HAW HAMBURG

Masterarbeit

Tom Schöner

Detecting Uncertainty in Text Classifications: A Sequence to Sequence Approach using Bayesian RNNs

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Tom Schöner

Detecting Uncertainty in Text Classifications: A Sequence to Sequence Approach using Bayesian RNNs

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr.-Ing. Marina Tropmann-Frick

Eingereicht am: 19. April 2020

Tom Schöner

Thema der Arbeit

Erkennung von Unsicherheit in Textklassifizierungen: Ein Sequence to Sequence Ansatz mit Bayesian RNNs

Stichworte

Unsicherheit, Textklassifikation, NLP, Sequenzanalyse, Bayesian LSTM

Kurzzusammenfassung

Deep Learning Modellen liegen Unsicherheiten zu Grunde, die häufig unbeachtet bleiben oder fehlinterpretiert werden. Die Zusammenführung von Recurrent Neural Networks (RNNs) und Bayesschen Methoden ist eine Möglichkeit, diese Unsicherheiten zu ermitteln. Am Beispiel der Textklassifikation evaluiert diese Arbeit einen neuartigen Ansatz, um Unsicherheitswerte auf Basis von Tokens zu quantifizieren. Ziel ist es, eine transparente Auswertung der Eingabesequenzen zu ermöglichen.

Tom Schöner

Title of Thesis

Detecting Uncertainty in Text Classifications: A Sequence to Sequence Approach using Bayesian RNNs

Keywords

Uncertainty, Text Classification, NLP, Sequence analysis, Bayesian LSTM

Abstract

Uncertainty is fundamentally connected to deep learning but is often disregarded or misinterpreted. Combining Recurrent Neural Networks (RNNs) with Bayesian methods is one way to detect these uncertainties. This work evaluates a novel approach to quantify uncertainty on a token-basis for text classification. The goal is to enable a transparent evaluation of input sequences.

Contents

List of Figures

List of Tables

Abbreviations

Adam Adaptive Moment Estimation.

BBB Bayes by Backprop.

BERT Bidirectional Encoder Representations from Transformers.

BNN Bayesian Neural Network.

BPTT Backpropagation Through Time.

CBOW Continuous Bag of Words.

CNN Convolutional Neural Network.

ELBO Expected Lower Bounds.

GloVe Global Vectors.

GRU Gated Recurrent Unit.

IMDB Internet Movie Database.

KL Kullback-Leibler.

LSTM Long Short-Term Memory.

MC Monte Carlo.

NER Named Entity Recognition.

NLP Natural Language Processing.

NLTK Natural Language Toolkit.

PCA Principal Component Analysis.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

TPU Tensor Processing Unit.

w.r.t. with respect to.

Symbols

Symbol	Description	Domain/Shape
H	hypothesis	
E	evidence	
$P(H)$	prior distribution	
$P(E)$	marginal likelihood	
$P(H \mid E)$	posterior distribution	
$P(E \mid H)$	likelihood	
x	input sequence	$\mathbb{N}^M$
y	true y; one hot encoded	$\mathbb{N}^{M \times K}$
D	data (x, y)	$(\mathbb{N}^M, \mathbb{N}^{M \times K})$
M	maximum sequence length (with padding)	$\mathbb{N}^+$
M'	actual sequence length of x (no padding)	$\mathbb{N}^+$
K	number of labels	$\mathbb{N}^+$
T	number of samples (forward passes)	$\mathbb{N}^+$
g	weights for weighted arithmetic means	${\mathbb{R}_{\geq 0}}^{M'}$
$\hat{y}$	output sequence with T forward passes	$[0,1]^{M' \times K \times T}$
$\bar{\mathbf{x}}_{\hat{y}}$	sequence of sample means for each label	$[0,1]^{M' \times K}$
$\hat{y}^*$	sequence of predictive labels	$\{0, \ldots, K-1\}^{M'}$
$\hat{y}_g$	weighted prediction	$\{0, \ldots, K-1\}$
$U(t, l)$	uncertainty at time step t for label l	$\mathbb{R}_{\geq 0}$
U_g	weighted uncertainty	${\mathbb{R}_{\geq 0}}^K$
$U_g(l)$	weighted uncertainty for label l	$\mathbb{R}_{\geq 0}$

Note: This notation is used in chapter 5 and following. If a definition changes in the context of a chapter, it will be marked as such.

1 Introduction

Machine learning achieves state-of-the-art performance in a great variety of Natural Language Processing (NLP) tasks [1, 2], most prominently sentiment analysis. Neural networks can solve complex NLP problems, although they are considered black-box models. Unfortunately, common approaches fail to quantify epistemic and aleatory uncertainty in predictions. This is aggravated by the fact that uncertainty often has various causes. They include noisy data, data that fits multiple labels, and mislabeled data at training time. Therefore, knowing the underlying uncertainty is advantageous. Its quantification should be part of the model.

Bayesian probability theory fills this gap. It provides methods to quantify uncertainty in a formalized way. Neural networks may implement these Bayesian methods to learn what they do not know. These types of networks are called Bayesian Neural Networks (BNNs). One way to determine the uncertainty in BNNs is to calculate the sample variance of multiple forward passes at inference time.

The following work focuses on quantifying epistemic uncertainty in text classification with Bayesian Long Short-Term Memory (LSTM) neural networks [3] using a novel sequence-to-sequence approach. It analyses the output sequence of a Bayesian LSTM to define intermediate predictions and uncertainty estimations for each input token. This approach supports binary and multiclass classification and is evaluated against Monte Carlo (MC) Dropout [4] and Bayes by Backprop (BBB) [5]. It can identify sensitive terms and uncertain parts in the output sequence while maintaining high accuracy.

1.1 Motivation

> *In classification, predictive probabilities obtained at the end of the pipeline [...] are often erroneously interpreted as model confidence.* [4, Yarin Gal and Zoubin Ghahramani]

In the above quote, Yarin Gal and Zoubin Ghahramani address the problem of falsely interpreting the predictive probability as a measurement of uncertainty. To solve that issue, they combine Bayesian methods with machine learning. In recent years, the community around Bayesian neural networks has been gaining traction with many new publications [6]. Yarin Gal [4, 7], Charles Blundell [5, 8], and others are leading researchers who aim to find practical ways to improve model transparency by introducing uncertainty estimations.

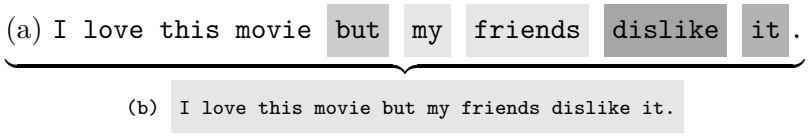


Figure 1.1: Local (a) and global (b) uncertainty over a sequence. High uncertainty is visualized with darker background colors. The first line estimates uncertainty per token. The second line estimates uncertainty per sequence. This example was created manually for demonstration purposes.

This Bayesian way of thinking applies to uncertainty estimations on a token-basis. Typically, classifications with Bayesian neural networks capture uncertainty by reducing the sequence information to one estimation. Fig. 1.1 shows an uncertainty distribution (a) over a sequence. It also shows a single uncertainty estimation (b) that applies to the whole sequence. (b) does not reveal the source of uncertainty. On the other hand, method (a) can locate uncertain sections because it knows the uncertainty of individual tokens.

These uncertainty estimations over the input sequence can be the missing link to locate uncertainty in text classification. Interpretability is an important aspect that builds trust in a model. It is this work's main motivation.

1.2 Goals

This work introduces a novel sequence-to-sequence approach to quantify uncertainty in text classifications. The goals are threefold.

1. The first goal is to reliably detect confident and uncertain sections in text classifications. Compared to other approaches with no or limited uncertainty estimations, predictions should be more transparent regarding the exact source of uncertainty.

2. The sequence-to-sequence approach should work with common classification datasets without the need for additional data labeling.

3. Finally, a visualization tool with an easy-to-use interface for fast evaluation should be created. The application should visualize token-based uncertainties.

1.3 Structure

The following work is structured such that the first chapters provide the theoretical background for all subsequent chapters.

Chapter 2: *A theoretical look at language processing with text preprocessing and word embeddings.* Besides conventional word embeddings, it demonstrates how Bidirectional Encoder Representations from Transformers (BERT) [9] can be used as a contextual word embedding (section 2.2.2). Recurrent Neural Networks (RNNs) are introduced (section 2.3) with a focus on Long Short-Term Memory (LSTM) neural networks (section 2.3.2). It is explained why LSTMs are superior to plain RNN implementations.

Chapter 3: *Quantifying uncertainty in neural networks.* First, section 3.1 distinguishes between aleatory (data) and epistemic (model) uncertainty. Sections 3.2 and 3.3 introduce Bayesian methods and variational Bayesian inference. Variational Bayesian inference is a practical approach to perform approximate probability computation. Sections 3.4 and 3.5 describe two ways to integrate variational Bayesian inference into neural networks and RNNs. The last section 3.6 addresses the pitfalls of approximate Bayesian methods.

Chapter 4: *Similar approaches and related publications.* This chapter mentions similar methods like Named Entity Recognition (NER) [10, 11] and transparency oriented methods [12].

Chapter 5: *The sequence-to-sequence concept.* Section 5.1 outlines the data preparation process. Section 5.2 describes the required LSTM architecture for token-based uncertainty quantification. Sections 5.3 and 5.4 provide a theoretical explanation for transforming multiple samples from the Bayesian LSTM into token-based predictions and uncertainty quantifications. This process is then illustrated with an exemplary calculation in section 5.5.

Chapter 6: *Experiments*. Section 6.1 lists essential frameworks and libraries. Section 6.2 introduces the Keras and BERT word embedding. Section 6.3 shows the implementation of Monte Carlo (MC) Dropout and Bayes by Backprop (BBB) for LSTMs in Tensorflow. The experiments are divided into binary (section 6.5) and multiclass (section 6.6) classification. The last section 6.7 shows the proof of concept application that supports multiple models.

Chapter 7: *Evaluation*. The evaluation starts with a model comparison that focuses on weighted uncertainty (eq. 5.11). Sections 7.2 and 7.3 evaluate the effect of stopwords and sensitive terms on uncertainty. Whether this implementation complies with the goals listed in section 1.2, is discussed in section 7.4. Section 7.5 lists known complications.

Chapter 8: *Summary and outlook*. Section 8.1 summarizes this work. The last section 8.2 lists ideas for future research, such as extending the Bayes by Backprop (BBB) implementation.

2 Language Processing with RNNs

In Germanic and Romance languages such as English and German, words are the fundamental elements to express ideas. Words either have contextual meaning or they are function words, such as `a` or `the`. Spoken languages are rapidly evolving. This makes capturing the nuances of a language challenging for computer programs. Therefore, Natural Language Processing (NLP) is the focus of many sophisticated studies in machine learning (e.g. [13]).

The required steps and methods to perform text classification with Recurrent Neural Networks (RNNs) are outlined in this chapter. Section 2.1 describes the process of preparing text documents for text classification. The goal is to identify and group similar words, which will effectively result in a reduced vocabulary. Section 2.2 explains how to transform a word into a word vector. This transformation step is crucial because it enables the neural network to interpret words by providing an abstract representation. Subsequently, section 2.2.2 outlines the functionality of the recently developed language representation method Bidirectional Encoder Representations from Transformers (BERT) and its connection to word embeddings.

Section 2.3 introduces text processing with RNNs. Even though an RNN tries to memorize the context of previous words in a sequence, in practice the naive implementation of an RNN often fails to do so. This is known as the vanishing gradient problem, described in section 2.3.1. The more intricate Long Short-Term Memory (LSTM) architecture tries to solve this problem with additional states and control flow gates (section 2.3.2).

2.1 Text Preprocessing

Text is the primary data type in Natural Language Processing (NLP). Informal texts like chat messages or forum entries are noisy and require text preprocessing to be interpretable by subsequent tasks. After the text preprocessing step, neural networks with

word embeddings (section 2.2) will work with simplified and normalized representations of words. This reduces complexity. Apart from the removal of stopwords and punctuation, this is achieved by applying various methods of which lemmatization, stemming, and normalization are presented in this section. To work with individual words, the text is separated by spaces and punctuation.

Stemming and lemmatization [14, 15] are language-dependent approaches to find root forms of words. They aim to reduce inflectional word forms including grammatical conjugations. The result may not reflect the true root form.

Stemming is a relatively simple yet effective process to replace common characters with fixed character sequences including the empty sequence. For example, the Porter stemmer [16] specializes in word suffixes. Among other things, it replaces all `ies` suffixes with the `ss` suffix and removes the `s` suffix. Therefore, `apples` is mapped to `apple` and `ponies` to `poni`, demonstrating the creation of a non-existing root form.

On the other side, lemmatization requires language-dependent information to perform root form reduction. Based on a dictionary, it could resolve more complex grammatical conjugations. For example, the present indicative `are` points to the present infinitive `be`.

To simplify texts further, the process of normalization can greatly reduce noise. Normalization techniques may group numerical values under a single value, resolve phonetic spelling in informal texts (`good n8` to `good night`) or emphasized expressions like `hiiii`. In contrast to lemmatization, normalization even works on unknown words. Sophisticated techniques might also correct spelling [17].

2.2 Word Embeddings

Unfortunately, neural networks do not directly understand string representations of words. As explained in section 2.2.1, a transformation step is necessary to create machine-readable word representations.

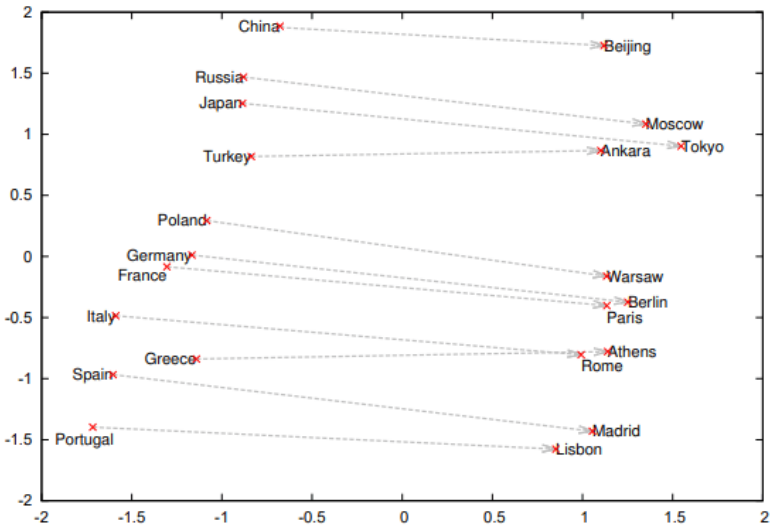


Figure 2.1: Relations between countries and their capitals in vector space, reduced from 1000 dimensions to two dimensions with PCA. [18]

2.2.1 Word Vectors

A word vector is an N-dimensional representation of a word. Typically, a word vector is located in a semantic vector space [19, 20] where the distance and/or angle between word vectors indicate their relation. For example, the distance between the word vector representations for `daughter` and `mother` should roughly be the same as for `mother` and `grandmother`. Fig. 2.1 demonstrates the relation between countries and their capitals in semantic vector space. Each vector may then be listed in a matrix with dimensions $M \times N$ where M is the size of the vocabulary.

Visualizing the high-dimensional vector space in a humanly readable two- or three-dimensional space requires a data projection. Dimensionality reduction algorithms, such as Principal Component Analysis (PCA), apply the data projection.

Word vectors do not strictly represent correlations. Although highly inefficient, it is possible to map each word to a scalar (a one-dimensional word vector). Doing so disregards all semantic meaning. Another method is one-hot encoding in which each word is represented as an independent sparse vector $[0, 0, ..., 1, ..., 0]$. The dimension is proportional to the size of the vocabulary, which causes high memory usage. More sophisticated approaches include the following [21]:

- **Word2Vec** [18] with Skip-Gram or Continuous Bag of Words (CBOW). Both are shallow, two-layer neural network approaches. Word2Vec with CBOW uses context words to predict the current word, given a window size. The Skip-Gram approach does the exact opposite. After training, the CBOW embedding matrix is extracted from the weights between the hidden layer and the output layer. The Skip-Gram embedding matrix is extracted from the weights between the input layer and the hidden layer. Similar words are grouped, measured by their cosine similarity.
- **Global Vectors (GloVe)** [20] is another unsupervised method. Given a corpus, GloVe creates a word-to-word co-occurrence matrix. The training objective is to minimize the ratios between co-occurrence probabilities and their word vector counterparts. The authors have published multiple official pre-trained word vectors based on Wikipedia, Twitter, and other resources. The dimensionality of word vectors in the pre-trained models ranges from 50 to 300 dimensions.

Obtaining a more dynamic and context-sensitive word embedding is possible by using BERT (section 2.2.2).

2.2.2 BERT

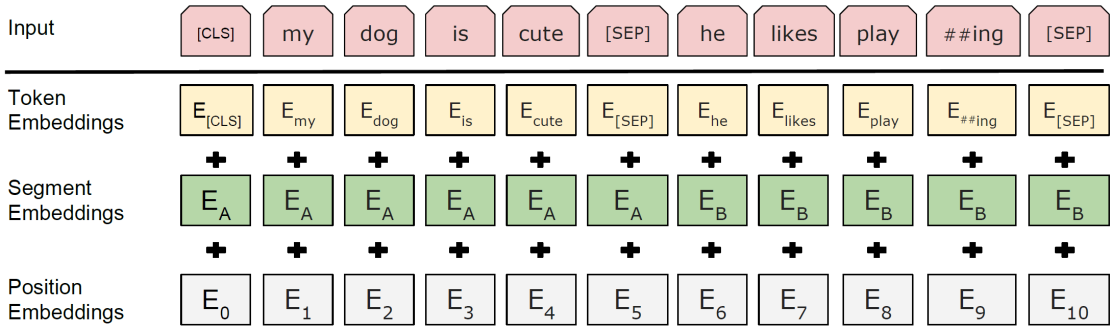


Figure 2.2: BERT input representation. [9]

Bidirectional Encoder Representations from Transformers (BERT) [9] is a new language representation method developed at Google. It claims state-of-the-art performance in various NLP tasks. Following Vaswani et al.'s [22] implementation for transformers, its architecture is a multi-layered bidirectional transformer with self-attention mechanisms. Its bidirectional nature enables BERT to learn contextual relations. For example, the word `bank` has a different meaning in `"A woman sits on a bank"` than in `"The bank will open in two hours"`.

BERT is trained without supervision, using masked language models and next sentence prediction. In masked language models, masking random words of a known sequence aids in predicting missing words. Next sentence prediction determines whether a given sentence A is followed by another sentence B. Given a large enough corpus, one can easily generate the data required by both training methods.

As shown in fig. 2.2, BERT transforms the textual input sequence into a token sequence. Each token sequence starts with a special `[CLS]` token to mark its beginning. Further, the `[SEP]` token indicates a separation between sentences. In the context of BERT, a sentence does not strictly follow its language representation and can be multiple English sentences.

As the name suggests, the architecture of BERT consists of multiple stacked encoder layers, also called transformer blocks. These layers apply self-attention [22].[1] BERT does not use a decoder. Each encoder layer returns a vector for each element in the sequence that is used as the input for the next encoder layer. In case of the last encoder layer, it is used as the output. The model's *hidden size* parameter determines the size of the output vector (BERT base models by Google use 768 dimensions). In comparison to other word embeddings, the hidden size corresponds to the embedding size.

Google released multiple regular- and large-sized pre-trained models for the English and Chinese language. They also released multilingual versions. The pre-trained models can be fine-tuned to improve the results in problem-specific tasks. [23, 24]

2.3 Recurrent Neural Networks

Contextual knowledge about past events can greatly improve future predictions. In spoken languages, the beginning of a sentence will most likely influence all subsequent words. This observation is the core idea behind Recurrent Neural Networks (RNNs) [25].

Given an input sequence x of length $\mathbf{T}$ (eq. 2.1), an RNN calculates its output $\hat{y}$ by performing the same set of instructions for each time step $t \in 1, \dots T$ and element $x_t \in x$.

[1] BERT's base model uses 12 encoder layers and its large model 16 encoder layers.

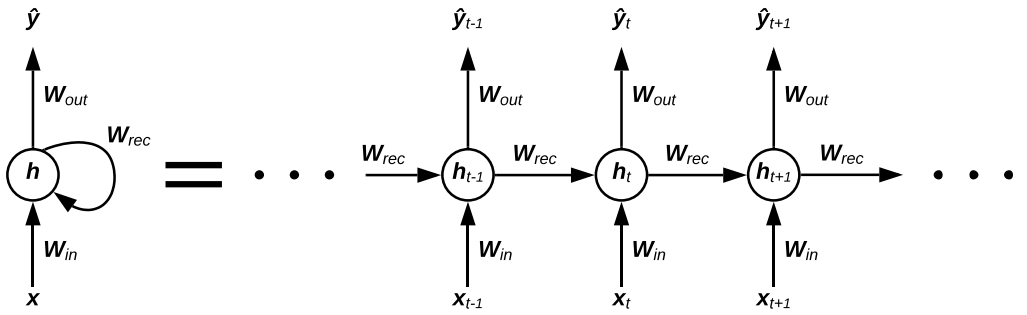


Figure 2.3: Two representations of an RNN. The left side shows an RNN as a loop, the right side shows its unfolded representation.

Alongside x_t, the hidden state h_{t-1} is passed on to the current calculation step. The output is the hidden state h_t. It encapsulates the information about past time steps.

$$x = (x_1,\ x_2,\ ...,\ x_{\mathbf{T}}) \tag{2.1}$$

The recurrent architecture of RNNs can be represented as a loop or unfolded (fig. 2.3). Therefore, the actual outputs of an RNN are $\hat{y}_1, ..., \hat{y}_{\mathbf{T}}$. In classification problems, it is common to focus on the last output $\hat{y}_{\mathbf{T}}$.

At time step t the hidden state h_t and output $\hat{y}_t$ is formally [26, 27] defined as:

$$h_t = \sigma(\mathbf{W}_{in}x_t + \mathbf{W}_{rec}h_{t-1} + \mathbf{b}) \tag{2.2}$$

$$\hat{y}_t = \mathbf{W}_{out}h_t \tag{2.3}$$

where σ is the activation function, $\mathbf{W}_{in}$ is the input weight matrix, $\mathbf{W}_{rec}$ is the recurrent weight matrix of the hidden layers, $\mathbf{W}_{out}$ is the weight matrix of the output, and $\mathbf{b}$ is the bias. More generally, given a dynamical system $F(h_{t-1}, x_t, \theta)$, θ describes all trainable parameters of F, i.e., $\mathbf{W}_{in}$, $\mathbf{W}_{rec}$, $\mathbf{W}_{out}$, and $\mathbf{b}$.[2] The weight matrices are independent of time step t, which greatly reduces the number of trainable parameters within the network.

To train an RNN, a variation of backpropagation called Backpropagation Through Time (BPTT), is used. The partial (eq. 2.4) and total cost (eq. 2.5) are determined by the

[2]The notation may vary in other papers, e.g., $\mathbf{W}_{rec}$ is often denoted as $\mathbf{U}$.

loss function $\mathcal{L}$:

$$\mathcal{E}_t = \mathcal{L}(x_t, \hat{y}_t) \tag{2.4}$$

$$\mathcal{E} = \sum_{t=1}^{\mathbf{T}} \mathcal{E}_t \tag{2.5}$$

After obtaining the costs for each time step in eq. 2.5, BPTT unfolds the RNN and calculates the gradients with respect to (w.r.t.) θ as a sum.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{t=1}^{\mathbf{T}} \frac{\partial \mathcal{E}_t}{\partial \theta} \tag{2.6}$$

As the main goal of RNNs is to learn long-term relations between data points of the same sequence, the vanishing gradient problem has to be addressed (section 2.3.1). With this in mind, new RNN architectures, particularly Long Short-Term Memory (LSTM) [3] and Gated Recurrent Unit (GRU) [28], were introduced. This work focuses on LSTMs.

2.3.1 Vanishing Gradient Problem

Sigmoid, tanh, and Rectified Linear Unit (ReLU) are common activation functions. They introduce non-linearity in neural networks. Updating weights via backpropagation requires each activation function's derivative. More specifically, backpropagation uses the chain rule to calculate the gradient of the loss function over the weights. Propagating the gradient back through each layer may result in a decreasing gradient.

In RNNs, the recurrent weight matrix used for all time steps is the primary cause of a vanishing gradient. The gradient component $\frac{\partial \mathcal{E}_t}{\partial \theta}$ (eq. 2.6) is partially calculated by the temporal component $\frac{\partial x_t}{\partial x_k}$ with t and k being time steps, and $t > k$ [27, eq. (4)]. Intuitively, $\frac{\partial x_t}{\partial x_k}$ propagates the error from time step t back to the earlier time step k:

$$\frac{\partial x_t}{\partial x_k} = \prod_{i=k+1}^{t} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{rec}^T diag(\sigma'(x_{i-1})) \tag{2.7}$$

Here, the multiplicand $\mathbf{W}_{rec}$ leads to the vanishing gradient problem. The weight matrix effectively becomes an exponential decay factor with $t-k$ as the exponent if its eigenvalue is less than 1.

2.3.2 Long Short-Term Memory

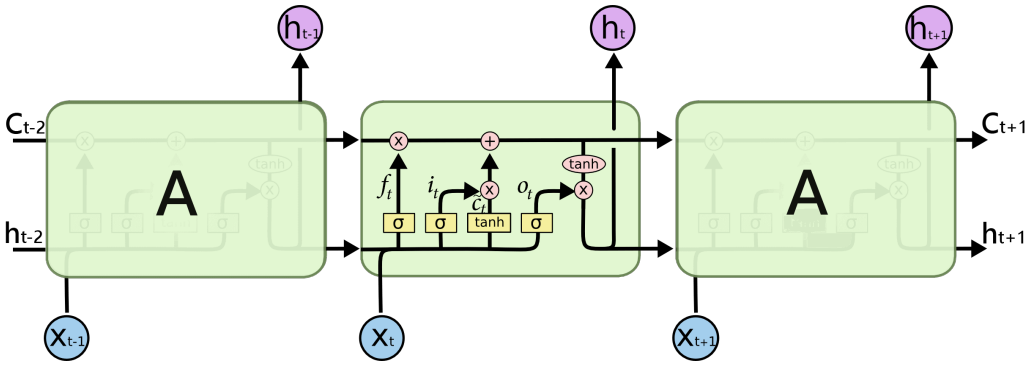


Figure 2.4: The repeating cell in an LSTM [29].

Long Short-Term Memory (LSTM) networks [3] solve the problem of vanishing gradients and are designed to understand long-term dependencies in sequences. They are a good fit for language modeling tasks because context-related words or sections may be separated over short or long distances. The recurrent unit in LSTMs is often referred to as a cell. In addition to the hidden state in plain RNNs, LSTMs cells have a second state: the cell state c_t. Its purpose is to store relevant information and forward said information to all upcoming cells.

LSTMs are more complex than regular RNNs because they introduce the concept of gates. These gates control the data flow in and from cells (fig. 2.4). This work utilizes the common LSTM architecture with a forget gate (eq. 2.8) [30]:

- The **input gate** i_t adds new information x_t to the cell state, regulated by $\widetilde{c}_t$. Depending on $\widetilde{c}_t$ and i_t, only a fraction of x_t or nothing at all is added.
- The **forget gate** f_t removes information from the cell state based on the last hidden state and new information x_t.
- The **output gate** o_t determines what information of the cell state should be added to the output at time step t. The output equals the hidden state h_t.

A gate's ouput depends on two weight matrices $\mathbf{W}_{i,f,o,c}$ and $\mathbf{U}_{i,f,o,c}$, a bias $\mathbf{b}_{i,f,o,c}$, and an activation function. The subscript i, f, o, c corresponds to the gate. Therefore, all

gates are traniable:

$$i_t = \sigma(\mathbf{W}_i h_{t-1} + \mathbf{U}_i x_t + \mathbf{b}_i) \tag{2.8}$$

$$f_t = \sigma(\mathbf{W}_f h_{t-1} + \mathbf{U}_f x_t + \mathbf{b}_f) \tag{2.9}$$

$$o_t = \sigma(\mathbf{W}_o h_{t-1} + \mathbf{U}_o x_1 + \mathbf{b}_o) \tag{2.10}$$

$$\widetilde{c}_t = \tanh(\mathbf{W}_c h_{t-1} + \mathbf{U}_c x_t + \mathbf{b}_c) \tag{2.11}$$

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c}_t \tag{2.12}$$

$$h_t = o_t * \tanh(c_t) \tag{2.13}$$

Here, $*$ is elementwise multiplication, t is the time step and σ is the sigmoid function. The total number of trainable parameters is $4(nm+n^2+n)$ with m being the size of the output vector and n being the size of the input vector x. Thus $\mathbf{W} \in \mathbb{R}^{n \times m}$, $\mathbf{U} \in \mathbb{R}^{n \times n}$, and $\mathbf{b} \in \mathbb{R}^n$.

The forget gate alleviates the vanishing gradient problem by preserving the recursive derivative of c_t. Think of the forget gate as a control mechanism; it is a trainable function after all. Only if it decides to forget, the gradient will vanish [31].

3 Quantifying Uncertainty

Missing or incomplete data aggravates the formulation of decisions. Unfortunately, it is the norm rather than the exception. Mathematically defining ways to quantify these uncertainties is crucial to making reliable predictions, especially in machine learning.

In section 3.1 the concept of uncertainty is further divided into aleatory and epistemic uncertainty. Sections 3.2 and 3.3 introduce the Bayesian world view, which is based on prior beliefs and updating said beliefs when new data becomes available. Section 3.4 presents the Bayes by Backprop (BBB) method to model the weights in neural networks as probability distributions. Utilizing dropout at inference time is another way to estimate uncertainty in neural networks. It is illustrated in section 3.5.

3.1 Aleatory and Epistemic Uncertainty

Whether the data is noisy or the underlying architecture has an incomplete understanding of the data, variance in the results is a direct consequence of those uncertainties. Uncertainty itself can be divided into the following two categories [32, 33]:

- **Aleatory uncertainty** refers to data uncertainty and is a measure of noise in observations or naturally occurring randomness. Detecting radio waves from a distant radio tower is a good example of aleatory uncertainty. Background noise eventually distorts data transmitted over any distance. Accumulating more data does not reduce aleatory uncertainty for the receiver. Aleatory uncertainty can be further subcategorized as homoscedastic and heteroscedastic uncertainty. The former is independent of the input, whereas the latter depends on the input of the model.
- **Epistemic uncertainty** refers to model uncertainty, usually within the model parameters. High epistemic uncertainty implies that the model's knowledge of

the data is insufficient. A neural network needs a tremendous amount of data to capture all important cases. Some patterns may still be unknown, causing epistemic uncertainty in the neural network. Other examples are the use of a suboptimal kernel in a Gaussian process and a neural network optimizer getting stuck at a local minimum.

Both types of uncertainty can be quantified under the law of total variance $\mathrm{Var}(y)$ for input x and output y [34]:

$$\mathrm{Var}(y) = \mathbb{E}[\mathrm{Var}(y \mid x)] + \mathrm{Var}(\mathbb{E}[y \mid x]) = U_d(y \mid x) + U_m(y \mid x) \tag{3.1}$$

$$\text{with} \quad U_d(y \mid x) = \mathbb{E}[\mathrm{Var}(y \mid x)] \tag{3.2}$$

$$U_m(y \mid x) = \mathrm{Var}(\mathbb{E}[y \mid x]) \tag{3.3}$$

where U_d is data (aleatoric) uncertainty and U_m is model (epistemic) uncertainty.

A more practical way to look at aleatoric and epistemic uncertainty is given by [6]:

$$\mathrm{Var}(y) \approx \underbrace{\frac{1}{T}\sum_{t=1}^{T} diag(\hat{p}_t) - \hat{p}_t^{\otimes 2}}_{\text{aleatoric}} + \underbrace{\frac{1}{T}\sum_{t=1}^{T}(\hat{p}_t - \bar{p})^{\otimes 2}}_{\text{epistemic}} \tag{3.4}$$

with $v^{\otimes 2} = vv^\intercal$. $\hat{p}_t$ is the tth randomly drawn sample from the neural network with the softmax function as the output activation function. $\bar{p}$ denotes the sample mean.

3.2 Bayesian Inference

Given some prior knowledge E of an event or a hypothesis H, the probability of H occurring can be formalized as:

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)} \tag{3.5}$$

also known as Bayes' theorem [35]. Its implications provide a broad range of tools in probability theory and statistics. The list of symbols at the beginning of this work outlines all symbols in eq. 3.5.

Bayesian inference is a continuous process to determine the parameters of a probability distribution or population. It closely incorporates Bayes' theorem. Whenever new data

becomes available, the prior beliefs will be updated, and the posterior becomes the new prior. Now the prior includes observations about recently obtained data. The prior beliefs are expressed through the prior distribution. A uniform distribution may be used if no prior knowledge exists.

The notation slightly changes in comparison to Bayes' theorem (eq. 3.6). $\mathbf{w}$ denotes the hypothesis H where $\mathbf{w}$ represents the set of all parameters. $\mathcal{D}$ denotes E. This notation is used in Bayesian neural networks and in the following sections:

$$P(\mathbf{w} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathbf{w})P(\mathbf{w})}{P(\mathcal{D})} = \frac{P(\mathcal{D} \mid \mathbf{w})P(\mathbf{w})}{\int_{\mathbf{w}} P(\mathcal{D}, \mathbf{w})\ d\mathbf{w}} \tag{3.6}$$

$$P(\mathbf{w} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \mathbf{w})P(\mathbf{w}) \tag{3.7}$$

The posterior is the probability distribution of the–currently unknown–parameters $\mathbf{w}$, given the observed data $\mathcal{D}$ in eq. 3.6. According to eq. 3.7, it is also proportional to the likelihood times the prior. $P(\mathcal{D})$ is a normalization constant that is hard to calculate. It is not needed when the proportionality characteristic suffices to solve the problem, i.e. in maximum likelihood estimations with uniform priors.

3.3 Variational Bayesian Inference

To overcome the computational overhead of Bayesian inference, approximate analytical methods were introduced [36, 37]. A new variational distribution q, hereafter called variational posterior, approximates the true posterior $P(\mathbf{w} \mid \mathcal{D})$. Simply calculating the true posterior requires integrating the marginal likelihood over the possibly high dimensional parameters $\mathbf{w}$ (eq. 3.6), which is intractable. Thus, a method to construct the variational posterior q is needed. In variational Bayesian inference, the Kullback-Leibler (KL) divergence between the true and variational posterior is the preferred measurement of similarity (closeness) between two probability distributions to quantify the proximity $q(\mathbf{w} \mid \theta) \approx P(\mathbf{w} \mid \mathcal{D})$:

$$\mathrm{KL}[q(\mathbf{w} \mid \theta) \mid\mid P(\mathbf{w} \mid \mathcal{D})] = \int q(\mathbf{w} \mid \theta) \log \frac{q(\mathbf{w} \mid \theta)}{P(\mathbf{w} \mid \mathcal{D})} d\mathbf{w} \tag{3.8}$$

The parameters θ are the result of variational learning to minimize the KL divergence. Although this does not immediately solve the intractable marginal likelihood integration problem, it does allow for further adjustments. Following the work presented in [5], the

optimization of the KL divergence ultimately derives the cost function $\mathcal{F}(\mathcal{D}, \theta)$, also known as the Expected Lower Bounds (ELBO):

$$\theta^* = \arg\min_\theta \int q(\mathbf{w} \mid \theta) \log \frac{q(\mathbf{w} \mid \theta)}{P(\mathbf{w})P(\mathcal{D} \mid \mathbf{w})} d\mathbf{w} \tag{3.9}$$

$$= \arg\min_\theta \mathrm{KL}[q(\mathbf{w} \mid \theta) \mid\mid P(\mathbf{w})] - \mathcal{L}(\mathcal{D}, \theta) \tag{3.10}$$

$$\mathcal{F}(\mathcal{D}, \theta) = \mathrm{KL}[q(\mathbf{w} \mid \theta) \mid\mid P(\mathbf{w})] - \mathcal{L}(\mathcal{D}, \theta) \tag{3.11}$$

$$\text{where } \mathcal{L}(\mathcal{D}, \theta) = \mathbb{E}_{q(\mathbf{w} \mid \theta)}[\log P(\mathcal{D} \mid \mathbf{w})] \tag{3.12}$$

The transformation step from eq. 3.9 to eq. 3.10 extracts the likelihood cost $\mathcal{L}$ so that the KL divergence does not depend on data $\mathcal{D}$ anymore. The first part in eq. 3.11 is called the complexity cost. The authors of [5] describe it as a "*[...] trade-off between satisfying the complexity of the data $\mathcal{D}$ and satisfying the simplicity prior $P(\mathbf{w})$*". As it is not the main focus of this work, a more in-depth explanation of deriving the cost function can be found in [38].

3.4 Bayes By Backprop

This section introduces the Bayes By Backprop algorithm [5]. It uses the cost function from eq. 3.11 as its optimization objective.

3.4.1 Method

In [5] the authors present Bayes by Backprop (BBB). It is a sophisticated Bayesian method for neural networks in which weights are drawn from a shared probability distribution to perform uncertainty estimations (fig. 3.1). Drawing single or multiple samples from the weight distribution resembles using an ensemble of neural networks. Further, this method implicitly regularizes the network while only doubling the number of trainable parameters.

Assume $\mathcal{D}$ to be a training dataset $(x,\ y)$ with features x and labels y. The weights $\mathbf{w}$ of a Bayesian Neural Network (BNN) are drawn from a complex posterior distribution $P(\mathbf{w} \mid \mathcal{D})$. It is non-trivial, and in most cases intractable, to exactly compute the posterior distribution. The dimensionality of $P(\mathbf{w} \mid \mathcal{D})$ is proportional to the number of

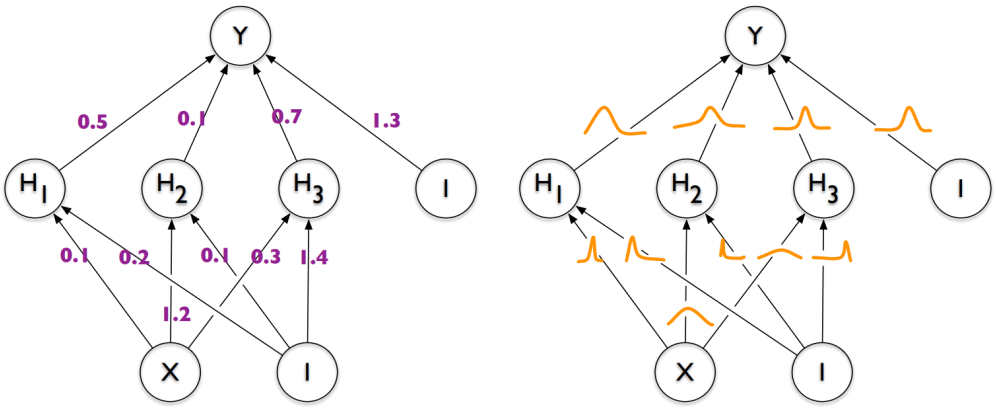


Figure 3.1: *left:* neural network with a weight matrix whose elements are fixed scalars, *right:* all elements of the weight matrix are sampled from a probability distribution [5].

weights and requires integrating a complex function over all dimensions. Thus, introducing approximation methods is critical for practical usage. Using variational Bayesian inference, as shown in section 3.3, lets us approximate $P(\mathbf{w} \mid \mathcal{D})$ by choosing a prior $P(\mathbf{w})$ and a variational posterior $q(\mathbf{w} \mid \theta)$. It is recommended to either use Gaussian distributions $\mathcal{N}$ or Gaussian mixture models:

$$P(\mathbf{w}) = \prod_i \mathcal{N}(\mathbf{w}_i \mid \mu, \sigma^2) \qquad \therefore \ \log P(\mathbf{w}) = \sum_i \log \mathcal{N}(\mathbf{w}_i \mid \mu, \sigma^2) \tag{3.13}$$

$$q(\mathbf{w} \mid \theta) = \prod_i \mathcal{N}(\mathbf{w}_i \mid \mu_\theta, \sigma_\theta^2) \qquad \therefore \ \log q(\mathbf{w} \mid \theta) = \sum_i \log \mathcal{N}(\mathbf{w}_i \mid \mu_\theta, \sigma_\theta^2) \tag{3.14}$$

where $\mathbf{w}_i$ is the ith weight of the network, μ is the mean, and σ is the standard deviation. In contrast to the prior, the mean and standard deviation of the variational posterior are trainable parameters as they are defined by θ:

$$\theta = (\mu, p) \tag{3.15}$$

$$\sigma = log(1 + e^p) \tag{3.16}$$

Building upon eq. 3.11, all samples are drawn from the weight probability distribution $\mathbf{w}$, where the ith Monte Carlo (MC) sample is denoted by $\mathbf{w}^{(i)}$. $q(\mathbf{w}^{(i)} \mid \theta)$ and $P(\mathbf{w}^{(i)})$ are known distributions (eq. 3.13 and 3.14). The likelihood $P(\mathcal{D} \mid \mathbf{w}^{(i)})$ is calculated via a loss function, e.g. cross-entropy:

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)} \mid \theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathcal{D} \mid \mathbf{w}^{(i)}) \tag{3.17}$$

Using backpropagation, the variational posterior and the weights are updated by the gradient w.r.t. $\mathbf{w}$. Therefore, it must be differentiable in $\mathbf{w}$. In practice, this is done by calculating $\frac{\partial \mathcal{F}(\mathbf{w},\theta)}{\partial \mathbf{w}}$ with $n = 1$ (one MC sample). The variational posterior is then shifted and scaled by the gradient w.r.t. to the mean and standard deviation by updating θ. Updating θ effectively adjusts the mean and standard deviation for each weight.[1]

Training the neural network with stochastic gradient descent causes another problem. Luckily, it can easily be fixed. Suppose the data $\mathcal{D}$ to be partitioned into M equally-sized subsets $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_M$. Since the cost function $\mathcal{F}(\mathcal{D}, \theta)$ applies the KL divergence M times for each minibatch, it has to be normalized by the factor M^{-1}:

$$\mathcal{F}_M(\mathcal{D}, \theta) \approx \sum_{m=1}^{M} \mathcal{F}_m(\mathcal{D}_m, \theta) \tag{3.18}$$

$$\text{where} \quad \mathcal{F}_m(\mathcal{D}_m, \theta) \approx \frac{1}{M} \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)} \mid \theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathcal{D}_m \mid \mathbf{w}^{(i)}) \tag{3.19}$$

where m denotes the index of the current minibatch. Without this normalization step, the regularization of the network would be too high and training would not be as effective.

3.4.2 Bayes by Backprop in RNNs

Section 3.4.1 discussed Bayes by Backprop for feedforward neural networks but the same idea applies to Recurrent Neural Networks (RNNs) [8]. All weights of an RNN are drawn from a distribution, as shown in fig. 3.2.

The loss function $\mathcal{L}(\theta)$ resembles eq. 3.11. The complexity cost, calculated by the KL divergence, and likelihood cost are defined as:

$$\mathcal{L}(\theta) = \mathrm{KL}[q(\theta) \mid\mid P(\theta)] - \mathbb{E}_{q(\theta)}[\log P(y_{1:T} \mid \theta, x_{1:T})] \tag{3.20}$$

where T is the sequence length. Here, all parameters are represented by θ, thus $\mathbf{w}$ is not directly referenced. x_t is the input and y_t the desired output at time step $t \in \{1, ..., T\}$. This approach requires unrolling the RNN and is not directly suitable for minibatches.

[1] Section *3.2. Gaussian variational posterior* [5] explains the process of updating the variational posterior in more detail.

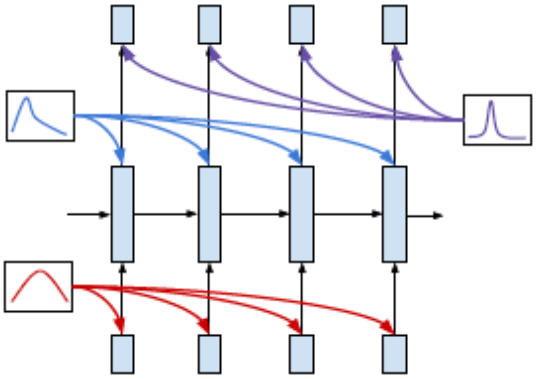


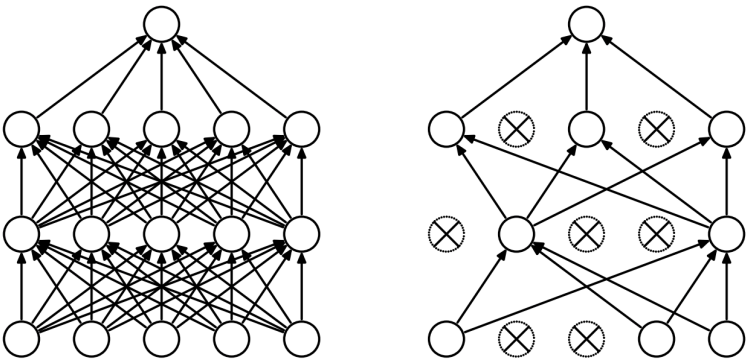
Figure 3.2: Bayes by Backprop applied to an RNN. [8]

Applying this principle to minibatches results in the *Truncated Bayes by Backprop Through Time* algorithm that is presented in the paper. It is similar to eq. 3.18 but adds the time dimension and truncated sequences. Furthermore, the authors of [8] propose the use of posterior sharpening to reduce the variance of the Bayesian RNN.

3.5 Monte Carlo Dropout

This section introduces Monte Carlo (MC) Dropout [4]. Its implementation differs from Bayes by Backprop's, but the objective is identical.

3.5.1 Method



(a) Standard Neural Net (b) After applying dropout.

Figure 3.3: Dropout in a dense neural network. Dropped units are marked with an **x**. [39]

MC Dropout utilizes dropout at inference time. Performing multiple forward passes approximates a deep Gaussian process. Dropout is applied to all layers of the neural network. Its main advantage over Bayes by Backprop (section 3.4) is the straightforward implementation for new and existing models because it does not introduce additional complexity. Typically, dropout is used for regularization, which prevents the neural network from overfitting, as some units are dropped randomly. The dropout probability is sampled from a Bernoulli distribution $\mathcal{B}(p_{drop})$. p_{drop} is a hyperparameter between zero and one where $p_{drop} = 0$ means *drop no units* and $p_{drop} = 1$ means *drop all units*. MC Dropout does not strictly require dropout while training.

The research paper *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks* [7] demonstrates the connection to Bayesian neural networks more clearly. The connection is given by the special variational posterior q, defined as a Gaussian mixture model with low variance and one Gaussian with a mean of 0.[2] Given a dropout probability $p \in \{0, 1\}$, small variance σ^2, and a variational parameter $\mathbf{m}_k$ for every weight matrix row $\mathbf{w}_k$, the variational posterior q is defined as:

$$q(\mathbf{w}_k) = p\mathcal{N}(\mathbf{w}_k; 0, \sigma^2 I) + (1 - p)\mathcal{N}(\mathbf{w}_k; \mathbf{m}_k, \sigma^2 I) \tag{3.21}$$

One can think of the variational parameter $\mathbf{m}_k$ as a part of θ in section 3.4.1. A major aspect of this distribution is the fact that it mimics dropout. $\hat{\mathbf{w}}_k \sim q(\mathbf{w}_k)$ with $p = 1$ roughly returns a zero-vector because the mean is 0 and the variance is negligible by design, whereas a random sample with $p = 0$ returns the weight matrix row $\mathbf{w}_k$ parameterized by $\mathbf{m}_k$. Further, the authors explain that the complexity cost can be approximated as L_2 regularization, thus providing a framework to interpret MC Dropout as Bayesian Inference in neural networks.

MC Dropout calculates the sample mean $\bar{\mathbf{x}}$ and sample variance s^2 of a new data point $\mathbf{x}$ with T stochastic forward passes. Assume $f_{nn}^{(r)}$ to be a deep neural network where some units are randomly dropped based on the dropout mask $r_{l,i} \sim \mathcal{B}(p_{drop})$ for layer $l \in \{1, ..., L\}$ and unit i [39]:

$$\bar{\mathbf{x}} \approx \frac{1}{T} \sum_{t=1}^{T} f_{nn}^{(r)}(\mathbf{x}) \tag{3.22}$$

$$s^2 \approx \frac{1}{T} \sum_{t=1}^{T} (f_{nn}^{(r)}(\mathbf{x}) - \mu)^2 \tag{3.23}$$

[2] This assumption comes with a few pitfalls mentioned in section 3.6.

A new dropout mask r is used for each forward pass. Unsurprisingly, each sample adds computational overhead, but small values for T (e.g. $T = 10$) already yield reasonable results. The calculations can be performed in parallel [4].

3.5.2 MC Dropout in RNNs

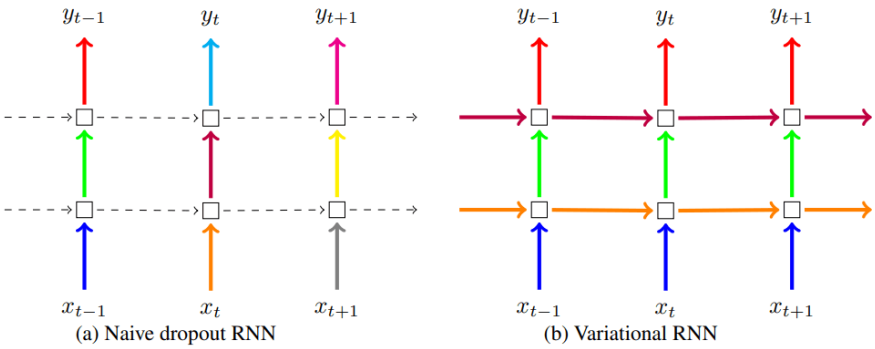


Figure 3.4: Application of dropout in RNNs. Each square is an RNN unit. The horizontal axis represents the time dimension and the vertical axis the input and output. Identically colored arrows represent the same dropout masks. A dotted arrow means that no dropout is applied. MC Dropout uses the right method (b) with recurrent dropout masks. [7]

After acquiring MC Dropout's theoretical background [4] (section 3.5.1), the next step is to transfer the method to RNNs [7]. The core idea remains the same, namely, applying dropout at inference time. To fully support this new interpretation of dropout in RNNs, the general approach of using dropout in RNNs has to be adjusted.

Regularizing RNNs with dropout is frequently studied. Many researchers argue that applying dropout to recurrent connections introduces model instability and should be avoided [40, 41]. To counteract this problem, the new approach by [7] uses dropout masks in all recurrent connections as well as in all input and output connections. This is demonstrated in fig. 3.4b. The dropout masks remain unchanged over the whole sequence. This is fundamentally different from earlier approaches (fig. 3.4a) where the dropout mask is updated between time steps.

Equivalent to eq. 3.22 and 3.23, T stochastic forward passes with active dropout approximate the mean and variance for new sequences.

3.6 Pitfalls of Approximate Bayesian Methods

To see the whole picture of approximate Bayesian methods, it is crucial to address their shortcomings. Measuring the epistemic uncertainty in sections 3.4 and 3.5 relies on MC sampling at inference time. This approach scales linearly but can be executed in parallel. An evident disadvantage is the computational cost at inference time. It is possible to achieve sampling-free uncertainty estimations [42] but is, admittedly, out of scope for this work.

Besides the computational overhead that might not be feasible in all scenarios, various scientific papers [43, 44, 45] criticize variational Bayesian dropout (MC Dropout) as an incomplete Bayesian framework. They also propose new concepts, such as a variation of the approximate inference objective (KL divergence). One point of criticism is that improper priors generally lead to an improper posterior distribution and overfitting. Nevertheless, the authors conclude that MC Dropout and the use of improper priors "*still provide good empirical results, albeit not because of the Bayesian [...] arguments.*" [43, section 6] They provide mathematical proofs to explain their empirical observations. One of the authors is Zoubin Ghahramani, who supported Yarin Gal in his foundational research paper *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning* [4].

4 Related Work

Text classification is a well-established task in machine learning with neural networks. The sequence-to-sequence classification approach presented in this work (chapter 5) is closely related to sequence tagging [46] and Named Entity Recognition (NER) [10, 11]. Table 4.1 compares the related approaches based on the method, whether it is Bayesian, interpretability, and the objective.

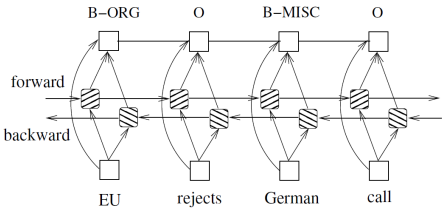


Figure 4.1: Bidirectional LSTM for sequence tagging. [46]

In sequence tagging, tags are assigned to individual elements in a sequence. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are commonly used architectures [10, 11]. Zhiheng Huang et al. [46] use Bidirectional LSTMs to assign tags like `Organization (B-ORG)` to words (fig 4.1).

Jonas Paul Winkler et al. [12] present a CNN-centered method. Their goal is to recognize sensitive words in the classification. They use document influence matrices as a way to create their interpretable text models. A document influence matrix states how strongly any word affects a class. In binary sentiment analysis these classes are `positive` and `negative`. To generate the document influence matrix, the neural network is reversed so that the output of the CNN can be traced back to its input.

Another approach involves hidden Markov models that closely resemble simple dynamic Bayesian networks. Filip Ginter et al. [47] use an unsupervised hidden Markov model to identify text segments and labels. The training data is not labeled. The authors' research area lies in the healthcare sector. They address the importance of detecting

health-related topics such as `breathing` and `consciousness` in a patient's report. There are also Hidden Markov models for labeled observations [48].

Cindy Wang [49] presents visualization techniques for hate speech detection with CNN-GRU models. This includes the detection of keywords associated with hate speech. One approach involves looking at the activation of distinct units in the CNN-GRU after predicting a whole dataset.

Quantifying uncertainty in neural networks is an active field of research. Most notably– and referenced in previous chapters–Charles Blundell et al. [5] and Yarin Gal et al. [4] provide methods of approximate Bayesian inference in neural networks that are later transferred to RNNs [8, 7].

Yijun Xiao et al. [34] propose a method to explore data and model uncertainty in Natural Language Processing (NLP) tasks, such as named entity recognition. They define a mathematical framework to quantify model and data uncertainty under the umbrella of total variance. They show that difficult tasks yield higher uncertainty while generally observing an improved model performance. Similar to [46], uncertainty estimations are obtained with bidirectional Bayesian LSTMs and CNNs with MC Dropout.

Paper	**Method**	**Bayesian**	**Inter-pretable**	**Objective**
Bidirectional LSTM-CRF Models for Sequence Tagging [46]	Bi-LSTM-CRF	✗	✗	POS tagging, chunking, NER
Named Entity Recognition with Bidirectional LSTM-CNNs [10]	Bi-LSTM, CNN	✗	✗	NER
Neural Architectures for Named Entity Recognition [11]	LSTM-CRF	✗	✗	NER
What Does My Classifier Learn? A Visual Approach to Understanding Natural Language Text Classifiers [12]	CNN, document influence matrix	✗	✓	binary and multiclass classification
Hidden Markov models for labeled sequences [48]	hidden Markov model	✓	✗	NER
Interpreting Neural Network Hate Speech Classifiers [49]	CNN-GRU	✗	✓	hate speech detection, visualization
Quantifying Uncertainties in Natural Language Processing Tasks [34]	Bi-LSTM, CNN	✓	✓	sentiment analysis, language modeling, NER

Table 4.1: Comparison of various Bayesian and non-Bayesian text classification methods.

5 Concept

This chapter introduces the sequence-to-sequence approach to text classification. Since a text source can be represented as a sequence, it is possible to measure the intermediate predictions in binary and multiclass classification. This also includes intermediate uncertainty estimations.

Previous chapters introduced language processing with LSTMs (chapter 2) in addition to Bayesian methods (chapter 3) and ways to achieve Bayesian approximation in LSTMs (sections 3.5.2 and 3.4.2). As mentioned in chapter 4, this approach is closely related to Named Entity Recognition (NER) [10]. It uses NER as a basis to classify whole text documents. The hypothesis is that additional information about intermediate predictions and their underlying uncertainties make an outcome analyzable if needed. Uncertain segments are easily identifiable because the model learns what it does not know. This also makes a model more trustable.

The following sections describe the sequence-to-sequence classification process, starting with data preparation and hyperparameters in section 5.1. For reference, commonly used symbols can be found in the list of symbols but are also explained throughout upcoming sections.

5.1 Data Preparation

This sequence-to-sequence approach requires some data preparation so that the data matches the input and output shape of the Bayesian Long Short-Term Memory (LSTM). The data is defined as $\mathcal{D} = (x, y)$. It does not involve manual data labeling.

The maximum input sequence length $M \in \mathbb{N}^+$ needs to be set accordingly. A good reference point is the average number of words per entry in the dataset. Doubling the maximum sequence length would approximately double the training time. If M is too

small, most inputs will be truncated, which leads to information loss. Further, to pad or truncate all features x and labels y, M must be known.

The process of padding and truncating the labels is straight forward. All labels must first be one-hot encoded. Then, to match the output shape of the neural network, the label itself has to become a repeating sequence of one-hot encoded labels of length M. For example, the one-hot encoded label $[0, 0, 1]^\intercal$ becomes $([0, 0, 1]^\intercal, [0, 0, 1]^\intercal, [0, 0, 1]^\intercal)$, given $M = 3$.

Features need to be tokenized first with the methods described in section 2.1. In section 6.2.1, a term frequency distribution is used to create a word index to encode word tokens as numeric tokens. Alternatives are the pre-trained word embedding GloVe and Word2Vec (section 2.2). To match the maximum sequence length M, all input sequences must be trimmed or padded with zeros accordingly.

5.2 Bayesian LSTM Structure

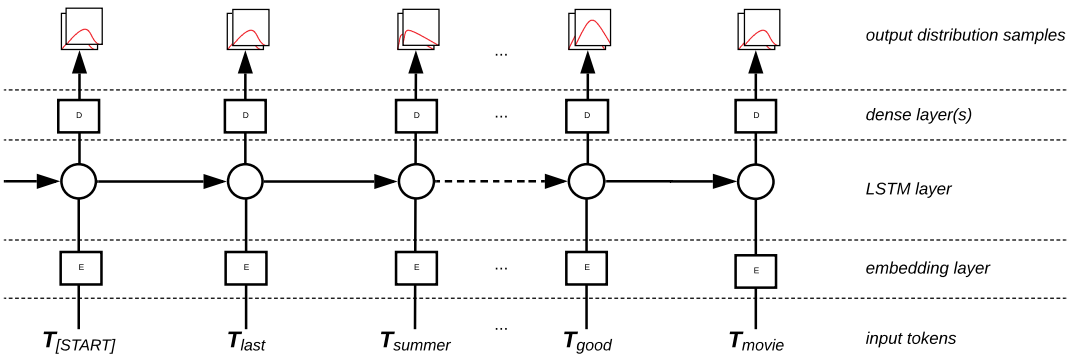


Figure 5.1: All layers of the Bayesian LSTM for sequence-to-sequence classification.

The neural network starts with an embedding layer (fig. 5.1) that accepts a sequence of numeric token representations. As shown in section 2.2, each token is embedded into word vector space. The embedding layer is not required if the input is in vector space.

The sequence of word vectors is then forwarded into a Bayesian LSTM layer that either uses MC Dropout (section 6.3.1) or Bayes by Backprop (section 6.3.2). Given a maximum sequence length M, at each time step $t \in \{1, 2, ..., M\}$ the hidden state h_t (eq. 2.13) of the LSTM is forwarded to one or more dense layers. Only forwarding the last hidden

state is not sufficient. This sequence-to-sequence classification approach requires all time steps to work.

The dense layers are a way to produce the correct output shape, given the number of labels $K = |\{label_1, \cdots, label_K\}|$ in the classification problem. The output shape of the neural network, and therefore of the last dense layer, is (*batch size*, M, K). If $K = 1$, the activation function of the last dense layer must be the sigmoid function. Otherwise the softmax function is required. Likewise, the loss function should be binary cross-entropy if $K = 1$ and categorical cross-entropy if $K > 1$.

5.3 Sequence to Sequence Classification

The Bayesian LSTM implementation allows for straightforward M to M sequence processing where M is the input and output sequence length. As demonstrated in fig. 5.1, each token x_t in the padded input sequence $x = (x_1, x_2, ..., x_M)$ associates with an element in the output sequence (the hidden state at the respective time step; eq. 2.13). During the evaluation, only the time steps up to the actual sequence length $M' = |x \setminus \texttt{[PAD]}|$ with $M' \leq M$ are relevant. $x \setminus \texttt{[PAD]}$ defines the actual input sequence without the padding elements.

The output will be a sequence of K-dimensional sample distributions where K is the number of labels. For binary sentiment analysis, K is 1. In other words, each input token x is assigned a univariate or multivariate probability distribution to quantify which label (or labels) is most probable, respecting the variance of the outcome. Since this approach uses Monte Carlo approximation to evaluate variational inference (section 3.4.1), the mean and variance are also approximations.

Given the number of samples T, the length of the input sequence M', and the number of labels K, the output y is calculated by the neural network function $f_{nn}(x)$:

$$\hat{y} = f_{nn}(x) = (\hat{y}^{(1)}, ..., \hat{y}^{(M')}) \tag{5.1}$$

$$\Leftrightarrow \qquad \hat{y} = \left(\begin{bmatrix} \hat{y}_{1,1}^{(1)} & \cdots & \hat{y}_{1,T}^{(1)} \\ \vdots & \ddots & \vdots \\ \hat{y}_{K,1}^{(1)} & \cdots & \hat{y}_{K,T}^{(1)} \end{bmatrix}, \ldots, \begin{bmatrix} \hat{y}_{1,1}^{(M')} & \cdots & \hat{y}_{1,T}^{(M')} \\ \vdots & \ddots & \vdots \\ \hat{y}_{K,1}^{(M')} & \cdots & \hat{y}_{K,T}^{(M')} \end{bmatrix} \right) \tag{5.2}$$

The superscript of $\hat{y}$ denotes the time step; the subscript states the *label* and *sample index*. In eq. 5.2, each row in one of the inner matrices represents T samples of a specific label. The number of rows equals the number of labels. In multiclass classification, each column adds up to 1, which is a direct result of the softmax activation function in the last dense layer. Following eq. 3.22, performing only one stochastic forward pass ($T = 1$) results in a variance of 0. T should be sufficiently large with $T > 1$ to perform uncertainty estimations that are based on the variance and the mean.

The sample means $\bar{\mathbf{x}}_{\hat{y}} = (\bar{\mathbf{x}}_{\hat{y}^{(1)}}, \cdots, \bar{\mathbf{x}}_{\hat{y}^{(M')}}) = (\bar{\mathbf{x}}_1, \cdots, \bar{\mathbf{x}}_{M'})$ are calculated for each time step t:

$$\bar{\mathbf{x}}_t = \frac{1}{T} \sum_{s'=1}^{T} \hat{y}_{*,s'}^{(t)} = \frac{1}{T} \begin{bmatrix} \sum_{s'=1}^{T} \hat{y}_{1,s'}^{(t)} \\ \vdots \\ \sum_{s'=1}^{T} \hat{y}_{K,s'}^{(t)} \end{bmatrix} \tag{5.3}$$

Similar to Named Entity Recognition (NER) (section 4), it is possible to assign a label $\hat{y}^{*(t)}$ to each time step by finding the maximum value of the sample mean vector $\bar{\mathbf{x}}_t$. Let $\hat{y}^{*(t)}$ be such that $\bar{\mathbf{x}}_{t,\hat{y}^{*(t)}} = max(\bar{\mathbf{x}}_t)$, where the second index of the subscript denotes the label:

$$\hat{y}^* = (\hat{y}^{*(1)}, \ldots, \hat{y}^{*(M')}) \tag{5.4}$$

Since eq. 5.4 does not address variance in the estimations, the uncertainty of $\hat{y}^{*(t)}$ can not yet be determined. Rather, the prediction $\hat{y}^*$ is a sequence of labels that are the most likely out of all possible permutations. This does not imply the predictions to be correct or certain. This is further explained in section 5.4.

Independent of uncertainty estimations, a feasible method to determine the label of $\hat{y}$ is to assign weights to each time step. This requires a weight sequence g of length M'. Each mean vector $\bar{\mathbf{x}}_t$ is scaled by g_t and normalized by g, also known as the weighted arithmetic mean:

$$\hat{y}_g = \frac{\sum_{t=1}^{M'} g_t \; \bar{\mathbf{x}}_t}{\sum_{t=1}^{M'} g_t} \tag{5.5}$$

$\hat{y}_g$ is a K-sized vector. The highest value indicates the prediction.

Initially, the LSTM states are uniformly random. This causes some noise in earlier time steps. Therefore, sequence weights that favor later time steps are preferable. Empirically,

the following sequence weights delivered good results:

$$g_{full} = (1, \cdots, 1) \tag{5.6}$$

$$g_{linear} = (\frac{0}{M'-1}, \frac{1}{M'-1}, \cdots, \frac{M'-2}{M'-1}, \frac{M'-1}{M'-1}) \tag{5.7}$$

$$g_{end} = (0, \cdots, 0, 1) \tag{5.8}$$

Choosing $g = g_{end}$ is the same as evaluating the last time step. These methods only use the sample mean. The uncertainty is yet to be determined.

5.4 Uncertainty Evaluation

There are various types of independently measurable uncertainties (section 3.1). This section introduces a method to evaluate epistemic uncertainty.

5.4.1 Uncertainty Quantification

Let $f_U(t)$ be a function to quantify epistemic uncertainty at each time step t in $\hat{y}$. Its implementation follows the definition from eq. 3.4:

$$f_U(t) = \frac{1}{T} \sum_{s'=1}^{T} (\hat{y}_{*,s'}^{(t)} - \bar{y}_t)(\hat{y}_{*,s'}^{(t)} - \bar{y}_t)^\intercal \tag{5.9}$$

The epistemic uncertainty at any time step is simply the covariance matrix of $\hat{y}^{(t)}$. Therefore, the variance s_t^2 of $\hat{y}^{(t)}$ at time step t is the diagonal of $f_U(t)$, and $s_t = \sqrt{s_t^2}$ is the standard deviation. The simplicity of the standard deviation makes it a practicable candidate for uncertainty quantification. Let $U(t, l)$ be a function that returns the standard deviation (i.e., the epistemic uncertainty) for label l at time step t:

$$U(t, l) = s_{t,l} = \sqrt{f_U(t)_{l,l}} \tag{5.10}$$

The subscript l, l accesses the lth variance on the diagonal of the covariance matrix returned by $f_U(t)$.

Similar to eq. 5.5, the uncertainty estimations should also be weighted by the weight sequence g. Thus, the weighted arithmetic mean U_g is formally defined as:

$$U_g(l) = \frac{\sum_{t=1}^{M'} g_t U(t,l)}{\sum_{t=1}^{M'} g_t} \tag{5.11}$$

5.4.2 Segmentation

Subdividing a prediction $\hat{y}$ into smaller segments supports the identification of uncertain parts.

By using a threshold $\alpha > 0$, a prediction at time step t, and label l is uncertain if $U(t,l) > \alpha$. Time steps t_1 and t_2 are connected and form a segment if they are neighbors with $t_1 + 1 = t_2$ or $t_1 - 1 = t_2$ and have the same uncertainty label:

$$\text{uncertain:} \quad U(t_1,l) > \alpha \wedge U(t_2,l) > \alpha \tag{5.12}$$

$$\text{confident:} \quad U(t_1,l) \leq \alpha \wedge U(t_2,l) \leq \alpha \tag{5.13}$$

5.4.3 Multivariate Gaussian Distribution

The sample mean $\bar{\mathbf{x}}_t$ and covariance matrix $\mathbf{K}_t = f_U(t)$ define an approximate version of a possibly multivariate Gaussian distribution, assuming that the predictive distribution can be modeled as a series of Gaussian distributions[1]:

$$(\mathbf{X^{(1)}} \sim \mathcal{N}_K(\bar{\mathbf{x}}_1, \mathbf{K}_1),\ ...,\ \mathbf{X^{(M')}} \sim \mathcal{N}_K(\bar{\mathbf{x}}_{M'}, \mathbf{K}_{M'})) \tag{5.14}$$

5.5 Example

In this example, a prediction of a two-class classification problem is given by:

$$\hat{y} = \left(\begin{bmatrix} 0.46 & 0.48 & 0.47 \\ 0.54 & 0.52 & 0.53 \end{bmatrix}, \begin{bmatrix} 0.55 & 0.45 & 0.47 \\ 0.45 & 0.55 & 0.53 \end{bmatrix}, \begin{bmatrix} 0.48 & 0.71 & 0.63 \\ 0.52 & 0.29 & 0.37 \end{bmatrix}, \begin{bmatrix} 0.86 & 0.83 & 0.76 \\ 0.14 & 0.17 & 0.24 \end{bmatrix} \right) \tag{5.15}$$

[1] Softmax and sigmoid limit the codomain to $[0, 1]$. Truncated Gaussian or Beta distributions are reasonable alternatives.

The sample means and standard deviations of $\hat{y}$ can be calculated for each time step and label:

$$\bar{\mathbf{x}}_{\hat{y}} \approx \left(\begin{bmatrix} 0.47 \\ 0.53 \end{bmatrix}, \begin{bmatrix} 0.49 \\ 0.51 \end{bmatrix}, \begin{bmatrix} 0.61 \\ 0.39 \end{bmatrix}, \begin{bmatrix} 0.81 \\ 0.19 \end{bmatrix} \right) \tag{5.16}$$

$$s_{\hat{y}} \approx \left(\begin{bmatrix} 0.0098 \\ 0.0098 \end{bmatrix}, \begin{bmatrix} 0.043 \\ 0.043 \end{bmatrix}, \begin{bmatrix} 0.095 \\ 0.095 \end{bmatrix}, \begin{bmatrix} 0.043 \\ 0.043 \end{bmatrix} \right) \tag{5.17}$$

Analogous to eq. 5.4, each time step is assigned a label based on the sample means. Using the weight sequence $g = g_{end} = (0, 0.\bar{3}, 0.\bar{6}, 1)$ to calculate the weighted arithmetic mean, the first entry of $\hat{y}_g$, which corresponds to the first label, is the most likely:

$$\hat{y}^* = (1, 1, 0, 0) \tag{5.18}$$

$$\hat{y}_g \approx \begin{bmatrix} \frac{0 \cdot 0.47 + 0.\bar{3} \cdot 0.49 + 0.\bar{6} \cdot 0.61 + 1 \cdot 0.81}{0 + 0.\bar{3} + 0.\bar{6} + 1} \\ \frac{0 \cdot 0.53 + 0.\bar{3} \cdot 0.51 + 0.\bar{6} \cdot 0.39 + 1 \cdot 0.19}{0 + 0.\bar{3} + 0.\bar{6} + 1} \end{bmatrix} \approx \begin{bmatrix} 0.69 \\ 0.31 \end{bmatrix} \tag{5.19}$$

To calculate $U_g(0)$ and $U_g(1)$, the standard deviations $s_{\hat{y}}$ are reduced to a single dimension per label:

$$\begin{aligned} U_g(0) = U_g(1) &\approx \frac{0 \cdot 0.0098 + 0.\bar{3} \cdot 0.043 + 0.\bar{6} \cdot 0.095 + 1 \cdot 0.043}{0 + 0.\bar{3} + 0.\bar{6} + 1} \\ &\approx 0.06 \end{aligned} \tag{5.20}$$

With $\alpha = 0.04$, the uncertain time steps are $t = 0$ and $t = 2$ for both labels.

6 Experiments

In this chapter, two publicly available datasets are examined. The first sections 6.1 to 6.4 describe the setup and implementation of the Bayesian LSTM with Monte Carlo (MC) Dropout and Bayes by Backprop (BBB) in Tensorflow 2. Sections 6.5 and 6.6 present the model performance and uncertainty implications in binary and multiclass classification with both datasets. Section 6.7 shows the real-time proof of concept application that was created alongside this work. The application supports multiple Bayesian models.

6.1 Software

The code is written in Python 3.7. The following list outlines critical frameworks and libraries used in the experiments:

- **Tensorflow v2.1.0** [50] is an open-source machine learning framework developed by Google. Internally, Tensorflow translates the user model into a flow graph for fast execution on CPUs, GPUs, and Tensor Processing Units (TPUs). To successfully transform the source code into a graph, Tensorflow provides many APIs and data objects. These data objects are called tensors and give Tensorflow its name. Tensorflow's high-level neural network API is Keras [51]. It is used throughout the experiments.
- **Tensorflow Probability v0.9.0** [52] is a probability theory extension to the Tensorflow framework. It adds probabilistic computation to Tensorflow. The neural network layers implement the common Keras layer interface. Through this interface, the layers can be used interchangeably with other Tensorflow layers. Tensorflow Probability also supports custom Bayesian layers by providing functions to create trainable multivariate posterior distributions and multivariate priors.

- **Natural Language Toolkit (NLTK) v3.4.5** [53] is a library for computational linguistics. It contains a collection of commonly used Natural Language Processing (NLP) tools to analyze and process texts. It also provides interfaces to access various corpora and other lexical resources.
- **BERT as a service v1.10.0** [54] is a word and sentence encoder that is based on Bidirectional Encoder Representations from Transformers (BERT). The service runs on a server with a pre-trained BERT model. To encode texts, a Python or HTTP client that connects to the service is needed. Thus, BERT as a service provides a convenient architecture to execute independent text encoding jobs with BERT.

6.2 Word Embeddings

The following sections present three approaches to implementing word embeddings. The first one being a trainable Keras word embedding layer [51] in section 6.2.1. The two subsequent approaches in sections 6.2.2 and 6.2.3 use a pre-trained BERT model. Table 6.3 compares the performance of all embeddings.

6.2.1 Keras Embedding Layer

Keras word embedding layer [51] turns positive integers into dense vector representations of a fixed size. The embedding size is a hyperparameter of the model. In the experiments, it is either set to 100 or 300. It is a trainable word embedding with a random initial state.

Text Preprocessing

Since the data is noisy from the perspective of a neural network, it requires multiple preprocessing steps. Moreover, the text needs to be tokenized so that it can be used as an input for the neural network. Based on the methods presented in section 2.1, the following steps are applied to all datasets (training, validation, and test) respectively.

1. **Text preparation and cleanup**: As the first step, the UTF-8 encoded text is lowercased. A character is removed when it does not fall into any of these categories: alphabetic, numeric, period, question mark, and exclamation mark.

2. **Tokenization**: Text can be split into a sequence of tokens using the default tokenizer API of the NLTK package. Internally, it uses the Penn Treebank [55] and the Punkt Sentence tokenizer [56] (section 6.1). Fundamentally, the NLTK tokenizer splits the text by whitespace and punctuation.

3. **Lemmatization**: Each token is then lemmatized by NLTK's WordNet lemmatizer that operates based on the WordNet Corpus [57]. If the word exists in the corpus, the algorithm will try to transform the input token by reducing it to its root form; otherwise, the unchanged input token is returned. For example, the token `dogs` becomes `dog` and the token `abaci` becomes `abacus`.

4. **Normalization**: In the last step, the tokens are further normalized to reduce their complexity. Numeric tokens like `402` and `1` are replaced by the placeholder `[NUMERIC]`. This step is justified by the limited size of the word index but causes information loss. It also reduces repeated characters to a maximum number of two. For example, `hellllo` becomes `hello`.

Word Index

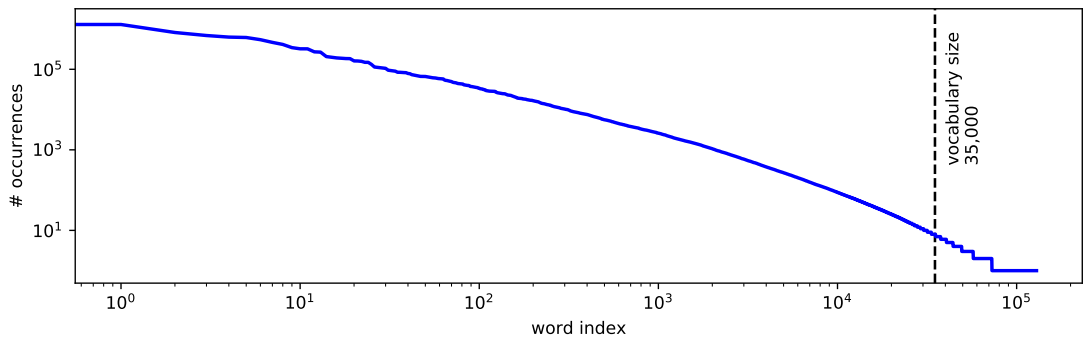


Figure 6.1: #Occurrences over the word index on a logarithmic scale. See also: table 6.1.

A word index efficiently transforms a token sequence into a numeric sequence by its index. Following the text transformations in the previous section 6.2.1, each token of the corpus is assigned a unique index, starting at zero. In this case, the corpus is

determined by large samples from both datasets. The order is based on a frequency distribution that records the number of times each token has occurred. This results in a total size of around 130,000 entries. Out of the 130,000 entries only the first 35,000 are evaluated in the embedding layer of the neural network (fig. 6.1). The word index is stored as a plain text file where each row contains the index in ascending order, the token, and the number of occurrences (table 6.3).

index	token	occurrences
0	.	1559374
1	the	1294457
2	a	819188
3	and	689231
4	i	629623
...		
4229	stopping	346
4230	household	346
4231	meaningful	345
...		
56141	regualr	3
56142	gismo	3
56143	framerates	3
...		

Table 6.1: Word index samples. Each row represents one line of the word index text file.

The start of a sequence, post-sequence padding, and any unknown words are substituted with one of the three special tokens `[START]`, `[PAD]`, and `[UNK]` respectively. Padding tokens are required because LSTMs need the input sequence to be of fixed length. If the sequence is too short, the remaining entries are set to `[PAD]`. Unknown words occur when the word index does not contain the word or when only a subset of the word index is used, limiting the vocabulary. Is that the case, the word is substituted with the `[UNK]` token. The `[START]`, `[PAD]`, and `[UNK]` tokens are prepended to the word index. This requires the word index to shift its index by 3.

6.2.2 BERT Embedding

To use BERT as a context-aware word embedding layer, it is recommended to choose one of multiple pre-trained BERT models [9]. Here, the uncased English base model with 12 layers, 768 hidden units, and 12 heads–totaling 110 million parameters–is used without further fine-tuning. Because of their size, a more flexible architecture to integrate

BERT as a word embedding is desirable. BERT can be separated into an independent service with *BERT as a service* [54] (section 6.1). The service manages the Tensorflow implementation, loading the pre-trained model, and BERT-specific tokenization. It does not require a text preprocessing step. Starting the service in a Docker container ensures reusability and scalability. As shown in figure 6.2, the *BERT as a service* instance returns a pair of word vectors and tokens for each input text. The word vectors are inputs to the Bayesian LSTM, which therefore does not need an additional word embedding layer. In practice, the first layer is a dense layer, directly followed by the Bayesian LSTM layer.

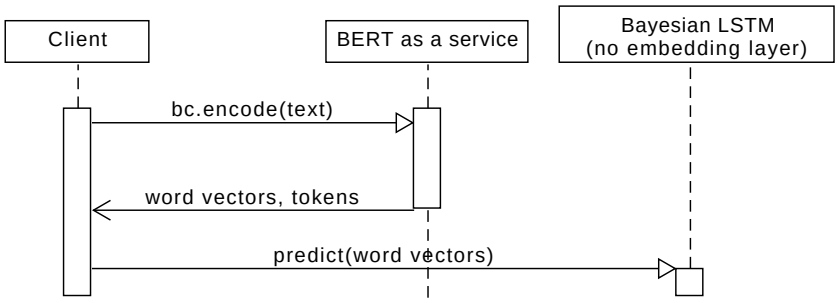


Figure 6.2: BERT embedding with *BERT as a service*.

For performance reasons, all responses are cached. This eliminates the need to recalculate the word embedding for the same dataset. The Python client which connects to the service abstracts the communication in fig 6.2 in a few lines of code:

```
1 from bert_serving.client import BertClient
2
3 def bert_encode_text(text):
4     with BertClient(ip="bert-as-service", timeout=30_000) as bc:
5         [word_vectors], [tokens] = bc.encode(
6             [text], show_tokens=True, is_tokenized=False)
7
8     return word_vectors, tokens
```

Listing 6.1: Text encoding with the *Bert as a service* client in Python.

6.2.3 BERT Embedding with PCA

All word vectors returned by the pre-trained BERT model in section 6.2.2 are 768-dimensional. This is drastically higher than the 100 and 300 dimensions used in the experiments with the Keras embedding (section 6.2.1). The output dimensions of pre-trained BERT models are not easily adjustable since it requires retraining over a huge corpus.

As a performance test, an incremental Principal Component Analysis (PCA) reduces the dimensions of each word vector to 128. The PCA kernel is trained using the training dataset. The idea is that most dimensions hold little to no relevant information. Table 6.3 shows the results of using PCA. The embedding process is almost identical to section 6.2.2. Figure 6.3 shows the new PCA component that acts as an intermediate layer.

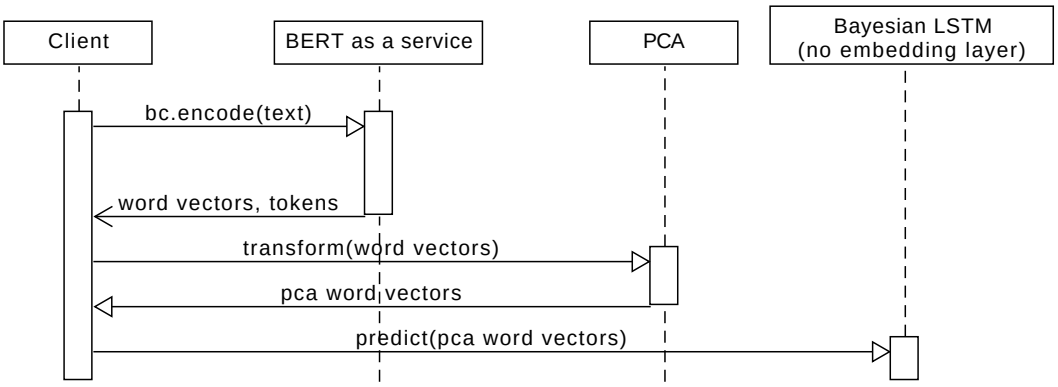


Figure 6.3: BERT embedding with *BERT as a service* using PCA to reduce the output dimensions before passing the word vectors into the Bayesian LSTM.

6.3 Bayesian LSTMs in Tensorflow 2

The implementation of variational Bayesian inference (section 3.2) in LSTMs differs heavily between the two methods MC Dropout and Bayes by Backprop. While MC Dropout uses a standard LSTM layer, Bayes by Backprop requires all weights of the LSTM to be modeled with probability distributions. The Tensorflow extension Tensorflow Probability [52] already provides a broad range of tools and variational layers

but lacks a Bayesian LSTM layer implementation. Section 6.3.2 explains the custom Bayesian LSTM layer implementation in detail.

6.3.1 Implementation of MC Dropout

As mentioned in previous chapters, the advantage of MC Dropout is its compatibility with existing architectures. As shown in listing 6.2, the high-level Tensorflow API Keras allows to set the `training` parameter to `True` in line 7 when the functional API is used. This effectively enables dropout at inference time. Setting `return_sequences` to `True` will result in the layer returning the hidden state (eq. 2.13) at each time step. This is required for sequence-to-sequence classification.

```
1 lstm = LSTM(units=64,
2           return_sequences=True,
3           stateful=False,
4           dropout=0.5,
5           recurrent_dropout=0.5,
6           recurrent_initializer=tf.initializers.glorot_uniform())
7 outputs = lstm(outputs, training=True)
```

Listing 6.2: MC Dropout layer with Keras' functional API

In the experiments, MC Dropout is tested with various dropout probabilities $p_{drop} \in \{0.1, 0.25, 0.5, 0.75\}$ (tables 6.3 and 6.6). The dropout mask does not change during a sequence. This is required by MC Dropout [7].

6.3.2 Implementation of Bayes by Backprop

The implementation of Bayes by Backprop (BBB) for LSTMs is an adaptation of Tensorflow's default LSTM implementation. It is instantiated just like any other layer class (listing 6.3) but requires a divergence function to calculate the divergence between the variational posterior $q(\theta)$ and prior $P(\theta)$: `divergence_fn()`. It is equivalent to the first part in eq. 3.20, stating that the KL divergence is defined by $\mathrm{KL}[q(\theta) \mid\mid P(\theta)]$. As noted in eq. 3.18, all models trained with minibatches have to scale the KL divergence of the loss function appropriately. Thus, the KL divergence is divided by `batch_size`.

```
def divergence_fn(q, p):
  return tfp.distributions.kl_divergence(q, p) / batch_size

lstm = BayesianLSTM(units=64,
                    divergence_fn=divergence_fn,
                    return_sequences=True,
                    unroll=True)
```

Listing 6.3: Initialization of a Bayesian LSTM layer

Listing 6.4 shows the Bayesian LSTM layer class. It has an `_apply_divergence()` function to add the KL divergence to the loss of the layer. After a sequence is processed by the `call()` function, `_apply_divergence()` is called three times to add the divergence w.r.t. the kernel $\mathbf{U}_{i,f,o,c}$, recurrent kernel $\mathbf{W}_{i,f,o,c}$, and bias $\mathbf{b}_{i,f,o,c}$ (eq. 2.8). Not shown is the `build()` function that instantiates the Bayesian LSTM cell in listing 6.5. It is a Bayesian implementation of the LSTM cell mentioned in fig. 2.4.

```
class BayesianLSTM(tf.keras.layers.RNN):
  # ...

  def call(self, inputs):
    cell_states = super(BayesianLSTM, self).call(inputs)
    # ... (_apply_divergence for kernel and bias)
    self._apply_divergence(
      self.divergence_fn,
      self.cell.recurrent_kernel_posterior,
      self.cell.recurrent_kernel_prior)

    return cell_states

  def _apply_divergence(self, divergence_fn, posterior, prior):
    kl_loss = divergence_fn(posterior, prior)
    self.add_loss(kl_loss, inputs=True)
```

Listing 6.4: Bayesian LSTM layer class (simplified)

The weights–namely the kernel, recurrent kernel, and bias–of the Bayesian LSTM are stored in a Bayesian LSTM cell instance (listing 6.5). They are modeled as (independent) multivariate Gaussians, representing the variational posterior in eq. 3.13. Training the

neural network with Bayes by Backprop (BBB) approximates the variational posterior to resemble the intractable and unknown true posterior. The variational posterior and its prior are trainable distributions in Tensorflow Probability. Their parameters are tracked in Tensorflow's computational graph and are updated by backpropagation, i.e., BBB.

The function `call()` of the `BayesianLSTMCell` class is called with the new input and the previous cell state at each time step t. The weights are estimated by MC sampling with a sample size of 1. Apart from that, it strictly follows eq. 2.8.[1]

```
1  class BayesianLSTMCell(tf.keras.layers.Layer):
2    # ...
3
4    def build(self, input_shape):
5      # ... (kernel, bias)
6
7      # recurrent kernel
8      recurrent_kernel_shape = (self.units, self.units * 4)
9      self.recurrent_kernel_posterior = mean_field_normal(
10       dtype, recurrent_kernel_shape, 'recurrent_kernel_posterior',
11       self.trainable, self.add_weight)
12
13     self.recurrent_kernel_prior = multivariate_normal(
14       dtype, recurrent_kernel_shape, 'recurrent_kernel_prior',
15       self.trainable, self.add_weight)
16
17   def call(self, inputs, states, training=None):
18     # previous memory state, previous carry state
19     h_tm1, c_tm1 = states
20
21     # MC sampling
22     kernel = self.kernel_posterior.sample()
23     recurrent_kernel = self.recurrent_kernel_posterior.sample()
24     bias = self.bias_posterior.sample()
25
26     z = K.dot(inputs, kernel)
27     z += K.dot(h_tm1, recurrent_kernel)
28     z = K.bias_add(z, bias)
29
```

[1] The implementation heavily uses numpy array notation. The variable `z` stores the inputs for all gates. The gates are later extracted into `z0`, `z1`, `z2`, and `z3` using numpy array slicing.

```
30      z0 = z[:, :self.units]
31      z1 = z[:, self.units: 2 * self.units]
32      z2 = z[:, 2 * self.units: 3 * self.units]
33      z3 = z[:, 3 * self.units:]
34
35      i = self.tf.sigmoid(z0)
36      f = self.tf.sigmoid(z1)
37      c = f * c_tm1 + i * self.tf.tanh(z2)
38      o = self.tf.sigmoid(z3)
39      h = o * self.tf.tanh(c)
40
41      return h, [h, c]
```

Listing 6.5: Bayesian LSTM cell class (simplified)

6.4 Hyperparameter

The binary and multiclass models are trained on a set of hyperparameters over 30 epochs with early stopping enabled:

- Bayesian method $\in$ {MC Dropout, BBB}
- LSTM units $u \in \{16, 64, 128\}$
- Embedding dimension $dims \in \{100, 300\}$
- Dropout probability $p_{drop} \in \{0.1, 0.25, 0.5, 0.76\}$
- Learning rate $= \begin{cases} 0.001, & \text{if epoch} < 5 \\ 0.001 \ exp(0.1(5 - epoch)), & \text{otherwise.} \end{cases}$
- Optimizer = Adaptive Moment Estimation (Adam)

The dropout probability is only used by MC Dropout. If BERT replaces the word embedding, the embedding dimension will be 768 (128 with BERT PCA).

In compliance with fig. 5.1, the last two dense layers have the output shape (batch size, M, $K \times 2 + 8$) and (batch size, M, K) respectively. Again, K denotes the number of labels in the classification and M the maximum sequence length (section 5.2).

6.5 Binary Classification

This section is about the sequence-to-sequence approach to sentiment analysis. It lists the performance of different models and hyperparameters introduced in previous sections. The experiments focus on epistemic uncertainty in various scenarios.

6.5.1 Dataset

The Internet Movie Database (IMDB) [58] sentiment classification dataset–subsequently referred to as IMDB dataset–holds 25,000 positive and 25,000 negative reviews. The reviews' average length is 234 words, or 251 tokens (table 6.2). Fig. 6.4 and appendix A.2 list text samples from the IMDB test dataset.

> **Negative**: *This was painful. I made myself watch it until the end, even though I had absolutely no interest in the plot, if there was one. My patience was not rewarded. The ending was even worse than the rest of the film. [...]*
> **Positive**: *It seems that no matter how many films are made on the subject, there is no shortage of stories that emerge from the Second World War. [...] The hardcore war film buff may find its exploration of relationships a bit off-putting, but it is on the whole an excellent film regardless of the bellicose element or not.*

Figure 6.4: Negative and positive IMDB reviews (shortened). The full reviews are available in figs. A.1 and A.2.

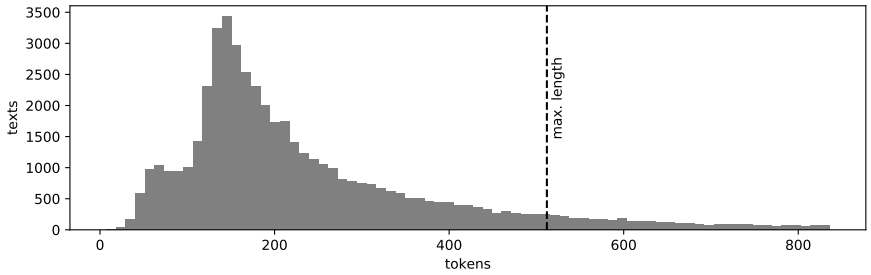


Figure 6.5: IMDB dataset: Number of tokens per text. The tokens are generated by a custom tokenizer (section 6.2.1). Here, the histogram only includes values up to the 98th percentile. The dotted line marks the maximum sequence length.

training size	22,500
test size	25,000
# tokens	
mean	251
25th percentile	139
50th percentile	189
75th percentile	304
98th percentile	836
labels	1 (sentiment)

Table 6.2: IMDB dataset statistics

6.5.2 Training

As mentioned in section 6.4, the models are trained over 30 epochs with early stopping and a decreasing learning rate. MC Dropout stops training based on its validation loss (figs. 6.6a, 6.6b, and 6.6c), whereas models with BBB use validation accuracy at the end of the sequence (figs. 6.6d, 6.6e, and 6.6f)[2].

Table 6.3 presents the performance scores. All scores are separated into a training score with g_{full}, test score with g_{full}, and test score with g_{end}. Higher scores at the end of the sequence justify the distinction between g_{full} and g_{end}.

Method	u	dims	p_{drop}	Accuracy %	Precision %	Recall %
MC Dropout	16	100	0.5	83.1/72.9/79.6	83.9/74.2/80.7	81.9/69.7/77.7
	64	100	0.1	85.5/76.2/84.1	85.1/78.5/86.8	86.2/71.6/80.3
	64	100	0.25	83.0/75.4/82.6	82.5/75.6/83.5	83.8/74.4/81.2
	64	100	0.5	88.3/73.6/80.8	88.7/76.1/83.2	87.8/68.4/77.0
	64	100	0.75	82.3/71.8/77.8	82.5/72.6/80.1	82.1/69.3/73.9
	64	300	0.5	86.8/73.5/80.4	88.6/74.9/81.1	84.4/70.3/79.3
	128	100	0.5	85.7/73.9/80.9	86.6/77.4/84.7	84.5/67.0/75.2
MC Dropout BERT	64	768	0.5	**90.7/88.3/89.8**	**90.6/88.1/89.3**	**90.8/88.6/90.5**
	64	128*	0.5	88.6/87.6/89.7	88.3/87.5/89.3	89.1/87.8/90.3
BBB	16	100	-	87.8/73.5/80.7	88.3/75.9/84.0	87.2/68.4/75.7
	64	100	-	83.9/71.1/77.8	82.5/71.2/78.4	86.2/70.1/76.7
	64	300	-	82.1/75.1/82.7	84.1/77.6/84.4	79.1/70.1/80.3
	128	100	-	83.4/72.0/78.7	84.0/70.8/76.7	85.9/74.3/82.2
BBB BERT	64	768	-	86.9/86.2/87.8	86.9/86.0/87.2	87.0/86.4/88.5
	64	128*	-	85.0/84.7/87.6	83.3/84.4/87.4	84.5/85.2/87.7

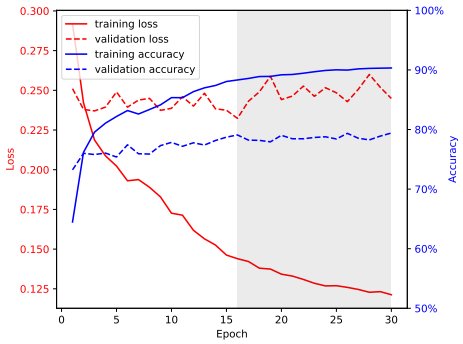
Table 6.3: Performance of IMDB models with MC Dropout and Bayes by Backprop. u: number of LSTM units in each cell; *dims*: word embedding dimension; p_{drop}: recurrent dropout rate. The metrics are divided into *training* (g_{full})/*test* (g_{full})/*test* (g_{end}) from 3 samples. **Bold scores** mark the overall best and underlined scores the best of each group. *The reduced embedding dimension $dims = 128$ is the result of a PCA transformation.

[2]Validation and training loss decrease monotonously for more than 30 epochs without noticeably improving the model's accuracy, precision, or recall score.

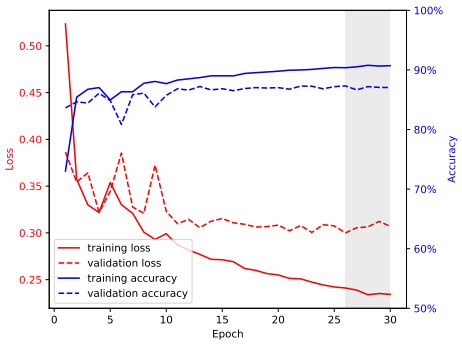
With BERT as a word embedding, MC Dropout and BBB achieve the best accuracy, precision, and recall scores. The training results are close to the test results with less than 2% difference. In the experiments, models with BERT are less likely to overfit. Interestingly, BERT performed only slightly better than BERT with PCA. Thus, the information needed in sentiment detection can be captured in much less than BERT's default 768 embedding dimensions for the base model. Nevertheless, there are disadvantages of using BERT combined with this approach, as demonstrated in section 7.5.4.

Models with MC Dropout and a trainable Keras word embedding showed the best results for lower dropout probabilities $p_{drop} = 0.1$ and $p_{drop} = 0.25$. To prevent the model with lower dropout rates from overfitting, it relies on early stopping. With $p_{drop} = 0.1$, the model trained over three epochs before early stopping kicked in. With $p_{drop} = 0.5$, it trained over 16 epochs.

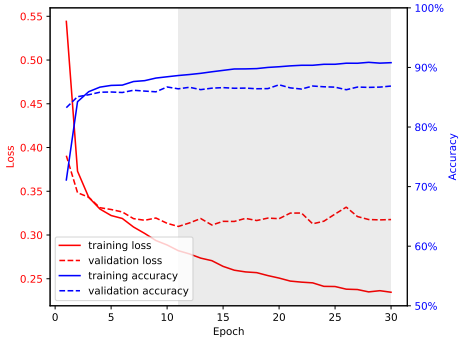
Models with BBB returned the highest test accuracy of 82.7% with 64 LSTM units and a 300-dimensional word embedding. Higher embedding dimensions and a greater number of LSTM units slightly increased the model performance but also increased the number of trainable model parameters.



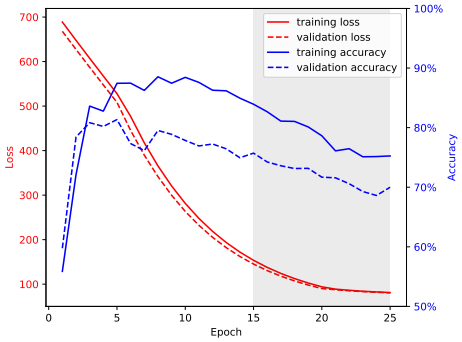
(a) MC Dropout, $u = 128$, $dims = 100$, $p_{drop} = 0.5$



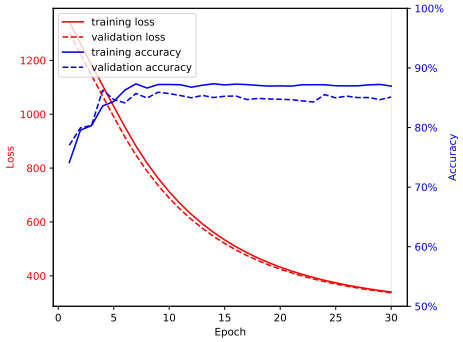
(b) MC Dropout (BERT), $u = 64$, $dims = 768$, $p_{drop} = 0.5$



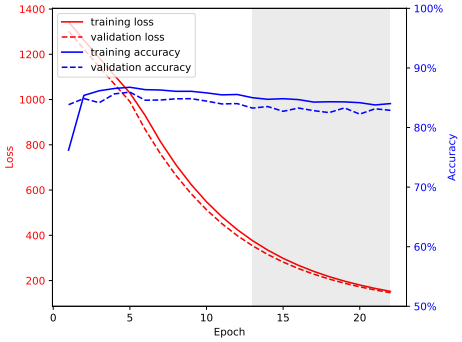
(c) MC Dropout (BERT), $u = 64$, $dims = 128$, $p_{drop} = 0.5$



(d) BBB, $u = 64$, $dims = 300$



(e) BBB (BERT), $u = 64$, $dims = 768$



(f) BBB (BERT), $u = 64$, $dims = 128$

Figure 6.6: Training and validation loss, and accuracy with early stopping enabled. Epochs in the gray area are ignored due to early stopping. (IMDB)

6.5.3 Benchmark Comparison

In 2019, XLNet [1] achieved state-of-the-art model performance with an accuracy of 96.21% (table 6.4). MC Dropout with BERT scored an accuracy of 89.8%, ranking last in comparison to XLNet and others. It performed best out of all the models that are experimented with. This approach's main goal is to detect uncertainty and not to replace highly optimized models. Therefore, the performance is sufficient.

Model	Accuracy %
XLNet (2019) [1]	96.21
BERT large + ITPT (2019) [23]	95.79
ULMFiT (2018) [59]	95.4
BCN + Char + CoVe (2017) [60]	91.8
MC Dropout BERT (own, 2020)	**89.8**
BBB 64 LSTM units, 300 dims (own, 2020)	**82.7**

Table 6.4: IMDB benchmarks

6.5.4 Input-Dependent Uncertainty

In this section, a Bayesian LSTM model is confronted with four different types of input uncertainty. Section **High Confidence** starts with a text input, that is easy to classify. If a review starts positively and ends negatively–or the other way around–it has mixed sentiment. This is studied in section **Mixed Sentiment**. Not every word is listed in the word index. Bad grammar and wrong spelling may cause unknown words. Section **Unknown Words** addresses their effect. To show the model's capabilities, a weather prediction is used as an input in section **Out-of-Distribution Example**.

Unless otherwise stated, they are evaluated against Bayes by Backprop (BBB) with a 300-dimensional Keras word embedding and 64 LSTM units with 75 samples.[3] The graphs are separated into two sub graphs. The first one tracks the sample mean in blue in addition to its standard deviation as a gray area around the sample mean. The second graph plots the course of the standard deviation along with the uncertainty threshold $\alpha = 0.2$ as a horizontal dotted line. All texts are manually selected. The results are representative of most inputs in the respective category.

[3]The evaluation chapter 7 includes a comparison between different models.

High Confidence

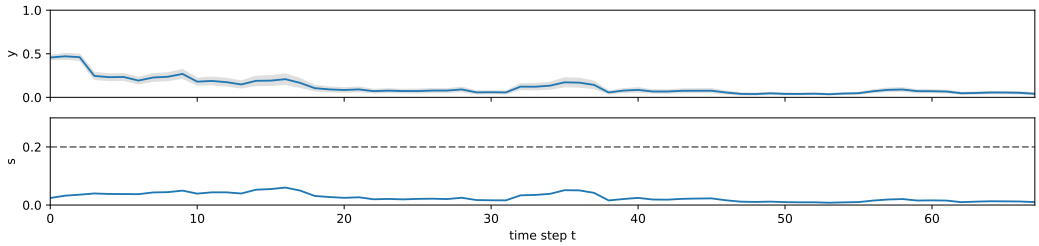


Figure 6.7: Negative sentiment with high confidence. The true label is negative. See also: fig. A.1.

In an unambiguous case, the model should be confident about its prediction. In fig. 6.7, the negative review fits this description with $U_{g_{linear}} \approx 0.02$ and $\hat{y}_{g_{linear}} \approx 0.08$. The token `painful` at time step 3 sets the sentiment for upcoming tokens. Section 7.3 also discusses the impact of sensitive terms. Fig. A.1 contains the full input text.

High Uncertainty

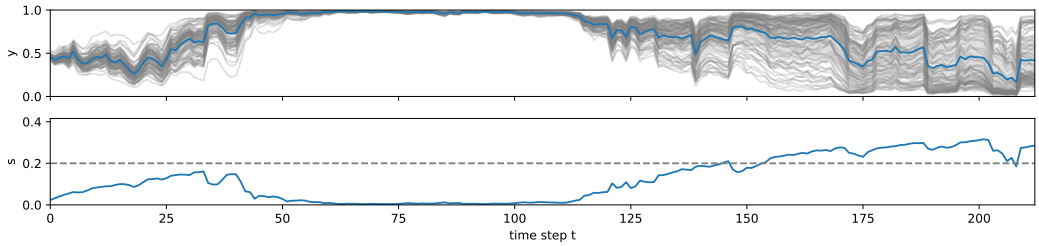


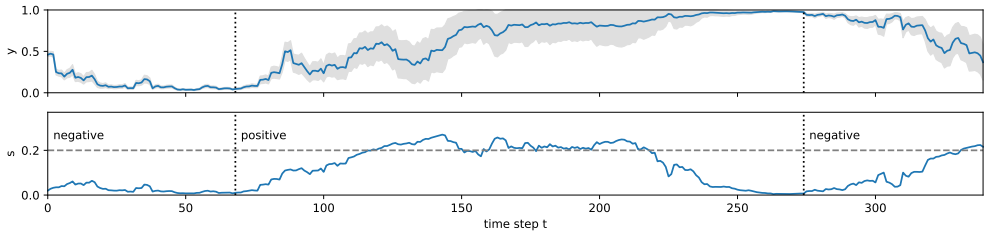
Figure 6.8: Mostly positive sentiment with high uncertainty at the end. The true label is negative. See also: fig. A.3.

Fig. 6.8 shows a sentiment analysis with high uncertainty in the second part of the review. First, the reviewer shortly summarizes the plot and points out some good aspects about the movie. The model classifies this part as mostly positive. In the second half, the reviewer rates the movie negatively. The model returns a high uncertainty for this transition phase with $U_{g_{full}} = 0.12$, $U_{g_{linear}} = 0.17$, and $U_{g_{end}} = 0.28$. The weighted predictions $\hat{y}_{g_{full}} = 0.7$, $\hat{y}_{g_{linear}} = 0.66$, and $\hat{y}_{g_{end}} = 0.41$ also reveal that later time steps tend towards a negative sentiment. It is similar to section **Mixed Sentiment**.

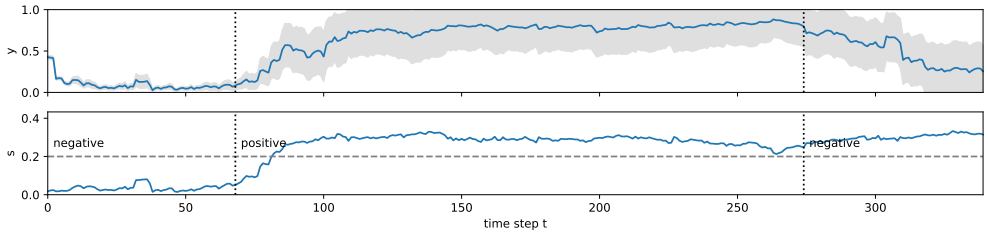
In addition to the standard deviation, every sample is plotted. If the model is uncertain,

the reviews often have a more complex syntax that might not be fully covered by the training dataset. Fig. A.3 contains the full input text.

Mixed Sentiment



(a) BBB



(b) MC Dropout

Figure 6.9: Mixed sentiment classification with high uncertainty in the second and third review. The true label is unknown. The negative review (fig. A.3) is included before and after the positive review. See also: fig. A.4.

Uncertainty estimations should reflect whenever the sentiment of a review changes drastically at certain time steps. In fig. 6.9, a negative review encloses a positive review such that it is inserted at the beginning and the end of the positive review. The respective time steps are marked in the figure.

After the dotted vertical line, both models display a rise in uncertainty. For MC Dropout, the prediction remains uncertain. BBB's uncertainty fades out at the end of the second review. BBB records $U_{g_{end}} = 0.21$ which is two times higher than $U_{g_{full}} = 0.1$. On the other hand, MC Dropout records $U_{g_{end}} = 0.31$ which is only 35% higher than $U_{g_{full}} = 0.23$ but also 48% higher than BBB's uncertainty estimation. Based on this experiment, the uncertainty in BBB models tends to fade out more rapidly if the review is long enough. MC Dropout seems to remember the first part of the review over a long period of time. Fig. A.4 contains the full input text.

Unknown Words

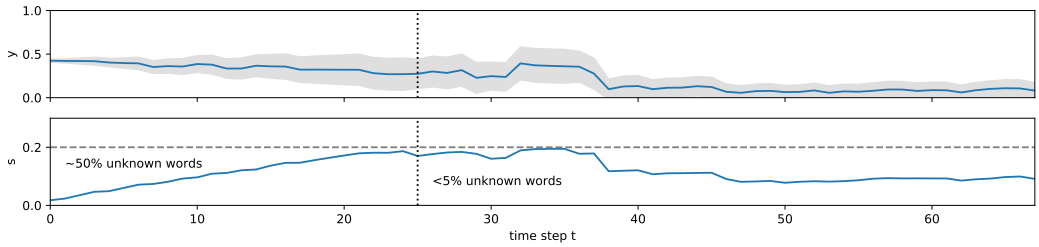


Figure 6.10: Sentiment analysis with a high rate of unknown words and high uncertainty. 50% of all words are obscured such that they are not listed in the word index (`[UNK]` token). The true label is negative. See also: fig. A.1.

Many user reviews contain spelling errors. In this experiment, 50% of the first 25 words are randomly replaced with the `[UNK]` token. The review in fig. A.1 serves as a basis:

```
[START] [UNK] [UNK] [UNK] . [UNK] [UNK] myself watch [UNK] until [UNK] end [UNK] though
[UNK] [UNK] absolutely [UNK] [UNK] [UNK] [UNK] plot if [UNK] wa one .  my patience wa not
rewarded .  the ending wa even worse (...)
```

In comparison to fig. 6.7, the uncertainty increases at the start. Interestingly, the prediction still tends towards a negative sentiment, which is the correct classification. Whether the `[UNK]` token increases uncertainty or does not affect it at all highly depends on the input and previous time steps. It is beneficial to know the unknown to known words ratio to evaluate uncertainty estimations.

Out-of-Distribution Example

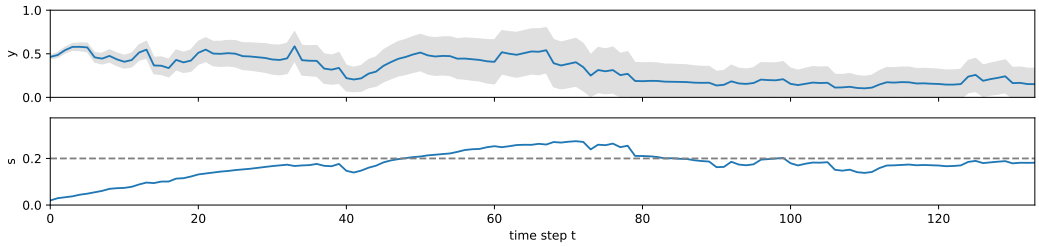


Figure 6.11: Sentiment analysis of an out-of-distribution example (weather forecast). The true label is unknown. See also: fig. A.5.

As a last experiment, the Bayesian LSTM is confronted with an out-of-distribution example (fig. 6.11). It is selected from a random weather forecast [61].

Ideally, the prediction should be uncertain with no preference towards positive or negative sentiment. The experiment shows that the uncertainty is relatively high with $U_{g_{full}}$, $U_{g_{linear}}$, and $U_{g_{end}} > 0.18$. The sample mean tends towards a negative sentiment, which results in $\hat{y}_{g_{linear}} = 0.25$. The uncertainty never really decreases noticeably. Fig. A.5 contains the full input text.

6.6 Multiclass Classification

This section contains experiments with multiclass classification. Similar to section 6.5, it lists the performance of different models and hyperparameters. Again, the experiments focus on epistemic uncertainty. Multiclass classification demonstrates the application of higher dimensional output distributions with $K > 1$.

6.6.1 Dataset

The Amazon customer review [62] multiclass dataset–subsequently referred to as Amazon dataset–provides 30+ different categories of product reviews. Six of which were chosen for the classification task. Unlike sentiment analysis, the objective is to classify the product category based on the user review. The motivation to use this dataset is its size of 130+ million reviews and the overall short reviews. For more details, refer to fig. 6.12 and table 6.5. Text samples from the Amazon test dataset are listed in fig. 6.13, as well as in appendix A.2.

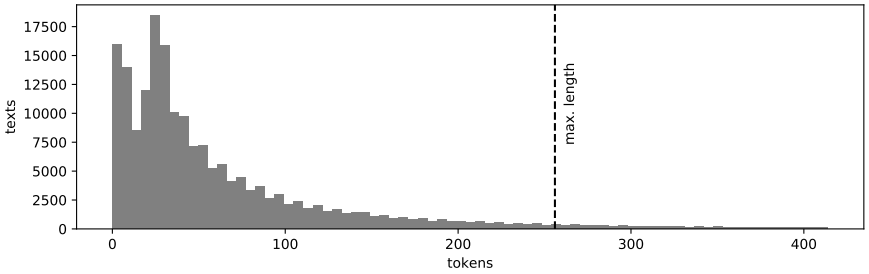


Figure 6.12: Amazon dataset: Number of tokens per text. The tokens are generated by a custom tokenizer (section 6.2.1). The histogram only includes values up to the 98th percentile. The dotted line marks the maximum sequence length.

training size	90,000
test size	90,000
# tokens	
mean	75
25th percentile	21
50th percentile	38
75th percentile	83
98th percentile	414
labels	6 (software, camera, home entertainment, outdoors, shoes, jewelry)

Table 6.5: Amazon dataset statistics. Only a subset of the 130+ million reviews is used in the experiments.

> *Fits both of my vehicles (2008 Chrysler Pacifica and 1996 Cadillac Fleetwood). Just finished a 1100 mile trip across country carrying my cycle. [...] I will reevaluate again once I add a second bike to the rack to see how it handles.*

Figure 6.13: Amazon review in the outdoors category. The full review is available in fig. A.6.

6.6.2 Training

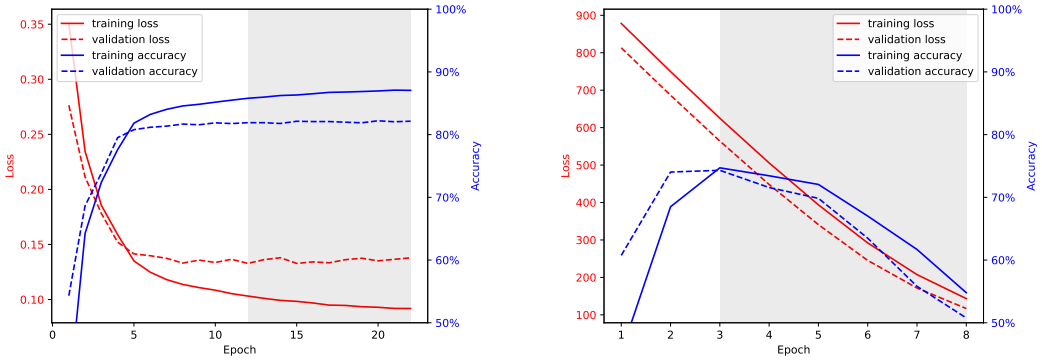
Table 6.6 shows that all MC Dropout models achieved similar test results with around 81.5% accuracy, 90.0% precision, and 77% recall for g_{end}. The ratio between these scores is about the same for BBB, although significantly lower in absolute terms. All models have a higher rate of *false negatives* than *false positives* demonstrated by high precision and relatively low recall scores:

$$\underbrace{\frac{true\ positives}{true\ positives + false\ positives}}_{\text{precision}} > \underbrace{\frac{true\ positives}{true\ positives + false\ negatives}}_{\text{recall}} \tag{6.1}$$

BBB performed worse than MC Dropout. In fig. 6.14b, BBB's accuracy starts to decrease after three epochs, and it stops training. This observation differs from the binary classification training results in section 6.5.2 in which both methods performed similarly. Section 7.5.3 discusses this problem in greater detail.

Method	u	dims	p_{drop}	Accuracy	Precision	Recall
MC Dropout	16	100	0.5	86.2/**82.7**/81.7	94.3/90.7/89.5	82.1/78.7/77.4
	64	100	0.1	86.7/82.6/81.7	94.9/90.9/90.0	82.7/78.5/77.1
	64	100	0.25	**87.0**/82.3/81.6	**95.1**/90.7/89.7	**83.0**/78.0/76.9
	64	100	0.5	85.8/82.1/81.1	94.1/90.6/89.6	81.6/77.9/76.3
	64	100	0.75	86.4/82.6/**81.8**	94.4/90.6/89.6	82.4/**78.7**/**77.4**
	64	300	0.5	85.5/82.6/81.7	94.2/90.8/89.9	81.1/78.6/77.1
	128	100	0.5	85.1/82.2/81.2	93.4/**91.2**/**90.3**	80.9/77.7/76.0
BBB	16	300	-	74.8/73.1/69.5	84.2/82.3/78.3	68.3/67.4/62.7
	64	100	-	73.3/71.2/67.8	83.4/81.7/78.3	66.6/64.6/60.1
	64	300	-	74.7/73.3/70.3	85.6/83.9/80.7	67.7/67.0/62.7
	128	300	-	74.5/73.6/71.3	84.0/82.8/80.0	68.3/68.5/65.6

Table 6.6: Performance of Amazon models with MC Dropout and Bayes by Backprop. u: number of LSTM units in each cell; *dims*: word embedding dimension; p_{drop}: recurrent dropout rate. The metrics are divided into *training* (g_{full})/*test* (g_{full})/*test* (g_{end}) from 3 samples. **Bold scores** mark the overall best and underlined scores the best of each group.



(a) MC Dropout, $u = 128$, $dims = 100$, $p_{drop} = 0.5$ **(b)** BBB, $u = 128$, $dims = 300$

Figure 6.14: Training and validation loss, and accuracy with early stopping enabled. Epochs in the gray area are ignored due to early stopping. (Amazon)

6.6.3 Correlation

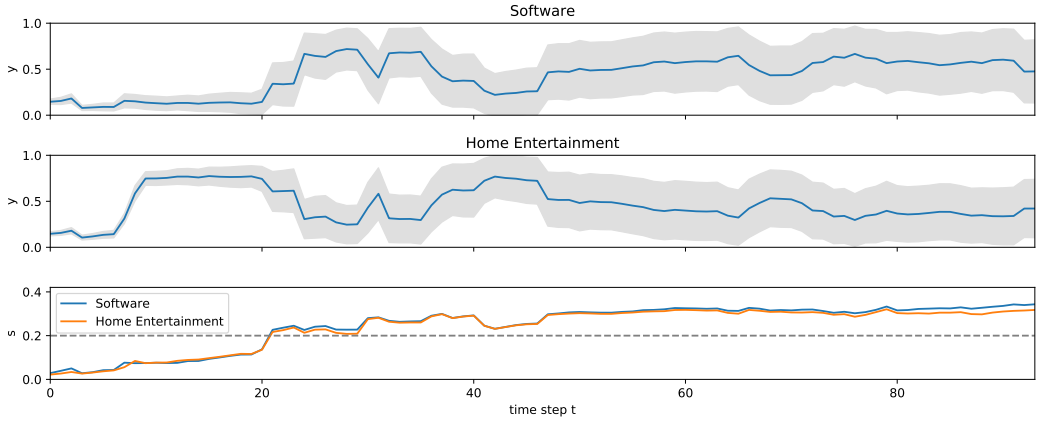


Figure 6.15: Correlation between two semantically similar labels. See also: fig. A.7.

One property of the softmax function is that all values must add up to one. Figs. 6.15 and A.7 show two negatively correlated labels. At any time step, the sum of both predictions is approximately one; the predictions of other labels are close to zero. The plot shows that the review is only associated with *Software* and *Home Entertainment*. They also share a similar standard deviation. The negative correlation between both labels causes the high uncertainty in the prediction:

$$U_{g_{linear}}(Software) \approx U_{g_{linear}}(Home\ Entertainment) \approx 0.32$$

In figs. 6.16 and A.8, the labels cannot be distinguished that easily. The setup is similar to the mixed sentiment experiment in section 6.5.4.

The first 79 tokens are predominantly labeled as *Shoes*. The model is confident until the *Software* review starts. Appending the *Software* review to the existing one demonstrates a multiclass classification that abruptly changes mid-sequence. Now, the sample mean of the label *Shoes* decreases and the uncertainty increases. The model favors all technology-related labels *Camera*, *Home Entertainment*, and *Software* until the true label *Software* is detected at time step 125. It is also this time step that sets the uncertainty of most labels close to zero except for *Home Entertainment*. At the last time step, the label *Software* has $\hat{y}_{g_{end}}(Software) \approx 0.97$ with very low uncertainty.

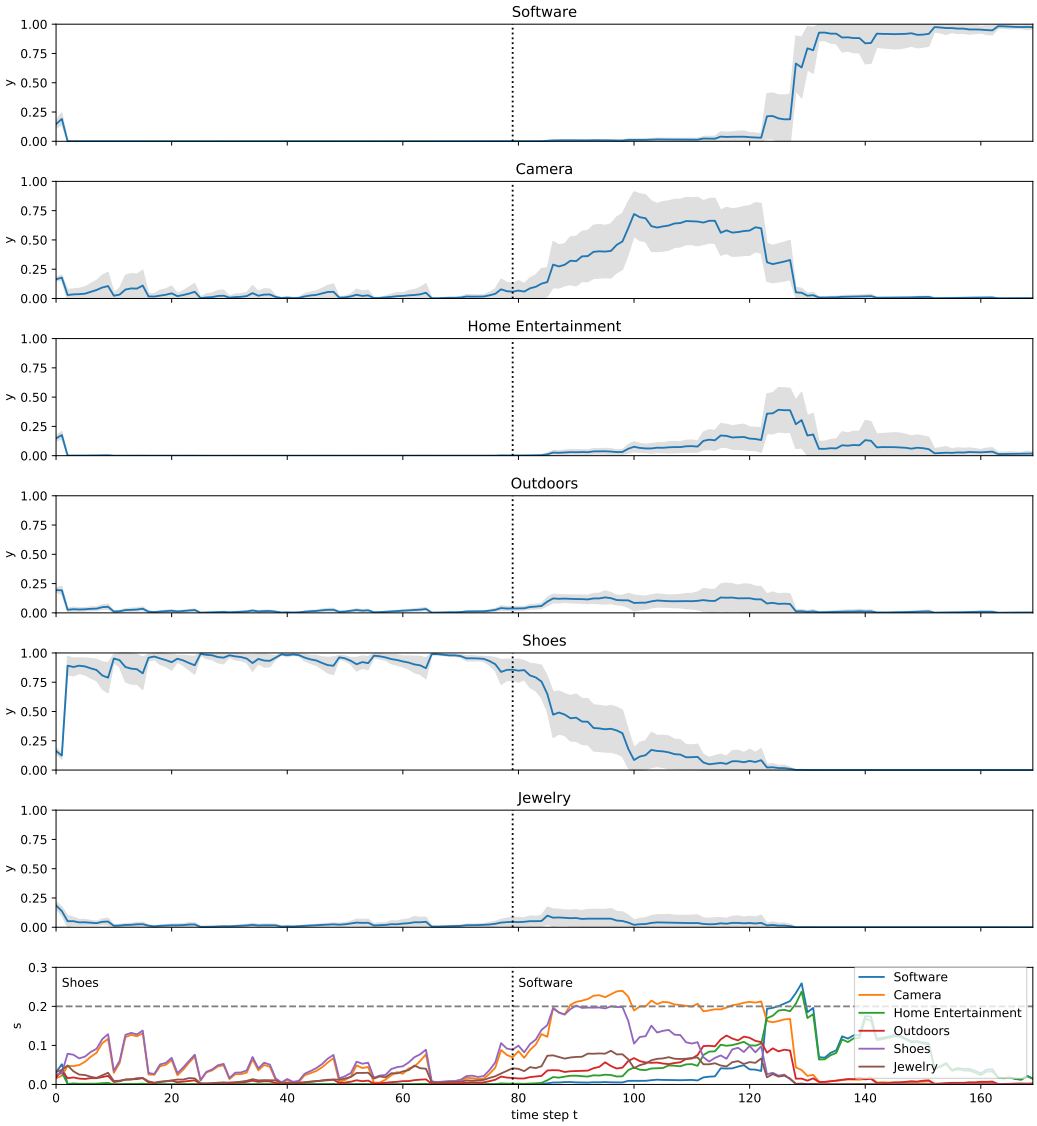


Figure 6.16: Uncorrelated labels. The first 79 tokens are from a review in the *Shoes* category. The remaining tokens are from a review in the *Software* category. See also: fig. A.8.

6.7 Proof of Concept

This section outlines the *proof of concept* application. It is designed to visualize sequence-to-sequence text classification in real-time. It utilizes a subset of the models presented in previous sections and supports binary and multiclass classification.

6.7.1 Software

The *proof of concept* application consists of multiple components, each bundled into a Docker [63] image and managed by Docker-Compose. Docker's virtual network allows for internal communication. External access is handled over HTTP with an NGINX [64] reverse proxy configuration.

The backend (fig. 6.17) is written in Python. It depends on the same libraries as the experiments (section 6.1). Additionally, the API uses the Flask framework [65] for its HTTP server.

The UI is created with React [66] and is written in TypeScript. React is a library to manage application state and component-based development. The application runs in a browser.

6.7.2 Architecture Overview

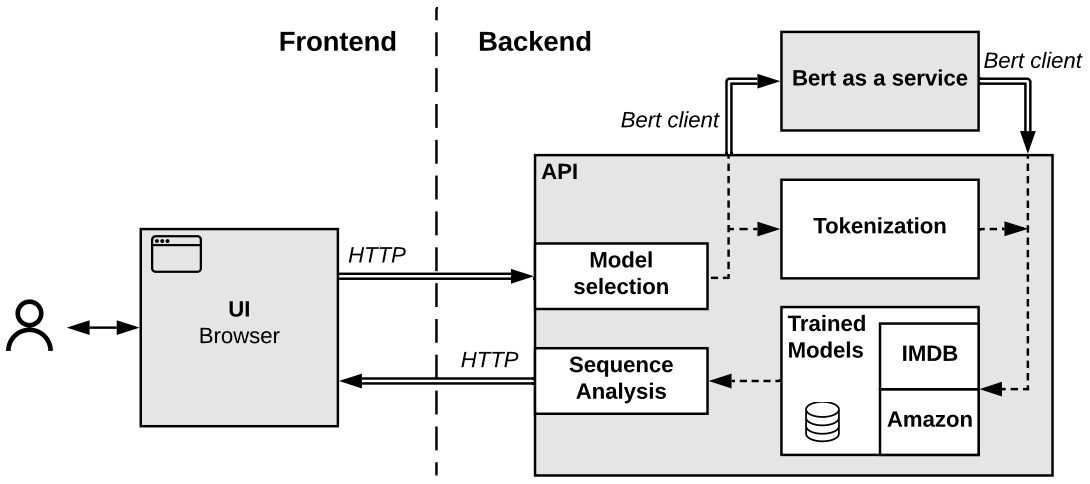


Figure 6.17: Proof of concept architecture. The gray boxes indicate independent components.

Fig. 6.17 shows the interaction between the user and the API. The user selects the classification problem and other parameters listed in section 6.7.3 and sends an HTTP request with the input text to the API. Based on the parameters of the HTTP request, the API determines the matching model and whether to use the BERT or Keras embedding layer. Then, the input text is tokenized and forwarded to the trained Tensorflow model. The model predicts the label sequence $\hat{y}$ (eq. 5.1) with the specified number of forward passes.

The API determines the weighted sample mean $\hat{y}_g$ (eq. 5.5) and uncertainty U_g (eq. 5.11) for each weighted sequence g_{full}, g_{linear}, and g_{end} after the prediction has been finished. Now, the client receives the sequence analysis in JSON format. Appendix A.1 contains the full response schema.

6.7.3 User Interface

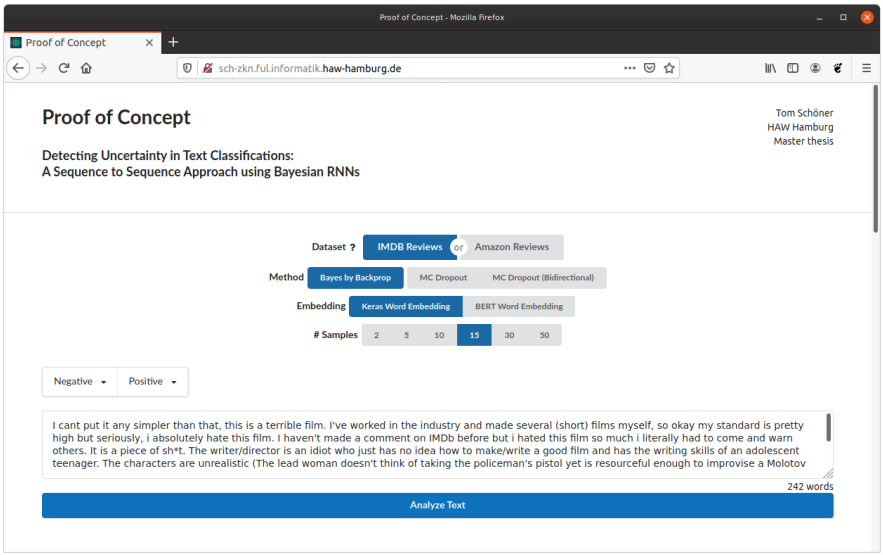


Figure 6.18: Proof of concept landing page with parameters and text area.

The user interface (fig. 6.18) is designed to provide an information-rich and visually pleasing sequence analysis. It supports multiple models and hyperparameters. A section at the top of the page allows the user to select the classification problem represented by the dataset, the Bayesian inference method, the word embedding, and the number of samples. Not all combinations are possible; for example, none of the Amazon models support the contextual BERT word embedding.

A text area is located below the parameter section. The dropdown menu above has a handful of text samples from both test dataset that are sorted by label. After formulating the input text, the user can click on the blue *Analyze Text* button to reveal the sequence analysis (figs. 6.19, 6.20, and 6.21).

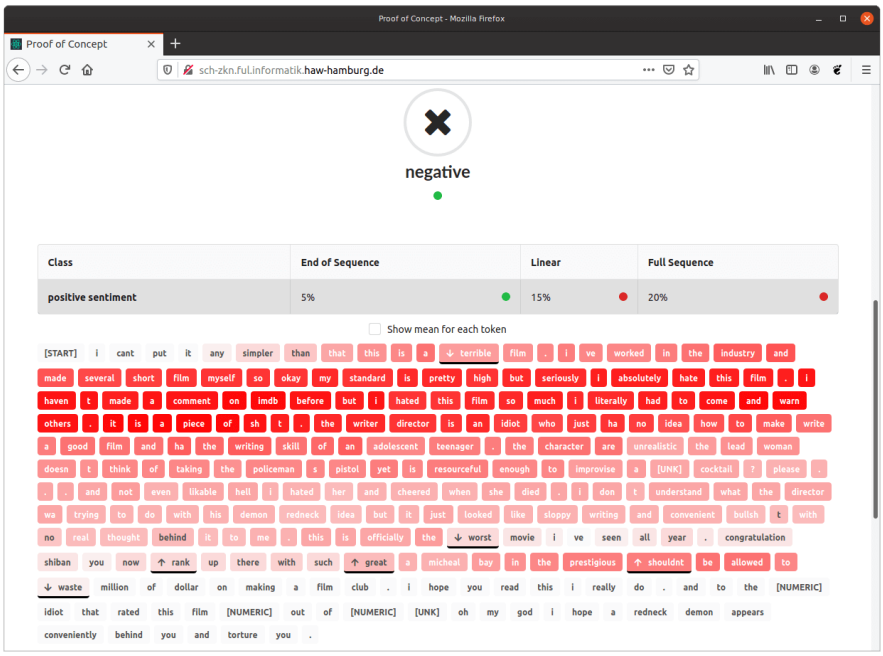


Figure 6.19: Binary prediction analysis.

The analysis section is divided into four subsections. Figs. 6.19 and 6.20 show screenshots of the first three subsections for binary and multiclass classification.

The first subsection highlights the predicted label with an icon, an uncertainty estimation as a color, and, in case of multiclass classification, a radar chart with all labels. The threshold α (section 5.4.2) determines the color (red to green) an uncertainty estimation is reduced to. Red signifies high uncertainty; green high confidence. The prediction and uncertainty estimation are calculated using the weight sequence g_{end}.

The table contains additional information about all labels. It includes predictions and uncertainty estimations for g_{end}, g_{linear}, and g_{full}. Clicking on a row selects the label for the upcoming subsections. This is only relevant for multiclass classification (fig. 6.20).

As mentioned above, the next two subsections depend on the selected label. The first one marks uncertain tokens with a shade of red. It shows confident and uncertain segments in the sequence. Selecting the checkbox above the list of tokens displays the sample

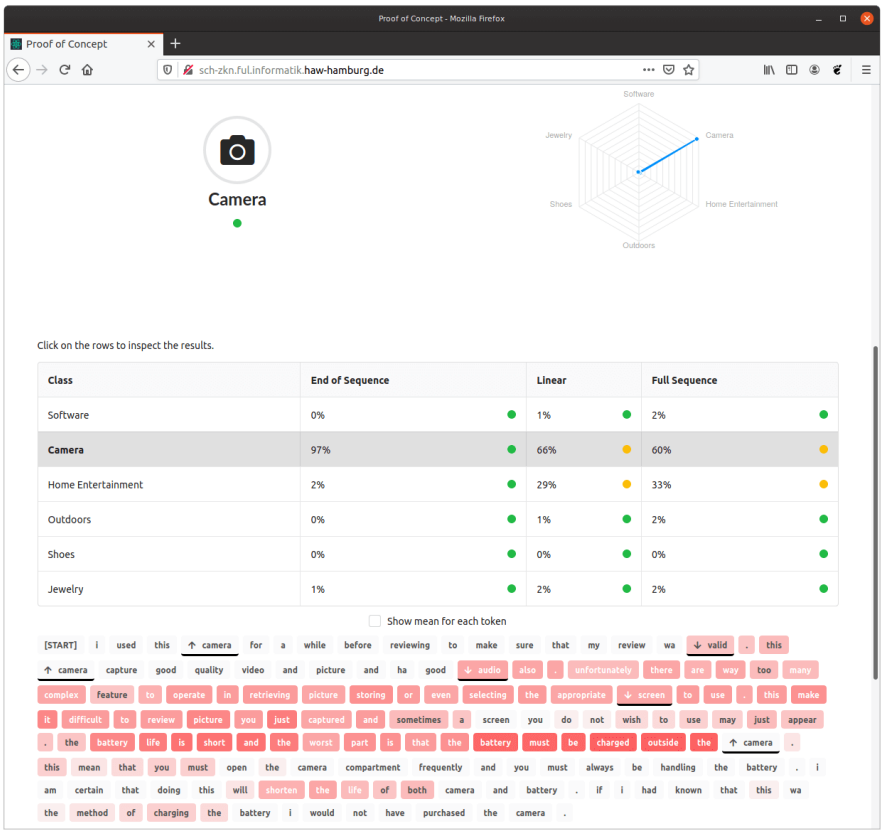
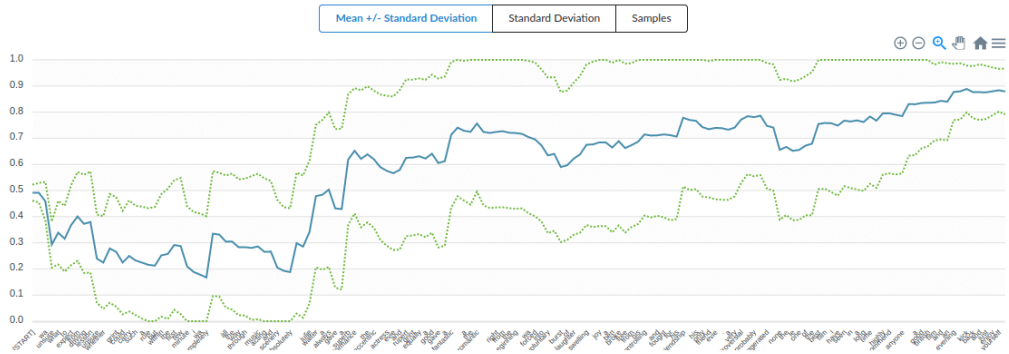


Figure 6.20: Multiclass prediction analysis.

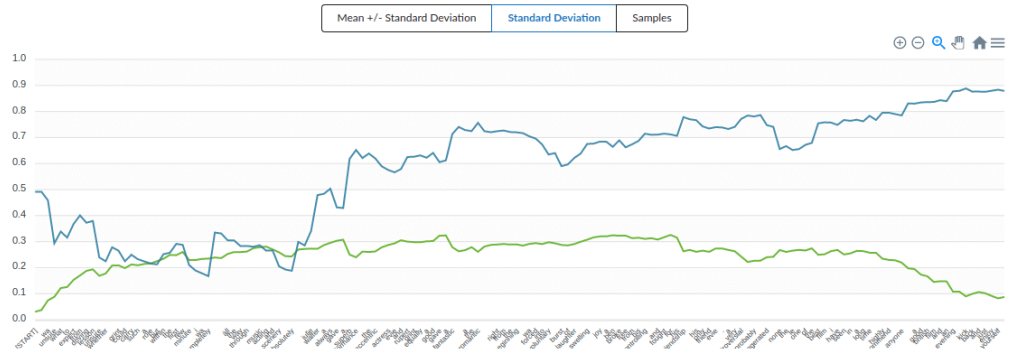
mean next to each token. Furthermore, the three most sensitive terms with the highest positive or negative impact are highlighted with a black border and an arrow to their left, indicating the type of change. For more information on sensitive terms, refer to section 7.3.

The last subsection (fig. 6.21) is an interactive graph over all time steps with multiple display modes: *mean* ± *standard deviation*, *standard deviation*, and *samples*. Simply displaying the samples can be hard to read. Therefore, the two other modes were introduced. The sample means in blue are visible in all modes.

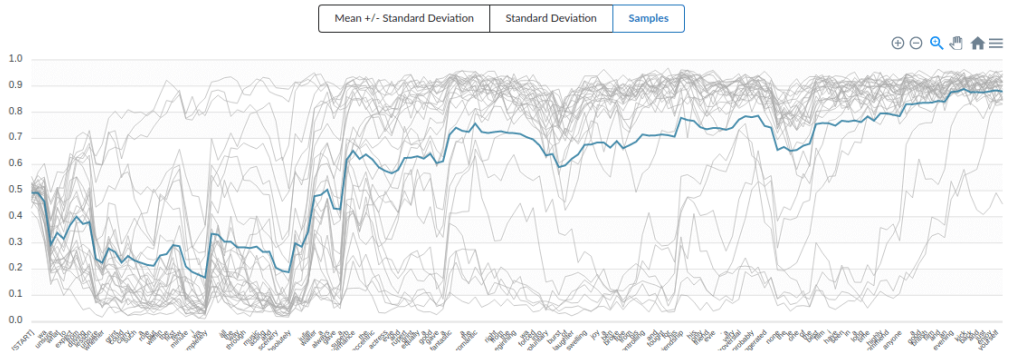
The view will update with each new request. The diverse data visualization tools help to understand uncertainty in text classification by reducing the available information into comprehensible representations.



(a) Mean ± standard deviation



(b) Standard deviation



(c) Samples

Figure 6.21: Interactive sequence graph over all tokens with multiple display modes.

7 Evaluation

This chapter evaluates the sequence-to-sequence classification approach to binary and multiclass classification with Bayes by Backprop (BBB) and Monte Carlo (MC) Dropout. It focuses on uncertainty estimations and the performance implications.

Section 7.1 starts with a model comparison. It compares the relation between confidence and accuracy, based on uncertainty thresholds. Section 7.1.2 compares the binary classification models using the distribution of uncertainty over various weighted predictions $\hat{y}_g$.

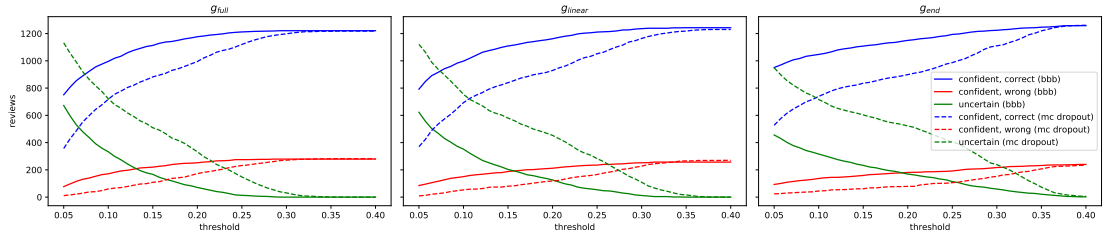
In the experiments, stopwords were not removed. Section 7.2 evaluates the implications of this decision. Unlike stopwords, sensitive terms are expected to affect the outcome of a prediction. Section 7.3 discusses the detection of sensitive terms and their influence on uncertainty.

The last two sections 7.4 and 7.5 evaluate whether the goals were reached and what known complications occurred during the training and experiments.

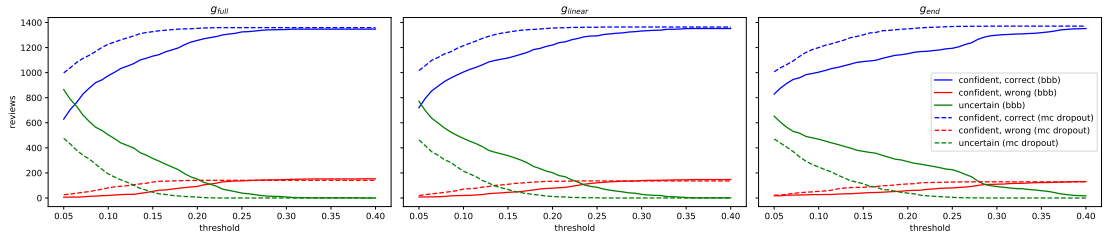
7.1 Model Comparison

The experiments have shown that Bayes by Backprop (BBB) and Monte Carlo (MC) Dropout are capable of estimating epistemic uncertainty over a sequence while maintaining acceptable performance scores (tables 6.3 and 6.6). BBB suffered in multiclass scenarios, performing approximately 15% worse than its MC Dropout counterpart. In comparison, using the Keras word embedding and $g = g_{end}$, BBB ($u = 64$ and $dims = 300$) achieved 82.7% accuracy in binary classification. This is only 1.7% worse than MC Dropout ($u = 64$, $dims = 100$, and $p_{drop} = 0.1$) with 84.1% accuracy and similar to MC Dropout models with higher dropout rates.

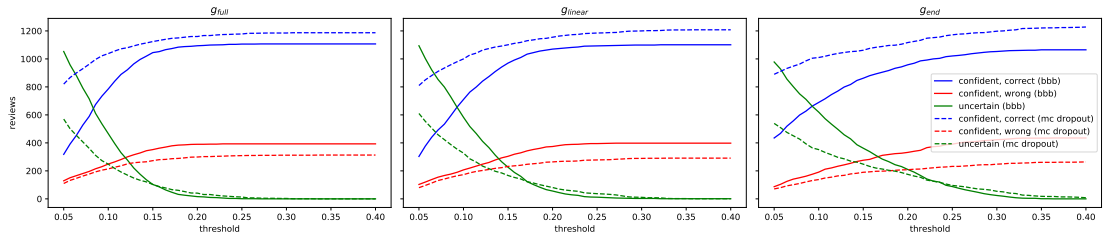
7.1.1 Confidence and Accuracy



(a) IMDB dataset (Keras embedding)



(b) IMDB dataset (BERT embedding)



(c) Amazon dataset (Keras embedding)

Figure 7.1: Model confidence with various thresholds. Each statistic consists of 1,500 data points from 8 samples. Fig. 7.1c only evaluates the uncertainty of the predicted label against the threshold; others are not shown.

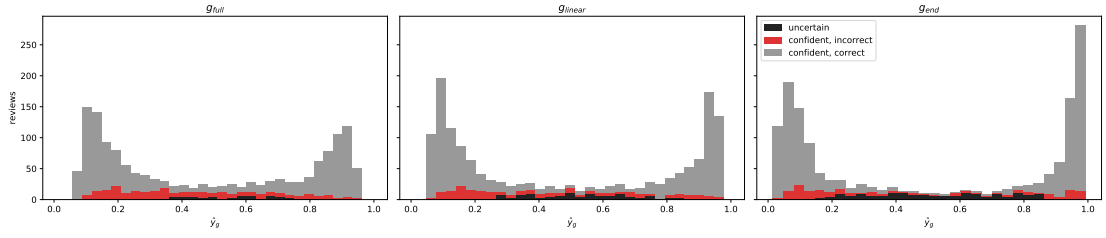
Section 5.3 introduced weighted arithmetic means that enable quantifying uncertainty over the whole sequence. Fig. 7.3 shows the number of confident correct predictions (blue line), confident incorrect predictions (red line), and uncertain predictions (green line) over different thresholds. BBB and MC Dropout are represented by a solid and a dotted line respectively. Confident incorrect predictions include false positives as well as false negatives. It is desirable to choose a threshold that has few confident incorrect predictions but maintains a high number of confident correct predictions. A greater threshold increases the number of correct and incorrect results with high confidence.

Overall, MC Dropout has fewer confident incorrect predictions than BBB. With BERT, MC Dropout also has lower uncertainty in binary classification and multiclass classifica-

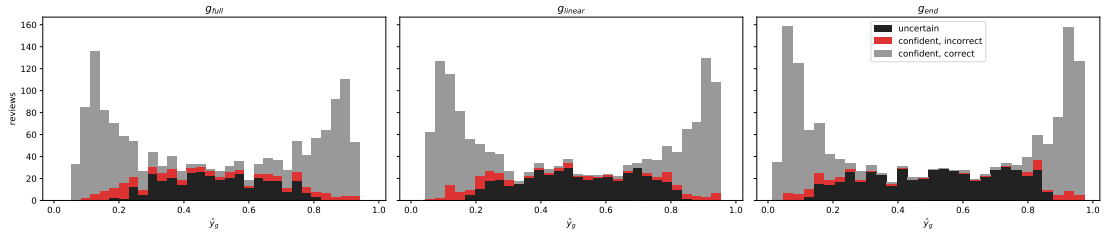
tion (figs. 7.1b and 7.1c). Yet, it also has up to 75% uncertainty in binary classification for small thresholds (fig. 7.1a). A high number of confident correct predictions with relatively few confident incorrect predictions indicates that the model can successfully identify known unknowns.

In figs. 7.1a and 7.1b, all identically colored lines converge to approximately the same value for larger thresholds. Large thresholds take uncertainty out of the equation. This shows that all binary classification models perform quite similarly but evaluate uncertainty differently. This is not the case in fig. 7.1c, in which MC Dropout performed significantly better.

7.1.2 Distribution of Uncertainty in Binary Classification



(a) BBB



(b) MC Dropout

Figure 7.2: Distribution of uncertainty over the weighted prediction in BBB and MC Dropout with $\alpha = 0.2$. It uses 1,500 data points from the IMDB test dataset with 8 samples.

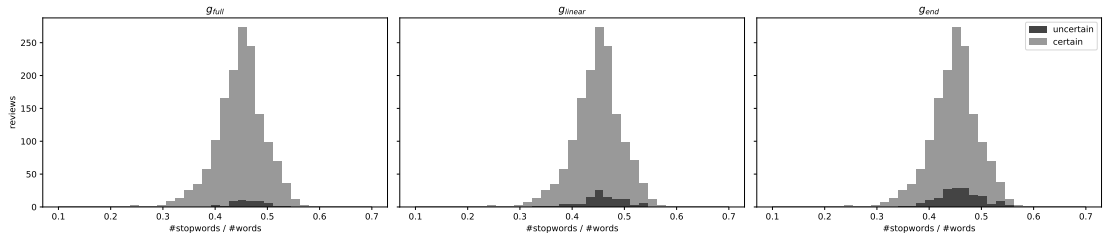
Fig. 7.2 reveals that the weighted uncertainty U_g is not uniformly distributed over $\hat{y}_g$. The figure represents the cross section at $\alpha = 0.2$ in fig. 7.1a. Predictions close to zero or one are very confident. Predictions close to 0.5 are mostly uncertain. This is further explained in section 7.5.2. The high number of uncertain predictions with MC Dropout is also apparent in fig. 7.2b.

All histograms show that the samples accumulate near zero and one if later time steps

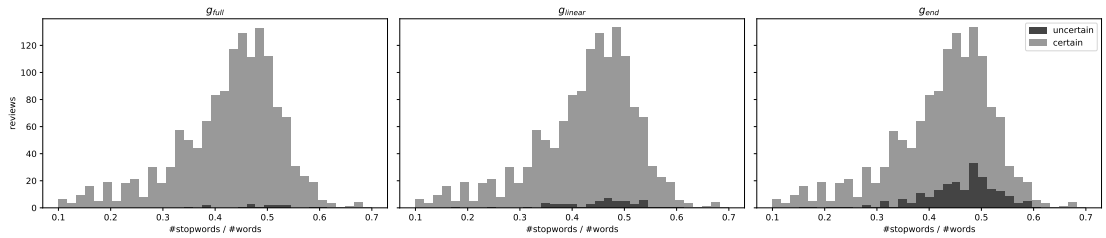
are prioritized by the weight sequence g. Despite BBB's low uncertainty, MC Dropout has fewer confident incorrect predictions close to zero and one. This coincides with the observations in section 7.1.1.

7.2 Stopwords

It is a common approach to remove stopwords from input sequences since they are believed to add little value to the classification. In the experiments, stopwords were not removed during preprocessing. Therefore, their effect on predictions and uncertainty can be evaluated. Fig. 7.3a shows the impact of various stopword-to-word ratios on the uncertainty in predictions. Stopwords make up almost half of all words.



(a) IMDB dataset



(b) Amazon dataset

Figure 7.3: Stopword-to-word distribution over 1,500 data points from 8 samples with $\alpha = 0.2$ (BBB).

In fig. 7.3b, the distribution is more diverse, mostly due to very short input sequences in the Amazon dataset, see fig. 6.12. In fig 7.3a, the mean value of stopword-to-words in uncertain predictions is 0.46 . It is only slightly larger than in confident predictions with 0.45 in the rightmost histogram (0.45 to 0.42 in fig. 7.3b). Therefore, stopwords increase uncertainty to some extent but do not seem to be its primary cause. They were negligible in sentiment analysis.

On the other hand, the removal of stopwords from the test dataset had a noticeable effect on model performance. In tables 6.3 and 6.6, the IMDB test scores varied by around 1%. The Amazon test scores improved by 2 - 4%.[1] Removing stopwords from the training dataset might have yielded even better results but was not pursued further.

7.3 Sensitive Terms

	Positive Term	$\frac{\Delta \bar{x}_t}{\Delta t}$	Count	Percentage	Negative Term	$\frac{\Delta \bar{x}_t}{\Delta t}$	Count	Percentage
1	wonderfully	0.25	131	0.10%	worst	-0.28	1664	1.33%
2	outstanding	0.22	190	0.15%	poorly	-0.24	335	0.27%
3	favorite	0.22	732	0.59%	awful	-0.23	878	0.70%
4	brilliant	0.21	683	0.55%	pointless	-0.22	226	0.18%
5	amazing	0.21	653	0.52%	seagal	-0.21	132	0.11%
6	perfectly	0.21	308	0.25%	disappointing	-0.21	214	0.17%
7	awesome	0.21	293	0.23%	terrible	-0.21	828	0.66%
8	excellent	0.21	1083	0.87%	horrible	-0.21	673	0.54%
9	loved	0.21	866	0.69%	poor	-0.20	950	0.76%
10	funniest	0.20	222	0.18%	terribly	-0.20	130	0.10%
11	friendship	0.20	132	0.11%	waste	-0.20	629	0.50%
12	rare	0.20	191	0.15%	suck	-0.20	234	0.19%
13	unusual	0.20	146	0.12%	badly	-0.20	285	0.23%
14	subtitle	0.19	140	0.11%	disappointment	-0.20	235	0.19%
15	surprisingly	0.19	222	0.18%	laughable	-0.20	171	0.14%
16	terrific	0.19	184	0.15%	hoping	-0.19	223	0.18%
17	superb	0.18	271	0.22%	rented	-0.19	235	0.19%
18	gem	0.18	210	0.17%	sorry	-0.19	391	0.31%
19	simple	0.18	552	0.44%	pathetic	-0.19	206	0.16%
20	vhs	0.18	164	0.13%	boring	-0.19	861	0.69%
21	fun	0.18	1552	1.24%	disappointed	-0.18	558	0.45%
22	genuinely	0.18	128	0.10%	ridiculous	-0.18	402	0.32%
23	surprised	0.18	515	0.41%	mediocre	-0.18	157	0.13%
24	lucky	0.18	132	0.11%	wasted	-0.18	270	0.22%
25	delivers	0.17	182	0.15%	dull	-0.18	343	0.27%
26	ride	0.17	267	0.21%	lame	-0.18	257	0.21%
27	powerful	0.17	261	0.21%	mess	-0.17	204	0.16%
28	perfect	0.17	678	0.54%	fails	-0.17	247	0.20%
29	unique	0.17	264	0.21%	annoying	-0.17	382	0.31%
30	wonderful	0.16	684	0.55%	bunch	-0.17	336	0.27%

Table 7.1: Top 30 most sensitive terms in the IMDB test dataset. For each review the top five most positive and negative sensitive terms are extracted. $\frac{\Delta \bar{x}_t}{\Delta t}$ denotes the mean delta over all occurrences.

In text classification, a subset of terms has a greater influence on predictions than others. In hate speech detection, these are typically called sensitive terms [49]. This description also fits sentiment analysis and is therefore used throughout this section.

[1] Without stopwords, BBB with 64 LSTM units and a 300-dimensional word embedding achieved 81.6% accuracy, 88.7% precision, and 72.4% recall in sentiment analysis. In the multiclass classification problem, it achieved 72.1% accuracy, 84.5% precision, and 64.8% recall. All scores are based on g_{end}.

In this sequence-to-sequence approach, sensitive terms are detected by a significant change in the prediction at time step t: $\frac{\Delta \bar{\mathbf{x}}_t}{\Delta t}$. The highest positive and lowest negative scores influence the prediction the most. This method does not capture all sensitive words. Predictions close to zero will not change if another negative term appears. Rather, this method detects terms that initiate a change in the prediction.

Based on the IMDB test dataset, table 7.1 lists the most relevant positive and negative sensitive terms in sentiment analysis. The table is sorted by $\frac{\Delta \bar{\mathbf{x}}_t}{\Delta t}$. Terms like `favorite` and `awful` have a high sentiment score, which was to be expected. The table also includes unexpected terms, such as the positive term `ride` at position 26.

In fig. 6.7, the sensitive term `painful` at the beginning of the review defines the sentiment of upcoming time steps. Removing it in fig. 6.10 increases the uncertainty and the model's ability to predict the sentiment.[2] Therefore, the proportion between positive and negative sensitive terms and their influence $\frac{\Delta \bar{\mathbf{x}}_t}{\Delta t}$ should reflect the sentiment of a review and whether the model can make confident predictions.

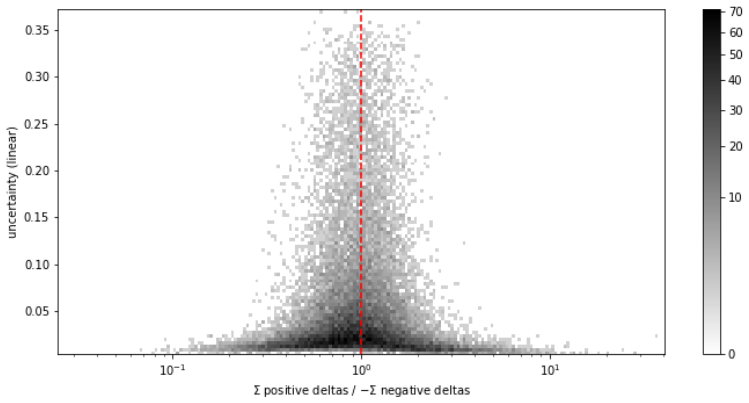


Figure 7.4: Influence of the two most positive and negative sensitive terms on uncertainty (IMDB test dataset, BBB). The value on the x-axis is calculated by dividing the two highest by the absolute value of the two lowest deltas. The y-axis uses g_{linear}.

Fig. 7.4 confirms that sensitive terms influence the model's ability to make confident predictions. When the two most positive and negative sensitive terms negate each other, the uncertainty is high on average. After reaching a certain imbalance, the model is predominantly confident. In fig. 7.4, this imbalance is reached at a ratio of about $2:1$ or $1:2$. The fact that the detection of this relation only requires two positive and negative sensitive terms shows their importance on uncertainty in text classification.

[2] For more examples, refer to appendix A.2.

7.4 Goals

This section evaluates the concept and implementation against the goals defined in section 1.2.

1. The goal to find a transparent method that quantifies the source of uncertainty in text classification was achieved by using Bayesian LSTMs with MC Dropout and BBB. In section 5.4.1, this work introduced the function $U(t, l)$ (eq. 5.10) to quantify epistemic uncertainty. This function is based on the work of Yongchan Kwon et al. [6] and makes use of the standard deviation, derived from multiple forward passes through a Bayesian LSTM. Additional concepts such as weight sequences supported the evaluation of different models and their uncertainty estimations. However, this work did not fully provide a reliable way to determine a suitable uncertainty threshold.
2. A dataset does not require manual labeling. The label of any text classification is assumed to be true for all time steps.
3. The proof of concept application (section 6.7) was implemented as a web application with user experience in mind. The evaluation provides multiple statistics to inspect certain parts of the prediction. It uses the same concepts as section 5. This includes weighted uncertainties and uncertainty estimations with thresholds. Therefore, this goal has been successfully reached.

7.5 Known Complications

There are several known complications with this approach that have to be considered. This section outlines these complications.

7.5.1 Assessment of Uncertainty

The assessment of uncertainty is not trivial, especially because uncertain parts are not marked in the training data. Even though all models provide token-based uncertainty estimations, the question of why parts are uncertain remains. In sentiment analysis, sensitive words like `painful` (fig. 6.4) lowered uncertainty in most cases. Unfortunately,

these types of observations depend on previous time steps since LSTMs are context-sensitive. The problem shifts from determining what parts are uncertain to questioning why those parts are uncertain. One way to counteract this problem is to measure the ratio between confident correct, confident incorrect, and uncertain predictions (section 7.1.1). This essentially quantifies trust in a model.

7.5.2 Softmax and Sigmoid Activation Function

The softmax and sigmoid activation function reduce the range of the neural network's codomain to values between zero and one. This codomain causes low uncertainty on any sample mean $\bar{\mathbf{x}}_t$ that is close to zero or one. As a consequence, the distance to those boundaries often correlates with the uncertainty estimation. In most cases, predictions close to 0.5 have high uncertainty. This is not directly an uncertainty quantification problem but has to be addressed to correctly interpret output sequences.

7.5.3 Training of BBB Models

The suboptimal performance of BBB models in multiclass experiments might have been caused by compatibility issues with Tensorflow Probability. Tensorflow might not have tracked all weights w.r.t. the time step correctly, which led to inaccurate results after a certain amount of epochs. In fig. 6.14b, the model accuracy decreased rapidly after only three epochs but the training and validation loss did not increase. Similar but less drastic behavior can be observed in fig. 6.6d. The effect on BERT models (figs. 6.6e and 6.6f) was negligible.

As mentioned in section 7.2, removing stopwords from the input sequence seemed to have improved model performance for BBB. It could counteract this problem to a certain extent.

7.5.4 Contextual Word Embeddings and BiLSTMs

While BERT achieved the best performance scores (table 6.3), it also has its flaws. Since the word embedding is contextual, each word vector depends on the whole sequence. In other words, identical tokens are represented differently at distinct time steps. This is caused by the multi-layered bidirectional transformers with self-attention mechanisms

in BERT (section 2.2.2) [22]. On a high level, BERT inspects the whole input sequence and then creates word vectors based on their context. Doing so shifts and blurs the uncertainty estimations of individual tokens using the sequence-to-sequence approach.

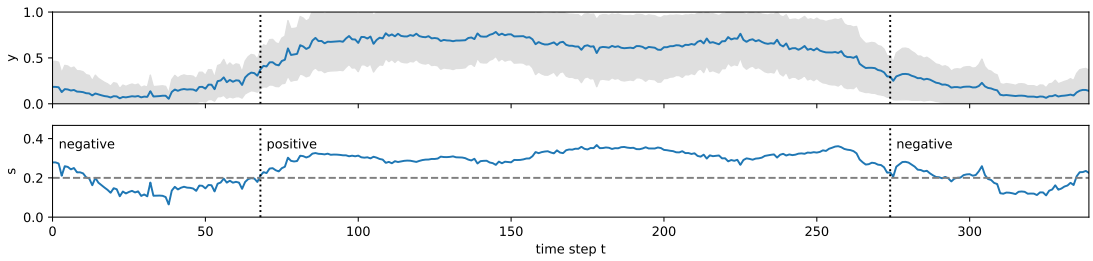


Figure 7.5: Bidirectional LSTM and mixed sentiment. The input sequence is listed in section 6.5.4.

A similar problem arises when using bidirectional LSTMs because they read the input sequence once from beginning to end and from end to beginning. Fig. 7.5 shows the evaluation of the same input sequence as in fig. 6.9 with a bidirectional LSTM. The graph has smooth transitions between the negative and positive reviews. The bidirectional LSTM uses the context before and after each time step. For performance reasons, bidirectional LSTMs are the preferred architecture (section 4) to solve several Natural Language Processing (NLP) problems. They still provide reasonable uncertainty estimations at a greater scale but do not provide the same level of detail for individual tokens. If this restriction is within reason, they are a viable extension to this approach.

8 Conclusion

This chapter summarizes this work and provides an outlook for future research in section 8.2.

8.1 Summary

Most text classification approaches focus on model performance but cannot determine the sources of uncertainty [1, 67]. By introducing transparent uncertainty estimations for individual tokens in the input sequence, the source of uncertainty was disclosed and made traceable.

This work showed a novel sequence-to-sequence approach to performing binary and multiclass text classification with per-token uncertainty estimations. Variational Bayesian LSTMs were implemented with the MC Dropout and BBB techniques. Multiple models were evaluated against various hyperparameters and word embeddings. Despite BBBs suboptimal training in multiclass experiments, all models achieved sufficient performance scores while providing insight into the uncertainties of individual tokens. Contextual word embeddings like BERT alongside BiLSTMs attained higher test accuracy but blurred results.

The impact of stopwords and detection of sensitive terms was studied by carrying out multiple forward passes to get an output sequence with uncertainty estimations. The experiments showed that this approach can detect uncertainties and sensitive terms in input sequences. The concept of weighted uncertainties and uncertainty thresholds simplified evaluating and comparing the models. Although this work did not fully provide a method to choose a meaningful threshold, empirical-based methods sufficed to optimize the ratio between confident incorrect and uncertain predictions.

The additional information in the predictions showed their importance in cases such as mixed sentiment. In these cases, the course of uncertainty changed throughout the

sequence. This approach is particularly helpful whenever a prediction appears as uncertain, and the system or end-user needs to locate the cause. Since the model learns what it does not know, predictions are more trustworthy. This plays an important role when it comes to more sensitive classification tasks like hate speech detection.

8.2 Future Work

Developing an uncertainty oriented text classification approach on a token-basis is a continuous process. Various parts can be improved.

As a first step, one could experiment with other lemmatizers and tokenizers to improve text preprocessing. Furthermore, training a Bayesian Long Short-Term Memory (LSTM) without stopwords may also improve the model performance, as empirically shown in section 7.2.

Gated Recurrent Units (GRUs) are a reasonable alternative to LSTMs [7]. It would be interesting to see the performance differences between these architectures. The implementation of MC Dropout should be straight forward.

There are other methods to quantify uncertainties in Bayesian neural networks, such as introducing two heads in the output layer for each label [33, 34, 68] as opposed to one. With this technique, one of the heads would estimate the variance explicitly.

Posterior sharpening is an extension to Bayes by Backprop (BBB) in Recurrent Neural Networks (RNNs) [8, chapter 4] that reduces variance in the variational posterior $q(\theta)$. It involves more complex sampling techniques at training time but could improve overall model performance. Therefore, it is an adequate candidate for future research concerning the implementation of BBB in Tensorflow 2.

In eq. 3.2, the total variance distinguishes between aleatory and epistemic uncertainty [6]. This work only focuses on epistemic uncertainty, leaving room for experiments with aleatory uncertainty and the relation between both types.

Finally, conducting an empirical study in which the contestants have to mark uncertain sections in unlabeled texts can create a reference point. This reference point can be used to evaluate uncertainty estimations.

Bibliography

[1] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

[2] Siddhartha Brahma. Improved sentence modeling using suffix bidirectional lstm. 2018.

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

[4] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1050–1059. JMLR.org, 2016.

[5] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1613–1622. JMLR.org, 2015.

[6] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Paik. Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. April 2018.

[7] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1027–1035, USA, 2016. Curran Associates Inc.

[8] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *ArXiv*, abs/1704.02798, 2017.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019. https://github.com/google-research/bert.

[10] Jason P.C. Chiu and Eric Nichols. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[11] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *HLT-NAACL*, 2016.

[12] Jonas Paul Winkler and Andreas Vogelsang. What does my classifier learn? a visual approach to understanding natural language text classifiers. In Matthias Tichy, Eric Bodden, Marco Kuhrmann, Stefan Wagner, and Jan-Philipp Steghöfer, editors, *Software Engineering und Software Management 2018*, pages 223–224, Bonn, 2018. Gesellschaft für Informatik.

[13] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

[14] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[15] José Camacho-Collados and Mohammad Taher Pilehvar. On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. In *BlackboxNLP@EMNLP*, 2017.

[16] M. F. Porter. Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[17] Bo Han and Timothy Baldwin. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for*

Computational Linguistics: Human Language Technologies, pages 368–378, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[19] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. Models are available at https://nlp.stanford.edu/projects/glove/.

[21] Marwa Naili, Anja Habacha Chaibi, and Henda Hajjami Ben Ghezala. Comparative study of word embedding methods in topic segmentation. *Procedia Comput. Sci.*, 112(C):340–349, September 2017.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[23] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *CCL*, 2019.

[24] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 09 2019. btz682.

[25] Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.

[26] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *ArXiv*, abs/1705.08209, 2018.

[27] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013.

[28] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[29] C. Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[30] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000.

[31] Justin Bayer. *Learning Sequence Representations*. PhD thesis, Technische Universität München, 2015.

[32] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31:105–112, 03 2009.

[33] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *NIPS*, 2017.

[34] Yijun Xiao and William Yang Wang. Quantifying uncertainties in natural language processing tasks. *ArXiv*, abs/1811.07253, 2019.

[35] G.E.P. Box and G.C. Tiao. *Bayesian inference in statistical analysis*. Addison-Wesley series in behavioral science: quantitative methods. Addison-Wesley Pub. Co., 1973.

[36] Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc., 2011.

[37] Charles W. Fox and Stephen J. Roberts. A tutorial on variational bayesian inference. *Artificial Intelligence Review*, 38(2):85–95, Aug 2012.

[38] Radford M. Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Proceedings of the NATO Advanced Study Institute on Learning in Graphical Models*, pages 355–368, Norwell, MA, USA, 1998. Kluwer Academic Publishers.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[40] Marius Pachitariu and Maneesh Sahani. Regularization and nonlinearities for neural language models: when are they needed? *ArXiv*, abs/1301.5650, 2013.

[41] Justin Bayer, Christian Osendorfer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *CoRR*, abs/1311.0701, 2013.

[42] Janis Postels, Francesco Ferroni, Huseyin Coskun, Nassir Navab, and Federico Tombari. Sampling-free epistemic uncertainty estimation using approximated variance propagation. *ArXiv*, abs/1908.00598, 2019.

[43] Jiri Hron, Alexander G. de G. Matthews, and Zoubin Ghahramani. Variational bayesian dropout: pitfalls and fixes. *ArXiv*, abs/1807.01969, 2018.

[44] Jiri Hron, Alexander G. de G. Matthews, and Zoubin Ghahramani. Variational gaussian dropout is not bayesian. 2017.

[45] Ian Osband. Risk versus uncertainty in deep learning : Bayes , bootstrap and the dangers of dropout. 2016.

[46] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *ArXiv*, abs/1508.01991, 2015.

[47] Filip Ginter, Hanna Suominen, Sampo Pyysalo, and Tapio Salakoski. Combining hidden markov models and latent semantic analysis for topic segmentation and labeling: Method and clinical application. *International Journal of Medical Informatics, Special Issue on Mining of Clinical and Biomedical Text and Data*, 78(12):e1–e6, 2009.

[48] A. Krogh. Hidden markov models for labeled sequences. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 140–144 vol.2, Oct 1994.

[49] Cindy Wang. Interpreting neural network hate speech classifiers. In *ALW*, 2018.

[50] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, 2015. Software available from tensorflow.org.

[51] François Chollet et al. Keras. https://keras.io, 2015.

[52] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. Tensorflow distributions. *ArXiv*, abs/1711.10604, 2017.

[53] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[54] Han Xiao. bert-as-service. https://github.com/hanxiao/bert-as-service, 2018.

[55] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, page 114–119, USA, 1994. Association for Computational Linguistics.

[56] Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. *Comput. Linguist.*, 32(4):485–525, December 2006.

[57] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[58] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[59] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *ArXiv*, abs/1801.06146, 2018.

[60] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *NIPS*, 2017.

[61] Icelandic Meteorological office. Weather forecast. https://en.vedur.is/weather/forecasts/text/.

[62] Amazon. Amazon customer reviews dataset. Registry of Open Data on AWS. https://s3.amazonaws.com/amazon-reviews-pds/readme.html.

[63] Inc. Docker. Docker. https://www.docker.com/.

[64] Igor Sysoev. Nginx. https://www.nginx.com/.

[65] Armin Ronacher. Flask. https://www.palletsprojects.com/p/flask/.

[66] Facebook Inc. React. https://reactjs.org/.

[67] Siyuan Chen, Chao Peng, Linsen Cai, and Lanying Guo. A deep neural network model for target-based sentiment analysis. *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2018.

[68] D. A. Nix and A. S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60 vol.1, June 1994.

A Appendix

A.1 Proof of Concept Response Schema

```
interface Token {
  token: string
  mean: number[]
  std: number[]
  samples: number[][]
}
interface GMetric {
  end: number
  full: number
  linear: number
}
interface Prediction {
  icon: string
  label: number
  name: string
  uncertainty: GMetric
  y: GMetric
  positive_sensitive_words: number[]
  negative_sensitive_words: number[]
}
interface ApiResponse {
  label: number
  prediction: Prediction[]
  tokens: Token[]
  uncertainty_threshold: number
}
```

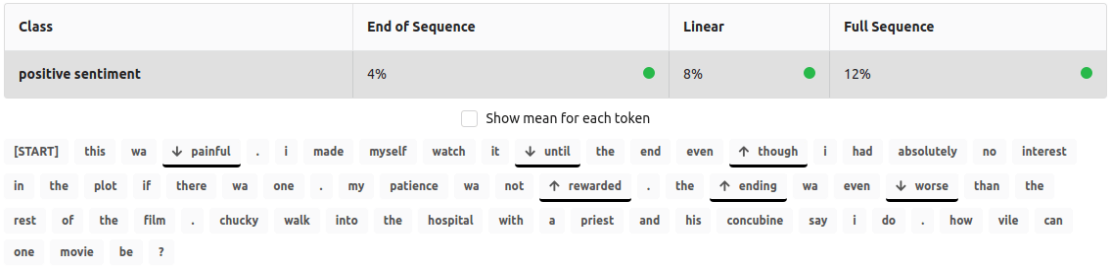
Listing A.1: Proof of concept response schema.

A.2 Text Evaluations

Note: The following text samples include token uncertainties using the proof of concept application. The sentiment analysis evaluation uses the Bayes by Backprop (BBB) model with a 300-dimensional Keras word embedding and 64 LSTM units over 50 forward passes. The proof of concept application included in the source code uses a BBB model with a 100-dimensional Keras word embedding for binary and multiclass classification.

This was painful. I made myself watch it until the end, even though I had absolutely no interest in the plot, if there was one. My patience was not rewarded. The ending was even worse than the rest of the film. Chucky walks into the hospital with a priest and his concubine says "I do". How vile can one movie be?

(a) Original text source

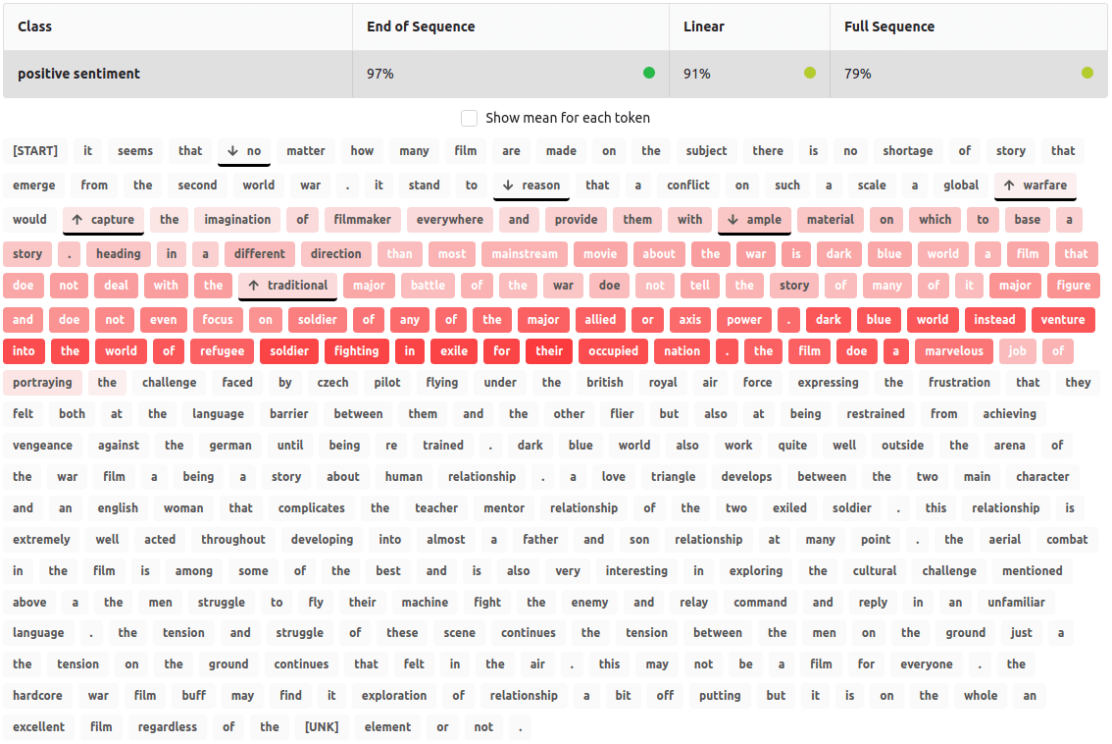


(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens.

Figure A.1: Negative IMDB review (BBB, 50 forward passes).

It seems that no matter how many films are made on the subject, there is no shortage of stories that emerge from the Second World War. It stands to reason that a conflict on such a scale as global warfare would capture the imagination of filmmakers everywhere and provide them with ample material on which to base a story. Heading in a different direction than most mainstream movies about the war is Dark Blue World, a film that does not deal with the traditional major battles of the war, does not tell the story of many of its major figures, and does not even focus on soldiers of any of the major allied or axis powers. Dark Blue World instead ventures into the world of refugee soldiers fighting in exile for their occupied nations. The film does a marvelous job of portraying the challenges faced by Czech pilots flying under the British Royal Air Force, expressing the frustration that they felt both at the language barrier between them and the other fliers, but also at being restrained from achieving vengeance against the Germans until being re-trained. Dark Blue World also works quite well outside the arena of the war film as being a story about human relationships. A love triangle develops between the two main characters and an English woman that complicates the teacher-mentor relationship of the two exiled soldiers. This relationship is extremely well acted throughout, developing into almost a father and son relationship at many points. The aerial combat in the film is among some of the best and is also very interesting in exploring the cultural challenges mentioned above as the men struggle to fly their machines, fight the enemy, and relay commands and replies in an unfamiliar language. The tension and struggle of these scenes continues the tension between the men on the ground, just as the tension on the ground continues that felt in the air. This may not be a film for everyone. The hardcore war film buff may find its exploration of relationships a bit off-putting, but it is on the whole an excellent film regardless of the bellicose element or not.

(a) Original text source

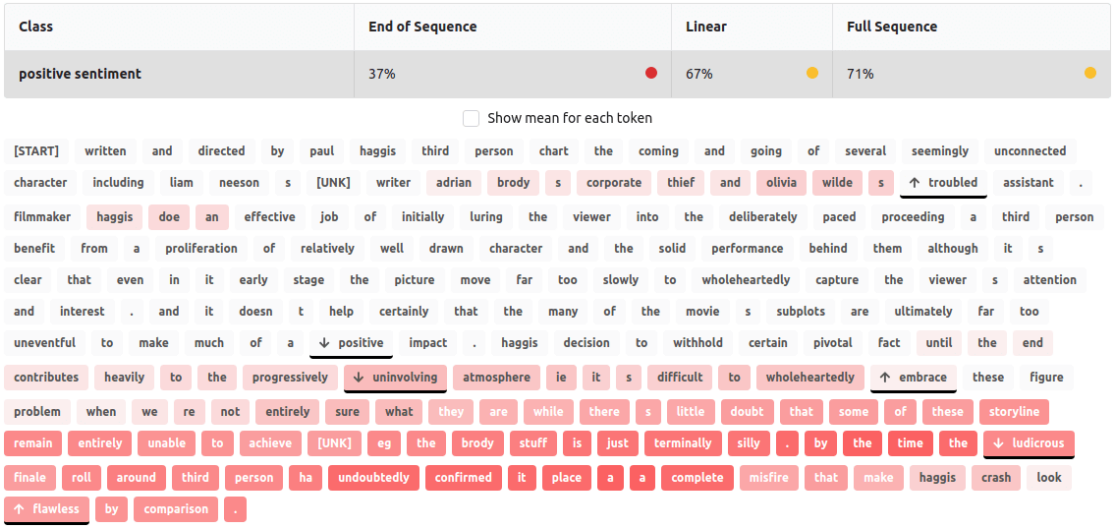


(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens.

Figure A.2: Positive IMDB review (BBB, 50 forward passes).

Written and directed by Paul Haggis, Third Person charts the comings and goings of several seemingly unconnected characters – including Liam Neeson's maladjusted writer, Adrian Brody's corporate thief, and Olivia Wilde's troubled assistant. Filmmaker Haggis does an effective job of initially luring the viewer into the deliberately-paced proceedings, as Third Person benefits from a proliferation of relatively well-drawn characters and the solid performances behind them – although it's clear that, even in its early stages, the picture moves far too slowly to wholeheartedly capture the viewer's attention and interest. (And it doesn't help, certainly, that the many of the movie's subplots are ultimately far too uneventful to make much of a positive impact.) Haggis' decision to withhold certain pivotal facts until the end contributes heavily to the progressively uninvolving atmosphere (ie it's difficult to wholeheartedly embrace these figures' problems when we're not entirely sure what they are), while there's little doubt that some of these storylines remain entirely unable to achieve liftoff (eg the Brody stuff is just terminally silly). By the time the ludicrous finale rolls around, Third Person has undoubtedly confirmed its place as a complete misfire that makes Haggis' Crash look flawless by comparison.

(a) Original text source

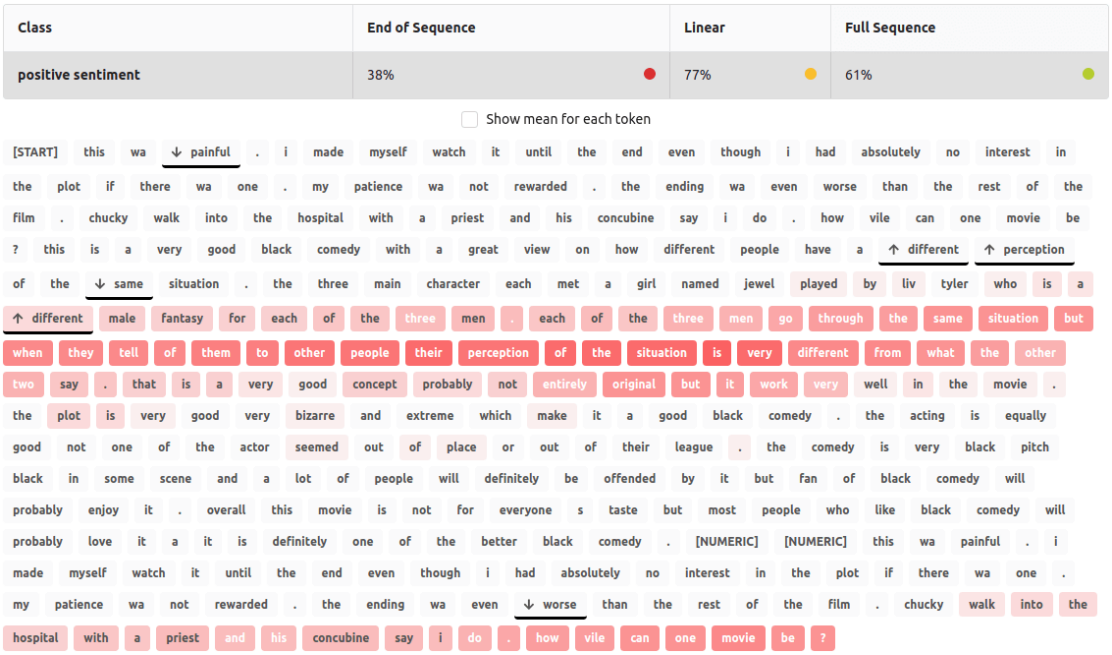


(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens.

Figure A.3: Negative IMDB review (BBB, 50 forward passes).

*(**Negative**) This was painful. I made myself watch it until the end, even though I had absolutely no interest in the plot, if there was one. My patience was not rewarded. The ending was even worse than the rest of the film. Chucky walks into the hospital with a priest and his concubine says "I do". How vile can one movie be?*
*(**Positive**) This is a very good black comedy, with a great view on how different people have a different perception of the same situations. The three main characters each met a girl named Jewel, played by Liv Tyler, who is a different male fantasy for each of the three men. Each of the three men go through the same situations, but when they tell of them to other people, their perception of the situation is very different from what the other two say. That is a very good concept, probably not entirely original but it works very well in the movie. The plot is very good, very bizarre and extreme, which makes it a good black comedy. The acting is equally good, not one of the actors seemed out of place or out of their league. The comedy is very black, pitch black in some scenes, and a lot of people will definitely be offended by it, but fans of black comedy will probably enjoy it. Overall, this movie is not for everyone's taste, but most people who like black comedy will probably love it, as it is definitely one of the better black comedies. 7/10*
*(**Negative**) This was painful. I made myself watch it until the end, even though I had absolutely no interest in the plot, if there was one. My patience was not rewarded. The ending was even worse than the rest of the film. Chucky walks into the hospital with a priest and his concubine says "I do". How vile can one movie be?*

(a) Original text source



(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens.

Figure A.4: Mixed IMDB review with unknown sentiment (BBB, 50 forward passes).

On Wednesday: Northeast 8-15 m/s and snowshowers in the north and east, but fair weather in the south and west. Frost 1 to 8 deg. C.

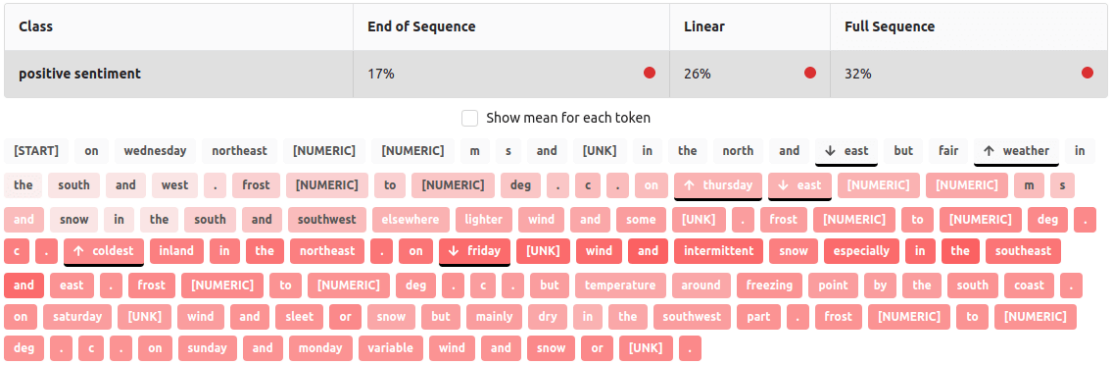
On Thursday: East 10-15 m/s and snow in the south and southwest, elsewhere lighter wind and some snowshowers. Frost 2 to 12 deg. C., coldest inland in the northeast.

On Friday: Easterly wind and intermittent snow, especially in the southeast and east. Frost 1 to 8 deg. C., but temperature around freezing point by the south coast.

On Saturday: Notheasterly wind and sleet or snow, but mainly dry in the southwest part. Frost 0 to 8 deg. C.

On Sunday and Monday: Variable wind and snow or snowshowers.

(a) Original text source

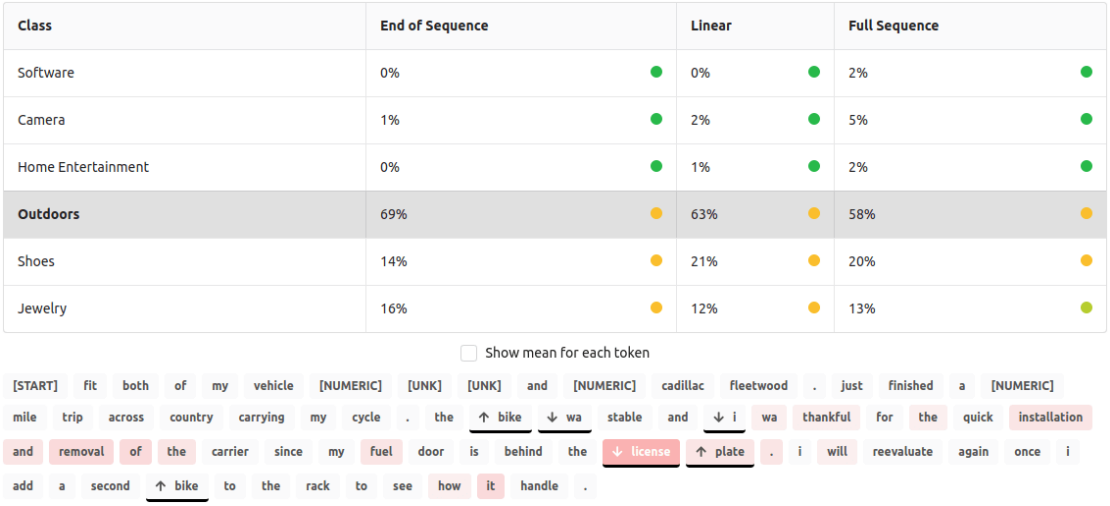


(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens.

Figure A.5: Out-of-distribution example [61] with unknown sentiment (BBB, 50 forward passes).

Fits both of my vehicles (2008 Chrysler Pacifica and 1996 Cadillac Fleetwood). Just finished a 1100 mile trip across country carrying my cycle. The bike was stable and I was thankful for the quick installation and removal of the carrier since my fuel door is behind the license plate. I will reevaluate again once I add a second bike to the rack to see how it handles.

(a) Original text source



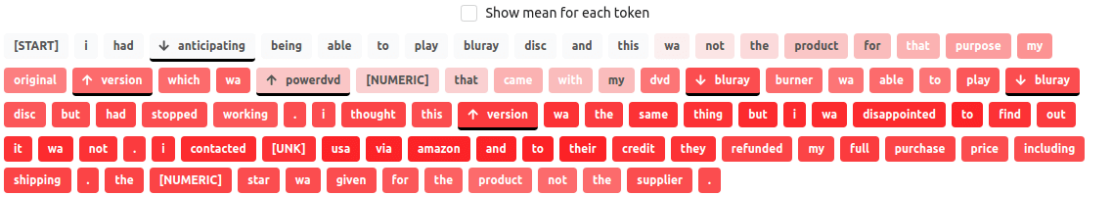
(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens for label *Outdoors*.

Figure A.6: Outdoors review (BBB, 50 forward passes).

I had anticipating being able to play bluray discs, and this was not the product for that purpose, My original version (which was PowerDVD 10 that came with my DVD/Bluray Burner) was able to play bluray discs but had stopped working. I thought this version was the same thing but I was disappointed to find out it was not. I contacted DiscountGiant USA via Amazon and to their credit they refunded my full purchase price including shipping. The 3 stars was given for the product, not the supplier.

(a) Original text source

Class	End of Sequence	Linear	Full Sequence
Software	69%	65%	56%
Camera	4%	2%	4%
Home Entertainment	25%	32%	35%
Outdoors	1%	0%	2%
Shoes	1%	0%	1%
Jewelry	0%	0%	1%



(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens for label *Home Entertainment*.

Figure A.7: Home Entertainment review (BBB, 50 forward passes).

*(**Shoes**) Finally, shoes that fit my 18 month old's feet! We have been to stride rite multiple times and they do not have shoes to fit his high instep. He has extra wide and very chunky feet! These are easy to put on and very cute. We also ordered a pair from the pipsqueaker website without a squeaker. The shoes run a little big. We are very happy with this product!*
*(**Software**) I have thought well of every operating system since DOS 3.1. This is without a doubt the worst system I have ever seen. Not only is it NOT graphicaly intuitive, it is incompatible with most all of my 32 bit software. Trying to use Quicken will crash the computer. It is going to cost a bundle to upgrade all of a person's softwarein order to use Vista. The only thing you will get out of running Vista is a lot of headaches.*

(a) Original text source

Class	End of Sequence	Linear	Full Sequence
Software	98%	61%	39%
Camera	0%	1%	1%
Home Entertainment	1%	1%	1%
Outdoors	0%	10%	14%
Shoes	0%	26%	44%
Jewelry	0%	1%	1%

Show mean for each token

[START] finally ↓ shoe that fit my [NUMERIC] month old s foot ! we have been to stride rite multiple time and
they do not have shoe to fit his high instep . he ha extra wide and very chunky foot ! these are easy
to put on and very cute . we also ordered a pair from the [UNK] website without a [UNK] . the shoe run
a little big . we are very happy with this product ! i have thought well of every operating system since do
[NUMERIC] . [NUMERIC] . this is without a ↓ doubt the worst system ↓ i have ever seen . not only is it not
[UNK] ↑ intuitive it is ↑ incompatible with most all of my [NUMERIC] bit ↑ software . trying to use quicken will
crash the computer . it is going to cost a bundle to upgrade all of a person s [UNK] order to use vista
. the only thing you will get out of running vista is a lot of headache .

(b) Weighted uncertainty with g_{end}, g_{linear} and g_{full} and uncertainty over tokens for label *Software*.

Figure A.8: Mixed review; Shoes and Software (BBB, 50 forward passes).

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Erkennung von Unsicherheit in Textklassifizierungen: Ein Sequence to Sequence Ansatz mit Bayesian RNNs

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original