Constraint Driven Learning

B. Tech. Project Report
2017-18

Submitted in the partial fulfilment of
the requirements for the degree of

Bachelor of Technology

By

Kalpesh Krishna
140070017
Department of Electrical Engineering

Under the guidance of

Prof. Preethi Jyothi
Department of Computer Science & Engineering

Acknowledgments

I would like to thank my advisor Prof. Preethi Jyothi for encouraging me to take up this research problem, and keeping me motivated throughout the journey. I am thankful to Prof. Preethi for providing me access to the CSALT's computational cluster which helped me run all my experiments. I am thankful to my examiner Prof. Manoj Gopalkrishnan for the helpful suggestions on the presentation day.

I am also grateful to Prof. Mohit Iyyer, Prof. Kevin Gimpel, Prof. Karen Livescu and John Wieting for several useful discussions at different stages of this project.

Finally, I thank Tim Rocktäschel for encouraging me to use Twitter. Following NLP researchers on Twitter has helped me keep up-to-date with the most important research happening in NLP. This led me to find out about ELMo in the last few months of the project, that helped us obtain the most interesting results of this thesis.

Abstract

Modern machine learning using neural networks is often completely data-driven and done in a domain-independent, end-to-end fashion. This is a paradigm shift compared to traditional algorithms which heavily relied on domain-specific priors or hand-crafted features decided by domain experts. In this thesis we explore algorithms designed to improve performance of end-to-end systems by using traditional techniques or domain-specific rules and priors. Our focus is on two important NLP tasks - Language Modelling and Sentiment Analysis.

Contents

1 Introduction

Machine learning is slowly becoming a ubiquitous component in modern software. Machine learning has been actively researched since the 1960s, but only recently have the computational resources caught up sufficiently to complement the algorithms. Machine learning is often described as Software 2.0, and we are moving towards a stage where our day-to-day lives would heavily rely on machine learning algorithms.
Modern, state-of-the-art algorithms in machine learning rely on neural networks, which are large non-linear function approximators having millions of parameters. A modern solution to machine learning problems involves an "end-to-end" neural network, where a single neural network directly converts inputs to outputs. The state-of-the-art performance for several tasks often uses this "end-to-end" approach. This design comes with its share of perks and problems,

- **Domain Independent** - The same architecture achieves good performance across domains (NLP, Vision, Speech). For instance ResNets, introduced in image classification [1] have been successfully used in lexicon-free speech recognition [2].
- **Uninterpretable** - Since end-to-end systems are so large, it's hard to understand which weights are interacting in what fashion to make particular decisions. Neural Networks are often termed as "black-boxes". Recent efforts are trying to use input gradients to demystify these "black-boxes" [3].
- **Unintuitive** - They are not very well understood in terms of performance differences with architectural changes. Several design decisions such as depth, hidden units, optimization and regularization are often ad-hoc and not theoretically motivated - decided based on trial-and-error. As a digression, we describe some optimization issues we faced in our Language Modelling experiments in Section 4.1.
- **Difficult to Encode** rules, traditional algorithms and intuitive ideas built over several decades of domain-specific research or traditional machine learning. We hope these rules are **learnt** from data!
- **Heavily Reliant on Data** - Without large datasets, it's impossible to train end-to-end systems. Large datasets are often hard and expensive to build and label.

Hence, rich datasets with several representative examples of domain-specific priors are critical for the success of "end-to-end" systems. In low-resource or zero-resource settings, "end-to-end" systems cannot be used.
This thesis explores algorithms which augment supervised "end-to-end" neural networks with information derived from traditional algorithms or explicit rules and constraints (and hence "Constraint-Driven Learning"). We focus on two tasks, Language Modelling and Binary Sentiment Classification.
We begin by describing the tasks and datasets used in Section 2, with Language Modelling in Section 2.1 and Sentiment Classification in Section 2.2. We then move to our constraint-driven learning algorithms in Section 3. We start with "Model Mimicking Loss Functions", an approach resembling neural distillation [4] in Section 3.1. We then move to Posterior Regularization (PR) techniques in Section 3.2, describing a PR framework for neural networks and the perceptron algorithm. We present our best results in Section 3.3 using the contextualized word embeddings algorithm, ELMo. Finally, we present Input Gradient Regularization algorithms in Section 3.4.
In addition to the main thesis, we present a set of digressions in Section 4, a set of impor-

tant issues extremely relevant to modern deep learning, which we faced in the course of this project. In Section 4.1 we describe issues with optimizers in Language Modelling experiments which significantly delayed our progress. We move on to an analysis describing the importance of averaging across random seeds with small datasets (like SST for Sentiment Classification) in Section 4.2. Finally, in Section 4.3, we raise concerns about the ambiguity in sentiment classification. This includes a revealing crowd-sourced evaluation of some of the difficult sentences in the SST dataset for sentiment classification.

2 Task Description

2.1 Language Modelling

Language Modelling are probabilistic models that measure the likelihood of a sentence in a particular language. Let's assume a language has a vocabulary V, which is the set of tokens in the language. A language model attempts to find out,

$$p(w_1 w_2 ... w_{|L|})$$

Where $w_i \in V$, and L is the length of the sentence. To allow us to model sentences of variable length, a special end-of-sentence token `<EOS>` is added to the vocabulary [5]. Language models can be used as next-token predictors using conditional distributions,

$$p(w_1 w_2 w_3 ...) = p(w_1) p(w_2|w_1) p(w_3|w_1 w_2) ...$$

Hence a language model tries to model $p(w_i|w_1^{i-1})$ for all i and all $w_i \in V$. Here w_1^{i-1} is a short form for $w_1 w_2 ... w_{i-1}$.

2.1.1 Algorithms

Traditional techniques for language modelling include n-grams, which work with the assumption,

$$p(w_i|w_1^{i-1}) = p(w_i|w_{i-n+1}^{i-1}) \approx \frac{c(w_{i-n+1}^{i})}{c(w_{i-n+1}^{i-1})}$$

Where $c(\cdot)$ is a count in the training corpus. Since the denominator is likely to be zero for rare contexts, smoothing techniques are applied [5]. The most popular smoothing technique is the Kneser-Ney smoothing. This smoothing technique uses context to determine n-gram distributions. A popular example to describe this technique is the bi-gram "San Francisco". If this bi-gram appears frequently in the dataset, the unigram frequency of "Francisco" will be high. Relying on only the unigram frequency to predict the frequencies of n-grams would lead to incorrect results. Kneser–Ney smoothing corrects this by considering the frequency of the unigram in relation to possible words preceding it.

Modern techniques use an LSTM [6] which predicts a probability distribution across all words at each timestep. LSTMs are trained using a cross-entropy loss term at each timestep,

$$L = -\frac{1}{T} \sum_{t=1}^{T} \log p(w_i|w_1^{i-1})$$

Note that the cross entropy computation is between a one-hot probability distribution (of size V) representing the ground truth w_i and the model's prediction.

2.1.2 Datasets

The most popular benchmark for language modelling in English is the PTB (Penn Treebank) [7]. This is an open-vocabulary dataset containing 1M tokens and a vocabulary of 10000 tokens. Owing to the small size of PTB, recently a few larger datasets are being preferred, such as the wikitext2 and the wikitext3 [8].

2.1.3 Benchmarks

RNNs were first applied to language models in a series of papers by Mikolov et al, compiled in his PhD Thesis [9]. A popular benchmark which is also a tutorial in the TensorFlow repository is [10]. More recently, [11] showed that well-trained LSTMs can out-perform more sophisticated architectures. [12] contains some useful tips for regularization and optimization of neural language models. Finally, the current state-of-the-art for PTB is held by [13], an interesting perspective on language models.

2.1.4 Evaluation

Language models are evaluated using "perplexity". Perplexity can be defined as,

$$\text{ppl} = 2^b = 2^{-\frac{1}{N}\sum_i \log p(w_i|w_1^{i-1})}$$

Information theoretically, b is an estimator of the average number of bits needed to encode the probability distribution. The best KN-smoothed n-gram reaches a perplexity of ≈ 140 [9] on PTB. The current state of the art on PTB is 47.69 [13].

2.2 Sentiment Classification

Sentiment Classification is a classification task (usually binary) which helps us determine how positive / negative a sequence of words is. In a binary setting, we assume every sequence has either got a positive sentiment or a negative sentiment. In a more fine-grained setting, there could be a list of five sentiments (strongly positive, weakly positive, neutral, weakly negative, strongly negative).

2.2.1 Linguistics

We focus on two types of discourse relations, namely *A-but-B* and negations. For *A-but-B* sentences, the sentiment of the whole sentence is generally positively correlated with the sentiment of B. For negation sentences, the sentiment of the whole sentence is generally negatively correlated with the sentiment of the negation scope, or the phrase being negated. [14].
Negative sentiments seem to be more common in negation style sentences [15]. This paper also talks about emphatic and attenuating words that influence the degree of sentiment of a sentence. This paper is an interesting compilation of a number of linguistic phenomenon that affect binary sentiments.

2.2.2 Algorithms

This thesis will focus on two baseline algorithms popularly used for sentiment classification. The first algorithm is a 1-layer, 1-D convolutional neural network [16]. An input sentence are converted first into a sequence of word embeddings (`word2vec`, [17]). A one-dimensional CNN (across time) is applied over these word embeddings followed by a max-pool layer. Finally, a fully connected layer with a softmax non-linearity projects the this intermediate

representation to a probability distribution across classes. This simple architecture shows good success and is a common baseline for sentiment classification research.
The second algorithm [18] analyzed in this thesis is an extension to the previous CNN. This framework is fairly general, and it can be used to augment any neural network with logic rules. We have described this algorithm in detail in Section 3.2.1.

2.2.3 Datasets

We focus on the Stanford Sentiment Treebank [19], which focusses on sentence level sentiment classification. Each sentence is a representative part of a movie review on Rotten-Tomatoes, originally mined in [20]. Three crowd workers have labelled each sentence from 1 to 25 depending on the degree of negative to positive sentiment. Two datasets have been constructed from this, a binary classification dataset SST2 (with two labels, positive or negative) and a 5-way classification dataset SST5 (very negative, slightly negative, neutral, slightly positive, very positive). Additionally, two variants of the training set have been provided. One with phrase-level labels and another with just whole-sentence labels. Another dataset we used was the MR dataset [20]. We present some data statistics in Table 1.

Number of	Phrases	Train	Dev	Test
Instances	76961	6920	872	1821
A-but-B	3.5%	11.1%	11.5%	11.5%
Negations	2.0%	17.5%	18.3%	17.2%
Discourse	5.0%	24.6%	26.0%	24.5%

Table 1: Statistics of SST2 dataset. Here "Discourse" includes both *A-but-B* and negation sentences. The mean length of sentences is in terms of the word count.

2.2.4 Benchmarks

Evaluation on SST2 / SST5 is a simple classification accuracy. The first few experiments on SST2 / SST5 were done in the papers which created the datasets, [19, 21]. The current state-of-the-art of SST2 is [22], a simple architecture which is pre-trained on a very large corpus before fine-tuning on the SST dataset. Another recent paper [23], which is an extension of the CNN architecture, analyzes initializations of CNN filters and achieves impressive results. Finally, a paper on contextualized word embeddings (called ELMo) holds the state-of-the-art in SST5 [24]. We have described our experiments with ELMo in Section 3.3.

3 Algorithms

3.1 Model Mimicking Loss Function

3.1.1 Idea

The standard cross entropy loss function L_1 uses the ground truth values as a one hot distribution.

$$L_1 = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t|x_1^{t-1})$$

Let's assume a trained teacher model distribution q_ϕ, and a student model distribution p_θ. We want to train the student model to emulate the distribution of a teacher model q_ϕ. To this end, we design a cross entropy loss function between the two distributions.
Let's study a language modelling task with a training set with T tokens and vocabulary V.

$$L_2 = -\frac{1}{T}\sum_{t=1}^{T}\sum_{y\in V} q_\phi(y|x_1^{t-1}) \log p_\theta(y|x_1^{t-1})$$

This loss function has a gradient similar to the standard one-hot cross entropy loss. Following the notation in Chapter 10 of [25], we obtain the gradient of the loss at timestep t with respect to the logits layer $\mathbf{o}^{(t)}$ as,

$$(\nabla_{\mathbf{o}^{(t)}} L_2^{(t)})_i = p_\phi(y_i|x_1^{t-1}) - q_\theta(y_i|x_1^{t-1})$$

For the special case of the standard cross-entropy loss, $q_\phi(y_i|x_1^{t-1}) = 1$ for the ground truth and zero elsewhere.
We can now interpolate a standard cross-entropy loss L_1 with our model mimicking loss L_2. This scheme resembles multi-task learning setups. We experiment with interpolation using the following techniques,

- **Direct Interpolation** - Similar to the neural distillation [4] formulation,

$$L = \lambda L_1 + (1-\lambda)L_2$$

- **Alternation** - Alternate weight updates on $L = L_1$ and $L = L_2$. This scheme slightly resembles GAN training [26] which alternates updates of the generator and the discriminator. However unlike GAN training, both updates are done on the same set of weights, using different objectives.

- **System Interpolation** - We train two different language models, one with $L = L_1$ and another with $L = L_2$. During inference, we interpolate the probabilities predicted by each model,

$$P(y|x_1^{t-1}) = \phi P_1(y|x_1^{t-1}) + (1-\phi)P_2(y|x_1^{t-1})$$

We hope to inject information derived from more traditional algorithms through the model mimicking loss function, hoping to leverage the benefits of both traditional algorithms and neural models.

3.1.2 Teacher Distribution

Several attempts have been made in the past to include n-gram language model statistics into neural architectures [27, 28]. In our experiments, we use a Kneser-Ney smoothed 3-gram language model [5] and use it as our teacher distribution.

3.1.3 Experiments

We train our models using SRILM [29], an n-gram language modelling toolkit. We created our language models using the command,

```
./srilm/bin/i686-m64/ngram-count -text ptb.train.txt -unk -interpolate -lm LM
-kndiscount -gt1min 1 -gt2min 1 -gt3min 1 -order 3
```

Inference using the command,

```
./srilm/bin/i686-m64/ngram -ppl ptb.test.txt -unk -lm LM -order 3
```

For the recurrent neural language models, we wrote a TensorFlow [30] implementation[1] of [10], a standard baseline for LSTM language models. In addition to this, we tie the weights of the embeddings and the final projection layer [31]. We train our models using the different schemes described in Section 3.1.1 on the Penn Treebank corpus (PTB).

We present our results in Table 2. We observe slight performance gains with the direct interpolation scheme ("L1 + L2") over the baseline ("L1"). Also notice the superior performance of the student model ("L2") over its teacher ("3-gram"). We attribute this success to better generalization of the model due to dropout.

Table 2: Perplexity on PTB test corpus for single models (no system interpolation). L1 / L2 represents alternation, whereas L1 + L2 represents the direct interpolation with $\lambda = 0.7$.

Model	Perplexity
3-gram	148.28
L1	76.67
L2	132.27
L1 / L2	82.92
L1 + L2	**76.06**

We now attempt to try out all possible combinations of system interpolation, and tune ϕ on the validation set for each pair of models. We present our results in Table 3.

[1]TensorFlow has a tutorial for this at `https://www.tensorflow.org/tutorials/recurrent`

Table 3: Perplexity on PTB test corpus for different system interpolations. Max Perplexity chooses the larger probability for each token = $\max\{P_1(x_t|x_1^{t-1}), P_2(x_t|x_1^{t-1})\}$. It is not a metric of performance, rather an upperbound over different dynamic interpolation schemes.

Model 1	Model 2	Perplexity	Max Perplexity
L1	L2	74.65	58.90
L1	3-gram	73.80	**57.19**
L1	L1 / L2	72.66	58.38
L1	L1 + L2	**70.73**	57.30
L2	3-gram	128.26	109.54
L2	L1 / L2	82.52	68.13
L2	L1 + L2	76.03	64.50
L1 / L2	3-gram	81.36	65.09
L1 / L2	L1 + L2	74.20	61.45
L1 + L2	3-gram	75.50	62.03

We notice that different system interpolation schemes improve performance over the baseline ("L1" in Table 2). Notably on the PTB dataset, we get the best performance by interpolating L1 with the direct interpolation model (L1 + L2).

Also notice the "Max Perplexity" column, which chooses the larger probability for every token = $\max\{P_1(x_t|x_1^{t-1}), P_2(x_t|x_1^{t-1})\}$. It is not a metric of performance, rather an upper-bound over different dynamic interpolation schemes. We notice "L1" combined with 3-gram has the best "Max Perplexity", which is indicative of the maximum diversity of the system interpolation - no neural network (3-gram) with true neural objective (L1).

We try to see the benefits of a system interpolation between L1 and 3-grams by plotting the probability of correct token in the L1 model against its probability in the 3-gram model, in Figure 1. While most of the points lie in the L1-dominated region, there are a decent number of tokens predicted more correctly by the 3-gram model than L1.

3.1.4 Technical Challenges

We faced a lot of trouble while optimizing the language models. We noticed significant performance improvement by using the SGD optimizer instead of the Adam optimizer. We have described our experience in Section 4.1.

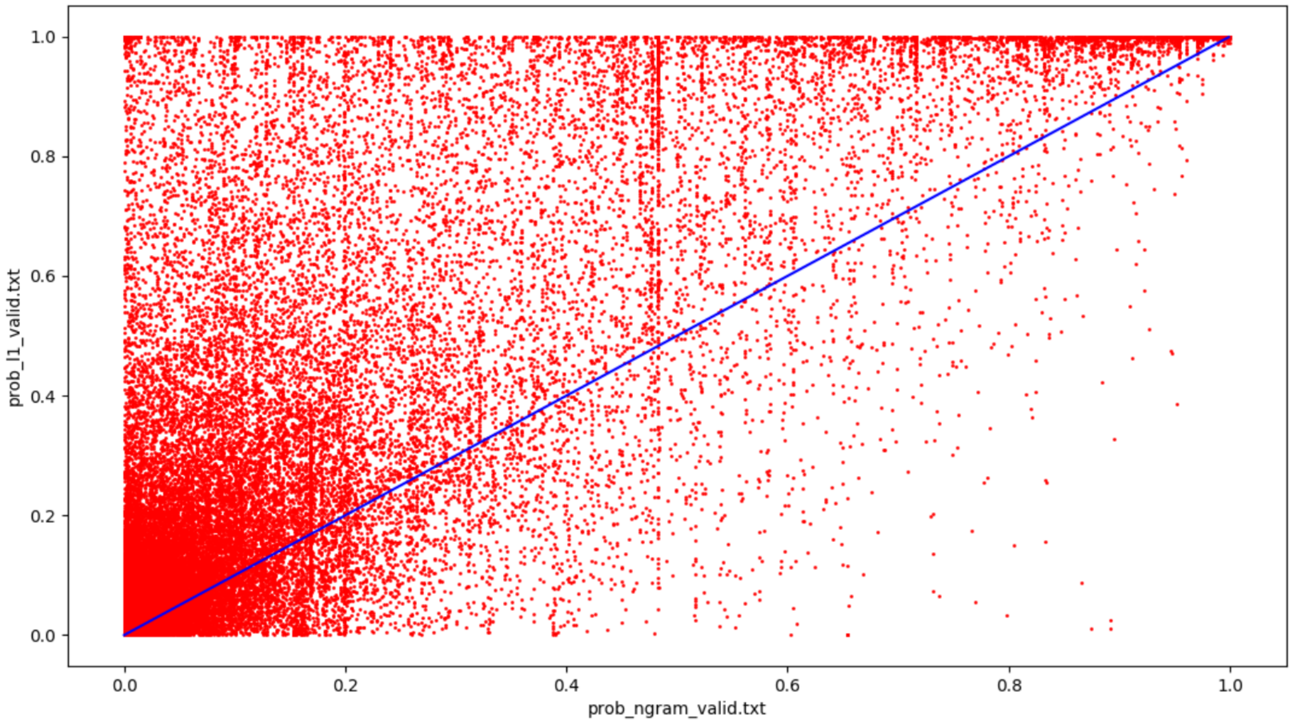
3.1.5 Future Work

As future work, we hope to investigate a few pressing questions,

- How important are baselines in Language Models, if you don't beat state-of-the-art (SotA)? How well do our ideas scale to SotA models?
- Why does the Adam Optimizer perform so badly? (ref: Section 4.1)
- Can we improve our student models by keeping a high temperature while training them from the teacher models? What is a good way to introduce temperature into probability distributions without logits?

- Does L2 have merits in non-PTB settings of larger vocabularies, in morphologically rich languages, or situations with less training data?
- What's the best way to leverage n-gram statistics in LMs?

Figure 1: Distribution of correct validation tokens' probabilities. Since a large number of tokens are above the blue line ($y = x$), most of the performance in the system interpolation of L1 and 3-gram is due to L1. The tokens below the blue line indicate all instances where a 3-gram model will provide more accurate predictions than the L1 model.



3.2 Posterior Regularization

3.2.1 PR for Neural Networks

This approach tries to encode logic rules (such as the *A-but-B* rule described in Section 2.2.1, via the loss function. The approach can be broken down into two components - Projection and Distillation.

Projection. The first technique is to *project a trained model into a rule-regularized subspace*, in a fashion similar to [32]. More precisely, a given model p_θ is projected to a model q_θ defined by the optimum value of q in the following optimization problem:[2]

$$\begin{aligned}&\min_{q,\xi\geq 0} \mathrm{KL}(q(X,Y)||p_\theta(X,Y)) + C\sum_{x\in X}\xi_x\\ &\text{s.t.}\quad (1-\mathbb{E}_{y\leftarrow q(\cdot|x)}[r_\theta(x,y)]) \leq \xi_x\end{aligned}$$

Here $q(X,Y)$ denotes the distribution of (x,y) when x is drawn uniformly from the set X and y is drawn according to $q(\cdot|x)$.

Iterative Rule Knowledge Distillation. The second technique is to transfer the domain knowledge encoded in the logic rules into a neural network's parameters. Following [4], a "student" model p_θ can learn from the "teacher" model q_θ, by using a loss function,

$$\pi H(p_\theta, P_{\text{true}}) + (1-\pi)H(p_\theta, q_\theta)$$

during training, where P_{true} denotes the distribution implied by the ground truth, $H(\cdot,\cdot)$ denotes the cross-entropy function, and π is a hyperparameter. [18] computes q_θ after every gradient update, by projecting the current p_θ, as described above. Note that both the mechanisms can be combined: After fully training p_θ using the iterative distillation process above, the projection step can be applied one more time to obtain q_θ which is then used as the trained model.

The whole algorithm can be summarized as,

```
input: number of epochs $E$, number of minibatches $B$
for $e = 1...E$ do
    for $b = 1...B$ do
        // forward pass, calculate $p_\theta$;
        // project $p_\theta$ to get $q_\theta$;
        // update $\theta$ using interpolated loss function
    end
end
// For inference, we can use the final learnt $p_{\theta *}$;
// or project the final $p_{\theta *}$ to get $q_{\theta *}$;
```

[2]The formulation in [18] includes another hyperparameter λ per rule, to control its relative importance; when there is only one rule, as in our case, this parameter can be absorbed into C.

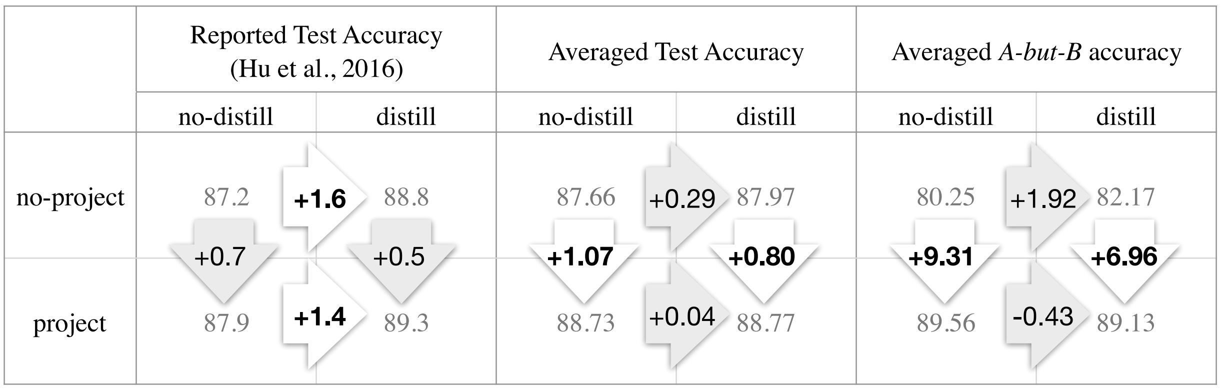
3.2.2 Perceptron Algorithm

We can augment the PR framework by learning a suitable value for C rather than a static C. This can be done using the perceptron algorithm [33], with individual features being the raw probabilities and the values r_i. We didn't achieve any significant improvements using this approach and will not include experimental results with this approach in the next section.

3.2.3 Experiments

We use the publicly available implementation of [18][3] and run it for a 100 different random seeds. (See Section 4.2 for more details on the importance of averaging).

Figure 2: Comparison of the accuracy improvements reported in [18] and those obtained by averaging over 100 random seeds. The last two columns show the (averaged) accuracy improvements for *A-but-B* style sentences. All models use the publicly available implementation of [18] trained on phrase-level SST2 data.



	Reported Test Accuracy (Hu et al., 2016)			Averaged Test Accuracy			Averaged *A-but-B* accuracy		
	no-distill		distill	no-distill		distill	no-distill		distill
no-project	87.2	**+1.6**	88.8	87.66	+0.29	87.97	80.25	+1.92	82.17
	+0.7		+0.5	**+1.07**		**+0.80**	**+9.31**		**+6.96**
project	87.9	**+1.4**	89.3	88.73	+0.04	88.77	89.56	-0.43	89.13

Our analysis reveals that the reported performance of the two methods proposed in that work (projection and distillation) is in fact affected by the high variability across random seeds. As it turns out, the more robust averaged analysis yields a somewhat different conclusion of their effectiveness.

In Figure 2, the first two columns show the reported accuracies in [18], for models trained with and without distillation (corresponding to using values $\pi = 1$ and $\pi = 0.95^t$ in the t^{th} epoch, respectively). The two rows show the results for models with and without a final projection into the rule-regularized space. We keep our hyper-parameters identical to [18] (in particular, for projection, $C = 6$).

The baseline system (no-project, no-distill) is essentially identical to the system of [16]. All the systems are trained on the phrase-level SST2 dataset with early stopping on the development set. The number inside each arrow indicates the improvement in accuracy by adding either the projection or the distillation component to the training algorithm. Note that the reported figures suggest that while both components help in improving accuracy, the distillation component is much more helpful than the projection component.

[3]`https://github.com/ZhitingHu/logicnn/`

The next two columns show the results of repeating the above analysis, but averaged over 100 random seeds. The averaged figures show slightly lower improvements, and more importantly, attributes the improvement almost entirely to the projection component rather than the distillation component. To confirm this, we repeat our averaged analysis restricted to the sentences which affect the rule that the implementation targets, namely, *"A-but-B"* style sentences (shown in the last two columns). We again observe that the effect of projection is pronounced, while, in comparison, distillation offers little or no advantage.

3.2.4 Future Work

- Investigate the perceptron algorithm in a more thorough manner.
- Analyze [18] for non-binary sentiment classification.
- Analyze [18] for other binary classification tasks.

3.3 Contextualized Word Embeddings

Traditional context-independent word embeddings like `word2vec` [17] or `GloVe` [34] are fixed vectors for every word in the vocabulary. In contrast, contextualized embeddings are dynamic representations, dependent on the current context of the word. We hypothesize that contextualized word embeddings might inherently capture these logic rules due to increasing the effective context size for the CNN layer in [16].

We follow the success of a recent contextualized embedding model, ELMo (Embeddings from Language Models) [24], which is a bidirectional language model trained on the 1 Billion Words language modelling benchmark [35]. Embeddings are created using a weighted combination of intermediate layer hidden representations, with the weights fine-tuned to the downstream task.

3.3.1 Experiments

We utilize the TensorFlow Hub implementation of ELMo[4] and feed these contextualized embeddings into our CNN model. As in Section 3.2, we check performance with and without the final projection into the rule-regularized space.
We present our results in Table 4. Switching to ELMo word embeddings improves performance by 2.91 percentage points on an average, corresponding to about 53.0 test sentences. Of these, 31.78 (60% of the improvement) sentences correspond to *A-but-B* and negation style sentences, which is substantial when considering that only 24.5% of test sentences include these discourse relations (Table 1). The baseline model (no-project, no-distill) gets 14.02% sentences incorrect on an average, which corresponds to 255.3 sentences in the test corpus. However, only 88.8 (34.8%) of these sentences are *A-but-B* style or negations.

Model		Test	*but*	*but* or *neg*
no-distill	no-project	85.98	78.69	80.13
no-distill	project	86.54	83.40	-
distill	no-project	86.11	79.04	-
distill	project	86.62	83.32	-
elmo	no-project	88.89	86.51	87.24
elmo	project	88.96	87.20	-

Table 4: Average performance (across 100 seeds) of ELMo on the SST2 task. We show performance on *A-but-B* sentences ("*but*"), negations ("*neg*"). The "distill" results have been trained on sentences and not phrase-level labels for a fair comparison with baseline and ELMo, unlike Section 3.2.

KL Divergence Analysis: We observe no significant gains by projecting a trained ELMo model into an *A-but-B* rule-regularized space, unlike the other models. We confirm that ELMo's predictions are much closer to the *A-but-B* rule's manifold than those of the other models by computing $\text{KL}(q_\theta || p_\theta)$ where p_θ and q_θ are the original and projected distributions: Averaged across all *A-but-B* sentences and 100 seeds, this gives $0.27, 0.26$ and 0.13 for the [16], [18] with distillation and ELMo systems respectively. Hence, a major chunk of ELMo's performance improvement is on contrastive conjunctions and negations.

[4]`https://tfhub.dev/google/elmo/1`

Performance vs Ambiguity: Given the inherent ambiguous nature of sentiment analysis, we check our models' performance on subsets of the *A-but-B* and negation sentences in SST2 based on a crowd-sourced evaluation of their ambiguity. We have described these experiments in Section 4.3.

3.3.2 Future Work

Check performance of other discourse relations using ELMo and investigate the benefits of ELMo + distillation. ELMo serves as a good baseline for the input gradient regularization technique in Section 3.4.

3.4 Input Gradient Regularization

3.4.1 Past Work

Input gradients were first introduced in [36] as "Double Backpropagation". Input gradients have been used to interpret models [3, 37, 38, 39], structured prediction [40], generating adversarial examples [41, 42, 43], training neural networks [44, 45] and Wasserstein GANs [46] and knowledge transfer [47, 48].

Here is a good summary of the work done with input gradients [49].

3.4.2 Model

Intuitively, we want the gradient of the final probabilities to be higher with respect to the "more important" input units. For *A-but-B* sentences, we would like $||\nabla_B P_+||_2$, $||\nabla_B P_-||_2$ to be high, and $||\nabla_A P_+||_2$, $||\nabla_A P_-||_2$ to be low. Here a gradient with respect to a word refers to the gradient with respect to each of the word embedding input units. We encode this in our model using the formulation outlined in [3]. For a single sentence,

$$L = \lambda_2 L_1 + (1 - \lambda_2)||\nabla_A \log P_+ + \nabla_A \log P_-||_2^2$$

3.4.3 Experiments

We present our results in Table 5. Our first experiment applies the gradient regularization to the baseline [16] with $\lambda_2 = 0.99$. While we do observe improvements over the baseline, these do not correspond to *A-but-B* sentences, but general regularization instead. We then apply our gradient loss function over ELMo, taking gradients with respect to the final ELMo embeddings (with $\lambda_2 = 0.999$). We notice modest improvements in performance, corresponding to *A-but-B* sentences.

Table 5: Average performance (over 100 seeds) on gradient-based regularization.

Model		Test	*but*
no-distill	no-project	85.98	78.69
no-distill	project	86.54	83.40
grad	no-project	86.54	78.96
grad	project	87.15	84.15
elmo	no-project	88.89	86.51
elmo	project	88.96	87.19
grad + elmo	no-project	89.05	87.40
grad + elmo	project	88.98	86.55

3.4.4 Future Work

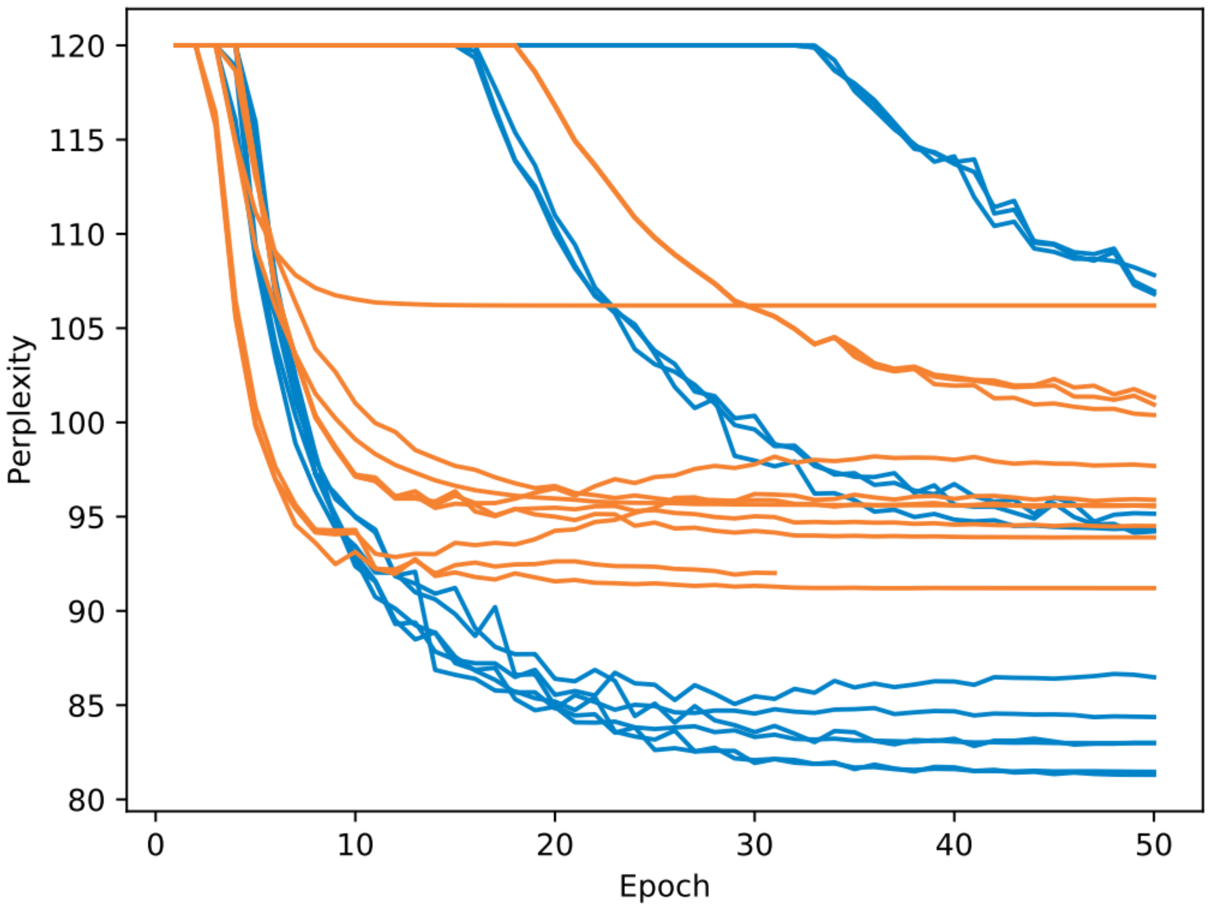
This idea has not been explored fully in this thesis. We should try out different variants the gradient loss function, along with a more thorough hyperparameter grid search.

4 Digressions

4.1 Optimization of Language Models

We described our experiments with Language Models in Section 3.1. We noticed that it was difficult to training language models using the Adam optimizer, irrespective of the learning rate. We compare the validation perplexity of different training configurations in Figure 3 for the baseline in [10]. Notice the large difference in performance between the best blue and orange curve.

Figure 3: Validation perplexity vs epochs of training for various settings of learning rate and decay schedules. The orange curves are trained using the Adam optimizer, whereas the blue curves have been trained using Stochastic Gradient Descent.



We discovered some literature talking about the disadvantages of adaptive gradient methods like Adam [50]. Notably, [51] presents a toy task where an adaptive gradient scheme will fail and provide empirical evidence on the benefits of Stochastic Gradient Descent over Adam. In the language modelling literature, [12] suggest using SGD instead of Adam for language modelling. They also introduce NT-ASGD, a new optimization scheme to train state-of-the-art language models.

The proofs in the original Adam paper have been re-analyzed in the ICLR best paper award winner [52]. This paper introduces AMSGrad, a simple modification to improve the optimization of Adam.

4.2 Averaging Across Random Seeds

For every sentiment analysis experiment in Section 3, we run each model 100 times with early stopping on validation performance in each run. We noticed a large variation from run-to-run, and carried out an analysis to assess the importance of averaging across random seeds.

We run the sentiment classification baseline CNN by [16] across 100 random seeds, training on sentence-level labels. We observe that, presumably due to the small size of the dataset, there is a large amount of variation from run-to-run. This can be seen in the density plot in Figure 4, which shows the range of accuracies (83.47 to 87.20) along with 25, 50 and 75 percentiles, when the models are trained with early stopping based on their performance in the development set. The figure also shows how the variance persists even after the average converges, by plotting (in gray) the accuracies of 100 models, as they are trained for 20 epochs each; their average accuracies are shown in red.

Figure 4: Variation in models trained on SST-2 (sentence-only). Accuracies of 100 randomly initialized models are plotted against the number of epochs of training (in gray), along with their average accuracies (in red, with 95% confidence interval error bars). The density plot in inset shows the distribution of accuracies when trained with early stopping.

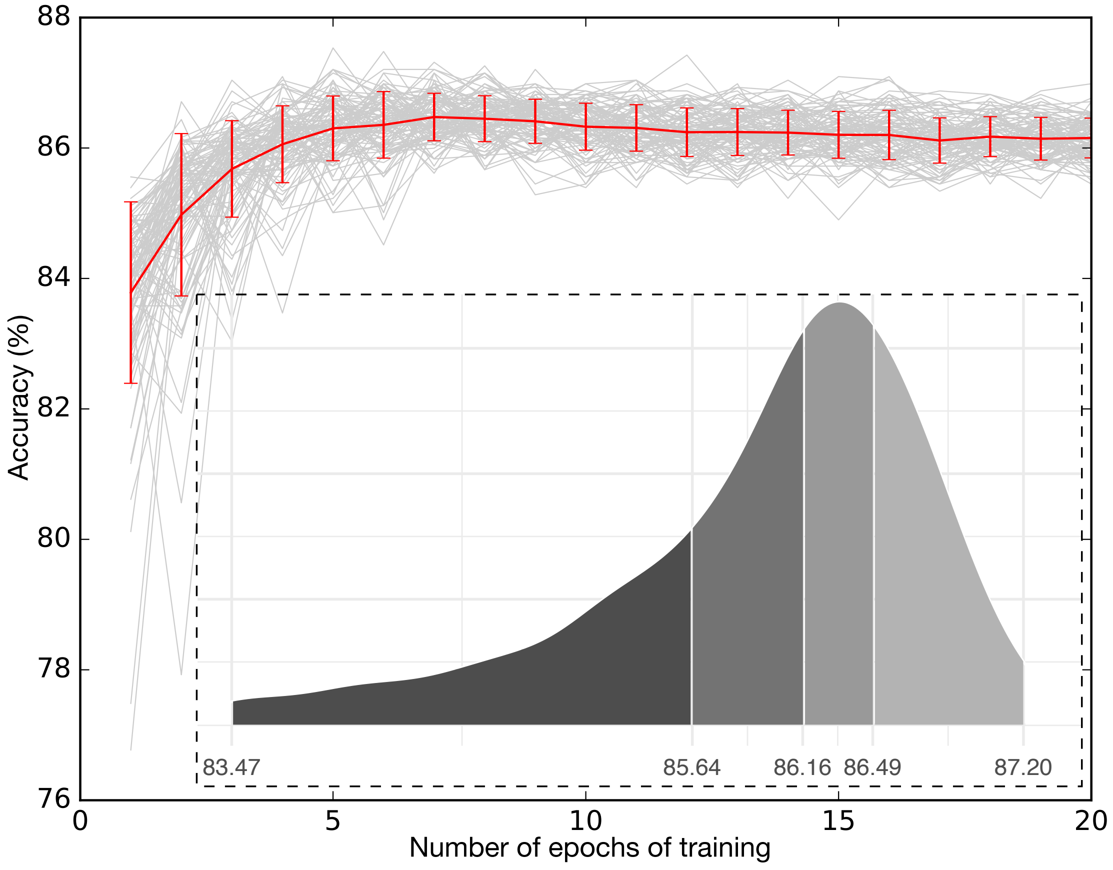
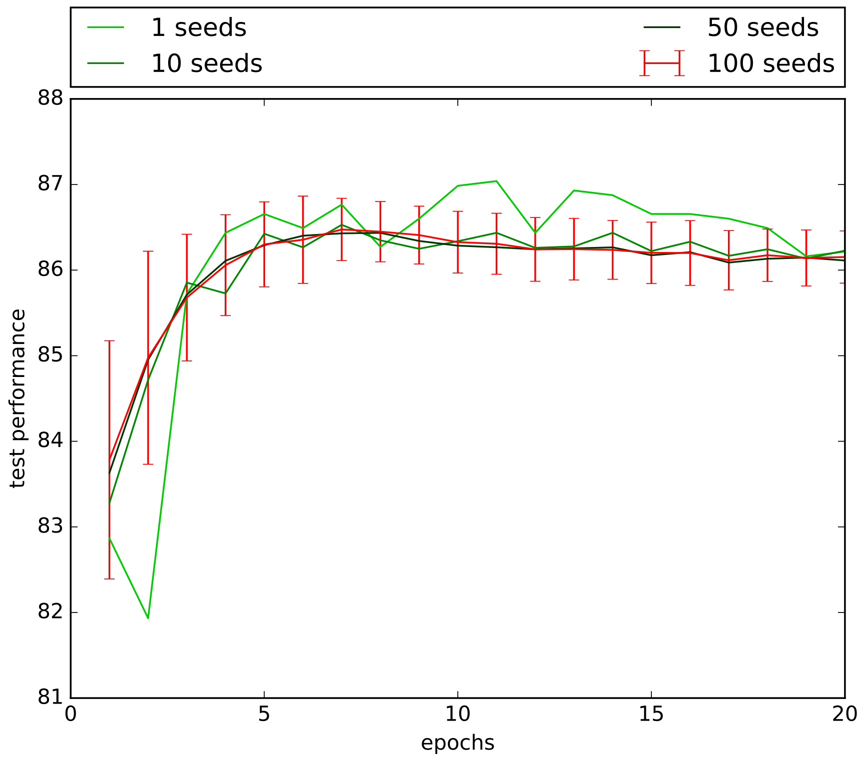


Table 6: Variance in SST-2 (sentence-only) performance in a pool of 100 runs. n is the number of seeds taken at a time.

	Dev Performance			Test Performance		
n	Worst n	Best n	Diff.	Worst n	Best n	Diff.
1	84.29 ± 0.00	86.35 ± 0.00	2.06	83.47 ± 0.00	87.20 ± 0.00	3.73
3	84.40 ± 0.09	86.28 ± 0.05	1.88	83.65 ± 0.19	87.10 ± 0.09	3.45
5	84.47 ± 0.12	86.15 ± 0.17	1.68	83.83 ± 0.26	87.04 ± 0.10	3.21
10	84.58 ± 0.14	85.95 ± 0.23	1.37	84.21 ± 0.44	86.91 ± 0.15	2.70
20	84.70 ± 0.16	85.76 ± 0.25	1.06	84.69 ± 0.58	86.80 ± 0.16	2.11
50	84.94 ± 0.24	85.54 ± 0.25	0.60	85.40 ± 0.71	86.54 ± 0.25	1.14
100	85.24 ± 0.38	85.24 ± 0.38	0.00	85.97 ± 0.78	85.97 ± 0.78	0.00

We reinforce the importance of having more seeds, we use common choices for number of seeds ($n = 1, 3, 5, 10$) and find the best and worst possible performance in the pool of 100 seeds by averaging across the chosen number of seeds. We present our results in Table 6. We notice a significant difference between the worst and best possible performance for all common choices of n. We note a similar trend in the training schedules in Figure 5, with a smooth ascent for the curve averaged across 100 seeds.

Figure 5: Test performance across epochs of training. We average performance across the specified number of seeds at each epoch.



We conclude that, to be reproducible, only averaged accuracies should be reported in this task and dataset. This mirrors the conclusion from a detailed analysis by [53] in the context of named entity recognition.

4.3 Ambiguity of Sentiment Analysis

Sentiment analysis is an inherently ambiguous task, with judgements varying from person to person. We report a crowdsourced analysis that reveals that the binary Stanford Sentiment Treebank (SST2) dataset has significant levels of ambiguity even for human labelers.

We attempt to compare the algorithms described in Section 3 across varying levels of ambiguity. We discover that ELMo's performance improvements over the baseline are robust across different levels of ambiguity, whereas the advantage of [18] is reversed in sentences of low ambiguity (restricting to *A-but-B* style sentences).

Our crowdsourced experiment was conducted on Figure Eight[5]. *Nine* workers scored the sentiment of each *A-but-B* and negation sentence in the test SST2 split as 0 (negative), 0.5 (neutral) or 1 (positive). (SST originally had *three* crowdworkers choose a sentiment rating from 1 to 25 for every phrase.) We average the scores across all users for each sentence. Corresponding to a threshold $x \in [0.5, 1)$, sentences with a score in the range $(x, 1]$ are marked as positive, in $[0, 1-x)$ marked as negative, and in $[1-x, x]$ are marked as neutral. (E.g., "*flat , but with a revelatory performance by michelle williams*" (score=0.56) is neutral when $x = 0.6$.) Higher thresholds correspond to less ambiguous sentences. Accuracy rate of a model is measured for each threshold by omitting the neutral sentences.

Table 7 shows that ELMo's performance gains in Table 4 extends across all thresholds. In Figure 6 we compare all the models on the *A-but-B* sentences in this set. Across all thresholds, we notice trends similar to previous sections: 1) ELMo performs the best among all models on *A-but-B* style sentences, and projection results in only a slight improvement; 2) models in [18] (with and without distillation) benefit considerably from projection; but 3) distillation offers little improvement (with or without projection). Also, across all models, with increasing threshold we see decreasing gains from projection. In fact, beyond the 0.85 threshold, projection degrades the average performance, indicating that projection is useful for more ambiguous sentences.

Threshold	0.50	0.66	0.75	0.90
Neutral Sentiment	10	70	95	234
Flipped Sentiment	15	4	2	0
no-distill, no-project	81.32	83.54	84.54	87.55
elmo, no-project	87.56	90.00	91.31	93.14

Table 7: Number of sentences in the crowdsourced study (447 sentences) which got marked as neutral and which got the opposite of their labels in the SST2 dataset, using various thresholds. Average accuracies of the baseline and ELMo (over 100 seeds) on non-neutral sentences are also shown.

[5] `https://www.figure-eight.com/`

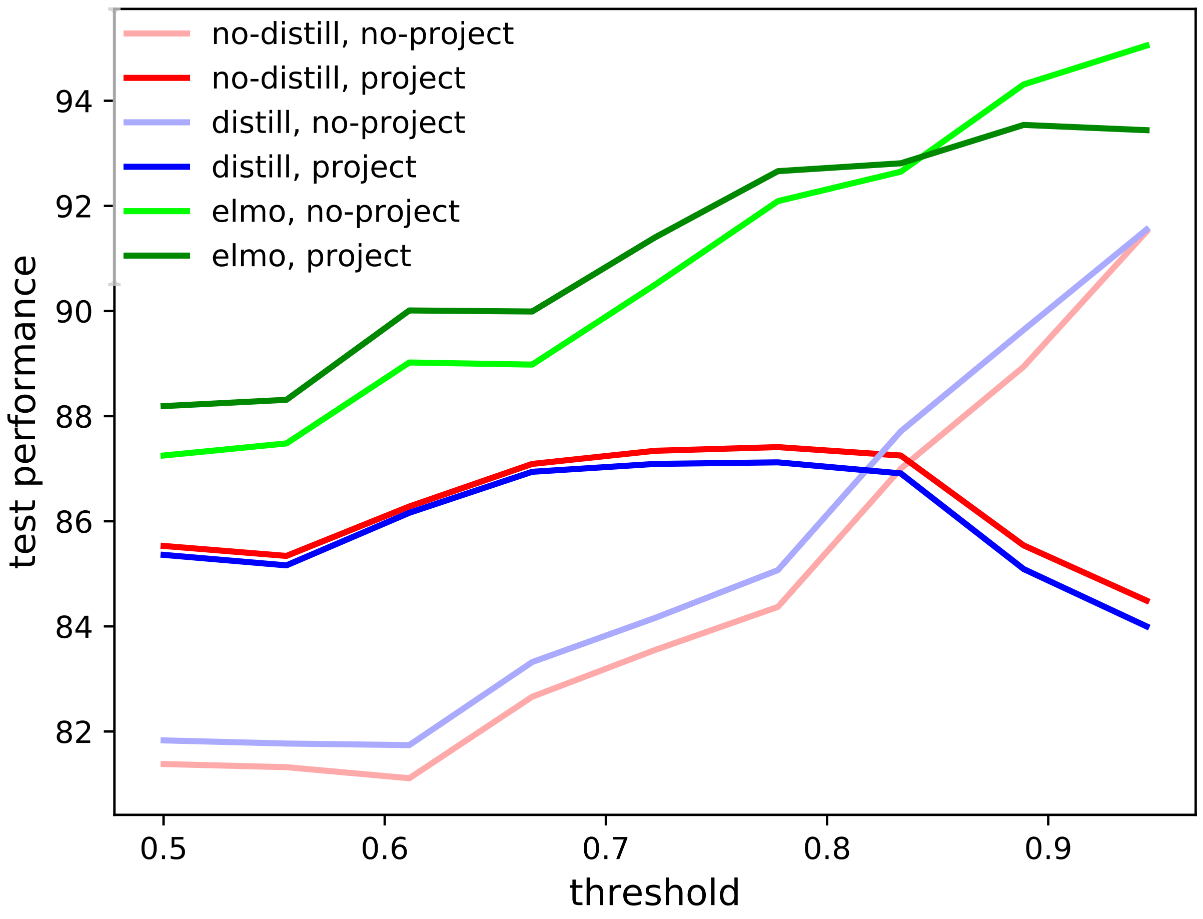


Figure 6: Average performance on the *A-but-B* part of the crowd-sourced dataset (210 sentences, 100 seeds)). For each threshold, only non-neutral sentences are used for evaluation.

5 Conclusion

We compare a suite of algorithms in language modelling and sentiment analysis which augment end-to-end systems with information learned from traditional algorithms and logic rules. All algorithms improve performance over the baseline model, with contextualized embeddings being the most effective. However, knowledge distillation does not seem beneficial as initially reported and contextualized embeddings do not explicitly encode logic rules. Input gradient techniques seem like a useful scheme to incorporate logic rules, but need further exploration.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[2] Kalpesh Krishna, Liang Lu, Kevin Gimpel, and Karen Livescu, "A study of all-convolutional encoders for connectionist temporal classification," *arXiv preprint arXiv:1710.10398*, 2017.

[3] Andrew Slavin Ross, Michael C Hughes, and Finale Doshi-Velez, "Right for the right reasons: Training differentiable models by constraining their explanations," *arXiv preprint arXiv:1703.03717*, 2017.

[4] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[5] Stanley F Chen and Joshua Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[6] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[8] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[9] Tomáš Mikolov, "Statistical language models based on neural networks," *Presentation at Google, Mountain View, 2nd April*, 2012.

[10] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[11] Gábor Melis, Chris Dyer, and Phil Blunsom, "On the state of the art of evaluation in neural language models," *arXiv preprint arXiv:1707.05589*, 2017.

[12] Stephen Merity, Nitish Shirish Keskar, and Richard Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[13] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen, "Breaking the softmax bottleneck: a high-rank rnn language model," *arXiv preprint arXiv:1711.03953*, 2017.

[14] Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric Xing, "Deep neural networks with massive learned knowledge," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1670–1679.

[15] Christopher Potts, "On the negativity of negation," in *Semantics and Linguistic Theory*, 2010, vol. 20, pp. 636–659.

[16] Yoon Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[18] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing, "Harnessing deep neural networks with logic rules," *arXiv preprint arXiv:1603.06318*, 2016.

[19] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[20] Bo Pang and Lillian Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 115–124.

[21] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng, "Semantic compositionality through recursive matrix-vector spaces," in *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics, 2012, pp. 1201–1211.

[22] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever, "Learning to generate reviews and discovering sentiment," *arXiv preprint arXiv:1704.01444*, 2017.

[23] Shen Li, Zhe Zhao, Tao Liu, Renfen Hu, and Xiaoyong Du, "Initializing convolutional filters with semantic features for text classification," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1884–1889.

[24] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[27] Graham Neubig and Chris Dyer, "Generalizing and hybridizing count-based and neural language models," *arXiv preprint arXiv:1606.00499*, 2016.

[28] Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel, "Frustratingly short attention spans in neural language modeling," *arXiv preprint arXiv:1702.04521*, 2017.

[29] Andreas Stolcke, "Srilm-an extensible language modeling toolkit," in *Seventh international conference on spoken language processing*, 2002.

[30] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., "Tensorflow: A system for large-scale machine learning.," in *OSDI*, 2016, vol. 16, pp. 265–283.

[31] Ofir Press and Lior Wolf, "Using the output embedding to improve language models," *arXiv preprint arXiv:1608.05859*, 2016.

[32] Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al., "Posterior regularization for structured latent variable models," *Journal of Machine Learning Research*, vol. 11, no. Jul, pp. 2001–2049, 2010.

[33] Michael Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002, pp. 1–8.

[34] Jeffrey Pennington, Richard Socher, and Christopher Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[35] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013.

[36] Harris Drucker and Yann Le Cun, "Improving generalization performance using double backpropagation," *IEEE Transactions on Neural Networks*, vol. 3, no. 6, pp. 991–997, 1992.

[37] Yotam Hechtlinger, "Interpretation of prediction models using the input gradient," *arXiv preprint arXiv:1611.07634*, 2016.

[38] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *See https://arxiv. org/abs/1610.02391 v3*, vol. 7, no. 8, 2016.

[39] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[40] David Belanger and Andrew McCallum, "Structured prediction energy networks," in *International Conference on Machine Learning*, 2016, pp. 983–992.

[41] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[42] Shixiang Gu and Luca Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[43] Andrew Slavin Ross and Finale Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients," *arXiv preprint arXiv:1711.09404*, 2017.

[44] II Ororbia, G Alexander, C Lee Giles, and Daniel Kifer, "Unifying adversarial training algorithms with flexible deep data gradient regularization," *arXiv preprint arXiv:1601.07213*, 2016.

[45] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues, "Robust large margin deep neural networks," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4265–4280, 2016.

[46] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017, pp. 5769–5779.

[47] Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu, "Sobolev training for neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 4281–4290.

[48] Suraj Srinivas and Francois Fleuret, "Knowledge transfer with jacobian matching," *arXiv preprint arXiv:1803.00443*, 2018.

[49] Dániel Varga, Adrián Csiszárik, and Zsolt Zombori, "Gradient regularization improves accuracy of discriminative models," *arXiv preprint arXiv:1712.09936*, 2017.

[50] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[51] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4151–4161.

[52] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[53] Nils Reimers and Iryna Gurevych, "Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging," *arXiv preprint arXiv:1707.09861*, 2017.