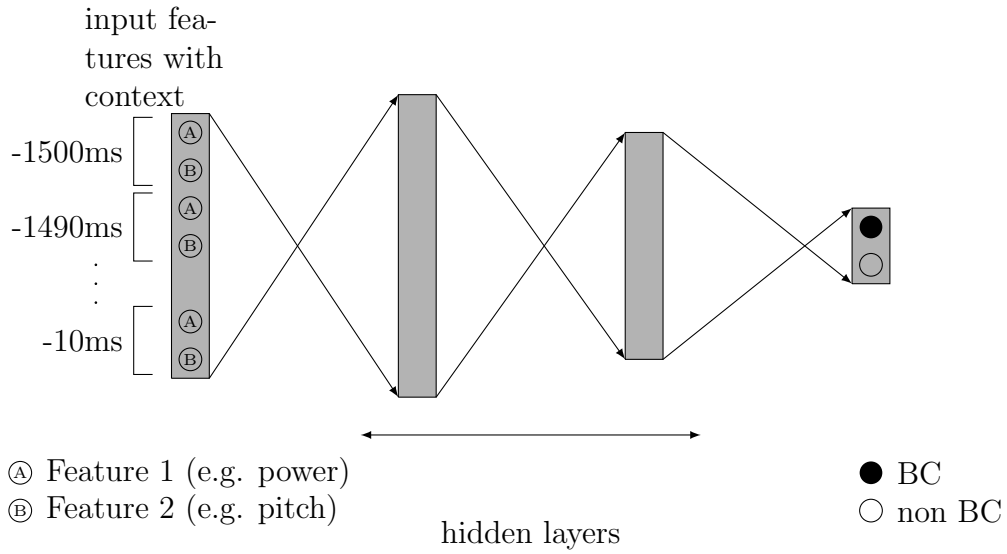

Figure 3.1: Neural Network architecture.

The placement of backchannels is dependent on previous backchannels: If the previous backchannel utterance was a long time ago, the probability of a backchannel happening shortly is higher and vice versa. To accommodate for this, we want the network to also take its previous internal state or outputs into account. We do this by modifying the above architecture to use Long-short term memory layers (LSTM) instead of dense feed forward layers. LSTM neurons are recurrent, meaning they are connected to themselves in a time-delayed fashion, and they have an internal state cell which is transmitted through time and which has set and clear functions which are triggered by any combination of their inputs. LSTM networks are trained in similar fashion as feed forward networks, with the time-stacked layer instances unrolled into individual copies with shared parameters before applying the backpropagation algorithm.

3.5 Postprocessing

Our goal is to generate an artificial audio track containing utterances such as "uh-huh" or "yeah" at appropriate times. The neural neural network gives us a noisy value between 0 and 1, which we interpret as the probability of a backchannel happening at the given time. To generate our audio track from this output, we need to convert the noisy floating value into discrete trigger timestamps. We first run a low-pass filter over the network output, which removes all the higher frequencies and gives us a less noisy and more continuous output function.

To ensure our predictor does not use any future information, the low-pass filter must be causal. A common low-pass filter is the gaussian blur, which folds the input function with a bell curve. This filter is symmetric, which in our case means it uses future information as well as past information. To prevent this, we cut the filter of asymmetrically for the right side (that would range in the future), with a cutoff at some multiple c of the standard deviation σ. Then we shift the filter to the left so the last frame it uses is ± 0 ms from the prediction target time. This means the latency of our prediction increases by $c \cdot \sigma\, ms$. If we choose $c = 0$, we cut off the complete right half of the bell curve, meaning we do not need to shift the filter, which keeps the latency at 0 at the cost of accuracy of the low-pass filter.

Another possible filter to use would be the Kalman Filter (Kalman 1960), which tries to predict the value of a function based on a noisy function masking the actual function. This filter has many parameters requiring tuning, so we used the described gaussian filter method.

After this filter we use a fixed trigger threshold to extract the ranges in the output where the predictor is fairly confident that a backchannel should happen. We trigger exactly once for each of these ranges. We have multiple possibilities to choose the trigger anchor with in each range.

The easiest method is to use the time of the maximum peak of every range where the value is larger than the threshold, but this requires us to wait until the value is lower than the threshold again before we can decide where the maximum is, which introduces another delay and is thus bad for live detection.

Another possibility is to use the start of the range, but this can give us worse results because it might force the trigger to happen earlier than the time the network would give the highest probability rating.

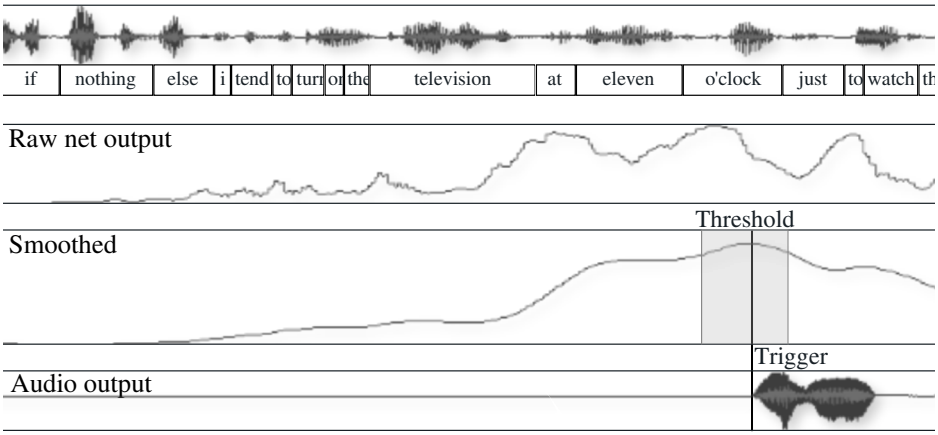

Figure 3.2: Postprocessing example

A compromise between the best quality and immediate decision is to use the first local maximum within the thresholded range. Because of the low-pass filter we mostly have no or few local maxima which differ from the global maximum within the given range, and it's easy to decide when the local maximum was reached by simply triggering as soon as the first derivate is < 0.

When we have the trigger anchor, we can either immediately trigger, or delay the actual trigger by a fixed amount of time, which can be useful if we notice the prediction would happen to early otherwise.

An example of this postprocessing procedure can be seen in fig. 3.2.

3.6 Evaluation

The switchboard dataset contains alternating conversation. Because our predictor is only capable of handling situations where one speaker is consistently speaking and the other consistently listening, we need to evaluate it on only those segments. We call these segments "monologuing segments".

For a simple subjective evaluation, we take a some random audio tracks from the training data and extract the monologuing segments. For each segment, we remove the original listener channel and replace it with the artificial one. This audio data is generated by inserting a random backchannel audio sample at every predicted timestamp. We get these audio samples from a random speaker from the training data, keeping the speaker the same over the whole segment so it sounds like a specific person is listening to the speaker.

To get an objective evaluation of the performance of our predictions, we again take a set of monologuing segments from the evaluation data set and compare the prediction with the ground truth, i.e. all the timestamps where a backchannel happens in the original data.

We interpret a prediction as correct if it is within a specific margin of the nearest real backchannel in the dataset. For example, with a margin of error of $[-100\,\text{ms}, 300\,\text{ms}]$, if the real data has a backchannel at 5.5 seconds, we say the predictor is correct if it also produces a backchannel within $[5.5\,\text{s} - 100\,\text{ms}, 5.5\,\text{s} + 300\,\text{ms}] = [5.4\,\text{s}, 5.8\,\text{s}]$.

In other research, varying margins of error have been used. We use a margin of 0ms to +1000ms for our initial tests, and later also do our evaluation with other margins for comparison with related research.

After aligning the prediction and the ground truth using the margin of error, we get two overlapping sets of timestamps. The set of predictions is called "selected elements", the set of true elements is called "relevant elements". The number of true positives is defined as $TP = |selected \cap relevant|$, which is all the backchannels the predictor correctly identified as such. The number of false positives is defined as $FP = |selected \setminus relevant|$, which is all the backchannels the predictor output that are not contained in the dataset. The number of false negatives is defined as $FN = |relevant \setminus selected|$, which is all the backchannels that the predictor should have found but didn't.

With these, we can now calculate the measures *Precision* and *Recall* commonly used in information retrieval and binary classification:

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

Both of these are values between 0 and 1. The *Precision* value is the fraction of returned values that were correct, and can thus be seen as a measure of the *quality* of a algorithm. The *Recall* value, also known as *sensitivity* is the fraction of relevant values that the algorithm output, thus it can be interpreted as a measure of *quantity*. Precision and Recall are in a inverse relationship, and it is usually possible to increase one of them while reducing the other by tweaking parameters of the algorithm. Recall can be easily maximized to 100% by simply returning true for every given timestamp. Precision can be maximized by never outputting anything, causing every predicted value to be correct. To solve this problem, we use the normalized harmonic mean of precision and recall, also known as the F1-Score or F-Measure:

$$F1\ Score = 2 \cdot \frac{1}{\frac{1}{Recall} + \frac{1}{Precision}} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

We use the F1-Score to objectively measure the performance of our prediction systems.

4. Experimental Setup

4.1 Dataset

We used the switchboard dataset (Godfrey and Holliman 1993) which consists of 2438 telephone conversations of five to ten minutes, 260 hours in total. Pairs of participants from across the United States were encouraged to talk about a specific topic selected from 70 possibilities. Conversation partners and topics were selected so two people would only talk once with each other and every person would only discuss a specific topic once. These telephone conversations are annotated with transcriptions and word alignments [**?**] with a total of 390k utterances or 3.3 million words. The audio data is given as stereo files, where the first speaker is on the left channel and the second speaker on the right channel. We split the dataset randomly into 2000 conversations for training, 200 for validation and 238 for evaluation. As opposed to many other datasets, the transcriptions also contain backchannel utterances like *uh-huh* and *yeah*, making it ideal for this task.

The transcriptions are split into *utterances*, which are multiple words grouped by speech structure, for example (slashes indicate utterance boundaries): "did you want me to go ahead / okay well one thing i- i- i guess both of us have very much aware of the equality / uh it seems like women are uh just starting to really get some kind of equality not only in uh jobs but in the home where husbands are starting to help out a lot more than they ever did um". The length of these utterances varies in length from one word to whole sentences. Each of these utterance has a start time and stop time attached, where the stop time of one utterance is always the same as the start time of the next utterance. For longer periods of silence, an utterance containing the text "[silence]" is between them.

The word alignments have the same format, except they are split into single words, each with start and stop time. Here the start and stop times are mostly exactly aligned with the actual word audio start and end. *[silence]* utterances are between all words that have even a slight pause between them.

To better understand and visualize the dataset, we first wrote a complete visualization GUI for viewing and listening to audio data, together with transcriptions,

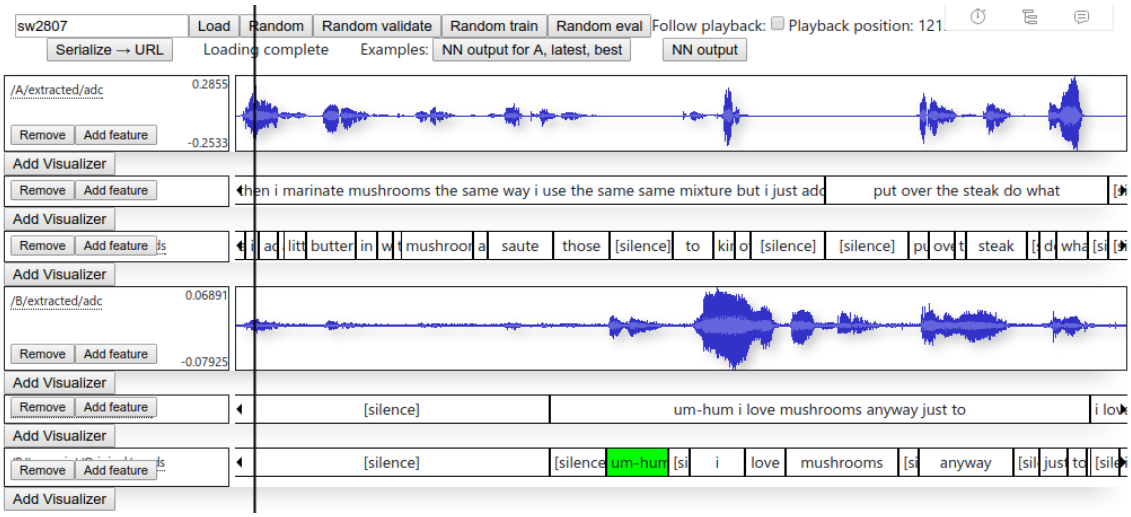

Figure 4.1: From top to bottom: Speaker A audio data, transcription, and word alignment, then the same for speaker B.

markers and other data. This proved to be very helpful. A screenshot of the UI inspecting a short portion of one of the phone conversations can be seen in fig. 4.1.

4.2 Extraction

4.2.1 Backchannel utterance selection

We used annotations from The Switchboard Dialog Act Corpus (Jurafsky, Van Ess-Dykema, and others 1997) to decide which utterances to classify as backchannels. The SwDA contains categorical annotations for utterances for about half of the data of the Switchboard corpus. An excerpt of the most common categories can be seen in tbl. 4.1.

Table 4.1: Most common categories from the SwDA Corpus

	name	act_tag	example	train_count	full_count
1	Statement-non-opinion	sd	Me, I'm in the legal department.	72824	75145
2	Acknowledge (Backchannel)	b	Uh-huh.	37096	38298
3	Statement-opinion	sv	I think it's great	25197	26428
4	Agree/Accept	aa	That's exactly it.	10820	11133
5	Abandoned or Turn-Exit	%	So, -	10569	15550
6	Appreciation	ba	I can imagine.	4633	4765
7	Yes-No-Question	qy	Do you have to have any special training?	4624	4727

We extracted all utterances containing one of the tags beginning with "b" (which stands for backchannels or backchannel-like utterances), and counted their frequency. Because the dataset also marks some longer utterances as backchannels, we only use those that are at most three words long to exclude those that transmit a lot of additional information to the speaker.

We chose to use the top 150 unique utterances from this set. For the most common

aggregated	self	count	category	text
77.84%	1.35%	607	b	okay
78.86%	1.02%	458	bk	okay
79.75%	0.89%	399	b	huh
80.57%	0.81%	364	b	sure
81.29%	0.72%	325	bk	oh okay
81.93%	0.64%	288	b	huh-uh
82.56%	0.63%	282	bh	oh really
83.15%	0.59%	264	ba	wow
83.73%	0.58%	259	b	um
84.16%	0.43%	193	bh	really
84.57%	0.41%	186	b	really
84.97%	0.39%	177	bk	oh

The SwDA is incomplete, it only contains labels for about half of the Switchboard dataset. Because we wanted to use as much training data as possible, we had to identify utterances as backchannels just by their text. As can be seen in tbl. 4.2, the SwDA also has some silence utterances marked as backchannels, as well as only laughter and noise, which we can't distinguish from normal silence, so we ignore them altogether. We manually removed some requests for repetition like "excuse me" from the SwDA list, and added some other utterances that were missing from the SwDA transcriptions but present in the original transcriptions, by going through the most common utterances and manually selecting those that seemed relevant, including but not limited to 'um-hum yeah', 'absolutely', 'right uh-huh'.

In total we now had a list of 161 distinct backchannel utterances. The most common backchannels in the data set are "yeah", "um-hum", "uh-huh" and "right", adding up to 41860 instances or 68% of all extracted backchannel phrases.

The transcriptions also contain markers indicating laughter while talking (e.g. "i didn't think that well we wouldn't still be [laughter-living] [laughter-here] so ..."), laughter on its own (`[laughter]`), noise markers (`[noise]`) for microphone crackling or similar and markers for different pronunciations (for example "mhh-kay" is transcribed as `okay_1`). To select which utterances should be categorized as backchannels and used for training, we first filter noise and other markers from the transcriptions, for example `[laughter-yeah] okay_1 [noise]` becomes `yeah okay` and then compare the resulting text to our list of backchannel phrases.

Some utterances such as "uh" can be both backchannels and speech disfluencies. For example: "... pressed a couple of the buttons up in the / uh / the air conditioning panel i think and uh and it reads out codes that way". Note that the first *uh* is it's own utterance and would thus be seen by our extractor as a backchannel. The second occurrence of "uh" has normal speech around in the same utterance it so we would already ignore it. We only want those utterances that are actual backchannels, so after filtering by utterance text we only choose those that have either silence or another backchannel before them.

This gives us the following selection algorithm:

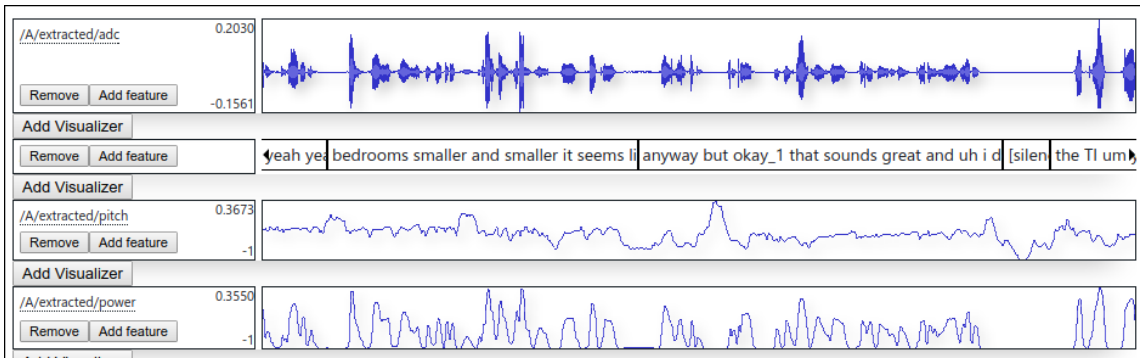

Figure 4.2: From top to bottom: Audio samples, transcription, pitch and power for a single audio channel. Note that the pitch value is only meaningful when the person is speaking.

```
def is_backchannel(utterance):
    text = noise_filter(utterance)
    if index(utterance) == 0:
        # no other utterance before this, can't be a backchannel
        return False
    previous_text = noise_filter(previous(utterance))
    return (text in valid_backchannels and
            (is_silent(previous_text) or is_backchannel(previous(utterance))))
```

This method gives us a total of 61645 backchannels out of 391593 utterances (15.7%) or 71207 out of 3228128 words (2.21%). Note that the percentage of words is much lower because backchannel utterance are on average much shorter than other utterances.

4.2.2 Feature extraction

4.2.2.1 Acoustic features

We used the Janus Recognition Toolkit (Levin et al. 2000) for the acoustic feature extraction (power, pitch tracking, FFV, MFCC).

These features are extracted for 32ms frame windows, with a frame shift of 10ms. This gives us 100 frames per feature per second.

A sample of the pitch and power features can be seen in fig. 4.2.

4.2.2.2 Linguistic features

In addition to these prosodic features, we also tried training Word2Vec (Mikolov et al. 2013) on the Switchboard dataset. Word2Vec is an "Efficient Estimation of Word Representations in Vector Space". After training it on a lot of text, it will learn the meaning of the words from the contexts they appear in, and then give a mapping from each word in the vocabulary to an n-dimensional vector, where n is configurable. Similar words will appear close to each other in this vector space, and it's even possible to run semantic calculations on the result. For example calculating $king - man + woman$ gives the result $= queen$ with the highest confidence. Because

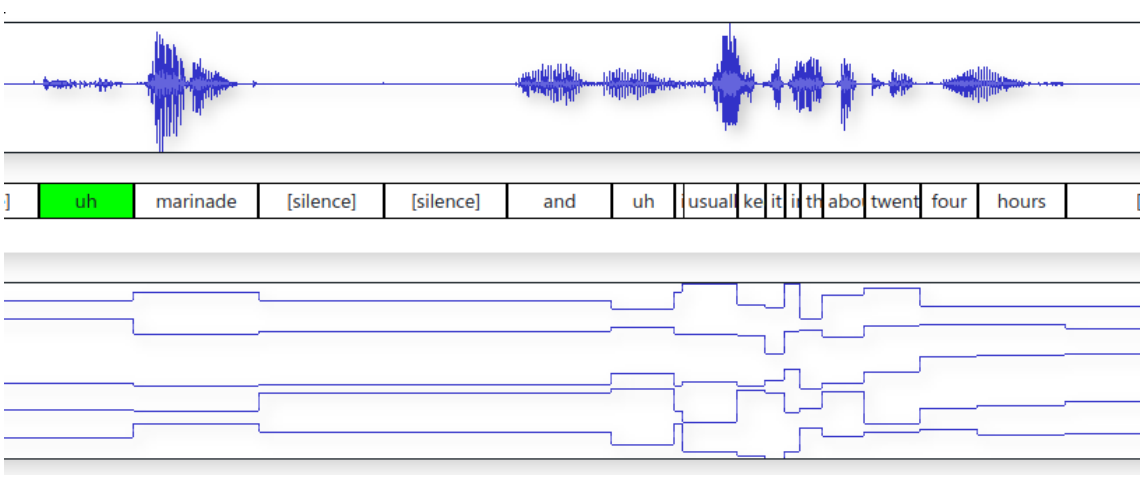

Figure 4.3: five-dimensional Word2Vec feature for some text. The encoding is offset by one word, for example the encoding for "twenty" is seen below the word "four", because we encode the word that *ended* before the current time. Note that with this method we indirectly encode the length of the words / utterances.

our dataset is fairly small, we used relatively small word vectors (5 - 20 dimensions). We train it only on utterances that are not backchannels or silence.

For simplicity, we extract these features parallel to those output by Janus, with a 10 millisecond frame shift. To ensure we don't use any future information, we extract the word vector for the last word that ended *before* the current frame timestamp. This way the predictor is in theory still online, though this assumes the availability of a speech recognizer with instant output. An example of this can be seen in fig. 4.3.

4.2.2.3 Context and stride

We extract the features for a fixed time context. Then we use a subset of that range as the area we feed into the network. As an example, we can extract the range [-2000ms, 0ms] for every feature, giving us 200 frames. We train the network on 1500ms of context, so we treat every offset like [-2000, -500ms], [-1990ms, -490ms], ..., [-1500ms, 0ms] as individual training samples. This gives us 50 training samples per backchannel utterance, greatly increasing the amount of training data, but introducing smear as the network needs to learn to handle a larger variance in when the backchannel cue appears in its inputs, and thus reducing the confidence of its output.

This turned out to not work very well, so in the end we settled on only extracting the features for the range [-w - 10ms, 0] where w is the context width, and training the network on [-w - 10ms, 10ms] and [-w, 0ms]. This gives us two training samples per utterance, reduces the smearing problem and at the same time force the network to learn to correctly handle when its inputs are the same or similar but offset by one.

We can also choose to only use every n-th timestep, which we call the "context stride". This works under the assumption that the input features don't change with a high frequency, which greatly reduces the input dimension and speeds up training. In practice a stride of 2 worked well, meaning one frame every 20 milliseconds. This works great in combination with the above. For example, with a stride of 2 and a

context size of 100ms, we would now get these two training samples (described as frame indices relative to the onset of the backchannel utterance):

1. [-10, -8, -6, -4, -2, 0]
2. [-11, -9, -7, -5, -3, -1]

4.3 Training

We used Theano (Theano Development Team 2016) with Lasagne v1.0-dev (Dieleman et al. 2015) for rapid prototyping and testing of different parameters. We trained a total of over 200 network configuration with various context lengths (500ms to 2000ms), context strides (1 to 4 frames), network depths ranging from one to four hidden layers, layer sizes ranging from 15 to 100 neurons, activation functions (tanh and relu), gradient descent methods (SGD, Adadelta and Adam), dropout layers (0 to 50%) and layer types (feed forward and LSTM).

In general, we used three variables to monitor training: The first is training loss, which is what the network optimizes, as defined in sec. 3.4. We expect this variable to decrease more or less monotonically while training, because the network is descending it's gradient. The second variable is validation loss, which is the same function a training loss, but on the separate validation data set. This allows us to tell whether the network is still learning useful information or starting to overfit. In general, we expect this to be about the same as the training loss, possibly a bit higher. If the training loss is still falling but validation loss is starting to increase again, the network is overfitting on the training data.

The third variable is the validation error, which we define as

$$1 - \frac{\sum_{s \in S} \{1 \text{ if prediction}(s) = \text{truth}(s) \text{ else } 0\}}{|S|},$$

where S is all the frames for all the samples in the validation data set, and prediction(s) = argmaxoutput. This means we take the network output, convert it into the category the network is currently most confident in, and compare it with the ground truth. For example, say we are training with two outputs, one for "Non BC" and one for "BC". The network will always give us two values between 0 and 1 that add up to one because of the softmax function as described in sec. 3.4. If the network output is [0.7, 0.3] for a specific sample we interpret it as "Non BC", because the confidence in the "Non BC" category is higher. If the ground truth for this sample is also "Non BC", the validation error is 0, otherwise 1. This gives us a statistical rating of the current state of the predictor that has less resolution than the loss function (because it throws away the "confidence" of the network), but is closer to our interpretation of the network output when evaluating. We use this value as an initial comparison between predictors and to decide which epoch of the network gives will probably give the best results. We will use a different threshold than 0.5 for evaluation, so the validation error is not a completely accurate measurement.

We train the network in epochs of minibatches. A minibatch is a set of N training samples that we feed into the network at once and update the gradient on their average loss. This means we only update the gradient every N training samples,

which reduces the probability of a single huge gradient disturbing training. We used a minibatch size of $N = 250$ training samples. One epoch is defined as one whole backward pass of all the training data through the network.

We started with a simple model with a configuration of pitch and power as input and 800 ms of context, giving us $80 \cdot 2 = 160$ input dimensions, hidden layers of 100 $\rightarrow$ 50 feed forward neurons. We trained this using many different gradient descent methods such as stochastic gradient descent (SGD), SGD with momentum, Nesterov momentum, Adadelta and Adam, each with fixed learning rates to start. The momentum methods add a speed variable to the descent. This can be interpreted similar to its physical name giver. Imagine a ball rolling down a mountain slope, for each time period, it keeps it's previous momentum and is thus able to jump over small dents in the ground (local minima). In our case, momentum worsened the results, so we stayed with SGD and Adam.

We tried different weight initialization methods. Initializing all weights with zero gave significantly worse results than random initialization, so we stayed with the Lasagne default of Glorot uniform initialization, which uses uniformly distributed random values, with the maximum value scaled so it makes statistical sense with the used layer dimension (Glorot and Bengio 2010). Another method to use would be layer-wise denoising autoencoder pretraining. With this method, the initial weights are created by training the layers individually to reproduce the input data with some dropout. The first layer is trained on its own, the second layer is trained with the first layer before it but fixed and so on. We did not try this, but it might give good results for this use case, especially for deeper networks with vanishing gradients.

We compared online prediction and offline "prediction", where offline prediction got 400 ms of past audio and 400 ms of future audio from the onset of the backchannel utterance, and online prediction got 800 ms of past audio. Offline prediction gave 18% better results, but of course we are more interested in online prediction.

The first simple LSTM we tested by simply replacing the feed forward layers with LSTM layers immediately improved the results by 16% without any further adjustments. But this showed the issues with a fixed learning rate, as the gradient regularily exploded after every 10 - 15 epochs, as can be seen in fig. 4.4. When adding FFV, increasing the input dimension per time frame from 2 to 9, SGD stopped working at all without manually tuning the learning rate. One solution to this would be to use a scheduler. A scheduler automatically adjusts the learning rate depending on some condition. One example is simple exponential decay, which exponentially decreases the initial learning rate with a fixed factor. Another method is newbob scheduling, which exponentially decreases the learning rate when the validation error stops decreasing or only decreases very little. These schedulers need parameter tuning, making them hard to use without experimenting more.

We solved the problem of automatically adjusting the learning rate by using Adam (Kingma and Ba 2014) instead of SGD, which is a gradient descent method related to Adadelta and Adagrad which also incorporates momentum in some way and intelligently adjusts the learning rate. No one really understands how this works, but Adam with a fixed learning rate of 0.001 worked great for us, so we did all further testing using Adam.

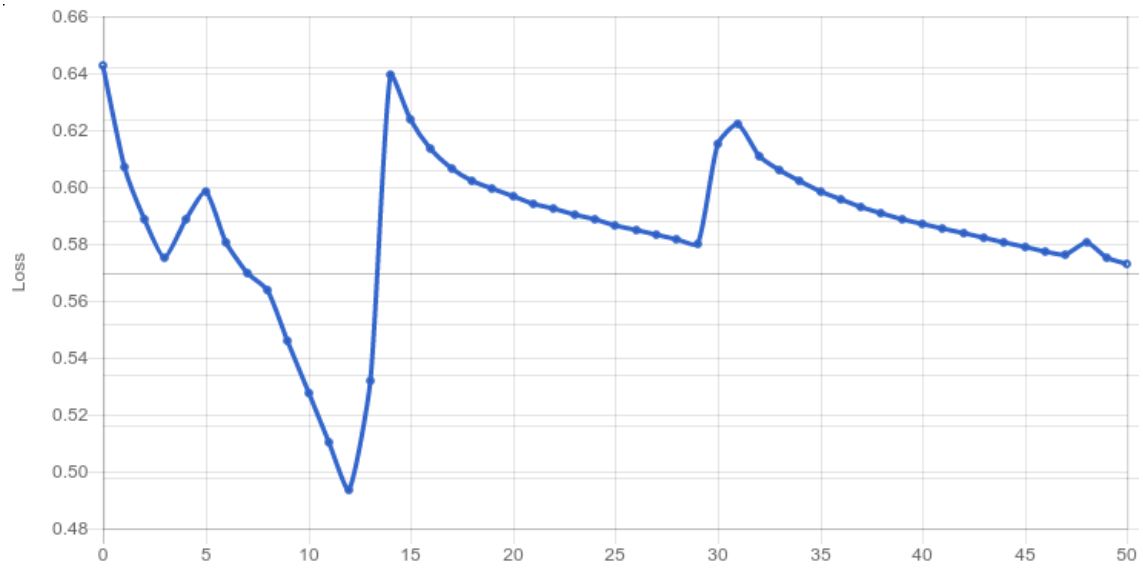

Figure 4.4: Exploding gradient while training a LSTM network. Shown is the training loss over epochs

The LSTM networks we tested were prone to overfitting very quickly, but they still provided better results after two to three epochs than normal feed forward networks after 100 epochs. Overfitting happens when the results still improve on the training data set, but plateau or get worse on the validation data set. This means the network is starting to learn specific quirks in the training data set by heart, which it then can't apply on other data.

We tried two regularization methods to reduce overfitting.

We tried adding dropout layers to the networks to try and avoid overfitting and to generally improve the results. Dropout layers randomly disconnect a specified fraction of neurons in a layer, different for every training batch. This should in theory help the network interpret it's inputs even when it is partially "blind". For validation the dropout is deactivated, so the network is able to take advantage of every single feature when actually using it as a predictor. In this case, we tried adding different dropout settings such as "input (20% dropout) → 125 neurons (50% dropout) → 80 neurons (50% dropout) → output" but this only increased the noise in the training loss and did not improve the results over a simple "input → 70 neurons → 45 neurons" configuration, both for feed forward and for LSTM networks.

The solution that worked was L2-Regularization, which reduced the overfitting problem greatly and slightly improved the results, as can be seen in the example in fig. 4.5.

4.4 Evaluation

The training data contains two-sided conversations. Because the output of our predictors is only relevant for segments with only one person talking, we only run our evaluation on monologuing segments.

For this we define the predicate `is_listening` (is only emitting utterances that are backchannels or silence) and `is_talking` (`= not is_listening`). An example of

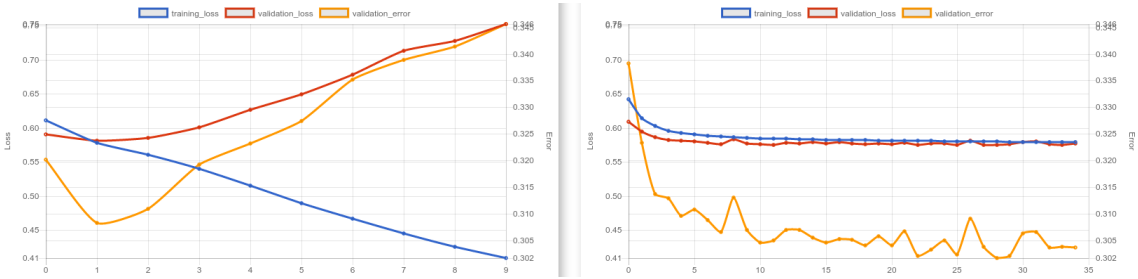

Figure 4.5: The same LSTM network trained without (left) and with (right) L2-Regularization. Note that without regularization the network starts overfitting after two epochs. With regularization training and validation loss mostly stay the same with regularization, and the validation error continues to improve. Training loss is blue, validation loss is red and validation error is orange.

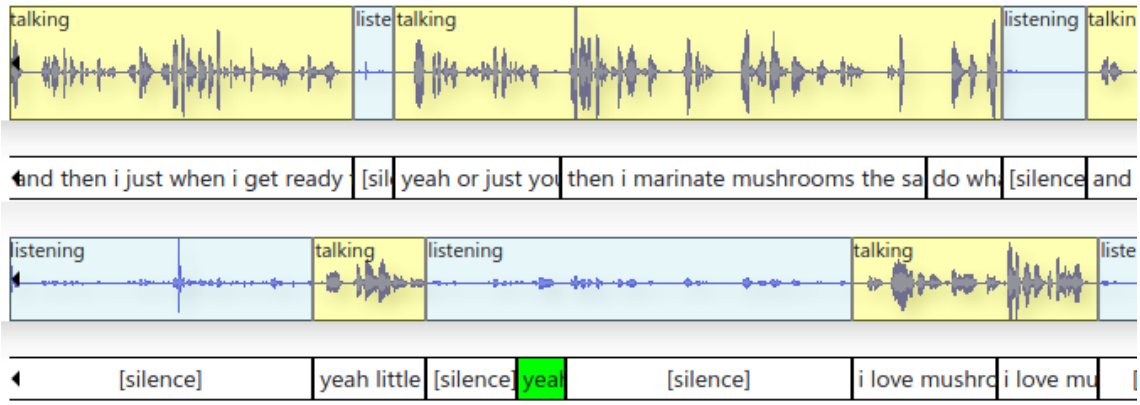

Figure 4.6: A short audio segment showing talking and listening areas. Note that a backchannel from Speaker B in the middle ("yeah") does not classify that area as speaking. At the end both people talk at the same time, so speaker B needs to repeat herself.

this can be seen in fig. 4.6. A monologuing segment is the maximum possible time range in which one person is consistently talking and the other only listening. We only consider segments of a minimum length of five seconds to exclude sections of alternating conversation. The results did not significantly change when adjusting this minimum length between two and ten seconds, though the amount of evaluation data and thus the accuracy of the evaluation values changed.

An interesting aspect is that in our tests the predictors had difficulty distinguishing segments of speech that indicate a backchannel and those that indicate a turn taking (speaker change). Subjectively, this makes sense because in many cases a backchannel can be seen as a replacement for starting to talk more extensively.

5. Results

We use "70 : 35" to denote a network layer configuration of input → 70 neurons → 35 neurons → output. All results in Figure 5.1 use the following setup if not otherwise stated: LSTM, configuration: (70 : 35), input features: power, pitch, FFV, context frame stride: 2, margin of error: 0,ms to +1000,ms. Precision, recall and F1-Score are given for the validation data set.

We tested different context widths. A context width of n ms means we use the range $[-n\,\text{ms}, 0\,\text{ms}]$ from the beginning of the backchannel utterance. The results improved significantly when increasing the context width from our initial value of 500,ms. Performance peaked with a context of about 1500,ms, as can be seen in Figure 5.1a. Longer contexts tended to cause the predictor to trigger too late.

We tested using only every n-th frame of input data. Even though we initially did this for performance reasons, we noticed that training on every single frame has worse performance than skipping every second frame due to overfitting. Taking every fourth frame seems to miss too much information, so performance peaks at a context stride of 2, as can be seen in Figure 5.1b.

We tested different combinations of features. Using FFV as the only prosodic feature performs worse than FFV together with the absolute pitch value. Adding MFCCs does not seem to improve performance in a meaningful way when also using pitch. See Figure 5.1c for more information. Note that using *only* word2vec performs reasonably well, because with our method it indirectly encodes the time since the last utterance.

Figure 5.1d shows a comparison between feed forward and LSTM networks. The parameter count is the number of connection weights the network learns in training. Note that LSTMs have significantly higher performance, even with similar parameter counts.

We compared different layer sizes for our LSTM networks, as shown in Figure 5.1e. A network depth of two hidden layers worked best, but the results are adequate with a single hidden layer or three hidden layers.

In Figure 5.2, our final results are given for the completely independent evaluation data set. We compared the results by Mueller et al. (2015) [**?**] with our sys-

tem. They used the same dataset, but focused on offline predictions, meaning their network had future information available, and they evaluated their performance on the whole corpus including segments with silence and with alternating conversation. We adjusted our baseline and evaluation system to match their setup. As can be seen in Figure 5.2a, our predictor performs significantly better. All other related research used different languages, datasets or evaluation methods, making a direct comparison meaningless.

Figure 5.2b shows the results with our own evaluation method. We provide values for different margins of error used in other research. Subjectively, missing a BC trigger may be more acceptable than a false positive, so we also provide a result with a balanced precision and recall.

Figure 5.1: Results on the Validation Set

(a) Results with various context lengths. Performance peaks at 1500 ms.

Context	Precision	Recall	F1-Score
500 ms	0.219	0.466	0.298
1000 ms	0.280	0.497	0.358
1500 ms	0.305	0.488	**0.375**
2000 ms	0.275	0.577	0.373

(b) Results with various context frame strides

Stride	Precision	Recall	F1-Score
1	0.290	0.490	0.364
2	0.305	0.488	**0.375**
4	0.285	0.498	0.363

(c) Results with various input features, separated into only acoustic features and acoustic plus linguistic features.

Features	Precision	Recall	F1-Score
power, pitch	0.307	0.435	0.360
power, pitch, mfcc	0.278	0.514	0.360
power, ffv	0.259	0.513	0.344
power, ffv, mfcc	0.279	0.515	0.362
power, pitch, ffv	0.305	0.488	**0.375**
word2vec$_{dim=30}$	0.244	0.478	0.323
power, pitch, word2vec$_{dim=30}$	0.318	0.486	0.385
power, pitch, ffv, word2vec$_{dim=15}$	0.321	0.475	0.383
power, pitch, ffv, word2vec$_{dim=30}$	0.322	0.497	**0.390**
power, pitch, ffv, word2vec$_{dim=50}$	0.304	0.527	0.385

(d) Feed forward vs LSTM

Layers	Parameters	Precision	Recall	F1-Score
FF (56 : 28)	40k	0.230	0.549	0.325
FF (70 : 35)	50k	0.251	0.468	0.327
FF (100 : 50)	72k	0.242	0.490	0.324
LSTM (70 : 35)	38k	0.305	0.488	**0.375**

(e) Comparison of different network configurations. Two LSTM layers give the best results.

Layers	Precision	Recall	F1-Score
100	0.280	0.542	0.369
50 : 20	0.291	0.506	0.370
70 : 35	0.305	0.488	**0.375**
100 : 50	0.303	0.473	0.369
70 : 50 : 35	0.278	0.541	0.367

Figure 5.2: Final best results on the evaluation set (chosen by validation set)

(a) Comparison with previous research. Mueller et al. did their evaluation without the constraints defined in section 4.4, so we adjusted our baseline and evaluation to match their setup

Predictor	Precision	Recall	F1-Score
Baseline (random)	0.0417	0.0417	0.0417
Müller et al. (offline) [**?**]	–	–	0.109
Our results (offline, context of −750 ms to 750 ms)	0.114	0.300	**0.165**
Our results (online, context of −1500 ms to 0 ms)	0.100	0.318	0.153

(b) Results with our evaluation method with various margins of error used in other research [**?**]. Performance improves with a wider margin width and with a later margin center.

Margin of Error	Constraint	Precision	Recall	F1-Score
−200 ms to 200 ms		0.172	0.377	0.237
−100 ms to 500 ms		0.239	0.406	0.301
−500 ms to 500 ms		0.247	0.536	0.339
0 ms to 1000 ms	Baseline (random)	0.111	0.0521	0.0708
	Balanced Precision and Recall	0.342	0.339	0.341
	Best F1-Score (only acoustic features)	0.294	0.488	0.367
	Best F1-Score (acoustic and linguistic features)	0.308	0.497	**0.380**

6. Technical Details

We implemented multiple software components to help us understand the data, extract the relevant parts, train the predictors and evaluate the results.

6.1 Web Visualizer

6.1.1 Description

The web visualizer is an interactive tool to display time-based data. It can display audio files as waveforms similar to other popular audio editing software. It can also show arbitrary data that has one or more dimensions per time frame of arbitrary length. We use this to show the features we extract from the audio and use for training of the neural networks, as well as the resulting network outputs. It can show these either as a line graph as seen in fig. 3.2 or as a grayscale color value, as seen in fig. 6.1.

In addition, it can also show time-aligned textual labels either as seperate features or as overlays over the other data. We use this to display the transcriptions below the audio, and to highlight ranges of audio, for example which areas we use as positive and negative prediction areas, or which areas we interpret as "talking" or "listening" as can be seen in fig. 4.6. This allows us to quickly check if the chosen prediction areas make general sense.

The user can choose a specific or random conversation from the training, validation or evaluation data set. Then they can choose to view a combination of the available features from a categorical tree fig. 6.2. The user can zoom into any section of the data and play the audio, the UI will follow the current playback position.

The user can also save the exact current GUI state including zoom and playback position to generate a unique URL that will restore the visualizer to that state when loaded.

Some state presets are also available, such as the default view which will show both channels of a conversation, together with the transcriptions, power, pitch and highlights for the training areas. Another meta preset is the NN output view, which

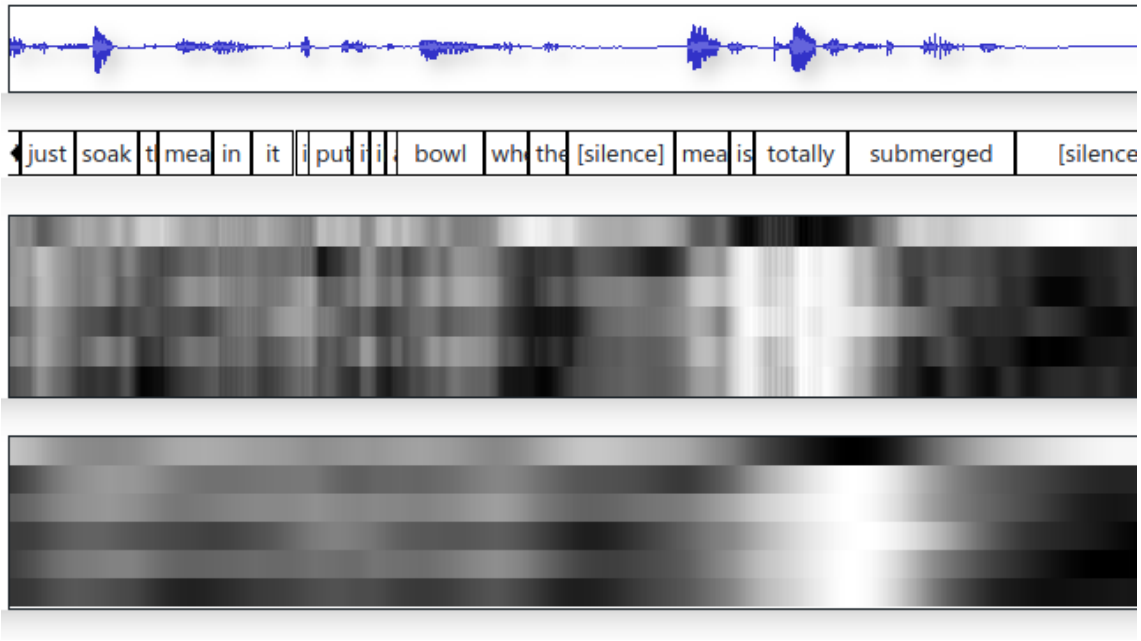

Figure 6.1: The output of a neural network trying to predict backchannel categories for a audio segment. The first category is "No Backchannel", so it is roughly inverse to the other categories (neutral, question, surprised, affirmative, positive). From top to bottom: Audio, Text, Raw NN output, Smoothed NN output. White means higher probabilities, black means lower probabilities.

includes a single audio channel and the raw output, smoothed output and resulting audio track for a trained network, as seen for example in fig. 3.2. The exact configuration can be chosen in a form fig. 6.3. Newly trained networks will automatically be available when the training is finished, allowing quick subjective evaluation of the results.

6.1.2 Implementation

The visualizer is split into two parts, the server side (backend) and the client side (frontend). The server is written in python. It accepts connections via Websockets. It sends a list of available conversation IDs and corresponding features. The client can then select a conversation and dynamically load any combination of the available

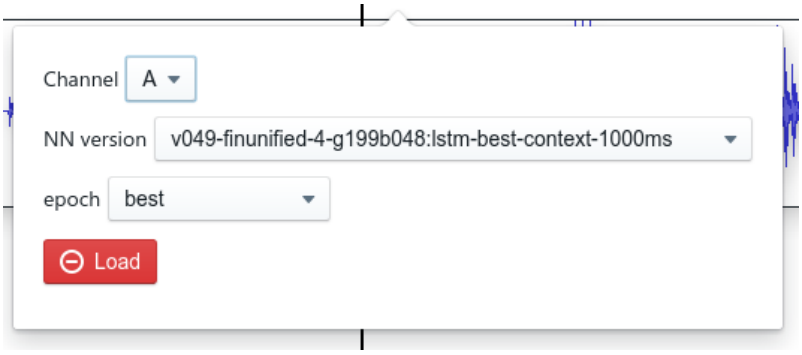

Figure 6.3: Loading the output of the best epoch of specific network for channel A of the current conversation

features, which the server will evaluate on demand while caching the most recently used features. The backend is in a shared code base with the extraction code, so it uses parts of the same code we also use for training and evaluation.

The client is written in TypeScript with MobX and React and runs in a web browser. It uses HTML canvas to draw the visualization of the numerical features. The other visualizations are drawn using DOM elements with CSS Flexbox. The drawn waveform shows the maximum and minimum as a dark blue color, and the root mean square ($\text{rms} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + \cdots + x_n^2)}$ of the signal (similar to the signal power) as a lighter blue overlay. Audio has many datapoints, for example, for 8kHz Audio of 10 minutes, the UI needs to iterate over 5 million data points. When the Audio is played back, the whole view shifts by some fraction of a pixel every 1/60th of a second, which is too much data for a browser client to handle. The trivial rendering of this takes $O(n)$ time for every rerender, where n is the total number of samples that are in the region that is currently on screen. To speed this up, the UI uses an intelligent data structure based on binary trees to cache the values for [min, max, count, RMS] for every range over the indices $[k \cdot 2^l, (k+1) \cdot 2^l - 1]$, where $k \in \mathbb{N}$ is the offset and $l \in \mathbb{N}$ is the "zoom level". Now every access takes only $O(\log n)$ time. For example, when requesting the range $[3, 13]$, the system will use the cached values for $[2, 3], [4, 7], [8, 11], [12]$. This works because min, max and rms can all be combined from partial results without needing to know the individual values. For example $max(5, 6, 1, 4, 7, 1, 5, 3) = max(max(5, 6, 1, 4), 7, max(1, 5, 3))$. For RMS, this can be done by saving both the square sum and element counts for every range.

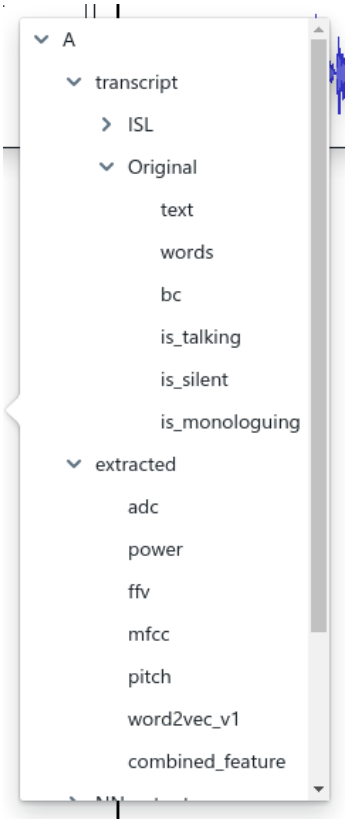

Figure 6.2: Selecting a feature to display in the Web Visualizer.

6.2 Extraction and Learning

We wrote our own parsing toolkit for the transcriptions and word alignments from the SwDA (Jurafsky, Van Ess-Dykema, and others 1997), which are formatted as plain text files. We used the Python interface of the Janus Recognition Toolkit, numpy, scipy and sklearn for feature extraction and filters. We implemented a small learning toolkit on top of Lasagne that reads the extraction and training configuration from a JSON file and outputs the training history as JSON. We used git to track the changes and git tags so every extraction and training output had the exact state of the code attached. This allowed us to easily reproduce different outputs. We also wrote a meta configuration generator, that takes a set of interesting configuration parameters and outputs all the relevant permutations, which can then be extracted, trained and evaluated in parallel.

6.2.1 Automatic caching

Many of the methods for extraction were written as pure functions, which allowed us to create a transparent caching system as a python function decorator that automatically caches the result of expensive functions on disk. For example, consider this function:

```
@functools.lru_cache
@DiskCache
def get_power(adc_filename: str, window_ms: int) -> Feature:
    return Feature(numpy.log10(load_adc(adc_filename).adc2pow))
```

When this function is called, the decorator function `DiskCache` is called first. This function creates a deterministic JSON object of the function name, function source code, and parameters. This JSON object is run through sha256. If a serialized (pickled) file with the hash as the filename exists in the cache folder, that file is loaded. Otherwise, the function is evaluated and its result is saved to the cache folder together with the meta json file. To circumvent issues with thousands of files in a single folder, the first two letters of the hash are used as a subdirectory name, like git does it for its object database. The first time this function is run it takes some time. The subsequent times, it is loaded from disk and not evaluated. Because of automatic disk caching by the OS, this becomes nearly instant for frequently accessed files. In addition, the lru_cache ensures that recently used function results are cached in RAM, which causes an additional speed boost because it circumvents the time otherwise spent unpickling the file.

This way, we do not need to explicitly seperate the extraction from training and write data files, we can simply run the training and the extraction only runs when the extraction parameters (features / context size) changes. For example, if we have already trained a network on the power and pitch features and we now add the FFV feature, the extraction code will only evaluate the FFV feature while using the cached data for power and picth.

We also used joblib, a python library for easy parallelization, to enable the extraction and evaluation processes to use all available CPU cores.

6.3 Evaluation Visualizer

The evaluation visualizer reads all the output JSON files from training and displays the resulting loss graphs as seen in fig. 4.5. It also shows a table of the evaluation results with various parameters for a every neural network sorted from best validation result to worst fig. 6.4. There is also a graph for detailed comparison of the postprocessing and evaluation methods, as shown in fig. 6.5, fig. 6.6, and fig. 6.7. This program is completely client-side and runs in the webbrowser.

The UI can show multiple graphs at the same time, and also shows an overview over all the best results from every network, which can be filtered by RegExp. In addition, it can automatically generate LaTeXtables from the shown filtered outputs for customizable columns.

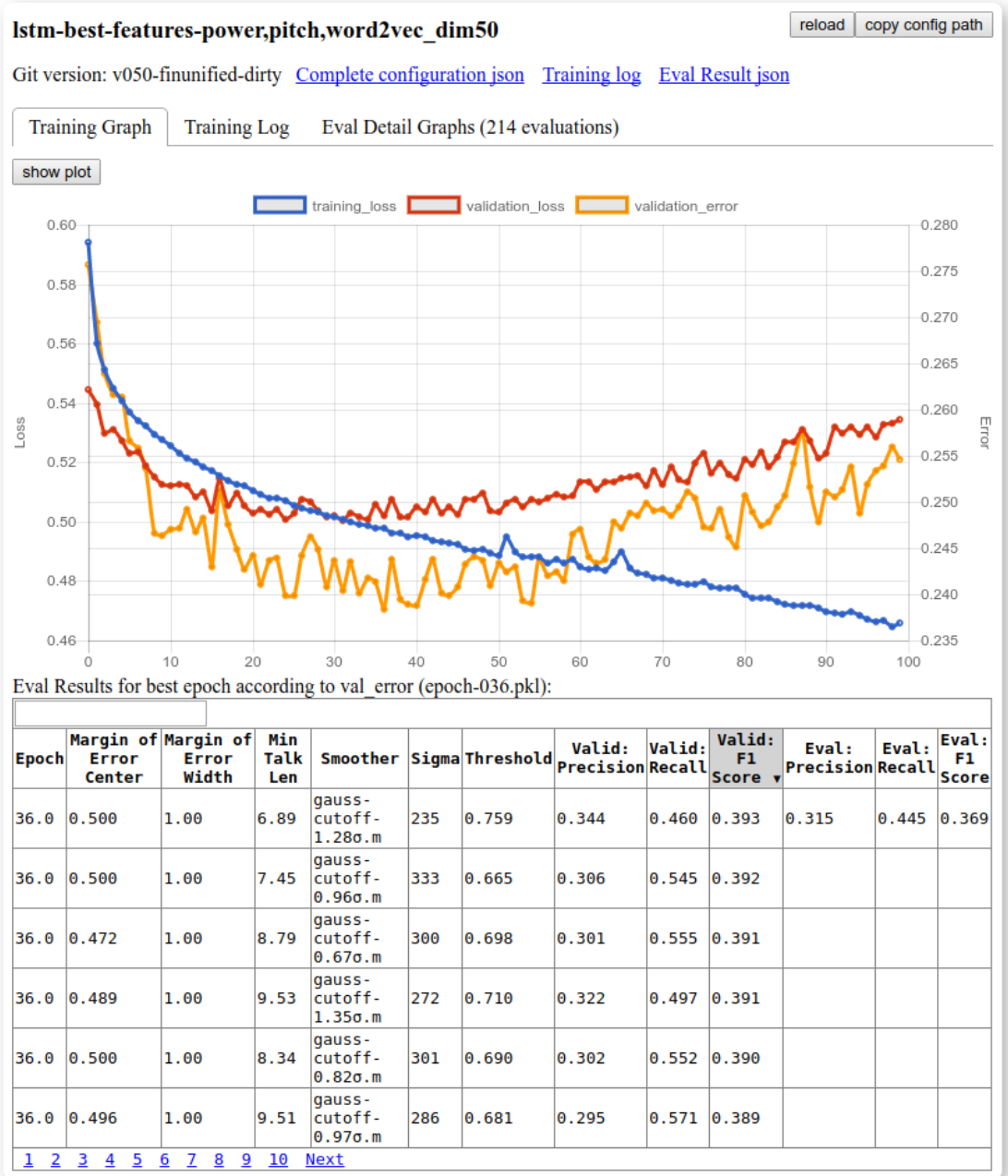

Figure 6.4: Full overview over a single training result, showing the loss graph for checking for overfitting issues and the evaluation results.

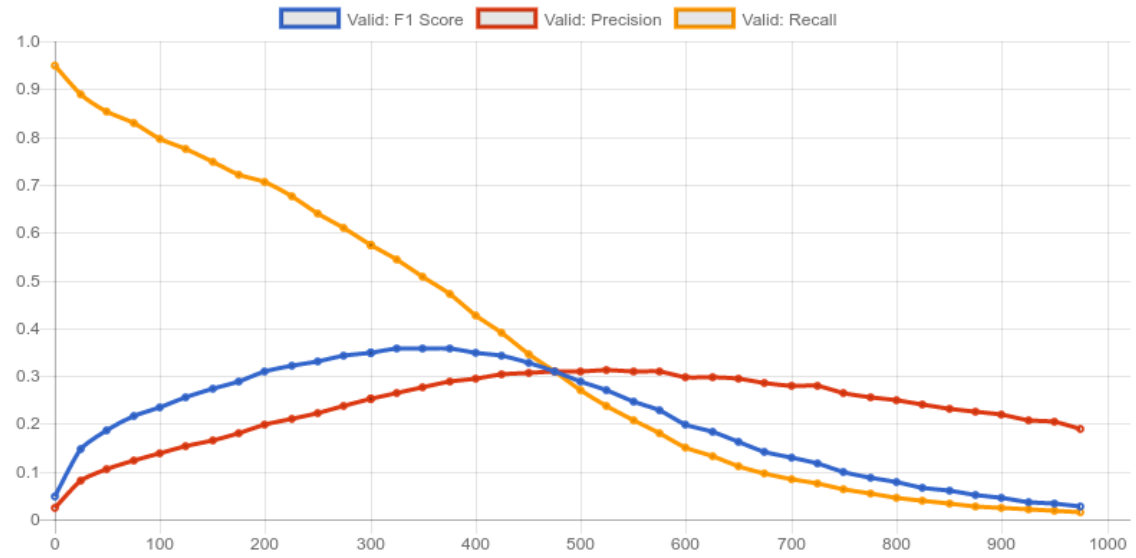

Figure 6.5: Precision, Recall and F1-Score depending on the smoothing sigma used. The F1-Score peaks at $\sigma = 350\,ms$, Precision peaks at $\sigma = 525\,ms$. With lower sigmas, the predictor triggers more often causing the recall to rise

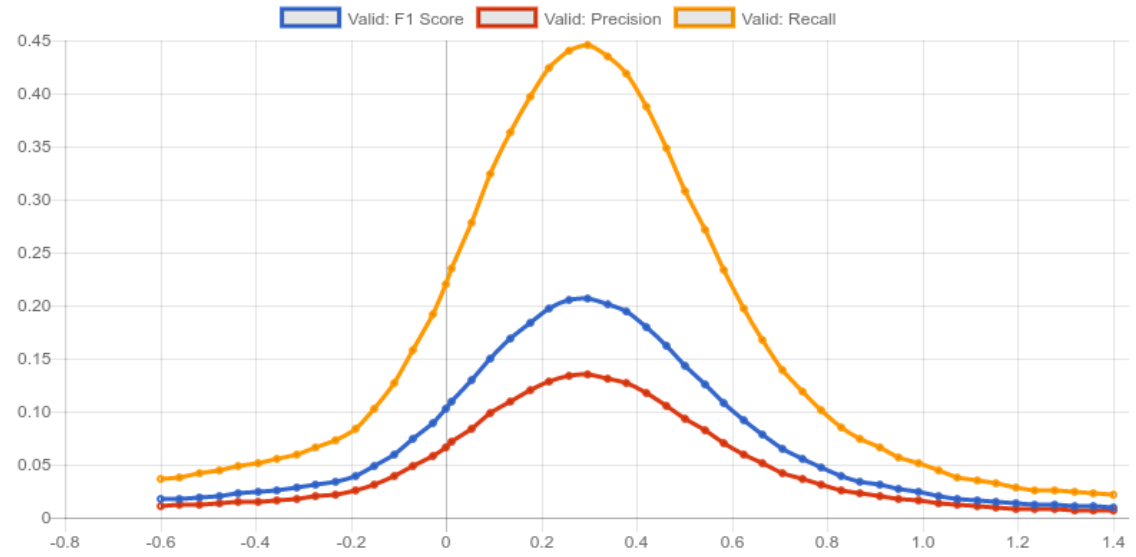

Figure 6.6: Precision, Recall and F1-Score depending on the margin of error center, showing a very clear normal distribution. The center of this distribution depends on the filter sigma and cutoff, meaning we can optimize it depending on the evaluation method.

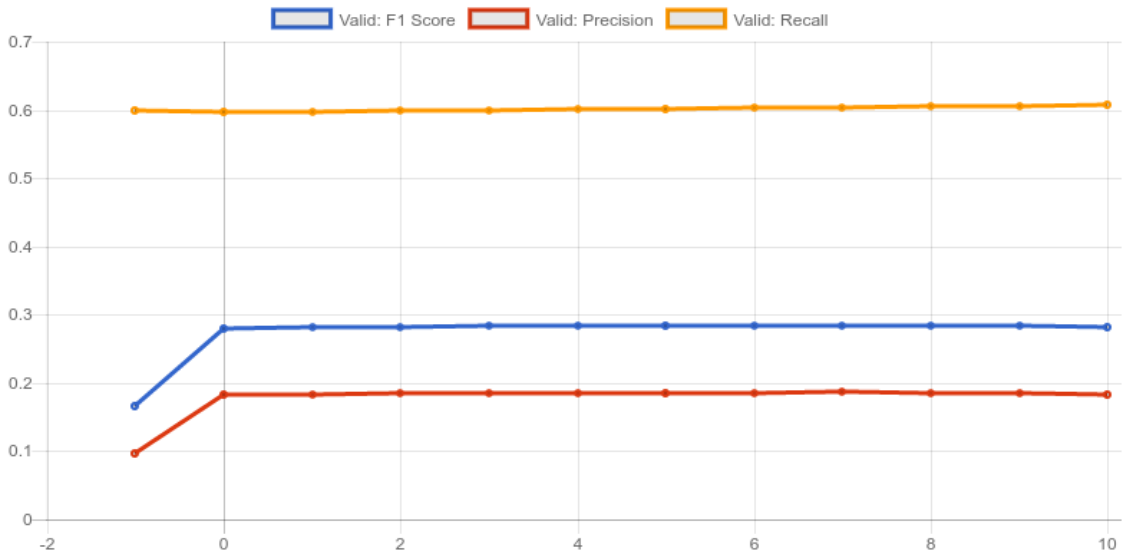

Figure 6.7: Precision, Recall and F1-Score depending on the minimum monologuing segment length. -1 means we evaluate on all data, including silence.

7. Conclusion and Future Work

We have presented... (paper umschreiben)

We only scraped the surface of adding linguistic features via word2vec because it assumes the availability of an instant speech recognizer. Further work is needed to evaluate other methods for adding word vectors to the input features and to analyse problems with our approach. We only tested feed forward neural networks and LSTMs, other architectures like convolutional neural networks may also give interesting results. Our approach of choosing the training areas may not be optimal because the delay between the last utterance of the speaker and the backchannel can vary significantly. A possibility would be to align the training area by the last speaker utterance instead of the backchannel start. Our initial tests of predicting multiple separate categories of BCs did not produce any notable results, further work is needed to analyse whether more context or features are needed to facilitate this.

Because backchannels are a largely subjective phenomenon, a user study would be helpful to subjectively evaluate the performance of our predictor and to compare it with human performance in our chosen evaluation method. Another method would be to find consensus for backchannel triggers by combining the predictions of multiple human subjects for a single speaker channel as described by Huang et al. (2010) as "parasocial consensus sampling" [**?**].

An interesting extension for use in Virtual Assistants would be to also predict speaker changes in addition to backchannels. Our current model already often predicts a backchannel at a time where the speaker stops talking and expects a longer response. Our current model of predicting [No BC, BC] could be extended to predict [No BC, BC, Speaker Change]. This would allow a virtual assistant to give natural sounding backchannel responses and detect when it is supposed to start answering the query.

Bibliography

Cathcart, Nicola, Jean Carletta, and Ewan Klein. 2003. "A Shallow Model of Backchannel Continuers in Spoken Dialogue." In *Proceedings of the Tenth Conference on European Chapter of the Association for Computational Linguistics - Volume 1*, 51–58. EACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics. doi:10.3115/1067807.1067816.

De Kok, Iwan, and Dirk Heylen. 2009. "Multimodal End-of-Turn Prediction in Multi-Party Meetings." In *Proceedings of the 2009 International Conference on Multimodal Interfaces*, 91–98. ACM. http://dl.acm.org/citation.cfm?id=1647332.

de Kok, Iwan, Dirk Heylen, and Louis-Philippe Morency. 2013. "Speaker-Adaptive Multimodal Prediction Model for Listener Responses." In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, 51–58. ICMI '13. New York, NY, USA: ACM. doi:10.1145/2522848.2522866.

de Kok, Iwan, Derya Ozkan, Dirk Heylen, and Louis-Philippe Morency. 2010. "Learning and Evaluating Response Prediction Models Using Parallel Listener Consensus." In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*, 3:1–3:8. ICMI-Mlmi '10. New York, NY, USA: ACM. doi:10.1145/1891903.1891908.

de Kok, Iwan, Ronald Poppe, and Dirk Heylen. 2014. "Iterative Perceptual Learning for Social Behavior Synthesis." *Journal on Multimodal User Interfaces* 8 (3): 231–41. doi:10.1007/s12193-013-0132-1.

Dieleman, Sander, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, and others. 2015. "Lasagne: First Release." August. doi:10.5281/zenodo.27878.

Fujie, Shinya, Kenta Fukushima, and Tetsunori Kobayashi. 2004. "A Conversation Robot with Back-Channel Feedback Function Based on Linguistic and Nonlinguistic Information." In *Proc. ICARA Int. Conference on Autonomous Robots and Agents*, 379–84.

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Godfrey, John, and Edward Holliman. 1993. "Switchboard-1 Release 2." https://catalog.ldc.upenn.edu/ldc97s62.

Huang, Lixing, Louis-Philippe Morency, and Jonathan Gratch. 2010. "Learning Backchannel Prediction Model from Parasocial Consensus Sampling: A Subjective

Evaluation." In *Intelligent Virtual Agents*, 159–72. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-15892-6_17.

———. 2011. "Virtual Rapport 2.0." In *Intelligent Virtual Agents*, edited by Hannes Högni Vilhjálmsson, Stefan Kopp, Stacy Marsella, and Kristinn R. Thórisson, 68–79. Lecture Notes in Computer Science. Springer Berlin Heidelberg. doi:10.1007/978-3-642-23974-8_8.

Jurafsky, Daniel, Carol Van Ess-Dykema, and others. 1997. "Switchboard Discourse Language Modeling Project."

Kalman, Rudolph Emil. 1960. "A New Approach to Linear Filtering and Prediction Problems." *Transactions of the ASME–Journal of Basic Engineering* 82 (Series D): 35–45.

Kingma, Diederik P., and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization," December. http://arxiv.org/abs/1412.6980.

Kitaoka, N., M. Takeuchi, Nishimura R, and Nakagawa S. 2005. "Response Timing Detection Using Prosodic and Linguistic Information for Human-Friendly Spoken Dialog Systems" 20 (3): 220–28. doi:10.1527/tjsai.20.220.

Laskowski, Kornel, Mattias Heldner, and Jens Edlund. 2008. "The Fundamental Frequency Variation Spectrum." *Proceedings of FONETIK 2008*, 29–32.

Levin, Lori, Alon Lavie, Monika Woszczyna, Donna Gates, Marsal Gavaldá, Detlef Koll, and Alex Waibel. 2000. "The Janus-III Translation System: Speech-to-Speech Translation in Multiple Domains." *Machine Translation* 15 (1): 3–25. doi:10.1023/A:1011186420821.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient Estimation of Word Representations in Vector Space," January. http://arxiv.org/abs/1301.3781.

Morency, Louis-Philippe, Iwan de Kok, and Jonathan Gratch. 2010. "A Probabilistic Multimodal Approach for Predicting Listener Backchannels." *Autonomous Agents and Multi-Agent Systems* 20 (1): 70–84. doi:10.1007/s10458-009-9092-y.

Morency, Louis-Philippe, Iwan de Kok, and Jonathan Gratch. 2008. "Predicting Listener Backchannels: A Probabilistic Multimodal Approach." In *Intelligent Virtual Agents*, 176–90. Springer, Berlin, Heidelberg. doi:10.1007/978-3-540-85483-8_18.

Mueller, Markus, David Leuschner, Lars Briem, Maria Schmidt, Kevin Kilgour, Sebastian Stueker, and Alex Waibel. 2015. "Using Neural Networks for Data-Driven Backchannel Prediction: A Survey on Input Features and Training Techniques." In *Human-Computer Interaction: Interaction Technologies*, 329–40. Springer, Cham. doi:10.1007/978-3-319-20916-6_31.

Nishimura, Ryota, Norihide Kitaoka, and Seiichi Nakagawa. 2007. "A Spoken Dialog System for Chat-Like Conversations Considering Response Timing." In *Text, Speech and Dialogue*, 599–606. Springer, Berlin, Heidelberg. doi:10.1007/978-3-540-74628-7_77.

Noguchi, Hiroaki, and Yasuharu Den. 1998. "Prosody-Based Detection of the Context of Backchannel Responses." In *ICSLP*.

Okato, Y., K. Kato, M. Kamamoto, and S. Itahashi. 1996. "Insertion of Interjectory Response Based on Prosodic Information." In*, Third IEEE Workshop on Interactive*

Voice Technology for Telecommunications Applications, 1996. Proceedings, 85–88. doi:10.1109/IVTTA.1996.552766.

Ozkan, D., and L. P. Morency. 2013. "Latent Mixture of Discriminative Experts." *IEEE Transactions on Multimedia* 15 (2): 326–38. doi:10.1109/TMM.2012.2229263.

Ozkan, Derya, and Louis-Philippe Morency. 2010. "Concensus of Self-Features for Nonverbal Behavior Analysis." In *Human Behavior Understanding*, 75–86. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-14715-9_8.

Ozkan, Derya, Kenji Sagae, and Louis-Philippe Morency. 2010. "Latent Mixture of Discriminative Experts for Multimodal Prediction Modeling." In *Proceedings of the 23rd International Conference on Computational Linguistics*, 860–68. COLING '10. Stroudsburg, PA, USA: Association for Computational Linguistics. http://dl.acm.org/citation.cfm?id=1873781.1873878.

Poppe, Ronald, Khiet P. Truong, Dennis Reidsma, and Dirk Heylen. 2010. "Backchannel Strategies for Artificial Listeners." In *Intelligent Virtual Agents*, 146–58. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-15892-6_16.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323 (6088): 533–36. doi:10.1038/323533a0.

Takeuchi, Masashi, Norihide Kitaoka, and Seiichi Nakagawa. 2004. "Timing Detection for Realtime Dialog Systems Using Prosodic and Linguistic Information." In *Speech Prosody 2004, International Conference*.

Theano Development Team. 2016. "Theano: A Python Framework for Fast Computation of Mathematical Expressions." *arXiv E-Prints* abs/1605.02688 (May). http://arxiv.org/abs/1605.02688.

Truong, Khiet P., R. W. Poppe, and D. K. J. Heylen. 2010. "A Rule-Based Backchannel Prediction Model Using Pitch and Pause Information." http://eprints.eemcs.utwente.nl/18627/.

Ward, Nigel. 1996. "Using Prosodic Clues to Decide When to Produce Back-Channel Utterances." In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, 3:1728–31. IEEE. http://ieeexplore.ieee.org/abstract/document/607961/.

Ward, Nigel, and Wataru Tsukahara. 2000. "Prosodic Features Which Cue Back-Channel Responses in English and Japanese." *Journal of Pragmatics* 32 (8): 1177–1207.

Watanabe, Tmio, and Naohiko Yuuki. 1989. "A Voice Reaction System with a Visualized Response Equivalent to Nodding." In *Proceedings of the Third International Conference on Human-Computer Interaction, Vol.1 on Work with Computers: Organizational, Management, Stress and Health Aspects*, 396–403. New York, NY, USA: Elsevier Science Inc. http://dl.acm.org/citation.cfm?id=92158.92234.