

UNIVERSITEIT VAN AMSTERDAM

MSC ARTIFICIAL INTELLIGENCE
MASTER THESIS

**Recoding latent sentence
representations**

Dynamic gradient-based activation modification in RNNs

by

DENNIS ULMER

11733187

August 9, 2019

36 ECTS

January 2019 - August 2019

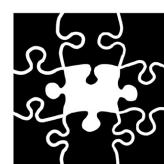
Supervisors:

DIEUWKE HUPKES

Dr. ELIA BRUNI

Assessor:

Dr. WILLEM ZUIDEMA



INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION
UNIVERSITY OF AMSTERDAM

Abstract

In Recurrent Neural Networks (RNNs), encoding information in a suboptimal or erroneous way can impact the quality of representations based on later elements in the sequence and subsequently lead to wrong predictions and a worse model performance. In humans, challenging cases like garden path sentences (an instance of this being the infamous *The horse raced past the barn fell*) can lead their language understanding astray. However, they are still able to correct their representation accordingly and recover when new information is encountered. Inspired by this, I propose an augmentation to standard RNNs in form of a gradient-based correction mechanism: This way I hope to enable such models to dynamically adapt their inner representation of a sentence, adding a way to correct deviations as soon as they occur. This could therefore lead to more robust models using more flexible representations, even during inference time.

The mechanism explored in this work is inspired by work of Julianelli et al., 2018, who demonstrated that activations in a LSTM can be corrected in order to recover corrupted information (*interventions*). This is achieved by the usage of “Diagnostic Classifiers” (DG) (Hupkes et al., 2018), linear models that can help to identify the kind of information recurrent neural networks encode in their hidden representations and correct them using their gradients. In this thesis, I present a generalized framework based on this idea to dynamically adapt hidden activations based on local error signals (*recoding*) and implement it in form of a novel mechanism. I explore signals which are either based directly on the task’s objective or on the model’s confidence, leveraging recent techniques from Bayesian Deep Learning (Gal and Ghahramani, 2016b; Gal and Ghahramani, 2016a; Pearce et al., 2018).

Crucially, these updates take place on an *activation* rather than a parameter level and are performed during training and testing alike. I conduct different experiments in the context of language modeling, where the impact of using such a mechanism is examined in detail.

To this end, I look at modifications based on different kinds of time-dependent error signals and how they influence the model performance. Furthermore, this work contains a study of the model's confidence in its predictions during training and for challenging test samples and the effect of the manipulation thereof. Lastly, I also study the difference in behavior of these novel models compared to a standard LSTM baseline and investigate error cases in detail to identify points of future research. I show that while the proposed approach comes with promising theoretical guarantees and an appealing intuition, it is only able to produce minor improvements over the baseline due to challenges in its practical application and the efficacy of the tested model variants.

Acknowledgements

春风化雨 - Spring wind and rain

Chinese idiom about the merits of higher education

Amsterdam, August 9, 2019

Nobody in this world succeeds alone. Giraffe calves are able to walk around on their own within an hour after their birth, while the whole process takes human babies around a year. And even then they require intense parental care throughout infancy, and, in modern society, often depend on others way into their adulthood.

In the end, the essence of our own personality is distilled from the sum of all of our experiences, which includes all the interactions we have with others. This forms a giant, chaotic and beautiful network of beings, constantly in flux. I therefore owe myself to the ones around me, a selection of which I want to thank at this point.

First and foremost, I want to thank my supervisors Dieuwke and Elia. Not only have they fulfilled their duties as such with excellence by providing guidance and constant feedback, they were also always available when I had questions or expressed doubts. They furthermore gave me the opportunity to participate in research projects and encouraged me to attend conferences and publish findings, essentially supporting my first steps into academia. I also want to thank Willem Zuidema for taking his time in assessing my thesis and taking part in my defense.

Secondly, I would like express my eternal gratitude to my family for all the ways that they have supported my throughout my entire life in a myriad of ways. My grandma Ingrid, my father Siegfried and my sister Anna; thank you for believing in me and supporting me and providing me with the opportunity to develop follow my interests. I want to especially thank my mother Michaela for constantly trying to enkindle my curiosity when growing up, always nudging me into the right direction

during my teenage years and offering guidance to this very day.

Next, I would like to thank all my friends who contributed to this work and the past two years of my life in a manifold of ways: Thank you Anna, Christina, Raj and Santhosh for all those beautiful memories that I will be fondly remembering for the rest of my life as soon as my mind reminisces about my time here in Amsterdam. Furthermore, thank you Caro for holding me accountable and being a pain in the neck of my inertia, and living through the highs and lows of the last two years. Thank you Putri for getting me addicted to coffee, something I had resisted to my whole life up until this point and for providing company in a constantly emptying Science Park. Also, thank you Mirthe for applying mathematical rigour to my derivations and to Atilla for being a super-human proof-reader.

I could extend this list forever but I have to leave some pages for my actual thesis, therefore let me state the following: Although you might have stayed unmentioned throughout this text, your contribution has not stayed unnoticed. And for the case that I have not made my appreciation to you explicit: Thank you. I am grateful beyond measure to be surrounded by some many great people that have supported me in times of doubt and that I know that I can always turn to and the many more that have enriched my life in smaller ways.

List of Abbreviations

BAE Bayesian Anchored Ensembling. 18, 45, 51–54, 56–58, 62, 63, 65, 88

BPTT Backpropagation Through Time. 29

DC Diagnostic Classifier. 22, 38

DNN Deep Neural Network. 17, 25

ELBO Evidence Lower Bound. 32, 72, 73

HMM Hidden Markov Model. 17

LSTM Long-Short Term Memory. 21, 22, 29, 30, 37–39, 44, 48, 49, 56, 61, 69

MAP Maximum a posteriori. 12, 23, 32, 36, 37, 74

MCD Monte Carlo Dropout. 12, 33, 51–53, 56–58, 62, 63, 65, 66, 88

MLE Maximum Likelihood Estimate. 31

MLP Multi-Layer Perceptron. 17, 28, 34, 43, 45, 56, 66

MSE Minimum Squared Error (loss). 26

NN Neural Network. 37

PTB Penn Treebank. 48, 51, 53, 54, 61, 88

ReLU Rectified Linear Unit. 26, 56

RNN Recurrent Neural Network. 10, 12, 17–19, 21, 28–30, 34, 35, 39, 40, 44, 45, 50, 66

Contents

1	Introduction	16
1.1	Research Questions	18
1.2	Contributions	19
1.3	Thesis structure	20
2	Related Work	21
2.1	Recurrent Neural Networks & Language	21
2.2	Model Uncertainty	22
2.3	Dynamic Models & Representations	23
3	Background	25
3.1	Deep Learning	25
3.2	Recurrent Neural Language Models	28
3.2.1	Recurrent Neural Networks	28
3.2.2	Language Modeling with RNNs	30
3.3	Bayesian Deep Learning	31
3.3.1	Variational inference	31
3.3.2	Estimating model confidence with Dropout	33
3.3.3	Estimating model confidence with ensembles	35
3.4	Recap	37
4	Recoding Framework	38
4.1	Interventions	38
4.2	Recoding Mechanism	39
4.3	Error Signals	41
4.3.1	Weak Supervision	41
4.3.2	Surprisal	41
4.3.3	MC Dropout	43
4.3.4	Bayesian Anchored Ensembling	45
4.4	Step Size	46
4.5	Recap	46
5	Experiments	48
5.1	Dataset	48
5.2	Training	48

5.3	Evaluation	50
5.3.1	Recoding Step Size	50
5.3.2	Number of samples	53
5.3.3	MC Dropout rate	55
5.3.4	Dynamic Step Size	56
5.3.5	Ablation	57
6	Qualitative Analysis	59
6.1	Recoding Behaviour	59
6.1.1	Cognitive Plausibility	60
6.1.2	Error signal reduction	61
6.1.3	Effect of overconfident predictions	62
7	Discussion	64
7.1	Step Size	64
7.2	Role of Error Signal	65
7.3	Approximations & Simplifying Assumptions	65
8	Conclusion	67
8.1	Future Work	67
8.1.1	Recoding signals	67
8.1.2	Step size	68
8.1.3	Efficiency	68
8.1.4	Training Procedure & Architecture	69
8.2	Outlook	69
8.2.1	Innovation over Scale	69
8.2.2	Inspiration from Human Cognition	70
Bibliography		72
A	Lemmata, Theorems & Derivations	79
A.1	Derivation of the Evidence lower bound	79
A.2	Approximating Predictive Entropy	80
A.3	Lipschitz Continuity	82
A.4	Theoretical guarantees of Recoding	83
A.5	Derivation of the Surprisal recoding gradient	86
A.6	Derivations fo the Predictive Entropy recoding gradient	88
B	Experimental Details & Supplementary Results	90
B.1	Pseudocode	90
B.2	Practical Considerations	91
B.3	Hyperparameter Search	92
B.4	Additional plots	93

List of Tables

5.1	Test perplexities for different recoding step sizes	52
5.2	Test perplexities for different numbers of samples	54
B.1	Training Hyperparameters	92
B.2	Sample ranges for hyperparameter search	93

List of Figures

3.1	Visualized loss surfaces of two deep neural networks	27
3.2	Comparison of a common application of Dropout to RNNs with Gal and Ghahramani, 2016a	34
3.3	Illustration of the randomized MAP sampling procedure	36
4.1	Illustration of the recoding framework	40
4.2	Illustration of different error signals.	42
5.1	Training losses for different recoding step sizes	51
5.2	Training losses for different numbers of samples	53
5.3	Uncertainty estimates for different numbers of samples	54
5.4	Results for using MCD recoding with different Dropout rates.	55
5.5	Overview over results with dynamic step size models.	57
5.6	Results for ablation experiments.	58
6.1	Model surprisal scores on two garden path sentences	60
6.2	Error reduction through recoding on garden path sentences	62
6.3	The effect of uncertainty recoding on surprisal	63
B.1	Pseucode for Recoding with Surprisal	90
B.2	Pseudocode for Recoding with MC Dropout	91
B.3	Pseudocode for Recoding with Bayesian Anchored Ensembles	94
B.4	Training loss for different dropout rates when using MCD recoding	95
B.5	Additional plots for dynamic step size models.	95

B.6 Additional plots for error reduction through recoding on garden path sentences	96
--	----

Note on Notation

In this short section I introduce some conventions about the mathematical notations in this thesis. This effort aims to make the use of letters and symbols more consistent and thereby avoid any confusion.

Statistics A random variable $X : \Omega \mapsto E$ is a function that maps a set of possible outcomes Ω to measurable space E . In this work, the measurable space E always corresponds to the set of real numbers \mathbb{R} . Parallel to the conventions in the Machine Learning research community, random variables are denoted using an uppercase letter, while realizations of the random variable from E are denoted with lowercase letters $X = x, x \in E$. Furthermore, the probability of this realization is denoted using a lowercase p like in $p(x)$, disregarding whether the underlying distribution is a probability mass function (discrete case) or probability density function (continuous case). In cases where a variable has multiple distributions (e.g. in the context of variational inference), another lowercase letter like $q(x)$ is used. The same holds when using conditional probabilities like $p(y|x)$. In case the lowercase realization is bolded, the distribution is multivariate, e.g. $p(\mathbf{x})$. $\mathbb{E}[\cdot]$, $\text{Var}[\cdot]$ denote the expected value and variance of a random variable, while $\mathbb{H}[\cdot]$ denotes the entropy of its corresponding distribution. If no subscript is used for the expected value, we are evaluating the expected value with respect to the same probability distribution; in other cases this is made explicit like in $\mathbb{E}_{q(x)}[p(x)] = \int q(x)p(x)dx$.

Linear Algebra Matrices are being denoted by using bold uppercase letters, e.g. \mathbf{A} or Γ , including the identity matrix, \mathbf{I} . Vectors however are written using lowercase bold letter like \mathbf{b} . Scalars are simply written as lowercase letters like y . Vector norms are written using $\|\dots\|$, using the subscript (∞) for the l_∞ norm. Without subscripts, this notation corresponds to the euclidean norm or inner product $\|\mathbf{x}\| = (\mathbf{x}^T \mathbf{x})^{1/2} = \langle \mathbf{x}, \mathbf{x} \rangle$.

Parameters Throughout this work, sets of (learnable) model parameters are denoted using bolded greek lowercase letters, where $\boldsymbol{\theta}$ denotes the set of all model parameters

and ω the set of all model weight matrices (without biases). In a slight abuse of notation, these two symbols are also used in contexts where they symbolize a *vector* of (all) parameters. In case a function is parameterized by a set of parameters, this is indicated by adding them as a subscript, e.g. $f_\theta(\cdot)$.

Indices, Sets & Sequences When using indexed elements, the lowercase letter is used for some element of an indexed sequence while the uppercase letter denotes the highest index, like $1, \dots, k, \dots, K$. The only exception to this is the index i , where N is used to write the total number of elements indexed by i . The index t is exclusively used to denote time. Sets of elements are written using a shorthand $\{x\}_1^K = \{x_1, \dots, x_k, \dots, x_K\}$. In case the elements are ordered, a sequence is written similarly using $\langle x \rangle_1^K = \langle x_1, \dots, x_k, \dots, x_K \rangle$. Both sets and sequences are written using caligraphic letters like \mathcal{D} or \mathcal{V} . In case of sets or sequences are used for probabilities or function arguments, the brackets are omitted completely for readability, as in $p(x_1^T) = p(x_1, \dots, x_t, \dots, x_T)$

Superscripts \cdot^* is used to denote the best realization of some variable or parameter. \cdot' is used to signifies some new, modified or updated version. \cdot^\dagger denotes an estimate, while \cdot^\sim is used for approximations or preliminary values. And index in parenthesis like $\cdot^{(k)}$ indicates the membership to some group or set.

Functions Here I simply list a short description of the most commonly used functions in this work:

- $\exp(\cdot)$: Exponential function with base e
- $\log(\cdot), \log_2(\cdot)$: Natural and base 2 logarithm
- $\mathcal{N}(\cdot | \cdot, \cdot)$: Uni- or multivariate normal distribution
- $\text{Bernoulli}(\cdot)$: Bernoulli distribution
- $\mathbb{1}(\cdot)$: Indicator function
- $\text{KL}[\cdot || \cdot]$: Kullback-Leibler divergence
- $\mathcal{L}(\cdot)$: Some loss function used for neural network training
- $\text{ELBO}[\cdot]$: Evidence lower bound of a probability distribution

Introduction

"Language is perhaps our greatest accomplishment as a species. Once a people have established a language, they have a series of agreements on how to label, characterize, and categorize the world around them [...]. These agreements then serve as the foundation for all other agreements in society. Rousseau's social contract is not the first contractual foundation of human society [...]. Language is."

Don't sleep, there are snakes - David Everett

Language forms the basis of most human communication. It enables abstract reasoning, the expression of thought and the transfer of knowledge from one individual to another. It also seems to be a constant across all human languages that their expressivity is infinite, i.e. that there is an infinite amount of possible sentences in every human language (Hauser et al., 2002).¹ Furthermore, natural language is often also ambiguous and messy (Sapir, 1921) and relies on the recipient to derive the right meaning from context (Chomsky, 1956; Grice et al., 1975), a fact that distinguishes it from unambiguous artificial or context-free languages like predicate logic or some programming languages (Ginsburg, 1966). It might therefore seem surprising that humans - given the constraints in time and memory they face when processing language - are able to do successfully. How they do so is still far from being fully understood, which is why it might be helpful to study instances which challenge human language processing.

As an example, take the following sentence: "The old man the boat" is an instance of a phenomenon called *garden-path sentences*.² These sentences are syntactically challenging, because their beginning suggests a parse tree that turns out to be wrong once they finish reading the sentence. In the end, it forces them to reconstruct the sentence's parse. In this particular instance, the sentence "The old man the boat" seems to lack its main verb, until it is realized by the reader that "The old man" is *not* a noun phrase consisting of *determiner - adjective - noun* but actually a verb phrase based on "(to) man" as the main verb and "The old" as the sentence's subject.

¹Although this also remains a controversial notion considering e.g. the Pirahã language, which seems to lack recursion (Everett et al., 2004). This is further discussed in Evans and Levinson, 2009, but not the intention of this work.

²The name is derived from the idiom "(to) be led down the garden path", as in being mislead or deceived.

Humans are prone to these ambiguities (Christiansen and Chater, 2016; Tessler et al., 2019), but display two interesting abilities when handling them:

1. They process inputs based on some prior beliefs about the frequency and prevalence of a sentence's syntactic patterns and semantics.
2. They are able to adapt and correct their inner representation based on newly encountered, incrementally processed information.

Insights like these acquired in linguistics can enable progress in the field of *computational linguistics*, which tries to solve language-based problems with computers. Since the surge of electronic computing in the 1940s, applying these powerful machines to difficult problems like automatic translation has been an active area of research (Abney, 2007). While early works in natural language processing and other subfields of artificial intelligence have focused on developing intricate symbolic approaches (for an overview see e.g. Buchanan, 2005; Jurafsky, 2000), the need for statistical solutions was realized over time. Early trailblazers of this trend can be found in the work of Church, 1989 and DeRose, 1988, who apply so-called Hidden Markov Models (HMMs) to the problem of Part-of-Speech tagging, i.e. assigning categories to words in a sentence. Nowadays, the field remains deeply intertwined with a new generation of statistical models: Artificial Neural Networks, also called Deep Neural Networks (DNNs) or Multi-Layer Perceptrons (MLPs). These connectionist models, trying to (very loosely) imitate the organization of neurons in the human brain, are by no means a recent idea. With the first artificial neuron being trained as early as the 1950s (Rosenblatt, 1958), effective algorithms to train several layers of artificial cells connected in parallel were developed by Rumelhart et al., 1988. However, after some fundamental criticisms (Minsky and Papert, 1969), this line of research fell mostly dormant. Although another wave of related research gained attraction in the 1980s and 1990s, it failed to meet expectations; only due to the effort of some few researches combined with improved hardware and the availability of bigger data sets, Deep Learning was able to produce several landmark improvements (Bengio et al., 2015), which sent a ripple through adjacent fields, including computational linguistics. Indicative for this is e.g. the work of Bengio et al., 2003, developing a neural network-based Language Model, which was able to outperform seasoned n -gram approaches by a big margin and also introduced the concept of *word embeddings*, i.e. the representation of words as learned, real-valued vectors.

In recent years, Recurrent Neural Networks (RNNs) have developed into a popular choice in Deep Learning to model sequential data, especially language. Although fully attention-based models without recurrency (Vaswani et al., 2017; Radford et al., 2018; Radford et al., 2019) have lately become a considerable alternative,

RNN-based models still obtain state-of-the-art scores on tasks like Language Modeling (Gong et al., 2018), Sentiment Analysis (Howard and Ruder, 2018) and more. However, they still lack the two aforementioned capabilities present in humans when it comes to processing language: They do not take the uncertainty about their predictions and representations into account and they are not able to correct themselves based on new inputs.

The aim of this thesis is to augment a RNN architecture to address both of these points: Firstly by developing a mechanism that corrects the model's internal representation of the input at every time step, similar to a human's correction when reading a garden path sentence. Secondly, this correction will be based on several methods, some of which like *Monte Carlo Dropout* (MC Dropout) or *Bayesian Anchored Ensembling* (BAE) were developed in a line of research called *Bayesian Deep Learning*, which is concerned with incorporating bayesian statistics into Deep Learning models. These methods help to quantify the degree of uncertainty in a model when it is making a prediction and can be used in combination with the correction mechanism to directly influence the amount of confidence the model has about its internal representations.

1.1 Research Questions

Building on the ideas of uncertainty and self-correction, this thesis is concerned with realizing them in a mathematical framework and followingly adapt the implementation of a RNN language model with a mechanism that performs the layed-out adaptations of hidden representations. From this intention, the subsequent research questions follow:

How to realize corrections? Gradient-based methods are a backbone of the success of Deep Learning since the inception of the first artificial neurons in the 1950s (Bengio et al., 2015) and its variants in their manifoldness still remain an important tool in practice today.³ However, these methods are usually only applied to the model *parameters* after processing a batch of data. Frequent local adaptions of activations have - to the best of my knowledge - not been attempted or examined in the literature, with the single exception of the paper this work draws inspiration from (Giulianelli et al., 2018).

On what basis should corrections be performed? Another research question that follows as a logical consequence of the previous one focuses on the issue of the mechanism's modality: The described idea requires to take the gradient of some

³E.g. refer to Ruder, 2016 for an overview over the most frequently used gradient-based optimizers.

“useful” quantity with respect to the network’s latent representations. This quantity can e.g. spring from additional data as in Julianelli et al., 2018, which results in a partly supervised approach. This supervision requires additional information about the task, which is not always available or suffers from all the problems that come with manually labeling data on scale: The need for expertise (which might not even produce handcrafted features that are helpful for models; deep neural networks often find solutions that seem unexpected or even surprising to humans) as well as a time- and resource-consuming process. This work thus examines different unsupervised approaches based on a word’s perplexity and uncertainty measures from the works of Gal and Ghahramani, 2016b; Gal and Ghahramani, 2016a; Pearce et al., 2018.

What is the impact on training and inference? In the work of Julianelli et al., 2018, the idea has only been applied to a fully-trained model during test time. The question still remains how using such an extended model would behave during training as adapting activations in such a way will have a so far unstudied effect on the learning dynamics of a model. Furthermore, it is the aim of this thesis to also shed some light on the way that this procedure influences the model’s behavior during inference compared to a baseline, which is going to be evaluated on a batch and word-by-word scale.

1.2 Contributions

In this thesis I make the following contributions:

- I formulate a novel and flexible framework for local gradient-based adaptions called *recoding*, including the proof of some theoretical guarantees.
- I implement this new framework in the context of a recurrent neural language model.⁴
- I examine potential sources of unsupervised “error signals” that trigger corrections of the model’s internal representations.
- Finally, I provide an in-depth analysis of the effect of recoding on the model during training and inference as well as a validation of theoretical results and an error analysis.

⁴The code is available online only under <https://github.com/Kaleidophon/tenacious-toucan>.

1.3 Thesis structure

The thesis is structured as follows: After this introduction, other relevant works about the linguistic abilities of RNNs, Bayesian Deep Learning and more dynamic deep networks are being summarized in chapter 2. Afterwards, I provide some relevant background knowledge for the approaches outlined in this thesis in chapter 3, including a brief introduction to Deep Learning, Language Modeling with Recurrent Neural Networks and Bayesian Deep Learning. A formalization of the core idea described above is given in chapter 4, where I develop several variants of its implementations based on different error signals. These are subsequently evaluated in an experimental setting in chapter 5, where the impact of the choice of hyperparameters concerning recoding and importance of different model components is studied. A closer look at the behavior of the models on a word level is taken in chapter 6. A full discussion of all experimental results is then given in chapter 7. Lastly, all findings and implications of this work are summarized in chapter 8, giving an outlook onto future research. Additional supplementary material concerning relevant lemmata, extensive derivations and additional figures as well as more details about the experimental setup are given in appendices A and B.

Related Work

“The world (which some call the Library) is made up of an unknown, or perhaps unlimited, number of hexagonal galleries, each with a vast central ventilation shaft surrounded by a low railing. From any given hexagon, the higher and lower galleries can be seen stretching away interminably.”

The Library of Babel - Jorge Luis Borges

In this chapter I review some relevant works for this thesis regarding the linguistic abilities of recurrent neural network models as well as the state-of-the-art approaches for Language Modeling. A small discussion is dedicated to the role of some techniques for “interpretable” artificial intelligence and the influences from the area of Bayesian Deep Learning that are the foundation for the framework presented in chapter 4. Lastly, I give an overview over a diverse set of ideas concerning more dynamic models and representations.

2.1 Recurrent Neural Networks & Language

Recurrent Neural Networks (Rumelhart et al., 1988) have long been established as a go-to architecture when modelling sequential data like handwriting (Fernandez et al., 2009) or speech (Sak et al., 2014). One particularly popular variant of RNNs is the *Long-Short Term Memory Network* (LSTM) (Hochreiter and Schmidhuber, 1997), where the network learns to “forget” previous information and add new to a memory cell, which is used in several gates to create the new hidden representation of the input and functions as a sort of long-term memory. This has some special implications when applying recurrent networks to language-related tasks: It has been shown that these networks are able to learn some degree of hierarchical structure and exploit linguistic attributes (Liu et al., 2018), e.g. in the form of numerosity (Linzen et al., 2016; Gulordava et al., 2018), nested recursion (Bernardy, 2018), negative polarity items (Jumelet and Hupkes, 2018), or by learning entire artificial context-free languages (Gers and Schmidhuber, 2001).

The most basic of language-based tasks lies in modeling the conditional probabilities of words in a language, called *Language Modeling*. Current state-of-the-art mod-

els follow two architectural paradigms: They either use a LSTM architecture, for instance in the case of Gong et al., 2018 or use another popular attention-centric approach which recently was able to produce impressive results. Attention, a mechanism to complement a model in a way that is able to focus on specific inputs was first introduced in the context of a recurrent encoder-decoder architecture (Bahdanau et al., 2015). In the work of Vaswani et al., 2017, this was extended to a fully (self-)attention-based architecture, which comprises no recurrent connections whatsoever. Instead, the model uses several attention mechanisms between every layer and makes additional smaller adjustments to the architecture to accommodate this structural shift. This new model type was further extended in Dai et al., 2019 and most recently in Radford et al., 2019.

However, due to the sheer amount of parameters and general complexity of models, the way they arrive at their predictions is often not clear. In order to allow the deployment of models in real-life scenarios with possible implications for the security and well-being of others, it is thus of paramount importance to understand the decision process and learned input representations of models (Brundage et al., 2018). One such tool to provide these insights into the inner workings of a neural network can be linear classifiers which are trained on the network's activations to predict some feature of interest (Hupkes et al., 2018; Dalvi et al., 2018; Conneau et al., 2018), called Diagnostic Classifiers (DC). Giulianelli et al., 2018 use this exact technique to extend the work of Gulordava et al., 2018 and show when information about the number of the sentence's subject in LSTM activations is being corrupted. They furthermore adjust the activations of the network based on the error of the linear classifier and are thus able to avoid the loss of information in some cases.

2.2 Model Uncertainty

Standard Deep Learning approaches do not consider the uncertainty of model predictions. This is especially critical in cases where deployed models are delegated important and possibly life-affecting decisions, such as in self-driving cars or medicine. Furthermore, incorporating the Bayesian perspective into Deep Learning reduces the danger of making overly confident predictions (Blundell et al., 2015) and allows for the consideration of prior beliefs to bias a model toward a desired solution. Therefore, a line of research emerged in recent years trying to harmonize established Deep Learning approaches with ideas from Bayesian inference. However, ideas that try sample weights from learned distributions (Blundell et al., 2015) are often unfeasible as they do not scale to the profound models and humongous datasets utilized in practice.

An influential work managed to exploit a regularization technique called Dropout (Srivastava et al., 2014), which deactivates neurons randomly during training in order to avoid overfitting the training data, to model the approximate variational distribution over the model weights (Gal and Ghahramani, 2016b), which in turn can be used to estimate the predictive uncertainty of a model. These results have also been extended to recurrent models (Gal and Ghahramani, 2016a) and successfully applied in cases like time series prediction in a ride-sharing app (Zhu and Laptev, 2017). Another promising approach to quantify uncertainty is exploiting model ensembles, as shown in Lakshminarayanan et al., 2017, which forgoes the variational approach. Technically, this approach cannot be considered Bayesian and was therefore extended using a technique called *Randomized Anchored MAP Sampling*, where uncertainty estimates can be procuded by employing an ensemble of Deep Neural Networks regularized using the same anchor noise distribution (Pearce et al., 2018).

2.3 Dynamic Models & Representations

Lastly, there have also been some efforts to make models more adaptive to new data. Traditionally, models parameters are adjusted during a designated training phase. Afterwards, when a model is tested and then potentially deployed in a real-world enviromment, these parameters stay fixed. This carries several disadvantages: If the distribution of the data the model is confronted with differs from the training distribution or changes over time, the model performance might drop. Therefore, different lines of research are concerned with finding a solution that either enable continuous learning (like observed in humans or animals) or more flexible representations.

Some works directly attack the problem of differing data distributions and the transfer of knowledge across different domains (Kirkpatrick et al., 2017; Achille et al., 2018). Further ideas about continuous learning have been explored under the “Life-long learning” paradigm e.g. in Reinforcement Learning (Nagabandi et al., 2019) or robotics (Koenig and Matarić, 2017). In Natural Language Processing, one such idea consists of equipping a network with a self-managed external memory, like in the works of Sukhbaatar et al., 2015 and Kaiser et al., 2017. This way allows for a way to adjust hidden representations based on relevant information learned during training. In contrast, the work of Krause et al., 2018 has helped models to achieve state-of-the-art results in Language Modeling by making slight adjustments to the models’ parameters when processing unseen data. It should be noted that these approaches directly target model parameters, while this work is concerned with adjusting model *activations*, which, to the best of my knowledge, has not been attempted in the way that it is being presented here.

The next chapter goes more into the technical details of related works.

Background

"As well as any human beings could, they knew what lay behind the cold, clicking, flashing face – miles and miles of face – of that giant computer. They had at least a vague notion of the general plan of relays and circuits that had long since grown past the point where any single human could possibly have a firm grasp of the whole. Multivac was self-adjusting and self-correcting. It had to be, for nothing human could adjust and correct it quickly enough or even adequately enough."

The Last Question - Isaac Asimov

In this chapter, I supply the reader with some background knowledge relevant to the content of this work, from the basic concepts of Deep Learning in section 3.1 and their extension to language modeling in section 3.2. In section 3.3 I try to convey the intuition of Bayesian Deep Learning, which helps to quantify the uncertainty in a network's prediction. I also outline approaches derived from it used in this thesis as possible signals for activation corrections which approximate this very quantity.

3.1 Deep Learning

Deep Learning has achieved impressive results on several problems in recent years, be it high-quality machine translation (Edunov et al., 2018), lip reading (Van Den Oord et al., 2016), protein-folding (Evans et al., 2018) and beating human opponents in complex games like Go (Silver et al., 2016), just to name a few examples among many. On a theoretical level, Deep Neural Networks (DNN) have been shown to be able to approximate any continuous function under certain conditions (Hornik et al., 1989; Csáji, 2001), which helps to explain their flexibility and success on vastly different domains.

These DNNs consist of an arbitrary number of individual layers, each stacked upon one another. The layers are feeding off increasingly complex *features* - internal, learned representations of the model realized in real-valued vectors - that are the output produced by the preceding layer. A single layer in one of these networks can

be described as an affine transformation of an input \mathbf{x} with a non-linear *activation function* σ applied to it

$$\mathbf{x}^{(l+1)} = \sigma(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)}) \quad (3.1)$$

where $\mathbf{x}^{(l)}$ describes the input of the l -th layer of the network and $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight and bias of the current layer, which are called learned parameters of the model. Using a single weight vector \mathbf{w} , this formulation corresponds to the original Perceptron by Rosenblatt, 1958, a simple mathematical model for a neuron cell which weighs each incoming input with a single factor in order to obtain a prediction. Using a weight *matrix* \mathbf{W} therefore signifies using multiple artificial neurons in parallel. The non-linear activation function $\sigma(\cdot)$ plays a pivotal role: Stacking up linear transformation results in just another linear transformation in the end, but by inserting non-linearities we can achieve the ability to approximate arbitrary continuous functions like mentioned above. Popular choices of activation functions include the sigmoid function, tangens hyperbolicus (\tanh), the rectified linear unit (ReLU) as well as the Softplus function, ReLU's continuous counterpart:

$$\text{sigmoid}(x) = \frac{e^x}{e^x + 1} \quad (3.2)$$

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (3.3)$$

$$\text{relu}(x) = \max(0, x) \quad (3.4)$$

$$\text{softplus}(x) = \log(1 + e^x) \quad (3.5)$$

The network itself is typically improved by first computing and then trying to minimize the value of an objective function or *loss*. This loss is usually defined as the deviation of the model's prediction from the ground truth, where minimizing the loss implies improving the training objective (a good prediction). A popular choice for this loss is the minimum squared error (MSE) loss, where we compute the average squared distance between the models prediction \hat{y}_i for a datapoint \mathbf{x}_i and the actual true prediction y_i using the l_2 -norm:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N \|y_i - \hat{y}_i\|^2 \quad (3.6)$$

However in practice, many other loss functions are used depending on the task, architecture and objective at hand. This loss can then be used by the Backpropagation

algorithm (Rumelhart et al., 1988), which recursively computes the gradient $\nabla_{\theta}\mathcal{L}_{\text{MSE}}$, a vector of all partial derivatives of the loss with respect to every single of the model's parameters θ . It is then used to move the same parameters into a direction which minimizes the loss, commonly using the gradient descent update rule:

$$\theta' = \theta - \eta \nabla_{\theta}\mathcal{L}_{\text{MSE}} \quad (3.7)$$

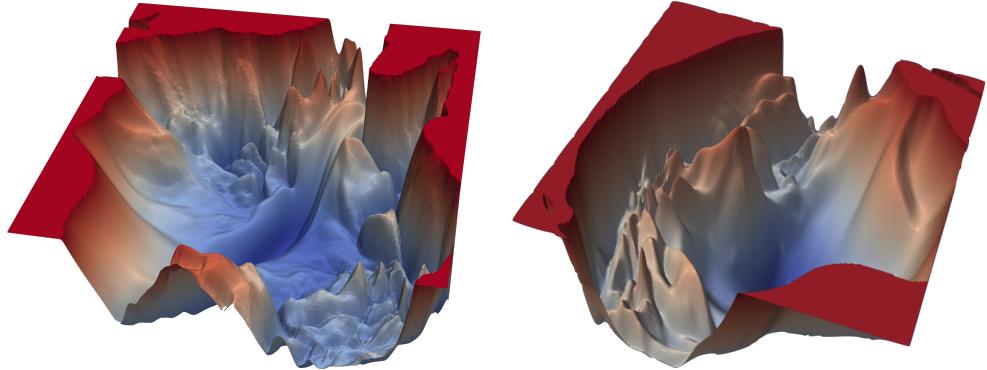


Figure 3.1.: Visualized loss surfaces of two deep neural networks. The “altitude” of the surfaces is determined by the value of the loss function the parameters of the models with the corresponding x - and y -values produce. This figure only depicts the projection of the loss surface into 3D space, as neural networks contain many more than just two parameters. Images taken from Li et al., 2018. Best viewed in color.

where η is a positive learning rate with $\eta \in \mathbb{R}^+$. This effectively means that we evaluate the quality of the network parameters at every step by looking at the prediction they produced. The gradient points into the direction of steepest ascend. Consequently, the antigradient $(-\nabla_{\theta}\mathcal{L}_{\text{MSE}})$ points us into the direction of the steepest *descend*. Updating the parameters into this direction means that we seek to produce a set of new parameters θ' which theoretically decreases the value of the loss function during the next step.

It is possible for convex loss functions to find their global minimum, i.e. the set of parameters that produces a loss that is lower than all other possible losses, in a single step (cp. Bishop, 2006 §3). However, as illustrated in figure 3.1, neural networks are known to have highly non-convex loss functions - highly dimensional, rugged loss surfaces with many local minima and an increasing number of saddle points as the dimensionality of the parameter space (the number of parameters in the model) increases (Dauphin et al., 2014). We therefore only take a step of length η into the direction of the antigradient. Using a suitable learning rate η , this is guaranteed to converge to a local minimum (Nesterov, 2018). Although this local minimum is rarely the global minimum, the found solution often suffices to solve the task at hand in a satisfactory manner.

3.2 Recurrent Neural Language Models

In this section I provide some insight into the motivation for and inner workings of Recurrent Neural Networks (3.2.1), an extension to standard MLPs especially suited for sequential data. Also, I outline their application to Language Modeling (3.2.2), where we try to model the probability distribution that human language is “generated” from and which will be the main objective of this thesis.

3.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a special type of Deep Neural Networks which comprises recurrent connections, meaning that they produce new representations based on their prior representations. Due to this recurrent structure, they present an inductive bias in their architecture which makes them especially suitable for modeling sequences of data (Battaglia et al., 2018).

Formally, a sequence of input symbols like words is mapped to a sequence of continuous input representations $\mathcal{X} = \langle \mathbf{x}_1, \dots, \mathbf{x}_T \rangle$ with $\mathbf{x}_t \in \mathbb{R}^M$ called *embeddings*, real-valued vector representations for every possible symbol that is used as an input to the model. When dealing with language, every word in the vocabulary of a corpus is mapped to a *word embedding*. These representations are initialized randomly and then tuned alongside the model’s parameters during training. Furthermore, a recurrent function $g(\cdot)$ parameterized by θ is used to compute hidden representations \mathbf{h}_t based on the current input representation \mathbf{x}_t and the last hidden representation \mathbf{h}_{t-1} with $\mathbf{h}_t \in \mathbb{R}^N$:

$$\mathbf{h}_t = g_{\theta}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (3.8)$$

Considering that the hidden activations \mathbf{h}_t depend on their predecessors thus allows the RNN to capture information across multiple processing steps. This recurrent function $g_{\theta}(\cdot)$ can be realized in many different ways. In its simplest, most commonly used form, it is stated as

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_{hx} + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_{hh}) \quad (3.9)$$

with $\theta = \{\mathbf{W}_{hx}, \mathbf{W}_{hh}, \mathbf{b}_{hx}, \mathbf{b}_{hh}\}$ as the learnable parameters. In a classification setting, where we try to predict the most likely output category, we can produce a categorical probability distribution over all possible classes in the following way: For every hidden state \mathbf{h}_t , an affine transformation is applied to create the distribution

over possible output symbols \mathbf{o}_t (eq. 3.10) with $\mathbf{o}_t \in \mathbb{R}^{|\mathcal{V}|}$, where \mathcal{V} denotes the set of all symbols, or vocabulary. As this distribution usually does not meet the requirements of a proper probability distribution, it is then typically normalized using the *softmax* function, so that the total probability mass amounts to 1 (eq. 3.11)

$$\tilde{\mathbf{o}}_t = \mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_{ho} \quad (3.10)$$

$$\mathbf{o}_{ti} = \frac{\exp(\tilde{\mathbf{o}}_{ti})}{\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj})} \quad (3.11)$$

where $\mathbf{W}_{ho} \in \mathbb{R}^{|\mathcal{V}| \times N}$ and $\mathbf{b}_{ho} \in \mathbb{R}^{|\mathcal{V}|}$.

When training this kind of network, it is often observed that gradients shrink more and more the longer the input sequence is, not allowing for any meaningful improvements of the parameters, which is called the *vanishing gradient problem*: When calculating the gradients for the network's parameters, a modification to the earlier introduced backpropagation algorithm, *backpropagation through time* (BPTT) has to be used, as the recurrency of the network conditions later outputs on earlier processes. If we now evaluate the resulting product of partial derivatives, we often find gradients to tend to zero when the network's error is sufficiently small.¹

The Long-Short Term Memory network (LSTM) by Hochreiter and Schmidhuber, 1997, which is the basis for the models used in this work, tries to avoid this problem by augmenting the standard RNN recurrency by adding several gates:

$$\underline{\mathbf{f}}_t = \sigma(\mathbf{W}_{h\underline{f}}\mathbf{h}_{t-1} + \mathbf{W}_{x\underline{f}}\mathbf{x}_t + \mathbf{b}_{h\underline{f}} + \mathbf{b}_{x\underline{f}}) \quad (3.12)$$

$$\underline{\mathbf{i}}_t = \sigma(\mathbf{W}_{h\underline{i}}\mathbf{h}_{t-1} + \mathbf{W}_{x\underline{i}}\mathbf{x}_t + \mathbf{b}_{h\underline{i}} + \mathbf{b}_{x\underline{i}}) \quad (3.13)$$

$$\underline{\mathbf{o}}_t = \sigma(\mathbf{W}_{h\underline{o}}\mathbf{h}_{t-1} + \mathbf{W}_{x\underline{o}}\mathbf{x}_t + \mathbf{b}_{h\underline{o}} + \mathbf{b}_{x\underline{o}}) \quad (3.14)$$

$$\mathbf{c}_t = \underline{\mathbf{f}}_t \circ \mathbf{c}_{t-1} + \underline{\mathbf{i}}_t \circ \tanh(\mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{W}_{xc}\mathbf{x}_t + \mathbf{b}_{hc} + \mathbf{b}_{xc}) \quad (3.15)$$

$$\mathbf{h}_t = \underline{\mathbf{o}}_t \cdot \tanh(\mathbf{c}_t) \quad (3.16)$$

¹Or in other words, $\lim_{d \rightarrow \infty} \nabla_{\theta} \mathcal{L}^d = 0$ holds for the gradient of the parameters and an input lag d , i.e. the number of time steps the error is being propagated back in time, if the spectral radius (or biggest absolute eigenvalue) of $\nabla_{\theta} \mathcal{L}$ is smaller than one (Hochreiter et al., 2001).

which are called the forget gate \mathbf{f}_t , input gate \mathbf{i}_t and output gate \mathbf{o}_t (not to be confused with the output *activations* \mathbf{o}_t). A new type of activations, the cell state \mathbf{c}_t , is also added. The activations of all gates consist of linear combinations of affine transformations of the last hidden state \mathbf{h}_{t-1} and the current input \mathbf{x}_t (eqs. (3.12) to (3.14)).

However, each of them is assigned a different role within the cell’s architecture: The input gate determines to which extent information is flowing into the cell state, the forget gates modulates the amount of information inside the cell state that is being erased and the output gate defines how much information flows from the cell state to the new hidden state \mathbf{h}_t (eq. 3.16). The role of the cell state \mathbf{c}_t is to act as a form of memory storage that is being updated based on the current input. Not only does this enable the LSTM to model more long-term dependencies in its input sequence², the formulation of the cell state update in eq. 3.15 allows gradients to still “flow” through the forget gate when computing the derivative $\partial \mathbf{c}_t / \partial \mathbf{c}_{t-1}$, which in turn avoids vanishing gradients.

3.2.2 Language Modeling with RNNs

In Language Modeling we try to model the probabilities of words in sequences. Considering a sequence of T words (or sentence) $\mathcal{S} = \langle w_1, \dots, w_T \rangle$, the question arises how to assign a probability to it. The joint probability $p(w_1^T)$ is commonly factorized into the product of the conditional probabilities of every word given all its predecessors

$$p(w_1^T) = p(w_1)p(w_2|w_1)p(w_3|w_2, w_1)\dots = p(w_1) \prod_{t=2}^T p(w_t|w_1^{t-1}) \quad (3.17)$$

As shown in the previous section, the output probability distribution over words \mathbf{o}_t is dependent on the hidden representation \mathbf{h}_t , which is in turn dependent on the current input embedding for a word \mathbf{x}_t and the previous hidden state \mathbf{h}_{t-1} . Therefore, we can see the RNN modeling this exact conditional probability $p(w_t|w_1^{t-1})$, where the output distribution tries to predict the next word in the sequence. That means that the probability of the next word w_{t+1} being the c -th word in the vocabulary \mathcal{V} assigned by the model can be written as follows:

$$p(w_{t+1} = c | w_1^t) = \mathbf{o}_{tc} \quad (3.18)$$

²This is what seems to give rise to all kinds of linguistic abilities in LSTMs, see e.g. Liu et al., 2018; Linzen et al., 2016; Gulordava et al., 2018; Bernardy, 2018; Jumelet and Hupkes, 2018.

To train the model parameters θ , a loss function \mathcal{L} then measures the difference of the output distribution \mathbf{o}_t to the target prediction, or in this case, the actual following word, w_{t+1} . Then, the gradient descent rule from the previous section (or any other update rule) is applied. For language modeling, the most popular choice of loss function is the cross-entropy loss:

$$\mathcal{L}_{CE} = -\frac{1}{T} \sum_{t=1}^T \sum_{c=1}^{|V|} \mathbb{1}(w_{t+1} = c) \log(\mathbf{o}_{tc}) \quad (3.19)$$

where $\mathbb{1}(\cdot)$ is a binary function that is 1 when \mathbf{o}_{tc} corresponds to the probability of the actually observed token at time step $t + 1$ and 0 otherwise. Intuitively, this loss measures the difference between two probability distributions and penalizes when the predicted output distribution \mathbf{o}_t deviates from a target distribution, where only the observed token should be assigned a probability of 1.

3.3 Bayesian Deep Learning

In this section I deliver some information about the incorporation of Bayesian statistics into Deep Learning. Bayesian statistics allows us to also express a degree of belief, such as a personal prior belief about the probability of an event happening. This differs from the purely frequentist approach, which simply observes the likelihood of certain events. Transferring this framework to Deep Learning introduces some problems - these and a possible mitigation are layed out in section 3.3.1. Methods to estimate the confidence of a model in its predictions using methods based on Bayesian Deep Learning are outlined in sections 3.3.2 and 3.3.3, which will be exploited in later chapters by the recoding mechanism.

3.3.1 Variational inference

Regular feedforward neural networks are prone to overfitting and do not incorporate uncertainty into their predictions, which often results in overly confident predictions (Blundell et al., 2015). This problem can be alleviated by trying to harmonize models with the Bayesian approach: The training regime explained in the previous section is equivalent to retrieve the *Maximum Likelihood Estimate* (MLE) of the model parameters θ given a data set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_1^N$ of data points \mathbf{x}_i and corresponding labels y_i :

$$\boldsymbol{\theta}_{MLE} = \arg \max_{\theta} \log p(\mathcal{D}|\boldsymbol{\theta}) = \arg \max_{\theta} \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \quad (3.20)$$

Intuitively, this solution retrieves the parameters that explain the data best, or maximize the *likelihood* $p(\mathcal{D}|\boldsymbol{\theta})$. However, one can also try to find the most likely parameters given a data set $p(\boldsymbol{\theta}|\mathcal{D})$, which can be obtained by using Bayes' theorem:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} \quad (3.21)$$

where we call $p(\boldsymbol{\theta}|\mathcal{D})$ the *posterior* distribution, $p(\boldsymbol{\theta})$ the *prior* and $p(\mathcal{D})$ the *evidence*. Maximizing this quantity (or obtaining the *Maximum a posteriori estimate* (MAP)) by finding the best model parameters $\boldsymbol{\theta}$, we can ignore the evidence, because it is not dependent on them:

$$\boldsymbol{\theta}_{MAP} = \arg \max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathcal{D}) = \arg \max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \quad (3.22)$$

Unfortunately, a problem arises when trying to perform exact Bayesian inference on a new data point \mathbf{x}' in order to predict its label y' . This requires one to evaluate the predictive distribution

$$p(y'|\mathbf{x}') = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[p(y'|\mathbf{x}', \boldsymbol{\theta})] = \int p(y'|\mathbf{x}', \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \quad (3.23)$$

because of the fact that the model parameters have to be marginalized out:

$$\int p(y'|\mathbf{x}', \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} = \int p(y', \boldsymbol{\theta}|\mathbf{x}')d\boldsymbol{\theta} = p(y'|\mathbf{x}') \quad (3.24)$$

This however is intractable in most practical scenarios as we have to integrate over all values of the network parameters $\boldsymbol{\theta}$. Overcoming this problem is the main motivation for using variational inference: We approximate the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ with an easy to evaluate variational distribution $q_{\phi}(\boldsymbol{\theta})$ parameterized by variational parameters ϕ . To find the optimal parameters ϕ^* (and therefore the distribution $q_{\phi}(\boldsymbol{\theta})$ that approximates the true posterior best), we try to minimize the evidence lower bound (ELBO) or variational free energy

$$\phi^* = \arg \min_{\phi} \text{KL}[q_{\phi}(\boldsymbol{\theta})||p(\boldsymbol{\theta})] - \mathbb{E}_{q_{\phi}(\boldsymbol{\theta})}[\log p(\mathcal{D}|\boldsymbol{\theta})] \quad (3.25)$$

which can be derived from evaluating $\text{KL}[q_{\phi}(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})]$ (for the full derivation see appendix A.1). $\text{KL}[\cdot||\cdot]$ here denotes the Kullback-Leibler divergence, an asymmetric similarity measure to quantify the distance between two probability distributions, in our case the real posterior distribution and the easier substitute we choose.

Blundell et al., 2015 uses this framework to propose *Bayes by Backprop*, a way to use the Backpropagation algorithm to learn *distributions* over weights instead of point estimates. However, this approach requires the model to store additional distribution parameters and results in overall slower training. Therefore I will showcase other works trying to find more scalable ways to achieve this in the next section.

3.3.2 Estimating model confidence with Dropout

Dropout (Srivastava et al., 2014) is a popular stochastic regularization technique utilized to avoid overfitting in neural networks. It is applied during training by randomly cutting out varying neurons, forcing the network to not rely on specific units for a successful prediction.

It was shown in the work of Gal and Ghahramani, 2016b that applying this scheme can be seen as variational inference in Deep Gaussian processes. The intractable posterior in equation 3.23 is now approximated by a variational distribution over the model weight matrices $q(\omega)$, whose elements are randomly set to zero³, according to the Dropout training regime. From there, they try to match the moments (i.e. the mean and variance) of the approximate predictive distribution using K Monte Carlo estimates

$$\mathbb{E}_{q(y'|\mathbf{x}')}[f_{\theta}(\mathbf{x}')] \approx \frac{1}{K} \sum_{k=1}^K f_{\theta}(\mathbf{x}', \omega^{(k)}) \quad (3.26)$$

$$\text{Var}_{q(y'|\mathbf{x}')}[f_{\theta}(\mathbf{x}')] \approx \tau^{-1} \mathbf{I}_D + \frac{1}{K} \sum_{k=1}^K f_{\theta}(\mathbf{x}', \omega^{(k)})^T f_{\theta}(\mathbf{x}', \omega^{(k)}) \quad (3.27)$$

which they refer to as *MC Dropout* (MCD). $f_{\theta}(\mathbf{x}', \omega^{(k)})$ simply denotes the prediction of the network for the new data point using a set of weight matrices $\omega^{(k)}$ with a different dropout mask. The mask itself is sampled from a multivariate Bernoulli distribution, according to the original description of the technique in Srivastava et al., 2014. The first term in the sum of eq. 3.27, τ , denotes the model precision, which corresponds to

$$\tau = \frac{pl^2}{2N\lambda} \quad (3.28)$$

This term results from using Dropout in order to obtain the approximate posterior distribution and using a weight prior of the form $p(\omega) = \mathcal{N}(0, l^{-2}\mathbf{I})$, with p being the

³As this distribution is not put onto the model's biases, we denote the set of weight matrices ω to distinguish it from all model parameters θ .

dropout probability, l the length scale of the weight prior distribution (expressing our prior belief about the realization of the model weights), N the number of training samples and λ the weight decay parameter⁴. In a regression setting, this finally results in a predictive variance approximated by

$$\text{Var}_{q(y'|\mathbf{x}')}[f_{\theta}(\mathbf{x}')] \approx \tau^{-1} \mathbf{I}_D + \frac{1}{K} \sum_{k=1}^K f_{\theta}(\mathbf{x}', \boldsymbol{\omega}^{(k)})^T f_{\theta}(\mathbf{x}', \boldsymbol{\omega}^{(k)}) - \mathbb{E}_{q(y'|\mathbf{x}')}[y']^T \mathbb{E}_{q(y'|\mathbf{x}')}[y'] \quad (3.29)$$

In other words, in order to approximate the predictive variance of the variational posterior, we conveniently only have to perform K forward passes with our model, each one of them using a different Dropout mask. This whole approach is applicable to MLPs. Because this work is concerned with Recurrent Neural Networks, the corresponding extension of this approach is also being discussed.

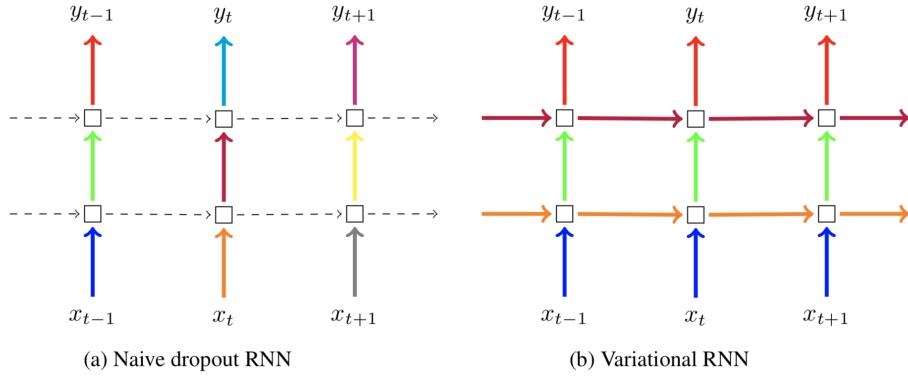


Figure 3.2.: Figure taken from Gal and Ghahramani, 2016a: While traditional approaches apply different kind of dropout masks to only a select set of connections (left), the variational RNN Dropout approach always applies the same set of masks to the same connection when processing a sequence.

When applying Dropout to RNNs, Gal and Ghahramani, 2016a show that it is inhibitive to sample a different mask for every connection at every time step. Instead, they propose a scheme where only a single set of Dropout masks is sampled for every batch, reusing the same mask for the same type connection while processing the sequence (see fig. 3.2). The authors refer to this as the *Variational RNN*. Thus the same way of estimating the predictive uncertainty as above can be used, with the exception that $f_{\theta}(\mathbf{x}', \boldsymbol{\omega}^{(k)})$ now refers to the prediction of the network at time step t and that $\boldsymbol{\omega}^{(k)}$ refers to the set of all weight matrices in the RNN with the same set of Dropout masks applied to it throughout the whole input⁵.

⁴Placing a Gaussian prior on the weights is known to result in l_2 -regularization, also called *weight decay* (MacKay and Mac Kay, 2003).

⁵ $\boldsymbol{\omega}_{\text{RNN}}^{(k)} = \{\mathbf{W}_{hx}^{(k)}, \mathbf{W}_{hh}^{(k)}, \mathbf{W}_{ho}^{(k)}\}$ for a vanilla RNN.

Lastly, because in Language Modeling we are trying to predict the probability of a class, the method to estimate the model's confidence has to be adapted accordingly, as the probability vector that is the output of the model does not capture model confidence (Gal, 2016). Instead, we can consider the *predictive entropy* of a model, which - in the case of a discrete probability distribution - is defined as

$$\mathbb{H}[y|\mathbf{x}, \mathcal{D}_{\text{train}}] = - \sum_c p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \quad (3.30)$$

Entropy is a measure from the field of information theory and captures the amount of information contained in a probability distribution. Minimum entropy is reached when all the mass of the distribution rests on a single class, the maximum value is attained when the underlying distribution is uniform. Again, the term $p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$ is intractable, but can be approximated using K forward passes of the model with different Dropout masks (a full proof of this is given in appendix A.2), where $p(y = c|\mathbf{x}, \omega^{(k)})$ denotes the probability of the c -th word produced by a single forward pass with a Dropout mask:

$$p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \approx \frac{1}{K} \sum_{k=1}^K p(y = c|\mathbf{x}, \omega^{(k)}) \quad (3.31)$$

Therefore, when the mean distribution of these K forward passes is almost uniform, the model is not confident which class to predict; conversely, if all passes amass all the probability on a single class, the model is confident that this class corresponds to the right prediction. In the next section I will lay out another, non-variational approach to obtain this quantity.

3.3.3 Estimating model confidence with ensembles

Pearce et al., 2018 choose another, Bayesian but non-variational approach to estimating the predictive uncertainty by using ensembles: Although using an ensemble of deep models had already been realized before (Lakshminarayanan et al., 2017), it lacked a Bayesian motivation. Pearce et al., 2018 achieve to overcome this blemish by a largely unknown inference method called *randomized MAP sampling*: It is known that for a multivariate normal likelihood with parameters μ_{like} , Σ_{like} and prior distribution with μ_{prior} , Σ_{prior} , the maximum a posteriori solution for the mean μ is given by

$$\mu_{\text{MAP}} = (\Sigma_{\text{like}}^{-1} + \Sigma_{\text{prior}}^{-1})^{-1} (\Sigma_{\text{like}}^{-1} \mu_{\text{like}} + \Sigma_{\text{prior}}^{-1} \mu_{\text{prior}}) \quad (3.32)$$

In randomized MAP sampling, a distribution of μ_{MAP} solutions is produced by using a noise-injecting mechanism. As the source of noise, we replace μ_{prior} with a random noise variable θ_0 , which produces a function f_{MAP} based on the current θ_0 :

$$f_{\text{MAP}}(\theta_0) = (\Sigma_{\text{like}}^{-1} + \Sigma_{\text{prior}}^{-1})^{-1}(\Sigma_{\text{like}}^{-1}\mu_{\text{like}} + \Sigma_{\text{prior}}^{-1}\theta_0) \quad (3.33)$$

This way, every time f_{MAP} is applied to a noise variable θ_0 , it produces a new μ_{MAP} solution. The parameters of the *anchor noise distribution* that the variable θ_0 is sampled from approximates the true posterior $\mathcal{N}(\mu_{\text{MAP}}, \Sigma_{\text{MAP}})$. It can then be found by (see Pearce et al., 2018, appendix A for full derivation)

$$\mu_0 = \mu_{\text{prior}} \quad (3.34)$$

$$\Sigma_0 = \Sigma_{\text{prior}} + \Sigma_{\text{prior}}^2 \Sigma_{\text{like}}^{-1} \quad (3.35)$$

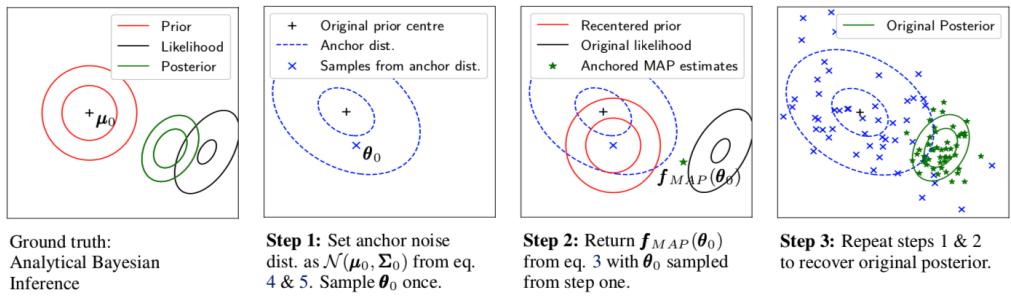


Figure 3.3.: Illustration of the randomized MAP sampling procedure, taken from the work of Pearce et al., 2018.

Therefore, repeatedly sampling $\theta_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ and returning $f_{\text{MAP}}(\theta_0)$ recovers the original posterior, as illustrated in figure 3.3. Transferring this concept to deep neural networks, it is known that a training loss with weight decay can be interpreted as MAP estimates with a normal prior centered at zero (MacKay and Mac Kay, 2003). In a classification setting, this results in a cross-entropy loss of the form

$$\mathcal{L}_{\text{CE}} = -\frac{1}{T} \sum_{t=1}^T \sum_c \mathbb{1}(y=c) \log p(y=c) + \frac{1}{T} \|\Gamma^{1/2} \theta\| \quad (3.36)$$

where Γ is a diagonal matrix of the weight decay parameter λ . We can modify this loss to return MAP estimates with prior estimates not centered at zero values:

$$\mathcal{L}_{\text{CE}}^{(k)} = -\frac{1}{T} \sum_{t=1}^T \sum_c \mathbb{1}(y=c) \log p(y=c) + \frac{1}{T} \|\Gamma^{1/2}(\theta^{(k)} - \theta_0^{(k)})\| \quad (3.37)$$

for every member k of a ensemble of K NNs. Finally, in order to draw $\theta_0^{(k)}$ with the parameters from equations 3.34 and 3.35, we approximate the latter as $\Sigma_0 = \Sigma_{prior}$, where Pearce et al., 2018 show that this approximation improves as the correlation between network parameter increases. Using this loss for every member of the ensemble simply ensures that we still obtain the real posterior.

In order to now estimate the confidence of the model, we can take the output distribution of every member in the ensemble to estimate the predictive entropy in eq. A.3 by using the approximation in eq. 3.31. A justification for this is given in appendix A.2.

3.4 Recap

In this chapter I layed out the theoretical foundations for the next chapter. The main model type used in this work, the LSTM network described in 3.2.1, is especially suited for the task of Language Modeling (section 3.2.2) due to its ability to be trained on modeling sequences while also being able to capture linguistic information. As recoding, the mechanism proposed in this thesis, aims to be unsupervised, we also explored two different approaches to approximate the true weight posterior distribution by using Dropout (section 3.3.2) and Bayesian anchored ensembles (section 3.3.3), both of which also enable the user to approximate model confidence in the form of predictive entropy. In the following chapter, I combine all these ideas into a single framework, called the recoder.

Recoding Framework

Having discussed related work and the necessary background for this thesis, I will use this chapter to layout the form and details of the proposed framework. The main idea stems from the work of Giulianelli et al., 2018, where activation corrections were named *interventions*, which is now being explored in more detail.

4.1 Interventions

In order to show how recoding is a generalization of *interventions*, let us first discuss the latter. Giulianelli et al., 2018 examine the ability of LSTMs to track subject-verb agreement, i.e. whether the models recognize the right verb number corresponding to the subject. Consider the sentence *She only talks gobbledegook* and its ungrammatical counterpart **She only talk gobbledegook*. In the second case, the grammatical violation manifests in the differing number between subject and verb, i.e. *she* requiring the 3rd person singular form *talks* when *talk* is used instead. If LSTMs store this information, then we would expect a Language Model to consistently assign a higher probability to the grammatical sentence than to the one in violation to the English grammar.

That this is indeed the case was already shown in the work of Gulordava et al., 2018, however, Giulianelli et al., 2018 extend these results further: They use Diagnostic classifiers (Hupkes et al., 2018; Dalvi et al., 2018; Conneau et al., 2018) (DCs), linear classifiers that are trained on the activations of the models in order to predict the subject's number. If the classifier reaches a high accuracy, it implies that the number information is reliably encoded in the activations and can thus be used for a successful prediction. Not only do they show that this information is indeed tracked, but also that it can be corrupted by *attractors*, intermediate nouns of a different number.

Their key contribution to this work now lies in trying to rectify these deviations by the use of *interventions*: Using the binary cross-entropy loss of the DC compared to the true subject number allows them to compute the gradient of this loss w.r.t. to the **activations**. After performing an update step akin to the gradient descent step they are able to conclude that this procedure indeed restores the damaged number information.

This idea will now be extended and generalized into a framework called *recoding*. Notwithstanding the similarities, I want to emphasize some key differences of the interventions in Julianelli et al., 2018 compared to the proposed framework:

1. **Interventions were only performed during inference** Intervention gradients were applied to an already trained LSTM, while in this work recoding will already be applied during training.
2. **Interventions were performed with a secondary objective in mind** While the model used in Julianelli et al., 2018 was also trained to model language, the main focus of the work was to examine the ability of the model to retain information of the subject's number. Interventions were helpful here to avoid the corruption of this information while processing a sequence without changing the models perplexity significantly. In contrast, this work aims to utilize recoding directly to improve the main performance metric of the task.

Therefore, when using additional labels to compute the error signal, it has to be considered to which extent the information contained in this labels is actually useful for the model to optimize the primary objective. Furthermore, useful secondary labels can be hard to come by or have to be created manually, which is undesirable for the generalizational ability of the recoding approach. The next sections thus focus on a more general concept.

4.2 Recoding Mechanism

Let us call the corrective mechanism that is added to the model the *recoding mechanism* or *recoder*. In its most abstract form, it simply consists of another computational step regarding the hidden activations of a model, which does not include any additional parameters. Let us define the mechanism as a function $r : \mathbb{R}^N \mapsto \mathbb{R}^N$ that maps a real-valued vector of activations to another vector of the same dimensionality.¹ In the following, we will focus on the case of a RNN that produces times-step dependent hidden activations $\mathbf{h}_t \in \mathbb{R}^N$. Thus, we can define the recoding function as a modification of the hidden state using the Delta-rule (Widrow and Hoff, 1960):

$$r(\mathbf{h}_t) \equiv \mathbf{h}'_t = \mathbf{h}_t - \alpha_t \nabla_{\mathbf{h}_t} \delta_t \quad (4.1)$$

where \mathbf{h}'_t denotes the modified hidden state that will subsequently be used by the network, α the recoding *step size* and δ_t a time-dependent *error signal*. Naturally, as the anti-recoding gradient $-\nabla_{\mathbf{h}_t} \delta_t$ points into the direction from \mathbf{h}_t that minimizes the error signal δ_t , taking a step α_t into its direction will result in a decrease or at

¹While it could also be possible to imagine a version that maps multiple activations to a single new one, we will only consider this version in this work for now.

least stagnation of the signal (see appendix A.4 for a full proof). Not specifying the kind of error signal enables us to consider vastly different methods, as we will see later. The step size can be chosen to be constant, learned as an additional parameter or even parameterized using a whole different network (more about this in section 4.4). It is no coincidence that this modification mirrors the gradient descent update rule. Using this particular form allows us to derive certain theoretical guarantees about its efficacy while leaving room for extensions (similar to how the gradient descent update rule was modified using e.g. momentum (Polyak, 1964) and can be used with any arbitrary objective function).

The whole procedure is illustrated in figure 4.1 for a normal, unidirectional single-layer RNN. However, the whole procedure can easily be extended to any RNN variant, either uni- or bidirectional, with an arbitrary amount of layers, albeit recoding gradients will become increasingly smaller for lower layers the more distant they are from the error signal δ_t , as the chain of partial derivatives necessary to evaluate the gradient grows longer.

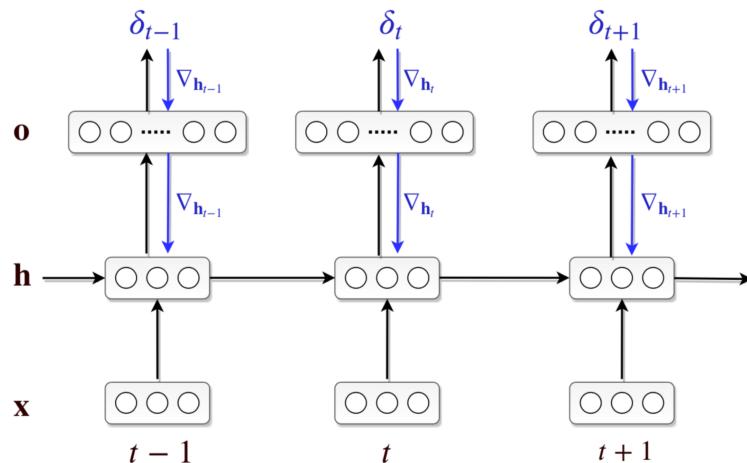


Figure 4.1.: Illustration of the recoding framework on a unidirectional, single-layer RNN. The error signal δ_t is computed based on the hidden activations h_t , or, in this case, the output activations o_t of every time step. Subsequently, it is used to compute the recoding gradient $\nabla_{h_t} \delta_t$ (blue) which is used in the update step to produce the recoded activations h'_t (*recoding*), which are the new activations utilized in the next RNN step.

The next sections are dedicated to exploring different choices of error signals (§4.3) and methods to determine the recoding step size (§4.4). Some additional information can be found in the appendix, e.g. theoretical guarantees for this framework in appendix A.4 and practical considerations during implementation in appendix B.2.

4.3 Error Signals

In this section, different sources for the error signal δ_t in a Language Modeling setting are explored. This ranges from using additional labels (4.3.1), to the “surprisal” of a model encountering a new token (4.3.2) or approximations of the predictive entropy of a model in sections 4.3.3 and 4.3.4.

4.3.1 Weak Supervision

Using this new framework, we can easily transfer the interventions from Julianelli et al., 2018 into a recoding setting. Additional binary labels $\langle y \rangle_1^T = \langle y_1, \dots, y_T \rangle$ with $y_t \in \{0, 1\}$ are used in combination with logistic regression classifiers to perform recoding. These classifiers simply predict the probability of one class by using an affine transformation of the hidden state. A loss function like binary cross-entropy loss can then be used to compute the difference between the predicted label and actual label:

$$\delta_t \equiv \mathcal{L}_{\text{BCE}}(\sigma(\mathbf{w}^T \mathbf{h}_t + \mathbf{b}), y_t) \quad (4.2)$$

where σ denotes the sigmoid function and \mathbf{w}, \mathbf{b} pre-trained weights and bias. These parameters have been trained based on the hidden activations of the model itself. This could further be generalized by using labels belonging to an arbitrary number of classes and using more complex classifiers than logistic regression. Because this still carries the problem of having to use additional labels, this approach is not regarded further in this thesis.

4.3.2 Surprisal

One of the easiest error signals to define in the context of language modelling would be the perplexity of the target token at a given time step, which we will call *surprisal* in order to distinguish it from our evaluation metric. In general, the surprisal for a sequence of events $\langle x_1, \dots, x_N \rangle$ is defined in terms of the entropy of their probability distribution $p(x)$:

$$\text{ppl}(x_1^N) = 2^{-\frac{1}{N} \sum_{x_i} p(x_i) \log_2 p(x_i)} \quad (4.3)$$

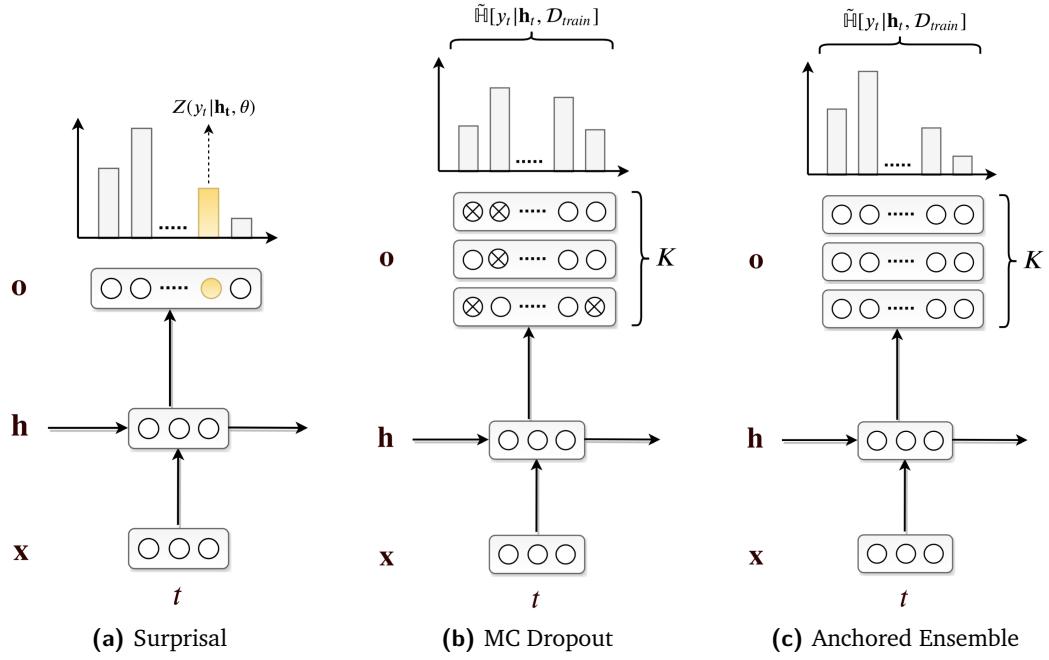


Figure 4.2.: Illustration of the different error signals used in this work. (a) After generating the output distribution, the probability of the gold token (actual next word) is selected to compute the surprisal score. (b) K forward passes with different Dropout masks are averaged, after which the predictive entropy of the resulting distribution is computed. (c) Like (b), but K distributions stem from several ensemble members.

If we now exclusively consider the probability of the gold token $p(w_{t+1}|w_1^t)$, this expression can be simplified. Let us denote the surprisal of a model as $Z(y_t|h_t, \theta)$, were $y_t = w_{t+1}$ in the language modelling case:

$$\delta_t \equiv Z(y_t | \mathbf{h}_t, \theta) = \sum_c \mathbb{1}(y_t = c) 2^{-\mathbf{o}_{tc} \log_2(\mathbf{o}_{tc})} - 1 = \sum_c \mathbb{1}(y_t = c) \mathbf{o}_{tc}^{-\mathbf{o}_{tc}} - 1 \quad (4.4)$$

where we simply add -1 so that the minimum surprisal value is 0. This does not influence the expression of the recoding gradient, which can be derived to be

$$\nabla_{\mathbf{h}_t} Z(y_t | \mathbf{x}_t, \boldsymbol{\theta}) = -\mathbf{A} \left(\log(\mathbf{o}_t) + \mathbf{1} \right) \exp \left(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t) \right) \left(\text{diag}(\mathbf{o}_t) - \mathbf{o}_t \mathbf{o}_t^T \right) \mathbf{W}_{ho} \quad (4.5)$$

The full derivation for this is given in appendix A.5. Intuitively, this error signal moves the hidden activations into a direction where the resulting surprisal score is being lowered, in other words the prediction based on these activations \mathbf{h}'_t would assign a higher probability to the gold token than the one based on the original \mathbf{h}_t .

Equipped with these results, we can apply surprisal recoding in the way described in fig. B.1 in appendix B.1 and illustrated in fig. 4.2a: For every time step, we use the probability of the gold token to compute the surprisal score, which in turn is used to compute its gradient w.r.t. to the hidden state \mathbf{h}_t . After the update step, the network continues with the recoded hidden activations. Two things should be noted here: Firstly, this approach is still supervised, but only uses the gold labels already included in the training set. Secondly, the output distribution \mathbf{o}_t cannot be recomputed based on the new recoded hidden activations \mathbf{h}'_t as this would trivialize the task.² However, in theorem 2 we have proven that recoding also has a positive influence on the value of future error signals.

4.3.3 MC Dropout

Another approach to determine an error signal lies in using the model’s predictive entropy. Here, MC Dropout (Gal and Ghahramani, 2016b) and Variational RNN Dropout (Gal and Ghahramani, 2016a), introduced in section 3.3, provide a procedure to estimate this measure by performing several forward-passes of the model using different Dropout masks. We apply these techniques in two distinct ways:

1. Apply Gal and Ghahramani, 2016b by making the simplifying assumption that the decoder weights \mathbf{W}_{ho} can be seen as a single-layer MLP. For every time step t , sample K Dropout masks to estimate the predictive entropy and use that in order to recode the hidden activations \mathbf{h}_t .
2. Apply Gal and Ghahramani, 2016a by sampling K Dropout masks at the beginning of each batch corresponding to every type of connection in the RNN. For every time step t , estimate the predictive entropy this way and use that in order to recode all hidden activations $\mathbf{h}_t^{(k)}$.

First, we focus on the simplified approach: Here, we apply the standard Dropout procedure by using a Dropout mask sampled from a Bernoulli distribution with probability p s.t.

$$\mathbf{Z}_{ij}^{(k)} \sim \text{Bernoulli}(p) \quad \forall i = 1, \dots, N \quad \forall j = 1, \dots, |\mathcal{V}| \quad (4.6)$$

$$\mathbf{W}_{ho}^{(k)} = \mathbf{W}_{ho} \odot \mathbf{Z}^{(k)} \quad (4.7)$$

In this case, the set of weights $\omega^{(k)}$ using for a stochastic forward pass is simply $\omega^{(k)} = \{\mathbf{W}_{ho}^{(k)}\}$ and the probability of a class given those weights $p(y_t = c | \mathbf{x}_t, \omega^{(k)}) = \mathbf{o}_{tc}^{(k)}$. Using eqs. A.3 and 3.31, we can define the error signal in terms of the

²The recoding gradient induces knowledge about the gold token into the model. Therefore recomputing the output predictions would give the model an unfair advantage, as it already possesses knowledge about its target.

approximate predictive entropy, now conditioned on the hidden state \mathbf{h}_t due to the recurrent setting:

$$\delta_t \equiv \tilde{\mathbb{H}}[y_t | \mathbf{h}_t, \mathcal{D}_{\text{train}}] = -\sum_{c=1}^{|\mathcal{V}|} \hat{\mathbf{o}}_{tc} \log(\hat{\mathbf{o}}_{tc}) = -\hat{\mathbf{o}}_t^T \log(\hat{\mathbf{o}}_t) \quad (4.8)$$

where

$$\hat{\mathbf{o}}_t = \frac{1}{K} \sum_{k=1}^K \mathbf{o}_t^{(k)} \quad (4.9)$$

The corresponding recoding gradient can be identified as

$$\nabla_{\mathbf{h}_t} \tilde{\mathbb{H}}[y_t | \mathbf{h}_t, \mathcal{D}_{\text{train}}] = -\left(\log(\hat{\mathbf{o}}_t) + \mathbf{1}\right) \frac{1}{K} \sum_{k=1}^K (\text{diag}(\mathbf{o}_t^{(k)}) - \mathbf{o}_t^{(k)} (\mathbf{o}_t^{(k)})^T) \mathbf{W}_{ho}^{(k)} \quad (4.10)$$

The full derivation is given in appendix A.6. On an intuitive level, this means that the recoding gradient based on this error signal moves the hidden activations in such a direction such that the *confidence* of the model's prediction based on these activations is increased. However, because the predictive entropy is approximated using Monte Carlo methods, the quality of the gradient depends on the number of samples K used. The procedure is illustrated in fig. 4.2b and written in pseudocode in fig. B.2 in appendix B.1. Note that because this error signal is being computed in an entirely supervised manner, we are allowed to recompute the output distribution \mathbf{o}_t here.

The approach using the variational LSTM Dropout is also tested: Here we can approximate the predictive entropy by sampling a different Dropout masks for every type of connection as illustrated in figure 3.2. To do so, we have to calculate the network's predictions under K different sets of Dropout masks. We can also imagine this as an ensemble of K RNNs, each equipped with a different set. Practically, although this results in less Dropout masks to sample in total³, it is most efficient to keep track of all intermediate hidden, cell and output activations by transforming batches into pseudo-batches of with $K \times B$ instances, where we save each batch instance with a different Dropout mask. This effectively slows down the model training again, but also provides more theoretically grounded estimates. The derived recoding gradient does not change, except that \mathbf{h}_t is now a concatenation of all

³ $K \times T$ masks in the simplified version vs. $3 \times K + (2 \times L \times K)$ for an LSTM with a number of layers L , therefore less masks are sampled when $3 + 2 \times L < T$.

hidden activations produced by the network under different sets of Dropout masks s.t. $\mathbf{h}_t = [\mathbf{h}_t^{(1)} \oplus \dots \oplus \mathbf{h}_t^{(K)}]$.

4.3.4 Bayesian Anchored Ensembling

Bayesian Anchored Ensembling (Pearce et al., 2018) (BAE), as introduced in 3.3, allows for another way to estimate predictive uncertainty that does not require the costly sampling of Dropout masks. In contrast, we use an ensemble of K decoders with decoder weights $\mathbf{W}_{ho}^{(k)}$ to produce a set of different output activations, which is depicted in fig. 4.2c. In the context of the recoding framework, this approach comes with some caveats: Firstly, the theoretical foundations of this approach have not been extended to RNNs yet and are only valid for MLPs. Straightforwardly, one could just ensemble multiple RNN cells by sampling anchor points for every set of weights in the cell. However, ensembling K full RNNs this way increases training time tremendously. We will therefore employ the same simplification from the previous subsection by only applying this technique to the decoder, although it is debatable whether certain assumptions still hold under this circumstances: Pearce et al., 2018 claim that Bayesian anchored ensembling gives good results for sufficiently wide NNs, as these are known to display an increasing correlation between parameters over the course of a training. Nevertheless, we proceed with this simplification for now.

The derivation of the recoding gradient $\nabla_{\mathbf{h}_t} \delta_t$ for anchored ensembles is equivalent to that of MC Dropout in section 4.3.3 and is therefore not repeated here. The main difference is that $\omega^{(k)}$ now does not denote the weights of the output layer with a specific Dropout mask but the weights of the k -th member of the anchored ensemble and $\mathbf{o}_t^{(k)}$ to the normalized output probabilities produced by these weights. A justification that this can also be used to approximate the predictive entropy is given in appendix A.2. Notwithstanding the formal resemblance to the gradient derived for MC Dropout recoding, the actual gradients differ in practice, as $\omega^{(k)}$ refers to the same set of weights with different Dropout masks in the case of MC Dropout, but to a set of independently initialized weights in this circumstance.

Again, the resulting recoding gradient based on this error signal points into a direction such that the *confidence* of the model’s prediction based on these activations is increased. In similar fashion to the MC Dropout error signal, the quality of the gradient is expected to depend on the number of members in the ensemble K .

The full pseudocode for this procedure is given in fig. B.3 in B.1. As calculating the additional weight decay loss term $\mathcal{L}_{\text{anchor}}^{(k)}$ and learning all different K members of the ensemble separately slows down training, I employ a variant of this approach

where we only sample a single anchor point for all ensemble members and average their losses to one single term:

$$\mathcal{L}_{\text{total}} = \frac{1}{K} \sum_{k=1}^K (\mathcal{L}_{\text{CE}}^{(k)} + \mathcal{L}_{\text{anchor}}^{(k)}) \quad (4.11)$$

This amortized form of Bayesian Anchored Ensembling actually proved to train significantly faster and produce better results in preliminary experiments.

4.4 Step Size

Lastly, we are going to focus on different choices of the step size α . The theorems in appendix A.4 rely on the optimal step size which can be derived from the smallest Lipschitz constant L . In practice however, this constant is hard to obtain (this will be discussed in more detail in section 8.1.2). I therefore experiment with three different alternatives:

1. **Constant step size** The simplest approach just treats the step size as a hyperparameter and uses the same value α across different time steps, layers and activation types.
2. **Learned step size** The next approach defines the step size as an additional parameter, which is learned via gradient descent alongside the model's other parameters. This has two advantages: We do not have to perform any hyperparameter search for the step size and we can add multiple step size parameters for every layer and activation type.
3. **Predicted step size** Here we parameterize the step size using an additional network with parameters \varkappa s.t. $\alpha_t = f_\varkappa(\mathbf{h}_t)$. This way, step sizes can flexibly be determined based on the content of the sentence encoded in \mathbf{h}_t . Furthermore, we can also use of these predictors for every activation type and layer, using the corresponding layer's hidden activations as input. These predictors can also be learned using gradient descent, but thus add additional training time.

4.5 Recap

In this chapter I presented an extension and generalization of the interventions in Julianelli et al., 2018 that performs gradient-based updates of activations, called *recoding*. I furthermore presented three error signals that will be examined in the next chapter: Surprisal, which based recoding gradients on the probability of the gold token and can therefore be classified as a supervised recoding approach, as

well as MC Dropout and Bayesian Anchored Ensembling recoding. The latter two function in an entirely unsupervised manner and approximate the predictive entropy of the model at every processing time step. I also introduced three distinct ways of determined the step size of the recoding update step. All of these variants will now be tested extensively.

Experiments

In this chapter I present the results of several experiments applying the models described in the previous chapter to a Language Modeling task, assessing the impact of important hyperparameters and model components.

The data set is being introduced first in section 5.1 along with the training conditions in section 5.2. Secondly, I evaluate the different models with a language modeling objective in section 5.3. More precisely, the effect of the most prominent hyperparameters on performances is dissected in sections 5.3.1, 5.3.2 and 5.3.3. Additional experiments in section 5.3.4 focus on the impact of the dynamic step sizes like introduced back in chapter 4.4. Lastly, some ablation experiments try to explain the interplay between the LSTM and the recoder in chapter 5.3.5.

Implementing these models in a framework like PyTorch come with some caveats, which are described in appendix B.2.

5.1 Dataset

The data set used for the experiments is the Penn Treebank (PTB) (Marcus et al., 1993), consisting of documents in American English originating from different sources like e.g. radio transcripts, governmental documents and financial news reports. The corpus is annotated with linguistic information like the syntactic tree structure and Part-of-Speech tags, which is discarded for the purpose of this work. End-of-sentence tokens are also added. Notwithstanding its small size compared to modern Deep Learning data sets (about 4.5 million words in total), it is still a benchmark for state-of-the-art models in language modelling and was purposefully chosen, as its limited size allows for rapid prototyping of new models and allows full training even for slower models, which helps to ensure comparability.

5.2 Training

Apart from the parameters of the model whose values are learning during the training, the procedure and model also comes equipped with a number of parameters that have to be determined beforehand, called *hyperparameters*. In this instance, hyperparameters are e.g. the number of layers of a model, the choice of optimizer,

the learning rate (schedule), and many more. Due to the sheer number of possible hyperparameters, models and model variations as well as computational resource constraints I conduct the hyperparameter search differently than performing the rather naïve grid search. Figure B.1 in the appendix gives a (non-exhaustive!) list of the main hyperparameters of the different models. For some models like recoding using MC Dropout, this results in 12 possible hyperparameters to tune. Just performing grid-search with 4 options per parameter would result in $12^4 = 20736$ runs to evaluate, which is intractable under the given circumstances.

For this reason, I employ two strategies: First, we reduce the search space by adapting the hyperparameters for the core LSTM found by Gulordava et al., 2018, which can be seen in table B.1a. While the authors do not necessarily have a strong motivation for their chosen options, they perform an extensive grid search. Secondly, sampling hyperparameters from a distribution instead of arranging them in a grid is more efficient, as not all parameters are equally important (Bergstra and Bengio, 2012), which is why this strategy is preferred over the common grid search. The details of this procedure are described in appendix B.3. Even this approach will not produce the best hyperparameters, but it should be made clear that this is not the intention of this work. Instead, the aim is to produce a *good enough* set of hyperparameters that allows us to examine the effect of adjusting individual options, make comparisons to the baseline and probe the models for interesting behavior and properties. The final hyperparameters listed in table B.2b in the appendix will stay fixed in the following unless explicitly mentioned otherwise.

With the final set of parameters, the models are trained on the corpus for 8 epochs with a learning rate schedule corresponding to Gulordava et al., 2018, where the value is halved when no improvement has been observed for an entire epoch using mini-batch stochastic gradient descent and up to four NVIDIA 1080TI graphics cards at a time, each running one separate instance. Models are selected using early stopping based on their perplexity score on a validation set. As it is common practice for Language Modeling, the data set is partitioned into batches of a certain size and sequence length, where sentences do not necessarily end at the end of a batch but continue into the same instance of the next batch. More details about the evaluation is given in the next section.

5.3 Evaluation

The models are evaluated on the test set consisting of about 3600 sentences based on their perplexity scores. For a sequence of T words $\langle w \rangle_1^T = \langle w_1, \dots, w_T \rangle$, the perplexity of a model is defined as

$$\text{ppl}(w_1^T) = 2^{-\frac{1}{T} \sum_{t=1}^T \log_2 p(w_t)} \quad (5.1)$$

In the following subsections, I explore the impact of different hyperparameters and model components on the performance of the models compared to the baseline. To account for variance between model runs due to their random initialization, the results are given as the average over four runs, including their standard deviation. A single-tailed Student's t-test is used to validate possible improvements over the baseline.

5.3.1 Recoding Step Size

I begin by examining the impact of the most integral parameter, the recoding step size α . At first, we just look at the effect of setting the step size to fixed values, the results obtained by learning or predicting the step sizes are given in a later section 5.3.4. Although the following experiments cover a wide range of step sizes (from 10^{-4} to 25), not all of those are applicable to all models. For example, the surprisal given the target token often results in value below 1. In contrast, the approximate predictive entropy reaches values in the hundreds, therefore not all of the step sizes that are experimented on are applicable to all models, which is why some of the results have been omitted in table 5.1, where all scores are listed. It should also be mentioned that using variational RNNs on the basis of Gal and Ghahramani, 2016a to compute the error signals produced results magnitudes worse than all other considered model variants in preliminary experiments and corresponding outcomes were therefore disregarded in this and later sections.

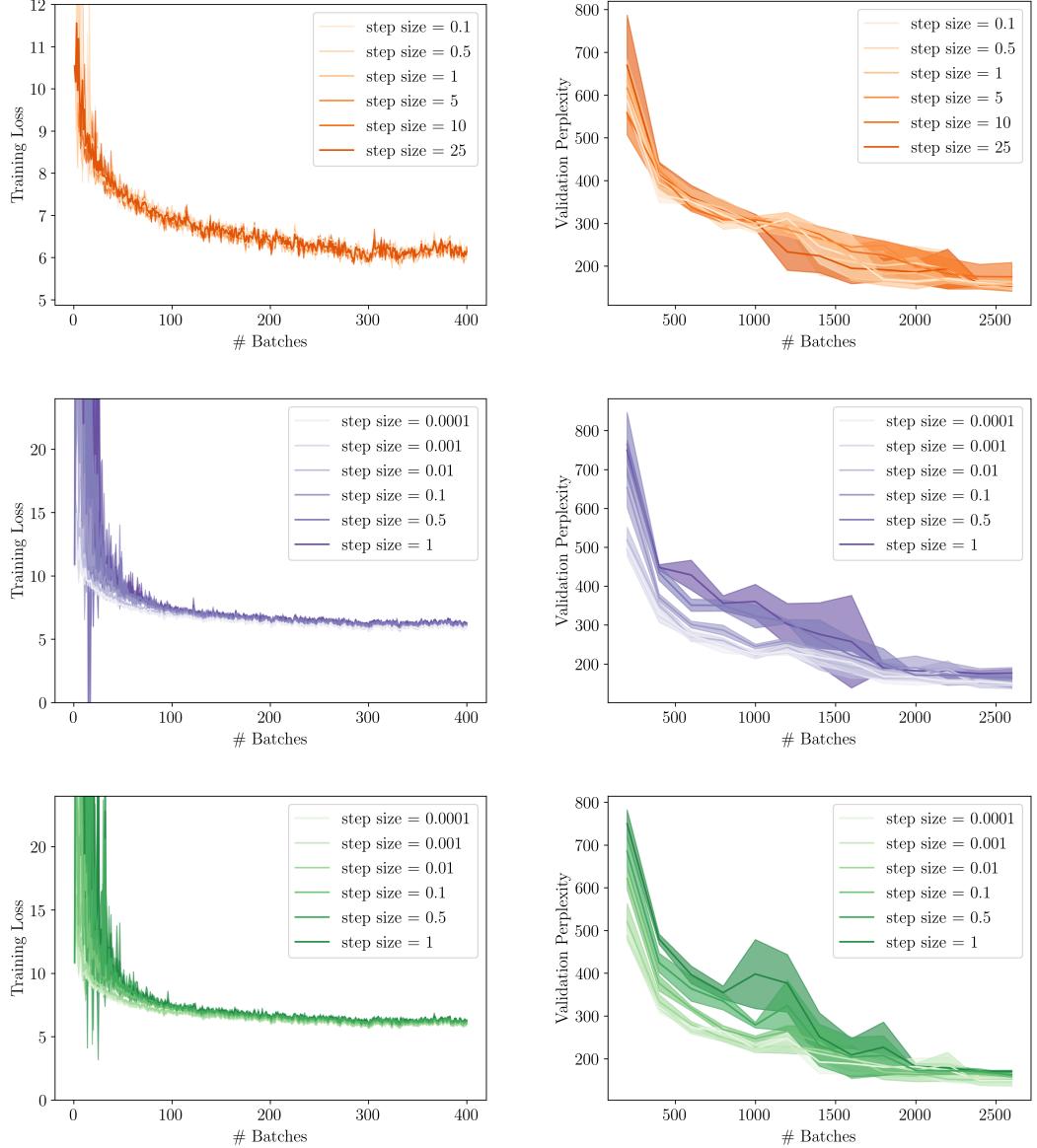


Figure 5.1.: Training loss (left column) and validation perplexities (right column) on PTB for different recoding step sizes using Surprisal, MC Dropout and Bayesian anchored ensemble recoding. Curves denote the mean over $n = 4$, with intervals signifying a standard deviation. Best viewed in color.

It seems immediately clear that there exists an advantageous range of values for the step size for all recoding types alike. Fig. 5.1 shows that for all recoding variants, more extreme values let models converge to worse training and validation scores. This is also reflected in the test perplexity scores in table 5.1, which are obtained for a step size of 5 for surprisal and 0.001 for MCD and BAE recoding. In the case of the latter two, a lower step size than 0.001 seems to result in a worse performance again.

Step size	Recoding Models			
	Surprisal	MC Dropout	BAE	Baseline
10^{-4}	-	144.03 ± 3.54	143.66 ± 8.79	126.60 ± 4.06
10^{-3}	-	139.61 ± 7.28	139.40 ± 7.24	
0.01	-	142.43 ± 8.12	144.29 ± 4.17	
0.1	126.93 ± 1.12	157.91 ± 22.03	151.61 ± 4.09	
1	126.94 ± 0.86	169.89 ± 7.36	164.79 ± 1.84	
5	125.33 ± 1.32	198.95 ± 7.66	204.00 ± 12.12	
10	170.51 ± 30.78	-	-	
25	235.14 ± 76.99	-	-	

Table 5.1.: Test perplexity on the test set for different models and a variety of recoding step sizes. Results are averaged of $n = 4$ runs. Best result overall is given in bold, best result per model in its corresponding color.

However, none of the novel models presented manage to outperform the baseline in a statistically significant manner. This could be due to one or a combination of the following reasons: Firstly, as seen in this line of experiments, the choice of step size seems to play a crucial role in the performance of the new models. Using a fixed step size therefore seems to be unlikely to be the best choice of the model under all circumstances encountered during testing. If we assume that decreasing the error signal δ_t has a positive impact on the model performance, then the theorems in appendix A.4 suggest that the possible improvements depend on using a step size that depends on the nature of the error function that produces δ_t . The “error signal surface”, however, changes with every parameter update of the model, and therefore the step size should be adjusted as well, but the exact adjustment is hard to determine (for a more in-depth discussion for this see chapter 7). Therefore, models using a more flexible step size are evaluated in section 5.3.4.

Another critical point to be considered is whether the chosen error signals actually provide the model with some advantage to solve the task. This is easier to argue for in the case of surprisal, because it is directly based on the probability of the target token. In the case of MCD and BAE recoding, i.e. when using predictive entropy, this is less clear. Here, recoding intuitively should improve the model’s confidence encoded in its activations, but it is uncertain whether increasing the confidence actually helps to avoid mispredictions later on and whether error occur by making very confident wrong predictions. These points are also considered in a later part of this work, namely chapter 6.1.3. To further assess the impact of certain hyperparameters, the effect of the number of samples K is evaluated next.

5.3.2 Number of samples

For MCD and BAE recoding, the error signal, i.e. the predictive entropy of the model, cannot be computed precisely but has to be approximated using a number of samples. To understand the extent to which the quality of this approximation influences the model performance, the two model variants are trained using an increasing amount of samples K . This happens on multiple levels: Similar to the previous section, first loss curves and test set results are given in fig. 5.2 and table 5.2, respectively.

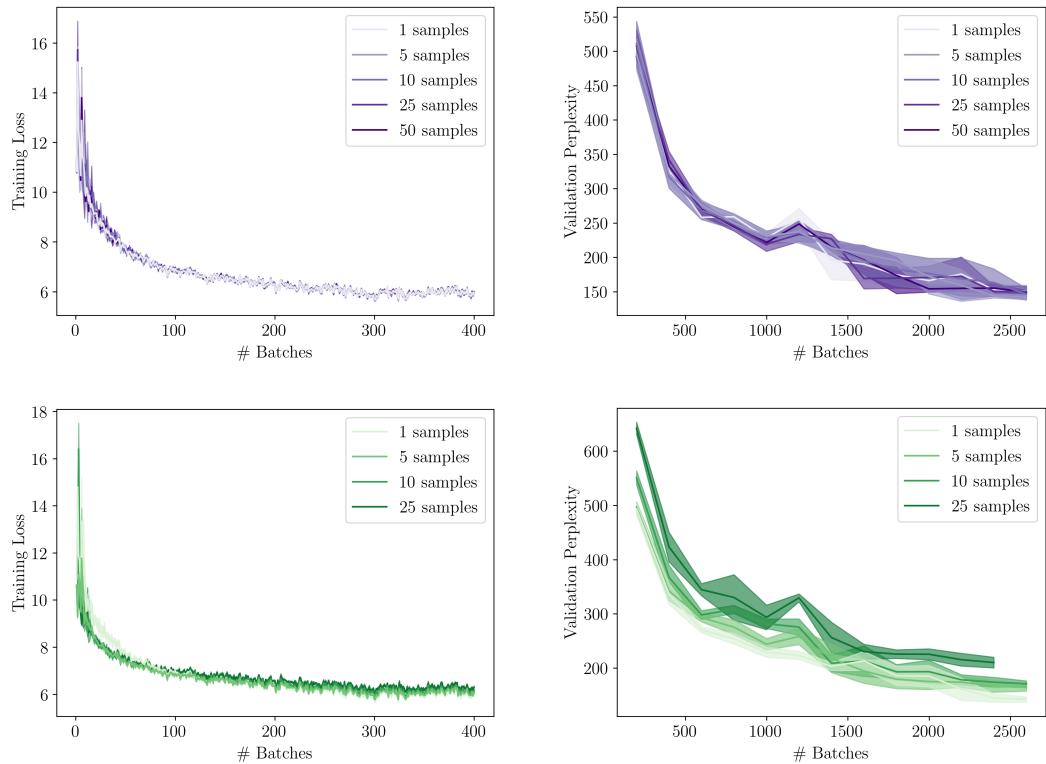


Figure 5.2.: Training loss (left column) and validation perplexities (right column) on PTB using a different amount of samples to approximate predictive entropy using [MC Dropout](#) and [Bayesian anchored ensemble](#) recoding. Curves denote the mean over $n = 4$, with intervals signifying the standard deviation. Best viewed in color.

Here we can identify very stark differences between the two models: Although training and loss curves seem to overlap almost completely, increasing the sample size seems to only yield diminishing returns after 5 samples for MCD recoding. For BAE recoding, the best result is achieved by just using a single (!) ensemble member. In both instances, the loss in speed is immediately apparent, as presented in the additional column of table 5.2: In the case of MC Dropout, additional Dropout masks have to be sampled for every k . In the case of Bayesian Anchored Ensembling, an additional member of the ensemble has to make a prediction. This combined with the additional gradient computations slow down both models by a factor of around

# Samples	Recoding Models				Baseline	
	MC Dropout		BAE		Baseline	
	Perplexity	\otimes Speed	Perplexity	\otimes Speed	Perplexity	\otimes Speed
1	145.77 \pm 8.99	66.85	138.54 \pm 4.24	59.15	126.60 \pm 4.06	406.35
5	138.02 \pm 2.79	48.3	161.30 \pm 9.22	32.2		
10	140.81 \pm 7.60	31.15	170.41 \pm 5.22	18.55		
25	143.88 \pm 7.03	14	215.23 \pm 12.16	8.05		
50	143.85 \pm 2.81	7.35	-	-		

Table 5.2.: Test perplexity on the test set for different models and a variety of recoding step sizes as well as average inference speed in tokens per second. Results are based on $n = 4$ runs. Speed is based running the evaluation on a single NVIDIA 1080TI graphics card.

6 in the case of a single k .

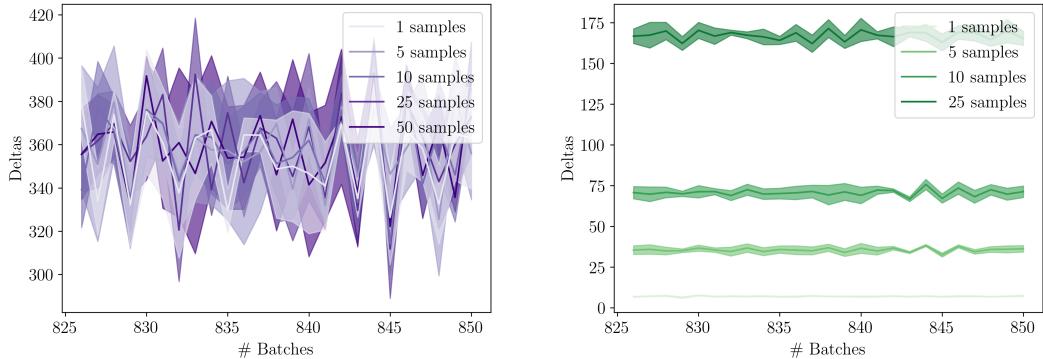
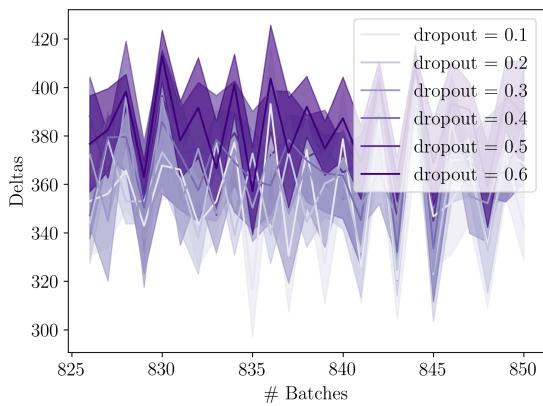


Figure 5.3.: Error signals on PTB using a different amount of samples on a later stage of the training using MC Dropout and Bayesian anchored ensemble recoding. Curves denote the mean over $n = 4$, with intervals signifying the standard deviation. Best viewed in color.

Lastly, let us briefly investigate the agreement between the uncertainty estimates of both models as well as the agreement among instances of the same model using a different number of samples. Fig. 5.3 displays the average estimated uncertainty per batch during a later stage in the training. These plots seem to elucidate why BAE recoding performs best only using a single member: While the different estimates produced by MC Dropout seem to highly correlate and to some degree be refined by more samples, they appear completely unrelated in the BAE case. This comes as a surprise given the theoretical justification for using BAE to estimate the predictive uncertainty in appendix A.2. Yet an amortization of the training loss was necessary to make training in the recurrent recoding context feasible, which appears to have a fatal effect on the quality of uncertainty estimates. It thus gives an explanation

Dropout	Perplexity
0.1	141.32 ± 4.83
0.2	140.37 ± 4.49
0.3	145.51 ± 4.97
0.4	139.14 ± 8.29
0.5	143.95 ± 4.17
0.6	143.25 ± 3.91

(a) Test perplexities.



(b) Uncertainty estimates.

Figure 5.4.: Results for different Dropout rates using MC Dropout recoding. (a) Perplexities on the test set. (b) Uncertainty estimates during a later stage in training. Curves denote the mean over $n = 4$ runs, with intervals signifying the standard deviation. Best viewed in color.

to the best score being achieved by a mere single ensemble member, corresponding simply to the entropy of the output distribution.

5.3.3 MC Dropout rate

Now we look at the impact of using different values for the MC Dropout rate. The test perplexities for different Dropout rates are given in table 5.4a. The results hint at the fact that the Dropout rate does not seem to produce significantly different outcomes for the options tested, a notion which is reinforced by the training and validation curves in fig. B.4 in the appendix.

More interesting however is to look at the uncertainty estimates for the same window during training as in the previous section, which is given in fig. 5.4b. Here we can clearly identify a relationship between the Dropout probability and the estimates of the predictive uncertainty: The lower the rate, the higher the uncertainty.

This interdependency can be explained by the following observation: An increased Dropout probability leads to more neurons whose activations become zero. As each neuron outputs the probability of a predicted word, averaging over K distributions with different masks, this will lower the probability of the same word when averaging samples. Ultimately, this leads to a flatter distribution (or increased uniformity) and therefore a higher predictive entropy.

Based on the results, it remains unclear how to select the “best” Dropout rate to estimate the predictive uncertainty. Gal and Ghahramani, 2016b note that for some of their experiments, using different Dropout rates the models actually converged

to the same uncertainty estimates over time. This cannot be said in our case, where the estimates differ in similar proportions even approaching the end of the training (not shown here). However, as the choice of Dropout rate does not seem to produce any significant differences, we proceed in the following experiments with the Dropout rates found during hyperparameter search. This value of 0.15 for the Language Model’s decoder is also in line with the experiments performed by Gal and Ghahramani, 2016b, who similarly choose small values for models of very limited size.

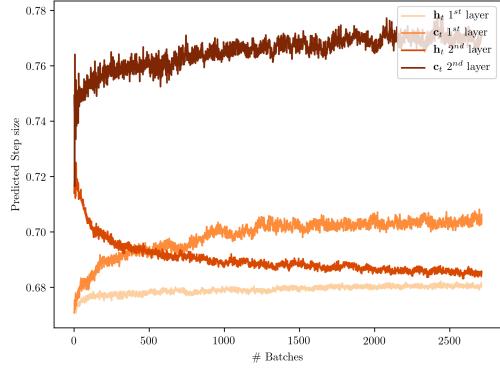
5.3.4 Dynamic Step Size

So far, we only considered a recoding step size α that stays fixed throughout training. However, as shown in theorem 1, a reduction of the error signal δ_t in the hidden activations \mathbf{h}_t is only guaranteed using the correct step size, which in turn depends on the error function’s Lipschitz constant L . This constant is not known, and although making the step size a learnable parameter is also not guaranteed to find the correct value, it does seem more promising than determining it manually (*learned step size*). Because we saw earlier that the recoding step size required varies highly depending on the model and should not be constrained to a certain range, regularization like weight decay - if applicable - is not applied to this parameter. Furthermore, the softplus function (eq. 3.5) is chosen to guarantee a positive step size.¹ We also consider the even more dynamic case described in chapter 4.4, where we parameterize the recoding step size at a time step t with a neural network with parameters \varkappa s.t. $\alpha_t = f_\varkappa(\mathbf{h}_t)$ (*predicted step size*). As before, the softplus function is used on the final output to restrict it to positive values. In this case, the predictor network was chosen to be a two-layer MLP with 300 and 100 hidden units, respectively.

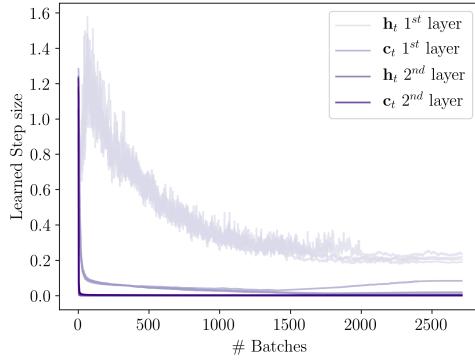
The results given in table 5.5a reveal the difficulty in determining the step size: No matter the model variant or manner to pick the recoding step, all models perform worse. Furthermore, all models disagree on the relative importance of correcting hidden activations: When predicting the update step with a MLP, the cherry-picked surprisal model depicted in fig. 5.5b places emphasis on recoding the cell states of the LSTM, preferring the upper layer, in contrast to the MCD and BAE models, which recode the hidden activations of the lowest layer the most and has other sizes tend close to 0. Curiously, the averaged step sizes - either learned or predicted - seem to be very similar in these two cases regardless of parameterization. This cannot be said for surprisal recoding, where in either cases both methods converge to distinct solutions in different runs (cp. with fig. B.5 in appendix). This seems to hint at a high variability in the training process of these recoding step parameters which does

¹It was observed in earlier experiments that applying a ReLU to the step size would tend to have the step size parameter dying off, resulting in a step size of 0 that would disable recoding and could not be recovered from.

Model	Step type	Perplexity	Gain
Baseline	-	126.60 ± 4.06	N/A
Surp.	Learned	126.25 ± 0.77	↓ -0.92
	Pred.	129.01 ± 2.42	↓ -3.68
MCD	Learned	140.50 ± 6.78	↓ -2.48
	Pred.	154.48 ± 14.32	↓ -16.46
BAE	Learned	142.20 ± 8.84	↓ -3.66
	Pred.	139.55 ± 6.30	↓ -1.01

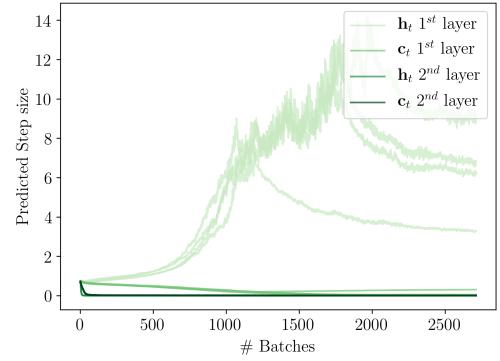


(a) Results for dynamic step sizes.



(c) Learned steps for $n = 4$ MCD models.

(b) Learned steps for single surprisal model.



(d) Predicted steps for $n = 4$ BAE models.

Figure 5.5.: Overview over the results with dynamic step size models. (a) Perplexities on the test set including the difference to best results for every model type so far. (b) - (d) Development of the average step size per batch for a certain layer and activation type during training for the three recoding models using either learned or predicted steps.

not find advantageous solutions and ultimately inhibits performance. This fragility could stem from multiple origins, including the flux of the error function surface the step size has to correspond to and the unregularized predictors. An overall reflection on the issue of step size is given later in chapter 7.1.

5.3.5 Ablation

After reviewing previous results, several questions arise: Why does surprisal encoding only perform marginally better? Can we improve an independently trained baseline model by adding the recoder? Does recoding using the model's approximate predictive entropy result in worse performance by virtue of the recoding mechanism or does training with the mechanism result in suboptimal weights?

I attempt to answer these questions by performing a series of ablation experiments: Here the original best evaluation scores per model type are compared against some modified instances, namely all recoding models *without* their respective recoders as

Model	Step	Modification	Result	Gain	Model	Step	Modification	Result	Gain
Surp.	Fixed		126.11 ± 1.38	↓ -0.78	BL	Fixed		125.95 ± 3.56	↑ +0.65
	Learned	w/o recoder	126.25 ± 0.77	→ ±0		Learned	w/ Surp. recoder	126.55 ± 3.63	↑ +0.05
	Pred.		129.01 ± 2.42	→ ±0		Pred.		126.59 ± 3.64	↑ +0.01
MCD	Fixed		138.01 ± 2.79	↑ +0.01	BL	Fixed		126.78 ± 3.67	↓ -0.18
	Learned	w/o recoder	145.74 ± 8.95	↓ -5.24		Learned	w/ MCD recoder	127.04 ± 4.11	↓ -0.44
	Pred.		159.60 ± 10.93	↓ -5.12		Pred.		127.11 ± 4.14	↓ -0.51
BAE	Fixed		138.54 ± 4.24	→ ±0	BL	Fixed		126.72 ± 3.65	↓ -1.12
	Learned	w/o recoder	142.36 ± 7.76	↓ -0.16		Learned	w/ BAE recoder	126.72 ± 4.22	↓ -1.12
	Pred.		140.51 ± 5.48	↓ -0.96		Pred.		126.72 ± 4.22	↓ -1.12

(a) Models without recoder.

(b) Baseline model with recoder.

Figure 5.6.: Test perplexity on the test set for different models using components from the baseline / recoding models. (a) Recoding models trained with the recoder but performing without it during testing. (b) The baseline model equipped with the recoding mechanisms of other models. Gains are calculated relative to the best results of this model type so far (i.e. Surprisal / MCD / BAE for (a) and the baseline for (b)).

well as the baselines *adding* the different mechanisms on top. All of this is solely performed during testing, using the models trained in earlier sections. The results of these experiments are given in fig. 5.6, where the unmodified models are used for comparison.

Table 5.6a brings an elusive interdependency between the recoder and the underlying model to light: In almost all cases, removing the recoder results in worse or the same performance. Yet the discrepancy does not appear as big as one might expect. This seems to suggest that the recoder loses some of its importance on the network’s prediction as the model matures during training and that the underlying weights adapt to the mechanism. Moreover, the resulting models perform worse than the baseline (126.60) in the case of MCD and BAE recoding, from which we can conclude that these methods actually have an adverse impact on the learning of the model weights. Only in the case of surprisal recoding, the models still perform better, although statistically non-significant.

The notion that recoding based on predictive uncertainty has a negative effect on performance is further reinforced by table 5.6b: Using the corresponding recoding mechanisms with the baseline model results in a small drop in performance. Surprisal recoding also produces lower test perplexities here, yet minuscule in nature. To get a better understanding of the reasons behind these observations, the next chapter makes some observations about the models on a smaller scale.

Qualitative Analysis

Last chapter has provided a comprehensive comparison between the baseline and all proposed model variants in terms of test perplexity and other metrics on a corpus-level. However, this singular perspective fails to give insight into the core differences between the models: Because we would like to gain more introspection into these models to explain their differences, this chapter contains several experiments highlighting key model traits on a sentence or even word level by performing qualitative analyses. To this end, section 6.1 provides a check of cognitive plausibility according to the motivation layed on in the introduction. It furthermore contains an empirical validation of theorems 1 and 2 by measuring the amount of reduction in error signal recoding produces in section 6.1.2 as well as an error analysis. Section 6.1.3 tries to elucidate the curious performance gap between the baseline and models using predictive entropy as their error signal.

6.1 Recoding Behaviour

To analyze model behavior, we can track certain quantities on a word-to-word basis: One of them is the *surprisal* of the Language Model. This approach has already been employed by other works to compare the surprisal of Language Models to human reading times (Van Schijndel and Linzen, 2018) as a proxy for the difficulty of a sentence. However, this surprisal score differs from the recoding signal of the same name defined in chapter 4.3.2, as it only consists of the base 2 log probability of the word $-\log_2(w_t|w_1^{t-1}) \equiv -\log_2(\mathbf{o}_{tc})$, where c corresponds to the entry of word w_{t+1} in the model's vocabulary. Also, we can track the recoding error signal δ_t emitted at some time step t . Besides these two pieces of information, we also record the surprisal of the model based on the recoded hidden activations $-\log_2(\mathbf{o}'_{tc})$, i.e. after re-computing the output distribution using \mathbf{h}'_t , along with the resulting new error signal δ'_t . The latter is not actually used for any computations, it just serves as a mean to disclose the impact of recoding.

Finally, experiments are conducted in a fashion that allows recoding only to be performed at specific time steps, which are appropriately marked or explained in the caption. This way, we can isolate the effect of single recoding actions and observe their side effects on the model later during a sequence.

6.1.1 Cognitive Plausibility

One motivation for the approach taken lies in the emulation of human behavior when processing challenging sentences. To test the cognitive resemblance of the new augmented model, a small corpus of 20 garden path sentences is used that were collected from multiple online sources as well as 64 sentence from the work of Sturt et al., 1999, resulting in a total of 84 sentences containing difficult structural ambiguities. Similar to the infamous “The horse raced past the barn fell” example mentioned in chapter 1, these sentences evoke a feeling of surprise when encountering a new pivotal word which does not correspond to the parse built up by the reader thus far. Accordingly, the sentence representation has to be adjusted, which was observed to lead to an increase in reading time among human participants (Milne, 1982). Accordingly, if the model is (less) susceptible to garden path effects, we can observe this by its surprisal scores.

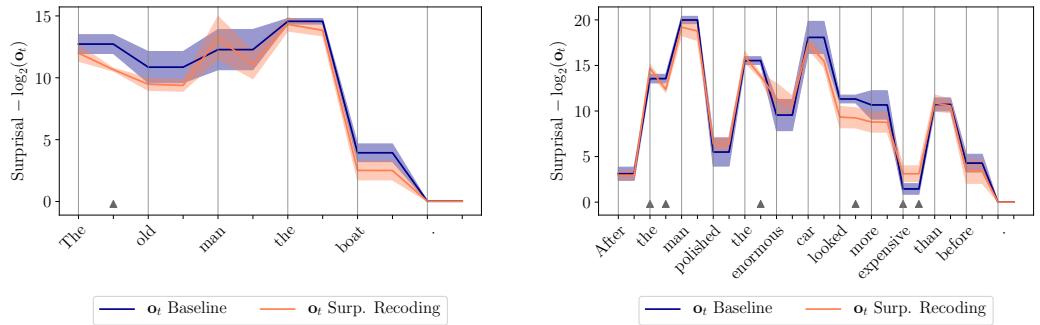


Figure 6.1.: Surprisal scores of the baseline and a surprisal recoding model on two garden path sentences. Curves denote the mean of $n = 4$ runs, while the intervals depict one standard deviation. In case of the recoding models, the surprisal score based on the re-decoded output distribution is shown *between* two time steps. Gray markers denote statistically significant differences (two-tailed Student’s t-test, $p = 0.05$). Best viewed in color.

In our case the processing time of new inputs is expected to remain nearly constant and does not depend on some notion of “difficulty”; instead we would expect to see an increase in the surprisal score and / or error signal δ_t when the model is confronted with an unexpected token, followed by a decrease thereof as a consequence of the recoding update step. To test these assumptions, fig. 6.1 illustrates the behavior of the baseline model compared to the best found surprisal-based recoding model (using a fixed step size of 5) and depicts the surprisal scores over the course of a sentence, using samples from the garden path corpus.

Here we can make the following observations: Both models highly correlate in their surprisal, while this specific recoding model (orange curve) achieves slightly lower

scores than the baseline (blue curve) most of the time. Furthermore, we can see in this instance that the surprisal scores improve or stay the same after recoding (data points between words), often in a statistically significant manner (gray triangular markers). There are examples where the baseline model produces lower scores (right plot, “expensive”), yet these are far outnumbered by the reverse case in which the recoding model succeeds. Using the more dynamic step size approaches also did not influence these observations (not shown here).

However, all tested models only show a limited degree of cognitive plausibility. While it is difficult to directly contrast these results with human reading time¹, we would intuitively assume for the model’s surprisal score to spike once they encounter the first word contradicting the sentences’ first parse assumed by a human. In the first example, this is somewhat the case, as the models display higher surprisal when processing “the”, which changes the preceding “man” from being the subject of the sentences with the attribute “old” to the former being the main verb and the latter to the subject (as in “the old [people]”). In the second more complex sentence, “looked” gives the decisive clue that “the enormous car” is not the object of the construction “the man polished” but indeed the subject of the main clause. Even though humans would be expected to take some more time to resolve this new-found ambiguity, the models actually emit *lower* surprisal scores.

Other works have already pointed out some limits of relating the results of Language Models on garden path sentences to processes in the human cognitive system: Van Schijndel and Linzen, 2018 find that the reading time estimates produced by LSTMs depend on the amount of training data and are incapable of completely modeling the processing time required by people. A similar observation is noted by Futrell et al., 2019, because their model trained on PTB displays significantly smaller garden path effects than instances trained on larger data sets. Overall, we can conclude that the similarity of the models with respect to human processing is limited in two regards: First, by the syntactic abilities of LSTMs (LSTMs were shown to learn a syntactic state - as described e.g. in chapter 2.1 - but it might insufficient) and secondly the grammatical knowledge induced simply by the amount of training data, which is small compared to most other Deep Learning data sets.

6.1.2 Error signal reduction

Now let us focus entirely on the error signal δ_t . By plotting its development during inference time several points about the methodology of this work can be verified: On one hand, that recoding under the right choice of step size reduces the error

¹No direct reading time data is available and the setup often differs in the human case, i.e. people reading the sentences in chunks instead of a word-by-word basis.

signal δ_t (cp. theorem 1) and the error signal at a later time step (cp. theorem 2), even when no additional recoding is used. On the other hand, this likewise gives an opportunity to analyze whether a suboptimal choice of step size actually leads to an *increase* of the error signal, breaking the assumptions of both theorems with respect to the error function's Lipschitz constant.

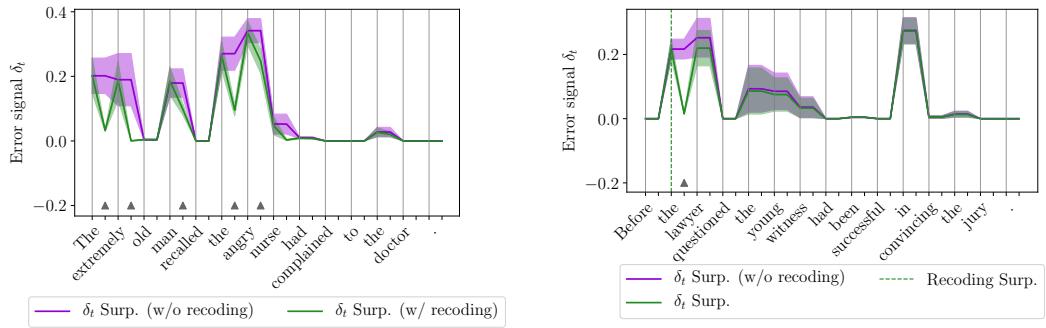


Figure 6.2.: Error reduction through recoding on garden path sentences for the surprisal-based recoding model with fixed step size for all time steps (left) and a single time step (right; dashed line). Curves denote the mean of $n = 4$ runs, while the intervals depict one standard deviation. Gray markers denote statistically significant differences (two-tailed Student's t-test, $p = 0.05$).

The following analysis focuses on the best observed surprisal-based recoding model.² In the left plot of figure 6.2, it can be observed that recoding indeed consistently reduced the error signal compared to the same model where the signal is recoded, but the recoding update step not performed. Although not impossible, it was also not observed in the given data set that recoding *increased* the error signal, it at most stagnates after the activation adaption. In the right plot, another sentence is used to illustrate the effect of only recoding at time step $t = 2$: Because recoding only takes place once, the difference between the surprisal scores fades over time, however, an offset is still noticeable even after the time step in question, which is likely to be explained by the findings of theorem 2. In many cases however, the effect afterwards is minimal or not noticeable, which can also be explained by the same theorem, which showed that the possible improvement becomes smaller by distance to the modification made by recoding. Lemon-picked examples of this outcome and more positive samples of both analyses are given in appendix B.6.

6.1.3 Effect of overconfident predictions

In the previous chapter, we observed that the recoding models using approximate predictive entropy as their error signals actually performed significantly *worse* than

²Due to the stochasticity involved in MCD a fair comparison of error signals for the same sequence cannot be guaranteed and the explanatory power of BAE results is questionable.

the baseline. The ablation study in chapter 5.3.5 revealed that this can at least partially be attributed to the fact that using those recoding mechanisms produces suboptimal weights (even without the mechanism, these models performed worse than the baseline). Nevertheless, another suspected reason for this discrepancy is that the confidence of the model approximated by MCD and BAE might lead to overconfident *mis*-predictions, leading the model astray after the correction step. In contrast to surprisal recoding, these two variants do not have knowledge about the gold token and only incentivize the model to concentrate their probability around their most probable predictions following the update step. To investigate this suspicion, some plots showing the development of the error signal jointly with the model surprisal scores compared to the baseline in figure 6.3. Because the range of estimated predictive entropy scores is much lower on the garden path data set than on PTB, the fixed step = 0.1 MCD recoding model from chapter 5.3.1 is reused.

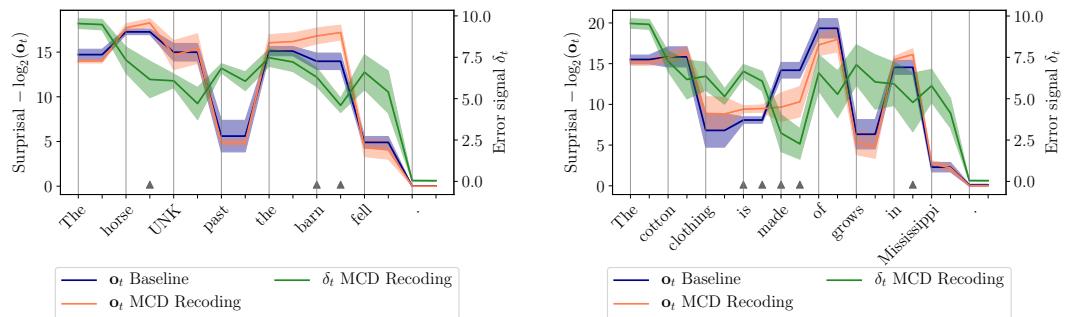


Figure 6.3.: The effect of uncertainty recoding with MCD on surprisal, using a fixed step size of 5 and $K = 10$ samples. Curves denote the mean of $n = 4$ runs, while the intervals depict one standard deviation. Gray markers denote statistically significant differences (two-tailed Student's t-test, $p = 0.05$).

Two observations can be made from both plots in fig. 6.3: First, we can see that recoding lowers the error signal δ_t , validating theorem 1 for this approach as well. In spite of this, the surprisal scores are **not** necessarily lowered by this. In the right plot, it does happen for the time step of the word “grow”, but in many other instances for the shown samples, the opposite occurs: In the left plot, recoding the activations at the time step of “horse” lowers predictive entropy but increases surprisal such that the gap to the baseline’s score becomes statistically significant. In the case of “barn” for the same sentence, the difference is exacerbated even further. This provides some evidence to the hypothesis that this behavior might contribute to the worse performance of MCD and BAE compared to the baseline. However, it is unclear whether this is an additional factor to the suboptimal weights identified in chapter 5.3.5 or actually the culprit behind this very same observation.

Discussion

"The more [the scientist] analyzes the universe into infinitesimals, the more things he finds to classify, and the more he perceives the relativity of all classification. What he does not know seems to increase in geometric progression to what he knows. "

The Wisdom of Insecurity - Alan W. Watts

The previous two chapters have tried to analyze the models and their operation from many angles. But even with all the experiments that have been conducted, this work cannot answer all questions that might be raised on the basis of the supplied results conclusively. Therefore I aggregate all the findings to far and try to distill some general conclusions from them while also pointing out some of the limits identified thus far.

7.1 Step Size

Many of the experiments in chapter 5 were dedicated to determine the effect of step size on the recoding models. The step seems like one of the core pieces of recoding, as an improvement of the error signal can only guaranteed using the right size under theorem 1. The results show that an advantageous choice of step size heavily depends on the value range of the error signal and possibly also on the data set, as e.g. uncertainty estimates in chapter 6.1.3 on the garden path data set differed from PTB. It also seems like a simplification to assume that all layers and activations types require the same constant step size at every point (the magnitude of correction by recoding probably should depend on the circumstances of the input).

To this end, it appears like a straightforward choice to parameterize the step size as a single learned parameter or by some secondary model, however the results from corresponding experiments in chapter 5.3.4 were inconclusive. For different models and different error signals, found solutions preferred to correct activations in an inconsistent hierarchical order when sorted by size. It is not certain which type of activation has to be recoded most to achieve the best improvement. Besides, the resulting models performed worse than their counterparts using constant update steps. It hence remains unclear to which extent a constant step size and having the magnitude of the recoding solely determined by the strength of the recoding

gradient $\nabla_{\mathbf{h}_t} \delta_t$ suffices. Perhaps the learning of step sizes requires more exploration of regularization and hyperparameter search in the predicted case which lie outside the scope of this work.

7.2 Role of Error Signal

The other moving part of the recoding framework is the choice of error signal δ_t , which also provides many points for discussion.

The key problem seems to be to identify an error signal that improves the objective, i.e. is “conjugate”. This cannot be said for predictive entropy: Both quantitative and qualitative analysis in the two previous chapters have shown that these approaches, at least using MCD and BAE, result in worse weights and confident mispredictions, resulting in higher perplexity than the baseline. These results underscore especially the difficulty of using unsupervised signals. In the original case of interventions (Giulianelli et al., 2018), replication experiments conducted in the context of this work showed that these corrections also had a statistically non-significant effect on perplexity. Even other works that improved drastically on the results of the number prediction task could not identify improvements in perplexity (Merrill et al., 2019), which seems to suggest that this information does not improve the model’s performance, either.

Even in the case of surprisal, which seems directly related to the cross-entropy objective as it involves the probability of the target token, only achieved minor improvements on the baseline in the best case. This could be explained by two reasons: Either the network does not profit from being supplied with the same learning signal twice (gold token probability during recoding **and** for the loss calculation) or the main source of error in a recurrent setting is not remedied by recoding. The process may correct corrupted information about the right prediction encoded in the hidden activations and therefore rectify the horizontal information flow in the RNN, but it does not influence any errors committed by the decoding layer of the LM itself or the insufficient encoding of the current input w_t by the embeddings and inside the hidden layer. That this vertical direction could be the source of many other model errors might explain the limited improvement attained by the best recoding model.

7.3 Approximations & Simplifying Assumptions

Moreover, I want to discuss the influence of approximations and simplifications on the results obtained in previous chapters.

Approximation Quality of Predictive Entropy In the case of MCD recoding, the quality of the approximation of the model’s predictive entropy relies on the number of samples and the MC Dropout rate. Although both factors seemed to not impact the performance to strongly (cp. chapters 5.3.2 and 5.3.3), the results should be taken with a grain of salt, as the sample size of $n = 4$ for different models runs was fairly small, although necessary due to enormous amount of models that had to be trained across all experiments. It has further been shown in chapter 5.3.2 that anchored ensembles do not produce correct uncertainty estimates when using the amortized training procedure. This simplification was paramount to make training of a decoder ensemble tractable in a recurrent setting; otherwise K different weight gradients have to be computed at the end of every batch, which backpropagate back through the entire sequence. As the experiments elucidated, this produces inconsistent estimates and seems to break the theoretical guarantee layed out in appendix A.2, which might explain the loss in performance compared to MC Dropout recoding. Recently, a faster method for ensembling derived from the topology of loss surfaces has been proposed (Garipov et al., 2018), but lacks a Bayesian foundation and an extension to RNNs. Until a scalable, Bayesian ensembling method in a recurrent setting is developed, this approach seems to be deemed to be outperformed by the other recoding model variants.

LM Decoders as Independent Models In order to transfer the ideas of MC Dropout and Bayesian Anchored Ensembling that apply to MLPs to a RNN setting, the simplifying assumptions was made to see the decoder layer of the Language Model as a sort of “mini-MLP”. While this allowed for an application of these techniques, it essentially results in only modeling the uncertainty of the decoder at a certain time step. For a recurrent model, we would like to model the uncertainty of the *whole model*, which depends on previous time steps. Gal and Ghahramani, 2016a provides an adaptation of MC Dropout to RNNs, yet preliminary experiments showed that this holistic solution is problematic in the recoding framework for two reasons: Ensembling multiple RNNs using distinct sets of dropout masks for every type of connection is memory-intense and becomes computationally prohibitive in connection with additional gradient calculations at every time step.

Conclusion

8

沧海桑田 - The blue sea turned into mulberry fields

Chinese idiom about the transformative changes in the world

The recoding framework proposed in this thesis comes with several theoretical guarantees and an appealing intuition. In spite of this, experimental results have shown that there are several caveats to the approach which make the application in practice challenging: The computation of the gradient imposes additional computational costs which are not justified by the results and only yield minor improvements in the best case. They also do not improve on the cognitive plausibility of the model. This shows that the most simple instantiation of the recoder is not sufficient to deliver on its promises. Nevertheless, this opens up further lines of research which are reflected on in the next section. The last part of this chapter, section 8.2, is withal dedicated to put this work into the greater context of current AI research trends.

8.1 Future Work

Nonwithstanding the analyses in previous chapters, the results should not be seen as the method's eulogy but as a first step that paves the road to future research ideas, some of which will be discussed here. The following sections hence deal with the issues of recoding signals (§8.1.1), step size (§8.1.2), efficiency (§8.1.3) and possible modifications of training and architecture (§8.1.4).

8.1.1 Recoding signals

This work explored three different kinds of recoding signals. While these were not able to produce any substantial improvements compared to the baseline, this does not have to be the case with other choices. Chapter 7.2 discussed the shortcomings of the explored options (surprisal, predictive entropy). Other error signals could be tested on a toy data set or proven mathematically to improve the objective function.¹ But it is also imaginable to use recoding to maintain certain desirable properties in the hidden activations - after all, the loss function only serves as proxy for the *real objective*, similar to how Neural Machine Translation models might be

¹This could e.g. be achieved by proving that there is a constant linear relationship between the error signals and the loss function - therefore a reduction of the error signal like derived in theorem 1 will linearly decrease the loss by the same factor.

trained using cross-entropy loss but are actually supposed to optimize BLEU or another, similar metric. Recoding could help here to retain certain linguistic features that are not considered during optimization but that improve some property of the translation. Apart from additional supervision, there is also a large space of unsupervised signals to explore, although my experiments have shown that this can turn out to be challenging.

8.1.2 Step size

Other research work could focus on the issue of step size by exploring other choices of parameterization and perform more expressive studies about the issue. In this work, one choice tried to predict the step size based on the current hidden activations. Other models could try to find a better solution by basing the prediction on the output activations or completely different features.

The step size could also be kept constant relative to the properties to the model. The improvement of the error signal is only guaranteed by the choice of the smallest Lipschitz constant L of the error function that produces the signal. This function maps the hidden activations to some signal and is parameterized by the decoder layer weights. Although the Lipschitz value of this function changes every time the parameters are updated, it is conceivable to compute this quantity to avoid misguided updates. Albeit recent work has shown that computing the exact Lipschitz constant is a NP-hard problem even in shallow neural networks, multiple other efforts have been dedicated to approximating it (Virmaux and Scaman, 2018; Oberman and Calder, 2018; Gouk et al., 2018). In this way, improvement would be guaranteed (up to the precision of the approximation) and the magnitude of the correction solely dependent on the recoding gradient.

8.1.3 Efficiency

One major drawback of the proposed method seems to lie in its efficiency. In table 5.2 it is noted that the models using predictive entropy as their recoding signal process inputs dramatically slower than the baseline. The disparity is further exacerbated when increasing the sample / ensemble size of the models. But even for the surprisal recoding model, a significant slow-down is observed based on the additional backward passes: Even though automatic differentiation engines like used in PyTorch are heavily optimized, the gradient calculation takes up a large portion of the training time. The recoder enlarges that portion, where the increase grows linearly with sequence length and network depth.

Based on the increasing resource requirements for state-of-the-art models, a recent position paper (Schwartz et al., 2019) has advocated to make efficiency an additional evaluation criterion of models. From this point of view, future research should evaluate whether the current model formulation could be improved by e.g. approximating the recoding gradient with the finite difference method. This has already successfully been applied in other approaches with gradient-heavy computations like differentiable architecture search (Liu et al., 2019).

8.1.4 Training Procedure & Architecture

In the tested models, the recoder was combined with the LSTM Language Model and jointly trained in the most simple way. To reduce the computational load, it could be sensible to integrate the recoder as some sort of special gate that is only used in specific circumstances. In this thesis, only modifications of the current hidden state were discussed; yet the recoder could also be extended to recode activations further back in the sequence, although this likely to result in additional computational costs. Inspiration for a solution could possibly be drawn from the work from Ke et al., 2018, who use a self-attention mechanism for sparse credit assignment instead of regular backpropagation through time to direct the gradient flow.

Given that joint training also seemed to produce suboptimal weights in the case of predictive entropy recoding, it could also be imagined to train the model using a variation of *teacher forcing* like commonly used in Neural Machine Translation, where the recoder is only used in a certain percentage of batches or time steps. Another possibility is to train the base model independently in some pretraining phase and add the recoder afterwards or exclusively during inference (the ablation study in chapter 5.3.5 showed that adding the surprisal recoder to the baseline during inference resulted in small improvements).

8.2 Outlook

This work is only a small piece in an incomprehensibly big landscape of Artificial Intelligence and Machine Learning research. As such, I would like to dedicate the last part of this thesis to embed it into a larger frame of reference, which points out possible research directions happening on a larger scale.

8.2.1 Innovation over Scale

Taking the position of a devil's advocate, someone might argue: "What is the value of models that do not outperform the current state-of-the-art?" or "Why try x when model y already produces very good results on this task?". Indeed, especially in

Natural Language Processing a line of recent works based on the transformer architecture by Vaswani et al., 2017 has established new state-of-the-art performances on Language Modeling.

These include Google’s BERT (Devlin et al., 2019), OpenAI’s GPT-2 (Radford et al., 2019) and XLNet (Yang et al., 2019). On the flip side, these models require huge amounts of data and computational resources. The training of XLNet has been estimated to have cost about than 61.000 \$.² This has several, worrying implications: First, with these resource requirements, scientific papers become hard to reproduce. These costs only allow training of these models in the context of well-funded institutions, namely top-tier universities and affluent tech giants. Secondly, the reliance on large-scale hardware produces a high electricity consumption along with a worrisome carbon footprint, which bears a certain irony: These models try to (loosely) imitate the human brain, a biological computer that is actually very energy efficient (Schwartz et al., 2019). Lastly, scaling up data sets and the number of parameters does not necessarily increase the semblance to human cognition: A recent paper found that BERT’s almost human-level performance in an argument reasoning Comprehension task was due to spurious statistical features in the data set (Niven and Kao, 2019).

All these reasons are reiterated to make the following point: While these models have tremendously propelled research and helped establish new cutting-edge results on different tasks, squeezing out a few more performance points by scaling up should not be the singular objective of research. Instead, there is merit in experimenting with completely different architectures - like attempted in this work. This might produce innovations to help make models process data better and use their parameters more efficiently, which in turn can thus produce more biologically plausible and energy-efficient models. And to get there, we can follow a model that already displays the desired behavior.

8.2.2 Inspiration from Human Cognition

The human brain is a marvelous object. It comprises 86 billion neurons (Azevedo et al., 2009), is able to perform all its complex computations while only consuming around 12.6 watts of energy per day.³ While scientists understand more and more about its modus operandi, their insights have provided a continuous stream of inspiration for AI research and vice versa, making the two fields historically entangled: From the beginnings of Deep and Reinforcement Learning to many current ideas

²Source: <https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/> (04.08.19).

³Source: <https://www.scientificamerican.com/article/thinking-hard-calories/> (04.08.19).

build on these insights, an overview of which is given in Hassabis et al., 2017. Given the line of reasoning induced in the previous section, I would like to end my thesis on the following note: Notwithstanding the rapid development of and impressive milestones achieved by Artificial Intelligence in recent years, the field has still a long way to go to actually achieve human-level intelligence. It was advocated to bridge this gap by equipping models with some of the innate cognitive properties of humans (Lake et al., 2017). And although it is undeniable that moving away from the biological example will always yield important results for research as well, this work should be contextualized as one small attempt to make progress by drawing inspiration from the one thing that we know proven to be a working example of general intelligence: Ourselves.

Bibliography

- Abney, Steven (2007). *Semisupervised learning for computational linguistics*. Chapman and Hall/CRC (cit. on p. 17).
- Achille, Alessandro, Tom Eccles, Loic Matthey, et al. (2018). „Life-long disentangled representation learning with cross-domain latent homologies“. In: *Advances in Neural Information Processing Systems*, pp. 9873–9883 (cit. on p. 23).
- Azevedo, Frederico AC, Ludmila RB Carvalho, Lea T Grinberg, et al. (2009). „Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain“. In: *Journal of Comparative Neurology* 513.5, pp. 532–541 (cit. on p. 70).
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). „Neural Machine Translation by Jointly Learning to Align and Translate“. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (cit. on p. 22).
- Battaglia, Peter W., Jessica B. Hamrick, Victor Bapst, et al. (2018). „Relational inductive biases, deep learning, and graph networks“. In: *CoRR* abs/1806.01261. arXiv: 1806 . 01261 (cit. on p. 28).
- Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin (2003). „A neural probabilistic language model“. In: *Journal of machine learning research* 3.Feb, pp. 1137–1155 (cit. on p. 17).
- Bengio, Yoshua, Ian J Goodfellow, and Aaron Courville (2015). „Deep learning“. In: *Nature* 521.7553, pp. 436–444 (cit. on pp. 17, 18).
- Bergstra, James and Yoshua Bengio (2012). „Random search for hyper-parameter optimization“. In: *Journal of Machine Learning Research* 13.Feb, pp. 281–305 (cit. on pp. 49, 92).
- Bernardy, Jean-Philippe (2018). „Can Recurrent Neural Networks Learn Nested Recursion?“ In: *LiLT (Linguistic Issues in Language Technology)* 16.1 (cit. on pp. 21, 30).
- Bishop, Christopher M (2006). *Pattern recognition and machine learning*. Springer (cit. on p. 27).
- Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015). „Weight Uncertainty in Neural Network“. In: *International Conference on Machine Learning*, pp. 1613–1622 (cit. on pp. 22, 31, 33).
- Brundage, Miles, Shahar Avin, Jack Clark, et al. (2018). „The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation“. In: *CoRR* abs/1802.07228. arXiv: 1802 . 07228 (cit. on p. 22).

- Buchanan, Bruce G (2005). „A (very) brief history of artificial intelligence“. In: *Ai Magazine* 26.4, pp. 53–53 (cit. on p. 17).
- Cheng, Xi, Bohdan Khomtchouk, Norman Matloff, and Pete Mohanty (2018). „Polynomial Regression As an Alternative to Neural Nets“. In: *CoRR* abs/1806.06850. arXiv: 1806.06850 (cit. on p. 81).
- Chomsky, Noam (1956). „Three models for the description of language“. In: *IRE Transactions on information theory* 2.3, pp. 113–124 (cit. on p. 16).
- Christiansen, Morten H and Nick Chater (2016). „The Now-or-Never bottleneck: A fundamental constraint on language“. In: *Behavioral and Brain Sciences* 39 (cit. on p. 17).
- Church, Kenneth Ward (1989). „A stochastic parts program and noun phrase parser for unrestricted text“. In: *International Conference on Acoustics, Speech, and Signal Processing*, IEEE, pp. 695–698 (cit. on p. 17).
- Conneau, Alexis, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni (2018). „What you can cram into a single \\$&!#* vector: Probing sentence embeddings for linguistic properties“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 2126–2136 (cit. on pp. 22, 38).
- Csáji, Balázs Csanád (2001). „Approximation with artificial neural networks“. In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24, p. 48 (cit. on p. 25).
- Dai, Zihang, Zhilin Yang, Yiming Yang, et al. (2019). „Transformer-XL: Attentive Language Models beyond a Fixed-Length Context“. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 2978–2988 (cit. on p. 22).
- Dalvi, Fahim, Nadir Durrani, Hassan Sajjad, and Stephan Vogel (2018). „Incremental Decoding and Training Methods for Simultaneous Translation in Neural Machine Translation“. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pp. 493–499 (cit. on pp. 22, 38).
- Dauphin, Yann N, Razvan Pascanu, Caglar Gulcehre, et al. (2014). „Identifying and attacking the saddle point problem in high-dimensional non-convex optimization“. In: *Advances in neural information processing systems*, pp. 2933–2941 (cit. on p. 27).
- DeRose, Steven J (1988). „Grammatical category disambiguation by statistical optimization“. In: *Computational linguistics* 14.1, pp. 31–39 (cit. on p. 17).
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186 (cit. on p. 70).
- Edunov, Sergey, Myle Ott, Michael Auli, and David Grangier (2018). „Understanding Back-Translation at Scale“. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 489–500 (cit. on p. 25).

- Evans, Nicholas and Stephen C Levinson (2009). „The myth of language universals: Language diversity and its importance for cognitive science“. In: *Behavioral and brain sciences* 32.5, pp. 429–448 (cit. on p. 16).
- Evans, R, J Jumper, J Kirkpatrick, et al. (2018). „De novo structure prediction with deeplearning based scoring“. In: *Annu Rev Biochem* 77, pp. 363–382 (cit. on p. 25).
- Everett, Daniel, B Berlin, MA Goncalves, et al. (2004). „L. 2005. Cultural constraints on grammar and cognition in Pirahã“. In: *Current Anthropology* 46.4 (cit. on p. 16).
- Fernandez, A Graves M Liwicki S, R Bertolami H Bunke, and J Schmidhuber (2009). „A novel connectionist system for improved unconstrained handwriting recognition“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5 (cit. on p. 21).
- Futrell, Richard, Ethan Wilcox, Takashi Morita, et al. (2019). „Neural language models as psycholinguistic subjects: Representations of syntactic state“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 32–42 (cit. on p. 61).
- Gal, Yarin (2016). „Uncertainty in deep learning“. PhD thesis. PhD thesis, University of Cambridge (cit. on pp. 35, 80).
- Gal, Yarin and Zoubin Ghahramani (2016a). „A theoretically grounded application of dropout in recurrent neural networks“. In: *Advances in neural information processing systems*, pp. 1019–1027 (cit. on pp. 3, 19, 23, 34, 43, 50, 66).
- (2016b). „Dropout as a Bayesian approximation: Representing model uncertainty in deep learning“. In: *International Conference on Machine Learning*, pp. 1050–1059 (cit. on pp. 3, 19, 23, 33, 43, 55, 56, 81).
- Garipov, Timur, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson (2018). „Loss surfaces, mode connectivity, and fast ensembling of dnns“. In: *Advances in Neural Information Processing Systems*, pp. 8789–8798 (cit. on p. 66).
- Gers, Felix A and E Schmidhuber (2001). „LSTM recurrent networks learn simple context-free and context-sensitive languages“. In: *IEEE Transactions on Neural Networks* 12.6, pp. 1333–1340 (cit. on p. 21).
- Ginsburg, Seymour (1966). *The Mathematical Theory of Context Free Languages.* [Mit Fig.] McGraw-Hill Book Company (cit. on p. 16).
- Giulianelli, Mario, Jack Harding, Florian Mohnert, Dieuwke Hupkes, and Willem Zuidema (2018). „Under the Hood: Using Diagnostic Classifiers to Investigate and Improve how Language Models Track Agreement Information“. In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 240–248 (cit. on pp. 3, 18, 19, 22, 38, 39, 41, 46, 65).
- Gong, Chengyue, Di He, Xu Tan, et al. (2018). „FRAGE: frequency-agnostic word representation“. In: *Advances in Neural Information Processing Systems*, pp. 1334–1345 (cit. on pp. 18, 22).
- Gouk, Henry, Eibe Frank, Bernhard Pfahringer, and Michael J Cree (2018). „Regularisation of Neural Networks by Enforcing Lipschitz Continuity“. In: *stat* 1050, p. 14 (cit. on p. 68).

- Grice, H Paul, Peter Cole, Jerry L Morgan, et al. (1975). „Logic and conversation“. In: 1975, pp. 41–58 (cit. on p. 16).
- Gulordava, Kristina, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni (2018). „Colorless Green Recurrent Networks Dream Hierarchically“. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pp. 1195–1205 (cit. on pp. 21, 22, 30, 38, 49, 92).
- Hassabis, Demis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick (2017). „Neuroscience-inspired artificial intelligence“. In: *Neuron* 95.2, pp. 245–258 (cit. on p. 71).
- Hauser, Marc D, Noam Chomsky, and W Tecumseh Fitch (2002). „The faculty of language: what is it, who has it, and how did it evolve?“ In: *science* 298.5598, pp. 1569–1579 (cit. on p. 16).
- Hochreiter, Sepp, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies* (cit. on p. 29).
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). „Long short-term memory“. In: *Neural computation* 9.8, pp. 1735–1780 (cit. on pp. 21, 29).
- Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). „Multilayer feedforward networks are universal approximators“. In: *Neural networks* 2.5, pp. 359–366 (cit. on p. 25).
- Howard, Jeremy and Sebastian Ruder (2018). „Universal Language Model Fine-tuning for Text Classification“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 328–339 (cit. on p. 18).
- Hupkes, Dieuwke, Sara Veldhoen, and Willem Zuidema (2018). „Visualisation and’diagnostic classifiers’ reveal how recurrent and recursive neural networks process hierarchical structure“. In: *Journal of Artificial Intelligence Research* 61, pp. 907–926 (cit. on pp. 3, 22, 38).
- Jumelet, Jaap and Dieuwke Hupkes (2018). „Do Language Models Understand Anything? On the Ability of LSTMs to Understand Negative Polarity Items“. In: *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pp. 222–231 (cit. on pp. 21, 30).
- Jurafsky, Dan (2000). *Speech & language processing*. Pearson Education India (cit. on p. 17).
- Kaiser, Lukasz, Ofir Nachum, Aurko Roy, and Samy Bengio (2017). „Learning to Remember Rare Events“. In: *CoRR* abs/1703.03129. arXiv: 1703.03129 (cit. on p. 23).
- Ke, Nan Rosemary, Anirudh Goyal ALIAS PARTH GOYAL, Olexa Bilaniuk, et al. (2018). „Sparse attentive backtracking: Temporal credit assignment through reminding“. In: *Advances in Neural Information Processing Systems*, pp. 7640–7651 (cit. on p. 69).
- Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, et al. (2017). „Overcoming catastrophic forgetting in neural networks“. In: *Proceedings of the national academy of sciences* 114.13, pp. 3521–3526 (cit. on p. 23).

- Koenig, Nathan and Maja J Matarić (2017). „Robot life-long task learning from human demonstrations: a Bayesian approach“. In: *Autonomous Robots* 41.5, pp. 1173–1188 (cit. on p. 23).
- Krause, Ben, Emmanuel Kahembwe, Iain Murray, and Steve Renals (2018). „Dynamic Evaluation of Neural Sequence Models“. In: *International Conference on Machine Learning*, pp. 2771–2780 (cit. on p. 23).
- Lake, Brenden M, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman (2017). „Building machines that learn and think like people“. In: *Behavioral and brain sciences* 40 (cit. on p. 71).
- Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). „Simple and scalable predictive uncertainty estimation using deep ensembles“. In: *Advances in Neural Information Processing Systems*, pp. 6402–6413 (cit. on pp. 23, 35).
- Li, Hao, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein (2018). „Visualizing the loss landscape of neural nets“. In: *Advances in Neural Information Processing Systems*, pp. 6389–6399 (cit. on p. 27).
- Linzen, Tal, Emmanuel Dupoux, and Yoav Goldberg (2016). „Assessing the ability of LSTMs to learn syntax-sensitive dependencies“. In: *Transactions of the Association for Computational Linguistics* 4, pp. 521–535 (cit. on pp. 21, 30).
- Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2019). „DARTS: Differentiable Architecture Search“. In: *International Conference on Learning Representations* (cit. on p. 69).
- Liu, Nelson F., Omer Levy, Roy Schwartz, Chenhao Tan, and Noah A. Smith (2018). „LSTMs Exploit Linguistic Attributes of Data“. In: *Proceedings of The Third Workshop on Representation Learning for NLP, Rep4NLP@ACL 2018, Melbourne, Australia, July 20, 2018*, pp. 180–186 (cit. on pp. 21, 30).
- MacKay, David JC and David JC Mac Kay (2003). *Information theory, inference and learning algorithms*. Cambridge university press (cit. on pp. 34, 36).
- Marcus, Mitchell, Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). „Building a large annotated corpus of English: The Penn Treebank“. In: (cit. on p. 48).
- Merrill, William, Lenny Khazan, Noah Amsel, et al. (Aug. 2019). „Finding Hierarchical Structure in Neural Stacks Using Unsupervised Parsing“. In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics, pp. 224–232 (cit. on p. 65).
- Milne, Robert William (1982). „Predicting garden path sentences“. In: *Cognitive science* 6.4, pp. 349–373 (cit. on p. 60).
- Minsky, ML and SA Papert (1969). *Perceptrons. An Introduction to Computational Geometry*. 1969, Expanded (cit. on p. 17).
- Nagabandi, Anusha, Ignasi Clavera, Simin Liu, et al. (2019). „Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning“. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019* (cit. on p. 23).
- Nesterov, Yurii (2018). *Lectures on Convex Optimization: Second Edition*. Vol. 137. Springer Science & Business Media (cit. on pp. 27, 82, 83).

- Niven, Timothy and Hung-Yu Kao (2019). „Probing Neural Network Comprehension of Natural Language Arguments“. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pp. 4658–4664 (cit. on p. 70).
- Oberman, Adam M. and Jeff Calder (2018). „Lipschitz regularized Deep Neural Networks converge and generalize“. In: *CoRR* abs/1808.09540. arXiv: 1808.09540 (cit. on p. 68).
- Pearce, Tim, Mohamed Zaki, Alexandra Brintrup, and Andy Neely (2018). „Uncertainty in Neural Networks: Bayesian Ensembling“. In: *stat* 1050, p. 12 (cit. on pp. 3, 19, 23, 35–37, 45, 80, 81).
- Polyak, Boris T (1964). „Some methods of speeding up the convergence of iteration methods“. In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17 (cit. on p. 40).
- Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). „Improving language understanding by generative pre-training“. In: URL <https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/languageunderstandingpaper.pdf> (cit. on p. 17).
- Radford, Alec, Jeffrey Wu, Rewon Child, et al. (2019). „Language models are unsupervised multitask learners“. In: *OpenAI Blog* 1, p. 8 (cit. on pp. 17, 22, 70).
- Rosenblatt, Frank (1958). „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6, p. 386 (cit. on pp. 17, 26).
- Ruder, Sebastian (2016). „An overview of gradient descent optimization algorithms“. In: *CoRR* abs/1609.04747. arXiv: 1609.04747 (cit. on p. 18).
- Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams, et al. (1988). „Learning representations by back-propagating errors“. In: *Cognitive modeling* 5.3, p. 1 (cit. on pp. 17, 21, 27).
- Sak, Haşim, Andrew Senior, and Françoise Beaufays (2014). „Long short-term memory recurrent neural network architectures for large scale acoustic modeling“. In: *Fifteenth annual conference of the international speech communication association* (cit. on p. 21).
- Sapir, Edward (1921). *An introduction to the study of speech*. Citeseer (cit. on p. 16).
- Schwartz, Roy, Jesse Dodge, Noah A. Smith, and Oren Etzioni (2019). *Green AI*. arXiv: 1907.10597 [cs.CY] (cit. on pp. 69, 70).
- Silver, David, Aja Huang, Chris J Maddison, et al. (2016). „Mastering the game of Go with deep neural networks and tree search“. In: *nature* 529.7587, p. 484 (cit. on p. 25).
- Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). „Dropout: a simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958 (cit. on pp. 23, 33, 81).
- Sturt, Patrick, Martin J Pickering, and Matthew W Crocker (1999). „Structural change and reanalysis difficulty in language comprehension“. In: *Journal of Memory and Language* 40.1, pp. 136–150 (cit. on p. 60).
- Sukhbaatar, Sainbayar, Jason Weston, Rob Fergus, et al. (2015). „End-to-end memory networks“. In: *Advances in neural information processing systems*, pp. 2440–2448 (cit. on p. 23).

- Tessler, Michael Henry, Karen Gu, and Roger Philip Levy (2019). „Incremental understanding of conjunctive generic sentences“. In: (cit. on p. 17).
- Van Den Oord, Aäron, Sander Dieleman, Heiga Zen, et al. (2016). „WaveNet: A generative model for raw audio.“ In: *SSW* 125 (cit. on p. 25).
- Van Schijndel, Marten and Tal Linzen (2018). „Modeling garden path effects without explicit hierarchical syntax.“ In: *CogSci* (cit. on pp. 59, 61).
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. (2017). „Attention is all you need“. In: *Advances in neural information processing systems*, pp. 5998–6008 (cit. on pp. 17, 22, 70).
- Virmaux, Aladin and Kevin Scaman (2018). „Lipschitz regularity of deep neural networks: analysis and efficient estimation“. In: *Advances in Neural Information Processing Systems*, pp. 3835–3844 (cit. on p. 68).
- Widrow, Bernard and Marcian E Hoff (1960). *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs (cit. on p. 39).
- Yang, Zhilin, Zihang Dai, Yiming Yang, et al. (2019). „XLNet: Generalized Autoregressive Pretraining for Language Understanding“. In: *CoRR* abs/1906.08237. arXiv: 1906 .08237 (cit. on p. 70).
- Zhu, Lingxue and Nikolay Laptev (2017). „Deep and confident prediction for time series at uber“. In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, pp. 103–110 (cit. on p. 23).

Lemmata, Theorems & Derivations

This part of the appendix contains many supplementary derivations and proofs that are not essential to the contents of earlier chapters but worth including or simply too long. Section A.1 derives the evidence lower bound for variational inference and gives some intuition about its implications. Section A.2 shows how a measure for model confidence in classification, predictive entropy, can be approximated using the methods described in chapters 3.3.2 and 3.3.3. Section A.3 gives a short introduction to the concept of Lipschitz continuity and derives an important lemma. Sections A.5 and A.6 derive the corresponding recoding gradients for surprisal and predictive entropy.

A.1 Derivation of the Evidence lower bound

This section described the derivation of the evidence lower bound (ELBO) or variational free energy. In variational inference, we try to approximate the infeasible posterior $p(\boldsymbol{\theta}|\mathcal{D})$ by an easier distribution $q_{\phi}(\boldsymbol{\theta})$ parameterized by parameters ϕ . We can cast this into the form of an optimization problem by trying to find the set of parameters ϕ^* that appromates the true posterior best

$$\phi^* = \arg \min_{\phi} \text{KL}[q_{\phi}(\boldsymbol{\theta}) || p(\boldsymbol{\theta}|\mathcal{D})] \quad (\text{A.1})$$

where $\text{KL}[\cdot || \cdot]$ is the Kullback-Leibler divergence defined as

$$\text{KL}[q_{\phi}(\boldsymbol{\theta}) || p(\boldsymbol{\theta}|\mathcal{D})] = \int q_{\phi}(\boldsymbol{\theta}) \log \left(\frac{q_{\phi}(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D})} \right) d\boldsymbol{\theta} \quad (\text{A.2})$$

Because $p(\boldsymbol{\theta}|\mathcal{D})$ is still not computable, we expand it in the following way

$$\begin{aligned}
\text{KL}[q_\phi(\boldsymbol{\theta}) || p(\boldsymbol{\theta} | \mathcal{D})] &= \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{q_\phi(\boldsymbol{\theta})}{p(\boldsymbol{\theta} | \mathcal{D})} \right) d\boldsymbol{\theta} \\
&= \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{q_\phi(\boldsymbol{\theta}) p(\mathcal{D})}{p(\boldsymbol{\theta}, \mathcal{D})} \right) d\boldsymbol{\theta} \\
&= \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{q_\phi(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \mathcal{D})} \right) d\boldsymbol{\theta} + \int q_\phi(\boldsymbol{\theta}) \log p(\mathcal{D}) d\boldsymbol{\theta} \\
&= \int q_\phi(\boldsymbol{\theta}) \log q_\phi(\boldsymbol{\theta}) d\boldsymbol{\theta} - \int q_\phi(\boldsymbol{\theta}) \log p(\boldsymbol{\theta}, \mathcal{D}) d\boldsymbol{\theta} + \log p(\mathcal{D}) \\
&= \mathbb{E}_{q_\phi(\boldsymbol{\theta})} [\log p(\boldsymbol{\theta}, \mathcal{D})] + \mathbb{H}[q_\phi(\boldsymbol{\theta})] + \log p(\mathcal{D}) \\
&\geq \mathbb{E}_{q_\phi(\boldsymbol{\theta})} [\log p(\boldsymbol{\theta}, \mathcal{D})] + \mathbb{H}[q_\phi(\boldsymbol{\theta})] = \text{ELBO}[q_\phi(\boldsymbol{\theta})]
\end{aligned}$$

where we omit the log evidence $\log p(\mathcal{D})$ and the quantity therefore becomes the lower bound of $\text{KL}[q_\phi(\boldsymbol{\theta}) || p(\boldsymbol{\theta} | \mathcal{D})]$. It can also be rewritten into the form given in an earlier chapter:

$$\begin{aligned}
\text{ELBO}[q_\phi(\boldsymbol{\theta})] &= \int q_\phi(\boldsymbol{\theta}) \log q_\phi(\boldsymbol{\theta}) d\boldsymbol{\theta} - \int q_\phi(\boldsymbol{\theta}) \log p(\boldsymbol{\theta}, \mathcal{D}) d\boldsymbol{\theta} \\
&= \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{q_\phi(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \mathcal{D})} \right) d\boldsymbol{\theta} \\
&= \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{q_\phi(\boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \mathcal{D}) p(\boldsymbol{\theta})} \right) d\boldsymbol{\theta} \\
&= \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{q_\phi(\boldsymbol{\theta})}{p(\boldsymbol{\theta})} \right) d\boldsymbol{\theta} - \int q_\phi(\boldsymbol{\theta}) \log \left(\frac{p(\boldsymbol{\theta}, \mathcal{D})}{p(\boldsymbol{\theta})} \right) d\boldsymbol{\theta} \\
&= \text{KL}[q_\phi(\boldsymbol{\theta}) || p(\boldsymbol{\theta})] - \mathbb{E}_{q_\phi(\boldsymbol{\theta})} [\log p(\mathcal{D} | \boldsymbol{\theta})]
\end{aligned}$$

This reformulation gives us more intuition about the ELBO: The first term encourages densities that are close to the prior distribution $p(\boldsymbol{\theta})$, the second term is the expected likelihood that rises when the latent variables or model parameters $\boldsymbol{\theta}$ explain the data well.

A.2 Approximating Predictive Entropy

In this section I re-state the proof given in Gal, 2016 showing that we can approximate the predictive entropy of a model using MC Dropout as well explicitly transferring the same proof to the work of Pearce et al., 2018. The problem lies in the fact that the predictive entropy of a model producing a discrete probability distribution defined as

$$\mathbb{H}[y | \mathbf{x}, \mathcal{D}_{\text{train}}] = - \sum_c p(y = c | \mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y = c | \mathbf{x}, \mathcal{D}_{\text{train}}) \quad (\text{A.3})$$

contains the term $p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}})$, which we cannot evaluate. However, both MC Dropout (Gal and Ghahramani, 2016b) and Bayesian Anchored Ensembling (Pearce et al., 2018) are able to approximate this quantity. In the former case, we can expand it to include the true posterior over the model weights

$$p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) = \int p(y = c|\mathbf{x}, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})d\boldsymbol{\omega} \quad (\text{A.4})$$

which can be approximated by using the variational distribution placed on the model's weights, which is obtained by Monte Carlo estimates of K model forward passes using different dropout masks (see chapter 3.3.2 for more details):

$$\begin{aligned} & \lim_{k \rightarrow \infty} \frac{1}{K} \sum_{k=1}^K p(y = c|\mathbf{x}, \boldsymbol{\omega}^{(k)}) \\ &= \int p(y = c|\mathbf{x}, \boldsymbol{\omega})q^*(\boldsymbol{\omega})d\boldsymbol{\omega} \\ &\approx \int p(y = c|\mathbf{x}, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D}_{\text{train}})d\boldsymbol{\omega} \end{aligned}$$

Which we can now insert into eq. A.3. In the latter case, we similarly have to consider the true posterior like in eq. A.4, but this time over all the model parameters $\boldsymbol{\theta}$ instead of just all model weights $\boldsymbol{\omega}$:

$$p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) = \int p(y = c|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D}_{\text{train}})d\boldsymbol{\theta} \quad (\text{A.5})$$

We saw in chapter 3.3.3 that the posterior distribution can be approximated by the MAP-solution generating function $f_{\text{MAP}}(\cdot)$. Let us denote the probability of a class predicted by the k -th member of the ensemble trained using the anchor parameters $\boldsymbol{\theta}_0^{(k)} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_{\text{prior}})$ as $p(y = c|\mathbf{x}, \boldsymbol{\theta}_0^{(k)})$. Then it can be shown that

$$\begin{aligned} & \lim_{\substack{k \rightarrow \infty \\ \uparrow \text{corr.}}} \frac{1}{K} \sum_{k=1}^K p(y = c|\mathbf{x}, \boldsymbol{\theta}_0^{(k)})p(\boldsymbol{\theta}_0^{(k)}) \\ &= \int p(y = c|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D}_{\text{train}})d\boldsymbol{\theta} \end{aligned}$$

where $\uparrow \text{corr.}$ stands for increasing correlation between the parameters $\boldsymbol{\theta}^{(k)}$, which is known to occur in deep neural networks during training (see Cheng et al., 2018 §7.1, Srivastava et al., 2014). Therefore, this method is equally compatible to approximate the predictive entropy in eq. A.3. The whole proof of insight is too elaborate to be included here, which is why I refer the reader to appendix A of Pearce et al., 2018 for the full derivation.

A.3 Lipschitz Continuity

Definition 1 (Lipschitz continuity). A function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is called Lipschitz continuous on \mathbb{R}^n with a non-negative constant L if

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L\|\mathbf{x} - \mathbf{y}\|_{(\infty)} \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \quad (\text{A.6})$$

where the l_∞ norm is defined as

$$\|\mathbf{x}\|_{(\infty)} = \max_{1 \leq i \leq n} |\mathbf{x}_i| \quad (\text{A.7})$$

Intuitively, a Lipschitz continuous function is limited in its slope $|f(\mathbf{x}) - f(\mathbf{y})|$ by some Lipschitz constant L . The set of Lipschitz continuous functions also is a superset of all continuously differentiable functions. There also exists a class of functions $f(\cdot)$ with a Lipschitz continuous gradient that follows a similar definition:

Definition 2 (Lipschitz continuous gradients). The first derivative of a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is called Lipschitz continuous on \mathbb{R}^n if

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \quad (\text{A.8})$$

The following lemma is adapted from Nesterov, 2018 §1.2.2:

Lemma 1. For every function $f : \mathbb{R}^n \mapsto \mathbb{R}$ with a Lipschitz continuous first derivative it holds that

$$|f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle| \leq \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2 \quad (\text{A.9})$$

Proof. We start off by realizing that the value of a function f at a point \mathbf{y} can be reached by starting at a point \mathbf{x} and adding all the slopes of ∇f on the $\mathbf{y} - \mathbf{x}$ line to $f(\mathbf{x})$:

$$\begin{aligned} f(\mathbf{y}) &= f(\mathbf{x}) + \int_0^1 \langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})), \mathbf{y} - \mathbf{x} \rangle d\tau \\ &= f(\mathbf{x}) + \int_0^1 \langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}) + \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle d\tau \\ &= f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \int_0^1 \langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle d\tau \end{aligned}$$

$$|f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle| = \left| \int_0^1 \langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle d\tau \right|$$

Now, due to the fact that $-|f(\mathbf{x})| \leq f(\mathbf{x}) \leq |f(\mathbf{x})| \rightarrow -\int |f(\mathbf{x})| d\mathbf{x} \leq \int f(\mathbf{x}) d\mathbf{x} \leq \int |f(\mathbf{x})| d\mathbf{x} \rightarrow |\int f(\mathbf{x}) d\mathbf{x}| \leq \int |f(\mathbf{x})| d\mathbf{x}$ (first line) and using the Cauchy-Schwartz inequality (second line) alongside the definition of Lipschitz continuous gradients above (third line) we can write

$$\begin{aligned} |f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle| &\leq \int_0^1 |\langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle| d\tau \\ &\leq \int_0^1 \|\nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x})\| \cdot \|\mathbf{y} - \mathbf{x}\| d\tau \\ &\leq \int_0^1 L \|\mathbf{x} + \tau(\mathbf{y} - \mathbf{x}) - \mathbf{x}\| \cdot \|\mathbf{y} - \mathbf{x}\| d\tau \\ &= \int_0^1 \tau L \|\mathbf{y} - \mathbf{x}\|^2 \\ &= \left[\frac{1}{2} \tau^2 L \|\mathbf{y} - \mathbf{x}\|^2 \right]_0^1 \\ &= \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2 \end{aligned}$$

□

A.4 Theoretical guarantees of Recoding

In the following I will show that recoding indeed reduces the error signal δ_t produced by some hidden activations \mathbf{h}_t , which parallels a proof described in Nesterov, 2018.

Theorem 1 (Encoded error reduction). *For some error signal δ_t based on some hidden activations \mathbf{h}_t and some error signal δ'_t based on the recoded hidden activations \mathbf{h}'_t obtained using the same error function $e(\cdot)$, it holds that*

$$\delta'_t \leq \delta_t$$

Proof. Let us denote the function producing a time-dependent error signal δ_t based on hidden activations \mathbf{h}_t as $e(\cdot)$. We can then apply the property for functions with Lipschitz continuous gradients described in appendix A.3, lemma 1:

$$|e(\mathbf{h}'_t) - e(\mathbf{h}_t) - \langle \nabla e(\mathbf{h}_t), \mathbf{h}'_t - \mathbf{h}_t \rangle| \leq \frac{L}{2} \|\mathbf{h}'_t - \mathbf{h}_t\|^2 \quad (\text{A.10})$$

Given the recoding update rule described in equation 4.1, we can rewrite this as

$$\begin{aligned}
|e(\mathbf{h}'_t) - e(\mathbf{h}_t) - \langle \nabla e(\mathbf{h}_t), \mathbf{h}'_t - \mathbf{h}_t \rangle| &\leq \frac{L}{2} \|\mathbf{h}'_t - \mathbf{h}_t\|^2 \\
|e(\mathbf{h}'_t) - e(\mathbf{h}_t) - \langle \nabla e(\mathbf{h}_t), -\alpha \nabla e(\mathbf{h}_t) \rangle| &\leq \frac{L}{2} \|\alpha \nabla e(\mathbf{h}_t)\|^2 \\
|e(\mathbf{h}'_t) - e(\mathbf{h}_t) + \alpha \|\nabla e(\mathbf{h}_t)\|^2| &\leq \frac{\alpha^2 L}{2} \|\nabla e(\mathbf{h}_t)\|^2 \\
e(\mathbf{h}'_t) &\leq e(\mathbf{h}_t) - \alpha \|\nabla e(\mathbf{h}_t)\|^2 + \frac{\alpha^2 L}{2} \|\nabla e(\mathbf{h}_t)\|^2 \\
&= e(\mathbf{h}_t) - \alpha(1 - \frac{\alpha L}{2}) \|\nabla e(\mathbf{h}_t)\|^2
\end{aligned}$$

In order to find the optimal step size α , we can identify the extremum of $-\alpha(1 - \frac{\alpha L}{2})$ as $\alpha^* = \frac{1}{L}$ by setting it to 0 and solving for α . Resubstituting into the result above yields

$$\begin{aligned}
e(\mathbf{h}'_t) &\leq e(\mathbf{h}_t) - \alpha(1 - \frac{\alpha L}{2}) \|\nabla e(\mathbf{h}_t)\|^2 \\
&= e(\mathbf{h}_t) - \frac{1}{L}(1 - \frac{L}{2L}) \|\nabla e(\mathbf{h}_t)\|^2 \\
e(\mathbf{h}_t) - e(\mathbf{h}'_t) &\geq \frac{1}{2L} \|\nabla e(\mathbf{h}_t)\|^2 \\
\delta_t - \delta'_t &\geq \frac{1}{2L} \|\nabla_{\mathbf{h}_t} \delta_t\|^2
\end{aligned}$$

This implies that the improvement in terms of the error signal after recoding is at least $\frac{1}{2L} \|\nabla_{\mathbf{h}_t} \delta_t\|^2$. Assuming the Lipschitz constant is $L > 0$, this implies that a theoretical new error signal δ'_t based on the recoded hidden activations \mathbf{h}'_t has at least the same value as the original error signal δ_t or lower.

□

In practice, this also entails that obtaining L can help us determine the optimal step size at every time step so that we obtain a guaranteed improvement. This means that setting a suboptimal step size still can result in a higher error signal encoded in the hidden activations, contrary to the implication of this theorem.

Theorem 2 (Encoded error reduction through time). *Given a sequence of time steps from t to $t + k$ where intermediate hidden activations $\mathbf{h}'_{t+1}, \dots, \mathbf{h}'_{t+k}$ are being recoded, we denote δ_{t+k} the error signal at time step $t + k$ where only the activations \mathbf{h}_t were not recoded and δ_{t+k}^* the error signal resulting from all the activations from t to $t + k$, including \mathbf{h}_t , being recoded. It then holds that*

$$\delta_{t+k}^* \leq \delta_{t+k}$$

Proof. Here I show how recoding activations at a time step t even influences the error signal produced at a later time step δ_{t+k} . To show this, we evaluate the difference to the case where recoding is not performed at time step t . Let us denote $g_{\theta}^{(t)}(\cdot)$ the recurrent function¹ from equation 3.8 and $r^{(t)}(\cdot)$ the recoding function used at time step t . We furthermore denote the hidden activations produced by the recurrent function based on recoded hidden activations as $\mathbf{h}'_{t+1} = g_{\theta}^{(t+1)}(\mathbf{x}_{t+1}, \mathbf{h}'_t) = g_{\theta}^{(t+1)}(\mathbf{x}_{t+1}, r^{(t)}(\mathbf{h}_t))$. We can thus expand the error signal terms as follows

$$\begin{aligned}\delta_{t+k}^* &= e(\mathbf{h}'_{t+k}) = e(g_{\theta}^{(t+k)}(\mathbf{x}_{t+k}, \mathbf{h}'_{t+k-1})) \\ &= e(g_{\theta}^{(t+k)}(\mathbf{x}_{t+k}, \dots, r^{(t+1)}(g_{\theta}^{(t+1)}(\mathbf{x}_{t+1}, \mathbf{h}'_t)))) \\ \delta_{t+k} &= e(g_{\theta}^{(t+k)}(\mathbf{x}_{t+k}, \dots, r^{(t+1)}(g_{\theta}^{(t+1)}(\mathbf{x}_{t+1}, \mathbf{h}_t))))\end{aligned}$$

where the **only difference** is that for δ_{t+k}^* , the recoding of \mathbf{h}_t produced \mathbf{h}'_t , while for δ_{t+k} , these activations were left untouched. We now define a composite function $e_t^{t+k}(\cdot)$ as

$$e_t^{t+k} = e \circ g_{\theta}^{(t+k)} \circ r^{(t+k-1)} \circ g_{\theta}^{(t+k-1)} \circ \dots \circ r^{(t+1)} \circ g_{\theta}^{(t+1)}$$

where we omit the intermediate inputs $\mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+k}$ in favor of a more compact notation for $e_t^{t+k}(\cdot)$ as they are not relevant for the recoding gradient². Using this definition, we can follow the same proof as in the last theorem: As it applies to any black-box function with Lipschitz continuous gradients and because the composition of functions with this property also possesses Lipschitz continuous gradients, we can show that

$$\begin{aligned}e_t^{t+k}(\mathbf{h}_t) - e_t^{t+k}(\mathbf{h}'_t) &\geq \frac{1}{2L} \|\nabla e_t^{t+k}(\mathbf{h}_t)\|^2 \\ \delta_{t+k} - \delta_{t+k}^* &\geq \frac{1}{2L} \|\nabla_{\mathbf{h}_t} \delta_{t+k}\|^2\end{aligned}$$

and therefore analogously $\delta_{t+k}^* \leq \delta_{t+k}$.

□

¹This is the same recurrent function, the superscript is simply used to keep track of the number of times the function was applied.

²This is an admitted abuse of notation, as composition for multivariate functions would be denoted here as $g_{\theta}^{(t+1)}|_{\mathbf{h}_t=r(\mathbf{h}_t)}(\mathbf{x}_{t+1}, r(\mathbf{h}_t))$. The notation above was therefore used to increase readability. The proof still holds as we are only interested in the gradient of the composite function w.r.t some \mathbf{h}_t and not \mathbf{x}_t and the derivative $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}'_t}$ does not involve \mathbf{x}_t .

This shows that the recoding even improves the error signal of later time steps in a recurrent context given a sensible choice of the step size α . However, it should be noted that this improvement might be very small, as $\nabla_{\mathbf{h}_t} \delta_{t+k}$ requires one to evaluate the chain of partial derivatives

$$\nabla_{\mathbf{h}_t} \delta_{t+k} = \frac{\partial \delta_{t+k}}{\partial \mathbf{h}_{t+k}} \frac{\partial \mathbf{h}_{t+k}}{\partial \mathbf{h}'_{t+k-1}} \frac{\partial \mathbf{h}'_{t+k-1}}{\partial \mathbf{h}_{t+k-1}} \dots \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}'_t} \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}_t}$$

which is reminiscent of the vanishing gradient problem.

A.5 Derivation of the Surprisal recoding gradient

This section describes the full derivation of the recoding gradient when using surprisal as the error signal δ_t . Let us start by rewriting the definition of surprisal given in eq. 4.4 into a form that is easier to take a derivative of:

$$\begin{aligned} Z(y_t | \mathbf{h}_t, \boldsymbol{\theta}) &= \sum_c \mathbb{1}(y_t = c) \mathbf{o}_{tc}^{-\mathbf{o}_{tc}} - 1 \\ &= \sum_c \mathbb{1}(y_t = c) \exp(-\mathbf{o}_{tc} \log(\mathbf{o}_{tc})) - 1 \\ &= \exp(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t)) - 1 \end{aligned}$$

where \mathbf{A} denotes a square $C \times C$ matrix where C is the number of classes³, which only contains zeros except for a single entry on the diagonal corresponding to the label y_t :

$$\mathbf{A} = \begin{bmatrix} \mathbb{1}(A_{11} = y_t) & 0 & \dots & 0 \\ 0 & \mathbb{1}(A_{22} = y_t) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbb{1}(A_{CC} = y_t) \end{bmatrix}$$

We can then write the recoding gradient in terms of its partial derivatives:

$$\nabla_{\mathbf{h}_t} Z(y_t | \mathbf{h}_t, \boldsymbol{\theta}) = \frac{\partial Z(y_t | \mathbf{h}_t, \boldsymbol{\theta})}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \tilde{\mathbf{o}}_t} \frac{\partial \tilde{\mathbf{o}}_t}{\partial \mathbf{h}_t}$$

This way, we can evaluate every partial derivative individually:

³In the language modelling case, this corresponds to the cardinality of the vocabulary $|\mathcal{V}|$.

$$\begin{aligned}
\frac{\partial Z(y_t | \mathbf{h}_t, \boldsymbol{\theta})}{\partial \mathbf{o}_t} &= \frac{\partial}{\partial \mathbf{o}_t} \exp \left(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t) \right) - 1 \\
&= -\mathbf{A} \frac{\partial}{\partial \mathbf{o}_t} \left(\mathbf{o}_t^T \log(\mathbf{o}_t) \right) \exp \left(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t) \right) \\
&= -\mathbf{A} \left(\log(\mathbf{o}_t) + \mathbf{o}_t^T \left(\frac{1}{\mathbf{o}_t} \right) \right) \exp \left(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t) \right) \\
&= -\mathbf{A} \left(\log(\mathbf{o}_t) + \mathbf{1} \right) \exp \left(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t) \right)
\end{aligned}$$

For the derivative of the softmax layer, we cannot take $\frac{\partial \mathbf{o}_t}{\partial \tilde{\mathbf{o}}_t}$ directly and therefore revert to its derivative on a scalar-level first using the quotient rule:

$$\begin{aligned}
\frac{\partial \mathbf{o}_{ti}}{\partial \tilde{\mathbf{o}}_{tj}} &= \frac{\partial}{\partial \tilde{\mathbf{o}}_{tj}} \frac{\exp(\tilde{\mathbf{o}}_{ti})}{\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj})} \\
&= \frac{\frac{\partial}{\partial \tilde{\mathbf{o}}_{tj}} \left(\exp(\tilde{\mathbf{o}}_{ti}) \right) \left(\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj}) \right)}{\left(\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj}) \right)^2} - \frac{\exp(\tilde{\mathbf{o}}_{ti}) \frac{\partial}{\partial \tilde{\mathbf{o}}_{tj}} \left(\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj}) \right)}{\left(\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj}) \right)^2} \\
&= \frac{\mathbb{1}(i=j) \exp(\tilde{\mathbf{o}}_{ti})}{\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj})} - \frac{\exp(\tilde{\mathbf{o}}_{ti}) \exp(\tilde{\mathbf{o}}_{tj})}{\left(\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj}) \right)^2} \\
&= \frac{\mathbb{1}(i=j) \exp(\tilde{\mathbf{o}}_{ti})}{\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj})} - \frac{\exp(\tilde{\mathbf{o}}_{ti})}{\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj})} \cdot \frac{\exp(\tilde{\mathbf{o}}_{tj})}{\sum_{j=1}^J \exp(\tilde{\mathbf{o}}_{tj})} \\
&= \mathbb{1}(i=j) \mathbf{o}_{ti} - \mathbf{o}_{ti} \mathbf{o}_{tj} = \mathbf{o}_{ti} (\mathbb{1}(i=j) - \mathbf{o}_{tj})
\end{aligned}$$

where the last step is performed by using the original definition of the softmax function. Finally, we can convert this result back into vector form by noticing that $\mathbb{1}(i=j)\mathbf{o}_{ti}$ corresponds to a diagonal matrix whose elements are taken from the vector \mathbf{o}_{ti} and that the value $\mathbf{o}_{ti}\mathbf{o}_{tj}$ corresponds to an entry in the matrix produced by multiplying \mathbf{o}_t with its transpose:

$$\frac{\partial \mathbf{o}_t}{\partial \tilde{\mathbf{o}}_t} = \text{diag}(\mathbf{o}_t) - \mathbf{o}_t \mathbf{o}_t^T$$

Finally, by deriving the last part

$$\frac{\partial \tilde{\mathbf{o}}_t}{\partial \mathbf{h}_t} = \frac{\partial}{\partial \mathbf{h}_t} \left(\mathbf{W}_{ho} \mathbf{h}_t + \mathbf{b}_{ho} \right) = \mathbf{W}_{ho}$$

we can identify the full recoding gradient as

$$\nabla_{\mathbf{h}_t} Z(y_t | \mathbf{h}_t, \boldsymbol{\theta}) = -\mathbf{A} \left(\log(\mathbf{o}_t) + \mathbf{1} \right) \exp \left(-\mathbf{A} \mathbf{o}_t^T \log(\mathbf{o}_t) \right) \left(\text{diag}(\mathbf{o}_t) - \mathbf{o}_t \mathbf{o}_t^T \right) \mathbf{W}_{ho}$$

□

A.6 Derivations fo the Predictive Entropy recoding gradient

In the following I provide a full derivation of the recoding gradient when using predictive entropy as the error signal δ_t . The predictive entropy is very helpful when judging the certainty of a model in a classification setting as it quantifies the average amount of information in the model's predictive distribution. Its maximum value is reached when the distribution is uniform, the entropy is 0 when only class has probability one. The predictive entropy is defined as

$$\mathbb{H}[y_t|\mathbf{x}_t, \mathcal{D}_{\text{train}}] = -\sum_c p(y=c|\mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y=c|\mathbf{x}, \mathcal{D}_{\text{train}})$$

Evaluating $p(y=c|\mathbf{x}, \mathcal{D}_{\text{train}})$ is not feasible, we can therefore estimate this quantity using a Monte Carlo estimate combining multiple predictive distributions $\mathbf{o}_t^{(k)}$ obtained under different circumstances (different sets of dropout masks for variational or dropout recoding or different ensemble members for Anchored Bayesian Ensembling). Section A.2 shows in his work how this approximates the real predictive entropy in the limit. We thus obtain

$$\begin{aligned} \hat{\mathbf{o}}_t &= \frac{1}{K} \sum_{k=1}^K \mathbf{o}_t^{(k)} \\ p(y=c|\mathbf{x}, \mathcal{D}_{\text{train}}) &\approx \hat{\mathbf{o}}_{tc} \\ \tilde{\mathbb{H}}[y_t|\mathbf{h}_t, \mathcal{D}_{\text{train}}] &= -\sum_c \hat{\mathbf{o}}_{tc} \log(\hat{\mathbf{o}}_{tc}) = -\hat{\mathbf{o}}_t^T \log(\hat{\mathbf{o}}_t) \end{aligned}$$

where condition on the hidden state \mathbf{h}_t due to the recurrent setting. Next, let us write the gradient $\nabla_{\mathbf{h}_t} \tilde{\mathbb{H}}[y_t|\mathbf{h}_t, \mathcal{D}_{\text{train}}]$ in terms of its partial derivatives:

$$\nabla_{\mathbf{h}_t} \tilde{\mathbb{H}}[y_t|\mathbf{h}_t, \mathcal{D}_{\text{train}}] = \frac{\partial \tilde{\mathbb{H}}[y_t|\mathbf{h}_t, \mathcal{D}_{\text{train}}]}{\partial \hat{\mathbf{o}}_t} \sum_{k=1}^K \frac{\partial \hat{\mathbf{o}}_t}{\partial \mathbf{o}_t^{(k)}} \frac{\partial \mathbf{o}_t^{(k)}}{\partial \tilde{\mathbf{o}}_t^{(k)}} \frac{\partial \tilde{\mathbf{o}}_t^{(k)}}{\partial \mathbf{h}_t}$$

where we can now evaluate the parts separately:

$$\begin{aligned} \frac{\partial \tilde{\mathbb{H}}[y_t|\mathbf{h}_t, \mathcal{D}_{\text{train}}]}{\partial \hat{\mathbf{o}}_t} &= \frac{\partial}{\partial \hat{\mathbf{o}}_t} \left(-\hat{\mathbf{o}}_t^T \log(\hat{\mathbf{o}}_t) \right) \\ &= -\left(\frac{\partial}{\partial \hat{\mathbf{o}}_t} \hat{\mathbf{o}}_t \right)^T \log(\hat{\mathbf{o}}_t) - \hat{\mathbf{o}}_t^T \left(\frac{\partial}{\partial \hat{\mathbf{o}}_t} \log(\hat{\mathbf{o}}_t) \right) \\ &= -\log(\hat{\mathbf{o}}_t) - \hat{\mathbf{o}}_t^T \left(\frac{1}{\hat{\mathbf{o}}_t} \right) \\ &= -\left(\log(\hat{\mathbf{o}}_t) + 1 \right) \\ \frac{\partial \hat{\mathbf{o}}_t}{\partial \mathbf{o}_t^{(j)}} &= \frac{\partial}{\partial \mathbf{o}_t^{(j)}} \left(\frac{1}{K} \sum_{k=1}^K \mathbf{o}_t^{(k)} \right) = \frac{1}{K} \end{aligned}$$

because we evaluated $\frac{\partial \mathbf{o}_t^{(k)}}{\partial \tilde{\mathbf{o}}_t^{(k)}}$ and $\frac{\partial \tilde{\mathbf{o}}_t^{(k)}}{\partial \mathbf{h}_t}$ already in the previous section for a single \mathbf{o}_t , $\tilde{\mathbf{o}}_t$, we obtain the final result as

$$\nabla_{\mathbf{h}_t} \tilde{\mathbb{H}}[y_t | \mathbf{h}_t, \mathcal{D}_{\text{train}}] = -\left(\log(\hat{\mathbf{o}}_t) + \mathbf{1}\right) \frac{1}{K} \sum_{k=1}^K (\text{diag}(\mathbf{o}_t^{(k)}) - \mathbf{o}_t^{(k)} (\mathbf{o}_t^{(k)})^T) \mathbf{W}_{ho}^{(k)}$$

□

Experimental Details & Supplementary Results

In contrast to the previous appendix, this chapter supplies the reader with additional, practical information about the models and experiments in this thesis. To this end, section B.1 lists pseudocode for the different models. Section B.2 ponders on practical issues that have to be considered when implementing the recoding procedure. Section B.3 gives more detail to the hyperparameter search that was employed to train the models in chapter 5 and section B.4 gives additional plots from chapters 5 and 6.

B.1 Pseudocode

This section contains the pseudocode for the main models used in this thesis. Fig. B.1 refers to the surprisal-based recoding approach described in chapter 4.3.2. The code that describes the implementation of MC Dropout recoding in chapter 4.3.3 and Bayesian Anchored Ensemble recoding in chapter 4.3.4 is given in figs. B.2 and B.3, respectively.

```

forall  $\mathbf{x}_t \in \mathcal{X}$  do
     $\mathbf{h}_t = g_{\theta}(\mathbf{x}_t, \mathbf{h}'_{t-1})$ 
     $\tilde{\mathbf{o}}_t = \mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_{ho}$ 
     $\mathbf{o}_t = \text{softmax}(\tilde{\mathbf{o}}_t)$ 
    /* Recoding
    Set  $\delta_t = Z(y_t | \mathbf{x}_t, \theta)$  based on gold label  $y_t$ 
    Calculate  $\nabla_{\mathbf{h}_t} \delta_t$ 
     $\mathbf{h}'_t = \mathbf{h}_t - \alpha \nabla_{\mathbf{h}_t} \delta_t$ 
end

```

Figure B.1.: Pseudocode for the forward pass of a model using surprisal as the recoding error signal δ_t .

```

/* Forward pass
forall  $\mathbf{x}_t \in \mathcal{X}$  do
     $\mathbf{h}_t = g_{\theta}(\mathbf{x}_t, \mathbf{h}'_{t-1});$ 
    /* Sample k different dropout masks and predict
    for  $k \in 1 \dots K$  do
         $\mathbf{Z}_{ij}^{(k)} \sim \text{Bernoulli}(p) \quad \forall i = 1, \dots, N \quad \forall j = 1, \dots, |V|$ 
         $\mathbf{W}_{ho}^{(k)} = \mathbf{W}_{ho} \odot \mathbf{Z}^{(k)}$ 
         $\tilde{\mathbf{o}}_t^{(k)} = \mathbf{W}_{ho}^{(k)} \mathbf{h}_t + \mathbf{b}_{ho}$ 
         $\mathbf{o}_t^{(k)} = \text{softmax}(\tilde{\mathbf{o}}_t^{(k)})$ 
    end
    /* Recoding
    Calculate predictive entropy  $\delta_t \equiv \tilde{\mathbb{H}}[y|\mathbf{h}_t, \mathcal{D}_{\text{train}}]$ 
    Calculate  $\nabla_{\mathbf{h}_t} \delta_t$ 
     $\mathbf{h}'_t = \mathbf{h}_t - \alpha_t \nabla_{\mathbf{h}_t} \delta_t$ 
    /* Recompute output distribution for loss
     $\tilde{\mathbf{o}}'_t = \mathbf{W}_{ho} \mathbf{h}'_t + \mathbf{b}_{ho}$ 
     $\mathbf{o}'_t = \text{softmax}(\tilde{\mathbf{o}}'_t)$ 
end

```

Figure B.2.: Pseudocode for the forward pass of a model using its predictive variance estimated with MC Dropout as the recoding error signal δ_t .

B.2 Practical Considerations

Lastly I elaborate on some practical issues when implementing recoding. Although we derived the recoding gradients $\nabla_{\mathbf{h}_t} \delta_t$ by hand in the previous appendix, this is luckily not necessary in modern Deep Learning frameworks like PyTorch or Tensorflow, as those come equipped with an automatic differentiation engine, which builds a computational graph on the network and is therefore able to efficiently compute partial derivatives. This gives rise to the question whether all the operations involved in the recoding should be added to the graph as well. In preliminary experiments this did not seem to make a significant difference in terms of performance. However when using PyTorch, including them in the graph sometimes lead to memory leakages in the automatic differentiation engine, the circumstances of which were rather inconsistent. It is possible to violate common practice and perform the recoding on the hidden activations' `.data` attribute. This way avoids any memory issues, but does not allow gradients to backpropagate far enough when using the learned or predicted step size. In the latter case, it should also be made sure that the hidden states that are used as the predictor's input are detached from the

Hyperparameter	Value	Hyperparameter	Recoding Models
Number of layers	2	Step size	All
Embedding size	650	# Samples	MC Dropout, BAE, Variational
Hidden size	650	Prior scale	MC Dropout, BAE, Variational
Batch size	64	Weight decay	MC Dropout, Variational
Learning rate	20	MC Dropout	MC Dropout, Variational
Gradient clipping	0.25	Learning rate	Surprisal
Sequence Length	35		

(a) Hyperparameters shared by all models.

(b) Hyperparameters shared by all or some recoding-based models.

Table B.1.: List of core hyperparameters based in the work of Gulordava et al., 2018 as well as an attribution of additional parameters to the recoding models.

graph to avoid memory errors (gradient will otherwise be backpropagated through them as well). As a final point, I found it useful to use the `torch.autograd.grad` function to compute recoding gradients instead of calling `.backward()` on the error signal tensor, as hidden activations do not have to be explicitly declared as variables that the autograd engine has to keep track off this way.

B.3 Hyperparameter Search

In this section I provide more detail about the hyperparameter search performed on the Penn Treebank Corporous using the different model variants described in this thesis. As described in chapter 5.2, many basic parameters are taken from the work of Gulordava et al., 2018, who performed extensive grid search over 68 different combinations in the Wikitext-2 data set.¹ The resulting parameters are listed in table B.1a.

Secondly, we have to consider the hyperparameters used by recoding models, which are listed in table B.1. For this purpose, I sample 64 combinations of hyperparameters for surprisal recoding and recoding with predictive entropy, respectively, following the idea of Bergstra and Bengio, 2012, from either uniform or exponential distributions, which are listed in fig. B.2a. These ranges and corresponding distributions are based on preliminary, manual experiments. Following the creation of a set of hyperparameters, a model is trained for 4 epochs on the training set. This point in time was chosen because it was also observed that models start to diverge during

¹For more details refer to the supplementary material of Gulordava et al., 2018 under https://github.com/facebookresearch/colorlessgreenRNNs/blob/master/supplementary_material.pdf.

Hyperparameter	Distribution	Range	Hyperparameter	Value
Step size	Exponential	[0.5, 75]	Step size	10.19 (Surp.)
# Samples	Uniform	[2, 50]		0.001 (rest)
Prior scale	Uniform	[0.1, 0.4]	# Samples	15
Weight decay	Exponential	[0.0001, 0.01]	Prior scale	0.29
Dropout	Uniform	[0.1, 0.6]	Weight decay	$4.82 \cdot 10^{-5}$
MC Dropout	Uniform	[0.1, 0.6]	Dropout	0.15
			MC Dropout	0.42

(a) Sample ranges.

(b) Best values found.

Table B.2.: Left: Ranges from which listed hyperparameters are sampled. In case they are sampled an exponential distribution, the mean value is set to $\beta = (\text{upper} - \text{lower})/4$. Right: Best values found for recoding hyperparameters, rounded to two decimal points.

this training phase which is indicative of their final (relative) performance. Lastly, the validation perplexity is recorded and used to select the best found parameters, which are given in table B.2b.

B.4 Additional plots

This section simply puts additional plots on display that were not essential to the conclusions drawn in the corresponding sections.

```

/* Before training
/* Initialize weights for every member of the ensemble by
for k ∈ 1...K do
    | Sample weights  $\mathbf{W}_{ho}^{(k)} \sim \mathcal{N}(0, \Sigma_{\text{prior}})$ 
    | Sample anchor points  $\mathbf{W}_0^{(k)} \sim \mathcal{N}(0, \Sigma_{\text{prior}})$ 
end

/* Forward pass
forall  $\mathbf{x}_t \in \mathcal{X}$  do
     $\mathbf{h}_t = g_{\theta}(\mathbf{x}_t, \mathbf{h}'_{t-1});$ 
    /* Use k different ensemble members to make predictions
    for k ∈ 1...K do
        |  $\tilde{\mathbf{o}}_t^{(k)} = \mathbf{W}_{ho}^{(k)} \mathbf{h}_t + \mathbf{b}_{ho}$ 
        |  $\mathbf{o}_t^{(k)} = \text{softmax}(\tilde{\mathbf{o}}_t^{(k)})$ 
    end

    /* Recoding
    Calculate predictive entropy  $\delta_t \equiv \tilde{\mathbb{H}}[y|\mathbf{h}_t, \mathcal{D}_{\text{train}}]$ 
    Calculate  $\nabla_{\mathbf{h}_t} \delta_t$ 
     $\mathbf{h}'_t = \mathbf{h}_t - \alpha_t \nabla_{\mathbf{h}_t} \delta_t$ 
    /* Recompute output distribution for loss
     $\tilde{\mathbf{o}}'_t = \mathbf{W}_{ho} \mathbf{h}'_t + \mathbf{b}_{ho}$ 
     $\mathbf{o}'_t = \text{softmax}(\tilde{\mathbf{o}}'_t)$ 
end

/* Loss
for k ∈ 1...K do
    | Calculate anchored weight decay loss:  $\mathcal{L}_{\text{anchor}}^{(k)} = \frac{1}{N} \|\Gamma^{1/2} \mathbf{W}_{ho}^{(k)} - \mathbf{W}_0^{(k)}\|$ 
end
/* Amortize Loss
 $\mathcal{L}_{\text{total}} = \frac{1}{K} \sum_{k=1}^K (\mathcal{L}_{\text{CE}}^{(k)} + \mathcal{L}_{\text{anchor}}^{(k)})$ 

```

Figure B.3.: Pseudocode for the forward pass of a model using its predictive variance estimated with Bayesian Anchored ensembling as the recoding error signal δ_t .

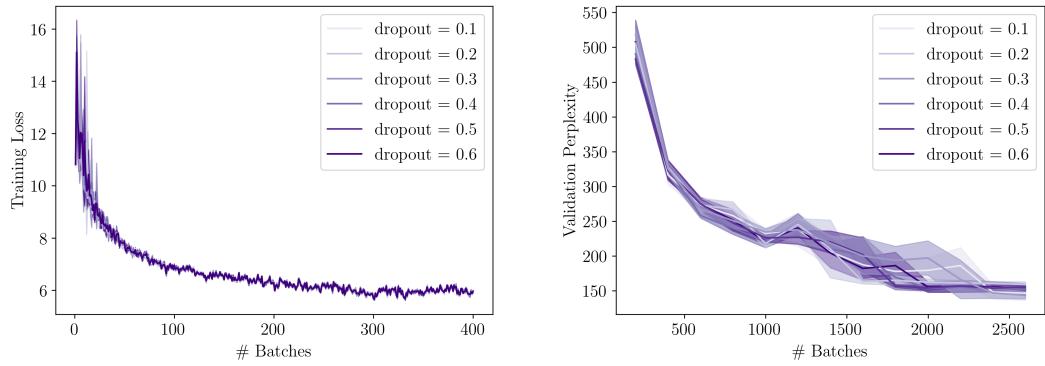


Figure B.4.: Training loss (left column) and validation perplexities (right column) on PTB using a different dropout rate to approximate predictive uncertainty using MC Dropout recoding. Curves denote the mean over $n = 4$ runs, with intervals signifying the standard deviation. Best viewed in color.

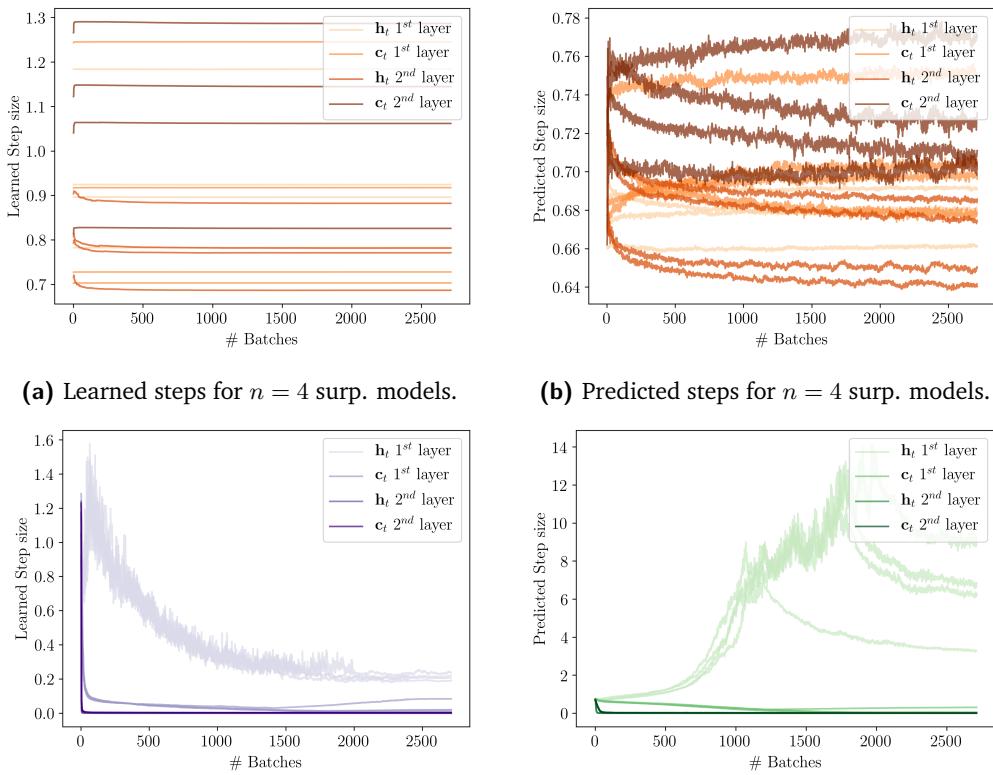


Figure B.5.: Additional plots for dynamic step size models showing the development of the average learned / predicted step size per batch over the course of the training.

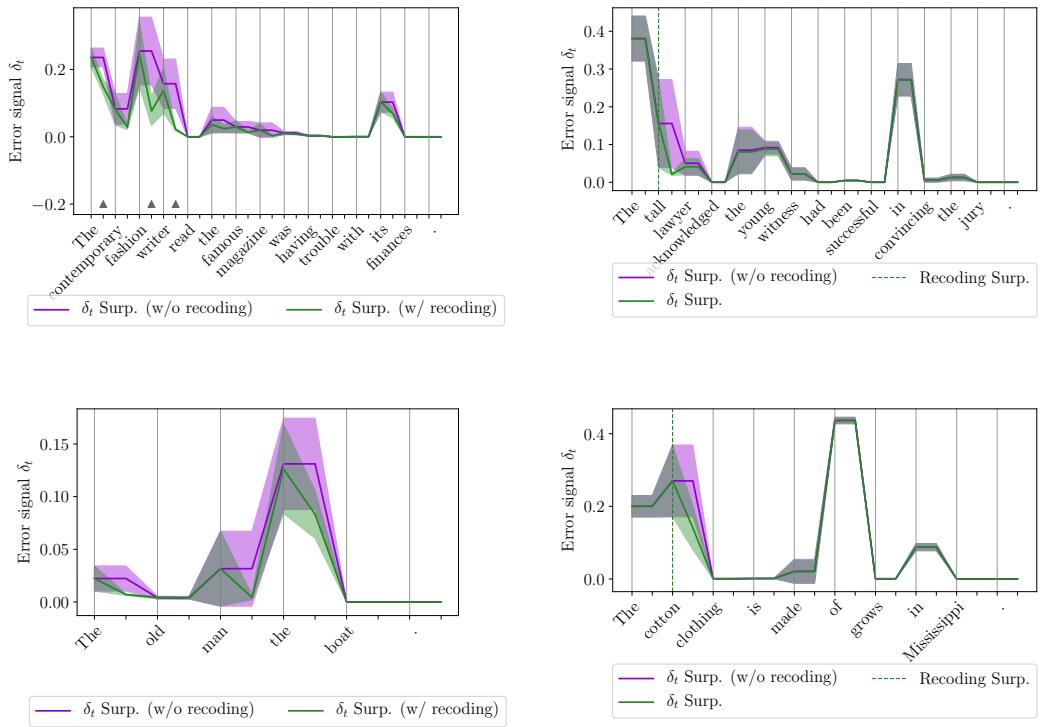


Figure B.6.: Additional plots for error reduction through recoding on garden path sentence for the surprisal based recoding model with fixed step size. In the bottom row, recoding at all (left) or a single time step (right) has no statistically significant effect on the error signal scores. Gray markers denote statistically significant differences (two-tailed Student's t-test, $p < 0.05$).